Федеральное государственное бюджетное учреждение науки
Институт автоматики и процессов управления
Дальневосточного отделения Российской академии наук

На правах рукописи

Крылов Дмитрий Александрович

Модели и методы реализации облачной платформы для разработки и использования интеллектуальных сервисов

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

Диссертация на соискание ученой степени
кандидата технических наук

Научный руководитель
д.ф.-м.н., профессор
Клещев Александр Сергеевич

Владивосток — 2013

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность темы. Интеллектуальные системы (ИС) или системы с базами знаний предназначены для решения таких задач, которые не могут быть решены с использованием традиционных алгоритмических методов. Главным ресурсом при решении таких задач являются профессиональные знания. Такие задачи возникают в разных предметных областях, в том числе медицине, химии, геологии, математике, технике и других.

Проблеме создания интеллектуальных систем посвящено множество работ. В них обосновывается структура ИС, состоящая из трёх компонентов: базы знаний, решателя задач и пользовательского интерфейса. К настоящему времени достигнуто понимание необходимости средств разработки ИС и повторного использования компонентов ИС.

Важной проблемой является предоставление доступа пользователей к ИС и средствам их разработки. Современный способ предоставления удобного доступа к программным системам представлен концепцией «облачных вычислений». Облачными вычислениями называются модель повсеместного сетевого доступа по требованию к общему пулу вычислительных ресурсов, которые могут быть оперативно предоставлены и освобождены с минимальными эксплуатационными затратами и обращениями к провайдеру этих ресурсов.

Прежде всего, созданы мощные редакторы информационного наполнения ИС, компоненты которых можно использовать при создании самостоятельных ИС. Разработаны и широко используются облачные платформы для программных систем общего назначения. Существующие платформы поддерживают почти весь цикл разработки облачных программных систем. При помощи таких облачных платформ возможно создание ИС, однако прямая поддержка этого не предоставляется.

Однако, на сегодняшний день нет полноценной облачной платформы, полностью поддерживающей разработку и функционирование всех трёх компонентов ИС (баз знаний, решателей задач, интеллектуального интерфейса). Соответственно, нет и поддержки всего жизненного цикла ИС. Почти все существующие облачные редакторы баз знаний ограничены механизмом редактирования снизу-вверх (от атомарных понятий к сложносоставным), не определяют порядок порождения, что не позволяет экспертам предметных областей формировать и сопровождать сложноструктурированные базы знаний без помощи посредников. Кроме того, хотя облачные платформы общего назначения и предоставляют средства синхронизации параллельно редактируемых баз данных, эти средства являются слишком низкоуровневыми для использования их при разработке ИС. Такие облачные платформы предоставляют также и средства распараллеливания программных систем, но они не являются достаточно высокоуровневыми для распараллеливания решателей ИС. Кроме того, существующие платформы (как облачные, так и не облачные) не предоставляют средства единого хранения и повторного использования информационного наполнения и компонентов ИС.

Поэтому актуальной является разработка облачной платформы, включающей средства поддержки всех компонентов облачных интеллектуальных систем, а также средств их интеграции.

Целью диссертационной работы является разработка средств для облачной поддержки, создания и использования облачных интеллектуальных систем. Для достижения этой цели необходимо решить следующие **задачи**:

1. Разработка общей концепции облачной платформы для создания облачных ИС;

2. Разработка моделей информационных ресурсов, решателя задач и интерфейса ИС;

3. Разработка методов реализации информационных ресурсов, решателей задач и интерфейсов облачных ИС;

4. Разработка технологии создания облачных ИС с использованием облачной платформы.

Методы исследования. В работе использовались методы, базирующиеся на аппарате теории графов, баз данных, системного анализа, теории формальных грамматик, искусственного интеллекта, объектно-ориентированного анализа и проектирования, теории параллельных процессов, а также методы веб-программирования.

Научная новизна работы состоит в следующем:

1. Предложена общая концепция развиваемого проекта на основе облачной платформы для создания облачных ИС, поддерживающая разработку и интеграцию всех компонентов ИС.

2. Разработаны модель информационных ресурсов ИС, структура которых специфицируется на логическом языке, обладающем согласованной логической и порождающей семантикой, модель решателей задач ИС в виде системы повторно-используемых агентов, взаимодействующих посредством сообщений, методы распараллеливания и обеспечения конфлюентности решателей задач, модель веб-интерфейса ИС.

3. Разработаны облачные методы реализации логической семантики языка описания структуры информационных ресурсов ИС, методы недетерминированной и управляемой параллельной поддержки функционирования агентов в облачной среде, методы реализации веб-интерфейса ИС, а также методы интеграции компонентов облачных ИС.

4. Разработана технология и её инструментальная поддержка создания облачных ИС на облачной платформе.

Практическая ценность и реализация результатов работы. Практическая значимость полученных в диссертационной работе результатов заключается в том, что разработанная облачная платформа IACPaaS позволяет создавать и сопровождать ИС различного назначения с использованием разработанной технологии и инструментальных средств.

На основе этой платформы и с использованием технологии разработаны инструментальные средства: информационно-административная система платформы; универсальный редактор онтологий, баз знаний и других информационных ресурсов, реализующий порождающую семантику языка описания структуры информационных ресурсов, ориентированный на работу экспертов без посредников; визуализатор информационных ресурсов; генератор облачных ИС; генератор агентов; загрузчик кода агентов; прогонщик тестов для тестирования агентов. Кроме того, на основе этой платформы, с использованием технологии и её инструментальной поддержки разработаны: облачные прикладные ИС, реализующие основные результаты диссертационной работы Л.А. Федорищева, в том числе средства разработки профессиональных виртуальных облачных сред и компьютерные обучающие тренажёры по классическим методам исследования в офтальмологии, виртуальная химическая лаборатория, виртуальная модель городского района; облачная база данных по диагностике трансформаторов.

Положения, выносимые на защиту:

1. Общая концепция развиваемого проекта на основе облачной платформы для создания облачных ИС, поддерживающая разработку и интеграцию всех компонентов ИС.

2. Модели информационных ресурсов, решателей задач и интерфейсов облачных ИС.

3. Методы реализации облачной платформы, фонда, информационных ресурсов, решателей задач и интерфейсов облачных ИС.

4. Технология разработки облачных ИС и её инструментальная поддержка на облачной платформе.

Обоснованность и достоверность полученных результатов обеспечиваются корректным применением использованных в работе методов исследования и подтверждаются эффективным практическим применением предложенных в диссертации моделей, методов и программных средств.

Апробация работы.

Основные положения диссертации докладывались и обсуждались на следующих конференциях и семинарах: «Инфокоммуникационные и вычислительные технологии и системы» (г. Улан-Удэ, 2010), Всероссийская научно практическая конференция «Информационные технологии и высокопроизводительные вычисления» (г. Хабаровск, 2013), Proceedings of the Distributed Intelligent Systems and Technologies Workshop (DIST’2013). (г. Санкт-Петербург, 2013), международный конгресс по интеллектуальным системам и информационным технологиям «IS&IT'12» (г. Москва, 2012), международная научно-техническая конференция «Открытые семантические технологии проектирования интеллектуальных систем» OSTIS-2011 (г. Минск, 2011), Конкурсе научных работ молодых учёных и специалистов ИАПУ ДВО РАН (г. Владивосток, 2010), а также на семинарах лаборатории интеллектуальных систем ИАПУ ДВО РАН (2010 – 2013 гг.).

Публикация результатов работы.

По материалам диссертации опубликовано 12 работ, из них 4 статьи в журналах, входящих в перечень ВАК РФ.

Структура и объем работы.

Диссертационная работа состоит из введения, четырёх глав, заключения, списка литературы, включающего 91 наименование, и 5 приложений. Основное содержание работы изложено на 173 страницах машинописного текста, содержит 73 рисунка.

1. ОБЛАЧНАЯ ПОДДЕРЖКА СОЗДАНИЯ И СОПРОВОЖДЕНИЯ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

В данной главе приведён обзор литературы по проблематике поддержки создания и сопровождения облачных интеллектуальных систем. В разделе 1.1 вводятся основные определения, в разделе 1.2 рассматриваются существующие в настоящее время средства разработки интеллектуальных систем, в разделе 1.3 рассматриваются средства разработки облачных программных систем, в разделе 1.4 — средства разработки облачных интеллектуальных систем. Главу завершает раздел 1.5, где приводятся выводы по проведённому обзору, обосновывается цель диссертационной работы и формулируются её задачи.

1.1. Основные понятия

Под интеллектуальной системой (далее — «ИС») в литературе [3] понимается программная система, способная решать задачи, считающиеся творческими, принадлежащие конкретной предметной области, знания о которой хранятся в памяти такой системы. Общепринятая структура ИС включает три компонента [37]: базу знаний, решатель задач, пользовательский интерфейс.

Под технологией создания ИС будем понимать совокупность методов разработки, интеграции и сопровождения этих трёх компонентов: баз знаний, решателей задач и пользовательского интерфейса. При этом особое внимание уделяется интеграции этих трёх компонентов.

Технология должна быть поддержана соответствующим программным инструментарием, который мы далее будем называть средствами разработки ИС.

Важной проблемой является предоставление доступа пользователей к программным системам (и к ИС в частности). Современным способом предоставления удобного доступа к программным системам являются «облачные вычисления» [74]. Облачными вычислениями называются модель повсеместного сетевого доступа по требованию к общему пулу вычислительных ресурсов, которые могут быть оперативно предоставлены и освобождены с минимальными эксплуатационными затратами и обращениями к провайдеру этих ресурсов [41].

В контексте модели облачных вычислений можно выделить «облачные сервисы» и «облачные платформы».

Облачные сервисы — программные системы с удалённым доступом. Обычно модель доступа к облачным сервисам в литературе называется SaaS (software as a service) [80]. Интерфейсом облачных сервисов является веб-интерфейс, и они, в целом, разрабатываются так, что пользователю для полноценной работы с ними нужен только веб-браузер и в этом их основное отличие от клиент-серверной архитектуры.

ИС, созданные с использованием модели облачных вычислений, будем называть «облачными ИС».

Облачные платформы [41] — программные системы, предназначенные для разработки и сопровождения облачных сервисов. Облачные платформы могут быть реализованы как облачные сервисы, т.е. с доступом только через веб-интерфейс, однако это необязательно. Модель предоставления услуг облачных платформ в литературе называется PaaS (Platform as a Service).

1.2. Средства разработки интеллектуальных систем

Ранним этапом развития ИС является создание и использование экспертных систем (ЭС), которые в подавляющем большинстве были основаны на базах правил [21]. Была разработана масса средств создания ЭС, чаще все-

го это были оболочки ЭС, предназначенные для дальнейшего заполнения инженерами знаний и экспертами [3]. Использование баз правил приводило к двум проблемам: 1) в базу правил приходилось включать не только предметные, но и процедурные знания о том, как достичь тех или иных результатов (ссылки); 2) реально используемые базы правил состояли из огромного количества правил, каждое из которых было понятно экспертам предметной области, но в целом понимание всей базы правил экспертами было недостижимо, что приводило к трудностям сопровождения, внутренним противоречиям и пр. [40].

Следующим шагом было появление гибридных ИС [22], которые сочетали в себе базы правил обычных ЭС и нейронные сети, методы генетических алгоритмов, нечёткой логики, статистические модели. Были созданы и средства их разработки [60], которые, как правило, представляли собой набор компонентов, из которых строилась конкретная гибридная ИС. Однако, эти системы обладали теми же недостатками, к которым прибавляется трудно-проверяемость такого вида машинного обучения.

Поэтому было достигнуто понимание, что в ИС предметные знания должны быть чётко отделены от процедурных [66].

Среди современных средства разработки ИС необходимо, прежде всего, выделить редакторы информаций разного уровня общности.

Прежде всего, это популярный редактор Protégé [48], который предоставляет возможности редактирования онтологий с моделью, предлагаемой протоколом OKBC [71]. Проект Protégé предоставляет также фреймворк (программный каркас) для построения баз знаний. Пользователям предоставляется масса плагинов (подключаемых модулей) для импорта, экспорта ресурсов (онтологий, баз знаний) в разные форматы, подключения к удалённым источникам ресурсов, визуализации ресурсов; плагины, реализующие иные машины вывода над знаниями и т.п. Проект является открытым и поддерживается значительным сообществом. Редактор Protégé, основной набор плаги-

нов, примеры онтологий доступен с официального сайта [83]. Кроме того, по построенной онтологии Protégé позволяет сгенерировать интерфейс для заполнения базы знаний.

Следует отметить также и другие редакторы онтологий, такие как, DOE [44], Neologism [67], MoKi [64], Knoodl [61], Ontowiki [70], Soboleo [79], PoolParty [72].

Перечисленные программные системы являются прежде всего редакторами разноуровневых ресурсов, и не поддерживают создание решателей (и, соответственно, не поддерживают весь жизненный цикл решателей), хотя и предоставляют программные интерфейсы для взаимодействия с ними. Вторым недостатком является порождение баз знаний «снизу-вверх», от простых, конкретных, понятий к более сложным, что может быть неудобно для экспертов.

1.3. Средства разработки облачных программных систем

Рассмотрим современные средства разработки облачных программных систем.

Современные облачные платформы, такие, как Heroku [57], Google App Engine [76], Amazon Web Services [89], Windows Azure [90] являются развитием сервисов хостинга сайтов и в настоящее время предоставляют следующие возможности:

1. Хранилище файлов,
2. Доступ к БД (как через реляционные СУБД PostgresSQL, MySQL, SQLite, так и альтернативные, NoSQL-решения, например, MongoDB или BigTable [42];
3. Запуск программ, как правило, написанных на интерпретируемых языках, таких как Ruby, Python;

4. Распределение нагрузки на несколько серверов; часто используется механизм MapReduce совместно с NoSQL СУБД;

5. Средства управления аккаунтом (консоль администратора, реализованная как веб-приложение);

6. Электронная почта;

Средства журналирования и просмотра журналов работы приложений.

Однако, существующие программные комплексы предоставляют низкоуровневые решения: низкая степень интеграции БД в программную систему, не предоставляется поддержка структурированных онтологией баз знаний. Платформы предоставляют только процедурный способ задания распараллеливания, который задаётся программистом вручную.

1.4. Средства разработки облачных интеллектуальных систем

В настоящее время среди существующих средств разработки облачных ИС можно выделить три: WebProtégé, TopBraid Live™ и Многоцелевой Банк компьютерных знаний.

Система WebProtégé [86], являющаяся ветвью развития популярного редактора онтологий Protégé, предоставляет богатые возможности разработки онтологий и баз знаний через браузер:

1. Создание новых ресурсов: онтологий и баз знаний,

2. Импорт ресурсов из разных форматов (таких как OWL, RDF),

3. Управление доступом к ресурсам,

4. Совместное редактирование, причём существует возможность просмотра внесённых каждым редактором изменений.

Однако, WebProtégé является лишь редактором и не предоставляет средств создания остальных компонентов ИС: решателей задач и интерфейса.

Компания TopQuadrant [85] предоставляет платформу TopBraid Live™ [84]. Она основана на редакторе TopBraid Composer и предоставляет сходную

с редактором Protégé функциональность, однако облачная поддержка этой платформы шире, чем у проекта WebProtégé.

Платформа предоставляет следующие возможности:

1. Прежде всего, редактор ресурсов: онтологий и баз знаний с возможностью импорта/экспорта из/в RDF, OWL и пр. Также предоставляется возможность коллективной разработки ресурсов.

2. Хранилище для ресурсов.

3. Поддержка SPARQL-запросов [77] к ресурсам в хранилище. Платформа предоставляет расширение SPIN [82], предназначенное для выполнения функций на стороне сервера (аналог хранимых функций в реляционных СУБД).

4. Возможность создавать простые приложения для взаимодействия с хранилищем. Платформа предоставляет IDE TopBraid Ensemble, которая позволяет создавать CRUD-формы для ресурсов. Эти формы могут состоять из нескольких компонентов, как простых, (например, кнопка), так и сложных, (например, древовидный редактор знаний, управляемый онтологией). Разработчик приложения конфигурирует компоненты, связывает их между собой и с ресурсами (причём для последнего связывания используется SPARQL в сочетании со SPIN), после этого TopBraid Ensemble генерирует Flex-приложение, которое выполняется в браузере клиента. Библиотека компонентов является расширяемой и конфигурироемой.

Данная платформа является расширением проекта TopBraid Composer для облаков. Проект поддерживает совместимость между этим расширением и стандартным, не облачным, инструментарием.

Однако, эта платформа больше ориентирована на представление и редактирование знаний, а не на полноценную поддержку создания ИС. Создание сложных решателей задач на языке SPARQL/SPIN представляется очень трудоёмким. Кроме того, существуют технические ограничения — например,

не вполне ясно, можно ли ускорить выполнение сложных SPARQL/SPIN-запросов за счёт распределения нагрузки на компьютерный кластер.

Проект МБкЗ [29] поддерживает весь жизненный цикл разработки информационного наполнения компьютерных банков знаний: получение, инженерия, хранение, управление и использование знаний. В рамках проекта [25] разработана концепция информационных ресурсов разного уровня общности, разработан редактор ИРУО (Информации Различного Уровня Общности) [26], используемый для редактирования этих инфоресурсов и веб-сайт для управления единицами информационного наполнения.

Важное преимущество редактора ИРУО состоит в том, что он устранял проблему Protégé–подобных редакторов, предоставляя возможность редактирования «сверху-вниз»: от абстракций предметных областей к конкретным терминам.

Однако, данный проект концентрировался вокруг обработки инфоресурсов, не предоставляя полноценной поддержки разработки остальных двух компонентов ИС — решателей и пользовательского интерфейса. Соответственно, не было предоставлено никакой поддержки всего жизненного цикла ИС. Программный интерфейс к информационному наполнению (в рамках проекта МБкЗ называемый «оболочкой») не предоставлял контроля соответствия метаинфоресурсов и порождаемых по ним ресурсов.

Кроме того, редактор ИРУО был реализован в виде Java-программы, что требовало установки как самого редактора, так и JVM (ссылка) на компьютер пользователя — поэтому редактор ИРУО не является облачным. Ещё одним недостатком редактора является его редактирование «в одной точке», когда пользователю виден только один элемент редактируемого информационного ресурса, из-за чего пользователь при редактировании «теряет» всю картину редактируемого инфоресурса. Отсюда же следовал другой недостаток: навигация также была одношаговой, и пользователю приходилось совершить много действий в интерфейсе редактора, чтобы перейти от одного фрагмента

инфоресурса к другому. В целом, такой подход может быть оправдан при линейном вводе данных от эксперта, но при редактировании уже существующего инфоресурса эти недостатки были особенно заметны.

Таким образом, разработка средств поддержки облачных программных систем является актуальной.

1.5. Выводы из обзора, обоснование цели и задач работы

Проведённый обзор литературы позволяет сделать следующие выводы:

1. Проблеме создания интеллектуальных систем посвящено множество работ. В этих работах обосновывается структура ИС, состоящая из трёх компонентов: баз знаний, решателя задач и пользовательского интерфейса.

2. Достигнуто понимание необходимости средств разработки ИС и повторного использования компонентов ИС. Прежде всего, созданы мощные редакторы информационного наполнения ИС, компоненты которых можно использовать при создании самостоятельных ИС.

3. В настоящее время разработаны и широко используются облачные платформы для программных систем общего назначения. Существующие платформы поддерживают почти весь цикл разработки облачных программных систем. При помощи таких облачных платформ возможно создание ИС, однако прямая поддержка этого не предоставляется.

Однако, у существующих систем присутствует ряд недостатков:

1. Нет полноценной облачной платформы, полностью поддерживающей все три компонента ИС (баз знаний, решателей, пользовательского интерфейса). Соответственно, нет и поддержки всего жизненного цикла ИС.

2. Рассмотренные редакторы ограничены редактированием снизу-вверх (от атомарных понятий к сложносоставным), не определяют порядок порождения, что является неудобным для экспертов предметных областей (исключением является редактор ИРУО проекта МБкЗ).

3. Хотя облачные платформы и предоставляют средства синхронизации параллельно редактируемых баз данных, эти средства являются слишком низкоуровневыми для использования их при разработке ИС. Точно также облачные платформы предоставляют средства распараллеливания программных систем, но они не являются достаточно высокоуровневыми для распараллеливания решателей ИС.

4. Существующие платформы (как облачные, так и не облачные) не предоставляют концепцию единого хранения и повторного использования информационного наполнения и компонентов ИС.

Поэтому актуальной является разработка облачной платформы, включающей средства поддержки всех компонентов облачных интеллектуальных систем, а также средств их интеграции.

Целью диссертационной работы является разработка средств для облачной поддержки, создания и использования облачных интеллектуальных систем. Для достижения этой цели необходимо решить следующие **задачи**:

1. Разработка общей концепции облачной платформы для создания облачных ИС;

2. Разработка моделей информационных ресурсов, решателя задач, интерфейса ИС;

3. Разработка методов реализации информационных ресурсов, решателей задач, интерфейсов облачных ИС;

4. Разработка технологии создания облачных ИС с использованием облачной платформы.

2. ОБЩАЯ КОНЦЕПЦИЯ ОБЛАЧНОЙ ПЛАТФОРМЫ И ОСНОВНЫЕ МОДЕЛИ

В настоящей главе решаются первые две задачи диссертационной работы: вводится общая концепция облачной платформы и приводятся модели информационных ресурсов, решателя задач, интерфейсов.

В разделе 2.1 рассматривается концепция всего комплекса и вводятся основные понятия. В разделе 2.2 рассматривается способ представления информационных ресурсов разного уровня общности. В разделе 2.3 рассматривается модель решателей задач. В разделе 2.4 предлагается модель построения пользовательского интерфейса.

2.1. Проект IACPaaS. Комплекс для разработки и использования интеллектуальных систем на основе облачных вычислений

2.1.1. Основные требования к проекту IACPaaS

Основной целью проекта IACPaaS является накопление интеллектуальных сервисов и информационных сущностей разного рода [8, 14, 15] в едином информационном пространстве, называемом **Фондом**. Фонд предназначен для непрерывного развития: в нем могут появляться новые сервисы, новые информационные единицы, обрабатываемые сервисами, существующие сервисы улучшаться, неактуальные — удаляться.

Перечислим основные требования, предъявляемыми к проекту:

1. Обеспечение доступа через Интернет к функциональности интеллектуальных систем без передачи пользовательских версий. Эта идея является одной из основополагающих для технологии облачных вычислений — предоставление пользователям сервисов вместо предоставления им непо-

средственно версий программных систем для установки на их компьютерах. Преимущества данного подхода широко обсуждаются в литературе [38, 74]. Дополнительное преимущество от использования данной технологии — возможность управления интеллектуальными системами в процессе их жизненного цикла [7], что подразумевает изменение с помощью высокоуровневых механизмов функциональных свойств интеллектуальных систем в соответствии с постоянно изменяющимися текущими требованиями пользователей, условиями эксплуатации, знаниями предметной области.

2. Создание единой среды для функционирования интеллектуальных систем, инструментальных средств разработки и управления ими. Многолетний опыт лаборатории интеллектуальных систем ИАПУ ДВО РАН [2] показал, что: во-первых, средства разработки интеллектуальных систем и управления ими, как правило, также являются интеллектуальными системами; во-вторых, прикладные и инструментальные интеллектуальные системы в ряде случаев используют одни и те же общие информационные ресурсы, а значит, с точки зрения функционирования, между ними нет различия. Поэтому прикладные и инструментальные интеллектуальные системы могут функционировать в единой среде.

3. Поддержка контролируемого доступа к функциональным возможностям программно-информационного комплекса и единой системы администрирования правами на использование прикладных и инструментальных систем. Контролируемый доступ к функциональным возможностям программно-информационного комплекса и единая система администрирования, прежде всего, подразумевает предотвращение несанкционированного доступа к интеллектуальным системам, средствам их разработки и управления ими. Большинство интеллектуальных систем предназначено для использования ограниченным кругом лиц (например, системы медицинской диагностики — врачами, а не больными, интеллектуальные системы в области химии — специалистами в области химии). Поэтому администраторы програм-

мно-информационного комплекса должны иметь информацию о пользователях интеллектуальных систем и целях использования систем. Разработка новых интеллектуальных систем в рамках программно-информационного комплекса должна соответствовать политике, поддерживаемой её администраторами, что предполагает контроль за использованием средств разработки. Очевидно, что такой контроль необходим и по отношению к средствам управления интеллектуальными системами.

4. Поддержка идеологии накопления и развития как интеллектуальных и инструментальных систем в целом, так и отдельных их компонентов. Контролируемый доступ и единая система администрирования программно-информационным комплексом, направлены на реализацию идеологии накопления и развития как непосредственно «законченных» прикладных и инструментальных интеллектуальных систем, так и отдельных их компонентов, на основе которых могут быть созданы новые прикладные и инструментальные интеллектуальные системы. Такими «отдельными» компонентами, прежде всего, являются информационные ресурсы различных уровней общности (базы знаний и данных, онтологии и метаонтологии); агенты, являющиеся составной частью решателей задач и выполняющие общие, независимые от конкретной системы вычисления над данными; шаблонные компоненты пользовательского интерфейса, которые могут быть как проблемно-зависимыми, так и не зависеть от специфики задачи, а определяться требованиями удобства использования.

5. Постепенная замена средств разработки прикладных интеллектуальных систем средствами управления ими. Основная идея, лежащая в основе управления программными средствами, заключается в том, чтобы рассматривать их разработку как начало управления ими, подобно тому, как сопровождение программных средств обычно рассматривается как продолжение их разработки [7, 69]. Таким образом, существующие модели облачных вычис-

лений дополняются ещё одной — управление как сервис (Control as a Service — CaaS) [27].

6. Создание условий для кооперативной деятельности пользователей интеллектуальных систем, экспертов, специалистов предметных областей, управляющих интеллектуальными системами, и программистов. Одним из важнейших аспектов жизни людей являются попытки разрешения противоречия между индивидуальным и коллективным, в частности, в профессиональной деятельности. В области вычислительной техники наиболее ярким примером является сосуществование концепций персонального компьютера и глобальной сети Интернет. Облачные вычисления являются ещё одной попыткой разрешения этого противоречия, когда при коллективном использовании ресурсов для каждого пользователя создаётся иллюзия его персонального обслуживания. Дальнейшим развитием этой концепции является поддержка кооперативной деятельности всех её участников, которая может проявляться не только в создании повторно используемых программных и информационных компонентов, но и управлении уже находящимися в эксплуатации программными средствами на основе результатов мониторинга процесса их использования.

2.1.2. Основные требования к Платформе IACPaaS и её концептуальная архитектура

Программная система, лежащая в основе Проекта, называется **Платформой IACPaaS** (или просто Платформой) [30].

Итак, проект IACPaaS — это непрекращающийся процесс развития Фонда, а Платформа IACPaaS есть программная поддержка для этого проекта, которая, будучи один раз созданной, дальше подвергается только сопровождению.

Платформа должна предоставлять услуги доступа:

1. Специалистам различных предметных областей к функциональности интеллектуальных систем;

2. Разработчикам интеллектуальных систем к средствам их разработки (проблемно-ориентированным и проблемно-независимым);

3. Управляющим интеллектуальными системами — к средствам управления ими.

Разработчиками и управляющими интеллектуальных систем могут выступать коллективы для создания интеллектуальных систем в различных предметных областях и управления ими.

Основные архитектурные компоненты программного обеспечения проекта IACPaaS приведены на Рис. 1. Интернет-комплекс состоит из трёх основных программных компонентов сайта, виртуальной машины и Фонда.

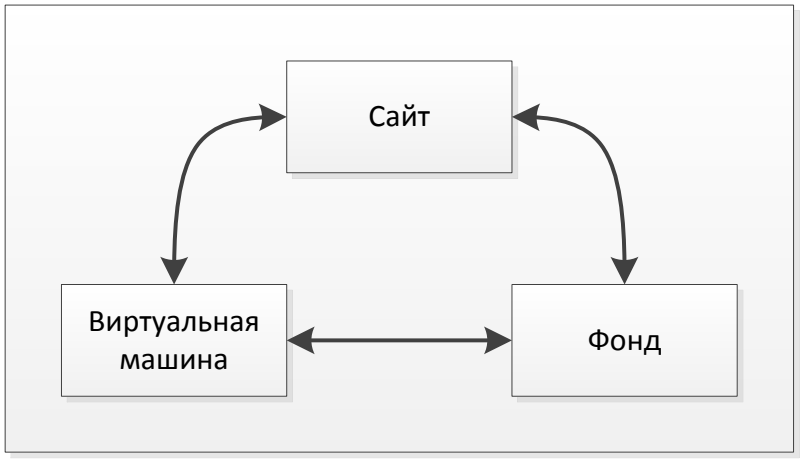

Рис. 1. Концептуальная архитектура проекта IACPaaS

Сайт предназначен для всех пользователей проекта. Через него они могут просматривать доступное им содержимое Фонда; подавать заявки на регистрацию, модификацию Фонда, а также получать и реализовывать свои полномочия.

Фонд составляет содержимое, интеллектуальное наполнение проекта; для удобства навигации он разделён на предметные области, а те, в свою очередь на разделы. Каждый раздел содержит относящиеся к нему единицы хранения:

прикладные и инструментальные средства (средства разработки и управления), агенты, информационные ресурсы, компоненты интерфейса.

Виртуальная машина представляет собой набор процессоров для запуска и выполнения средств администрирования, а также для реализации полномочий пользователей: интеллектуальных систем, средств разработки и управления.

2.1.2.1. Сайт проекта IACPaaS

Через сайт все пользователи могут просмотреть общую информацию о проекте и новости, просмотреть доступное содержимое фонда, получать и реализовать свои полномочия, а также взаимодействовать с сопровождающим персоналом проекта или некоторого раздела Фонда.

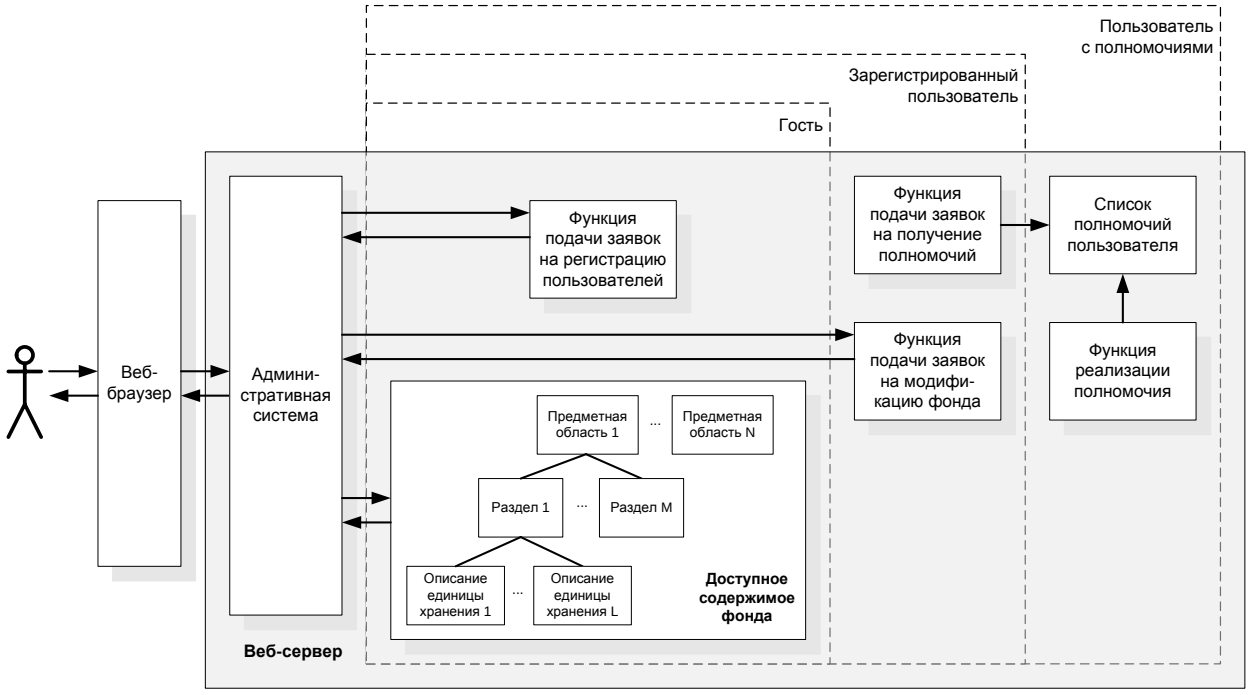

Рис. 2. Типы пользователей проекта IACPaaS

Существует три основных типа пользователей проекта IACPaaS (Рис. 2):

1. *Гость* может просматривать доступное ему содержимое Фонда — предметные области, их разделы и описания связанных с ними единиц хра-

нения, а также может подать заявку на регистрацию в одной из предметных областей через соответствующее средство администрирования;

2. *Зарегистрированный пользователь* обладает правами гостя на просмотр содержимого Фонда, на регистрацию в других предметных областях, а также может подать заявку на получение полномочий в тех предметных областях, в которых он уже зарегистрирован, или заявку на модификацию Фонда;

3. *Пользователь с полномочиями* обладает всеми правами зарегистрированного пользователя, а также может реализовывать полномочия из своего списка полномочий. Его полномочия могут относиться к следующим возможным классам: пользователь интеллектуальной системы, разработчик и сопровождающий кода программных компонентов, управляющий декларативной компонентой агента, информационными ресурсами, коллектив управляющих прикладным или инструментальным средством, а также администратор проекта IACPaaS и администратор предметной области (Рис. 3).

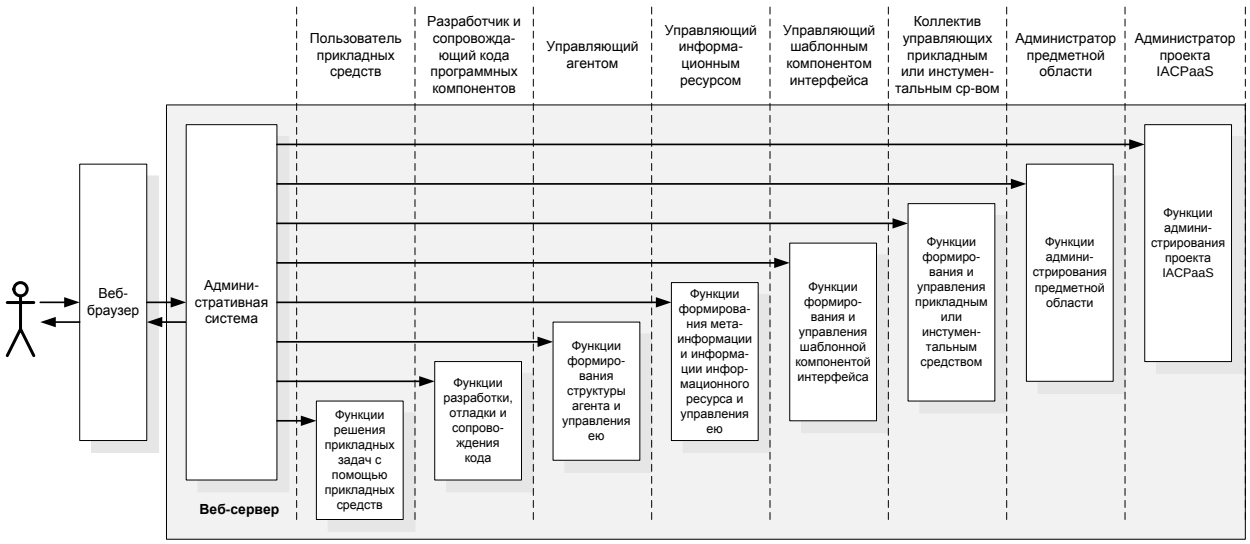

Рис. 3. Классы пользователей с полномочиями проекта IACPaaS

2.1.2.2. Фонд проекта IACPaaS

Основными функциями Фонда является аккумулирование в едином информационном пространстве информационных ресурсов различных типов, а

также поддержка их коллективного развития в различных предметных областях для решения задач в практической, научной и образовательной деятельности.

Информационные ресурсы фонда могут быть проблемно-зависимыми и проблемно-независимыми. Проблемно-зависимыми являются прикладные интеллектуальные системы, онтологии и метаонтологии предметных областей, базы знаний и данных. Проблемно-независимыми являются инструментальные системы, предназначенные для создания интеллектуальных систем в различных предметных областях и управления ими, а также проблемно-независимые информационные ресурсы, например, онтологии пользовательского интерфейса.

Программные компоненты — агенты, интеллектуальные сервисы — также представляются информационными ресурсами со специальным содержимым.

2.1.2.3. Виртуальная машина проекта IACPaaS

Виртуальная машина проекта IACPaaS состоит из процессора информационных ресурсов, процессора решателей задач и процессора пользовательских интерфейсов, каждый из которых представляет собой набор функций для поддержки соответствующих компонентов интеллектуальных систем (Рис. 4).

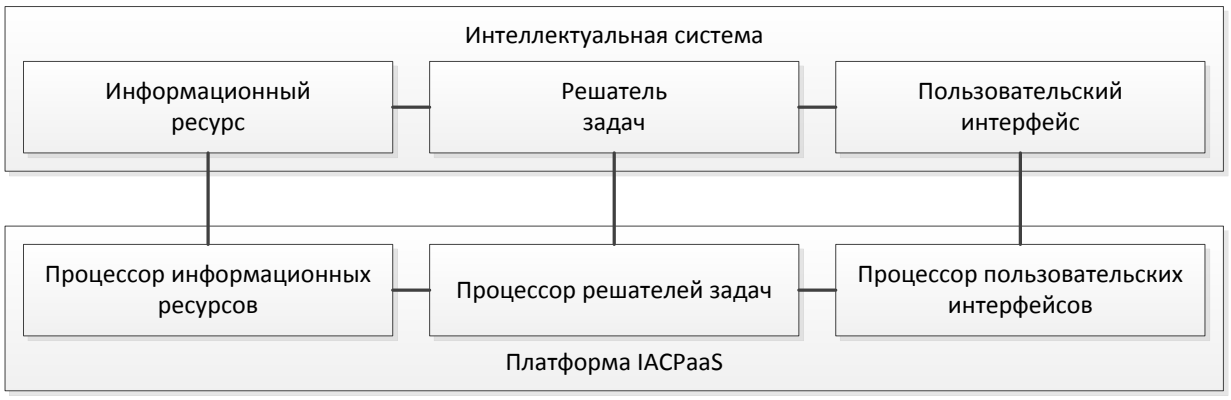

Рис. 4. Связь виртуальной машины с интеллектуальной системой

Процессор информационных ресурсов представляет собой набор функций обработки информационных ресурсов, доступный разработчикам и сопровождающим программных компонентов, хранимых в фонде. Процессор решателей задач обеспечивает запуск полномочий пользователей и средств администрирования, их завершение, приостановку и возобновление выполнения, осуществляет выполнение кода решателей задач, взаимодействие между его компонентами, реализованными как совокупность агентов. Процессор пользовательского интерфейса обеспечивает диалог с пользователем на основе информации из его модели и информации, полученной от решателя задач.

2.2. Модель информационных ресурсов

2.2.1. Требования

В работе [24] проанализированы уровни информации и особенности её редактирования, из чего следуют *принципиальные* требования:

Требование 1. Информационные ресурсы разных уровней общности должны представляться семантическими сетями [81]. Напомним, что семантическая сеть есть граф, т.е. набор вершин (представляющих понятия), и дуг (представляющих отношения между понятиями) между ними. Представлением инфоресурсов семантическими сетями достигается универсальность их обработки.

Требование 2. Спецификация классов информационных ресурсов должна задаваться метаинформацией. Любой информационный ресурс должен принадлежать некоторому классу. Так как информационный ресурс есть граф, состоящий из вершин и дуг, то метаинформация должна специфицировать эти вершины и дуги.

Требование 3 (требование локальности). Каждой вершине информационного ресурса должна соответствовать одна и только одна вершина мета-

информации, множество исходящих дуг метаинформации должно соответствовать множеству исходящих дуг информации.

Требование 4. Метаинформация должна использоваться для управления пользовательским интерфейсом при порождении или редактировании соответствующего ей информационного ресурса.

Требование 5 (контекстная зависимость для повторного использования). Метаинформация должна содержать средства для указания повторно-используемых фрагментов других информационных ресурсов с подходящей структурой.

2.2.2. Концепция информационных ресурсов и метаинформации

В соответствии с **требованием 2**, всё множество инфоресурсов разделено на классы, определяемые метаинформациями [6, 56]. Каждый такой класс инфоресурсов порождается исчислением, правила порождения которого определяются метаинформацией. Иными словами, метаинформация задаёт порождающую семантику для инфоресурсов.

Для определения множества метаинформаций введено исчисление формул метаинформаций (см. раздел 2.2.4). Формулы с поименованной начальной вершиной есть метаинформации.

Формулы имеют логическую семантику: формула истинна для тех и только тех инфоресурсов, которые принадлежат классу, задаваемому этой метаинформацией.

Формулы также имеют порождающую семантику: начальным состоянием порождающего процесса является инфоресурс в начальном состоянии, единственная вершина которого является активной; с каждой вершиной порождаемого инфоресурса связывается формула метаинформации; на очередном шаге порождающего процесса из множества активных вершин выбирается одна, из которой выполняется очередной шаг порождения в соответствие с формулой, связанной с этой вершиной; при этом вершина, из которой

выполнен шаг порождения, перестаёт быть активной, но могут возникнуть новые активные вершины; порождающий процесс заканчивается, когда в порождаемом графе нет ни одной активной вершины.

Логическая и порождающая семантики метаинформации связаны следующим естественным условием — по каждой формуле метаинформации может быть порождено множество тех и только тех информационных ресурсов, относительно которых эта формула истинна.

2.2.3. Информационные ресурсы

Введём информационные ресурсы (далее «инфоресурсы») следующим образом:

Инфоресурс есть граф, задаваемый четвёркой $<C, R, A, root>$, где

C — есть множество вершин графа, называемых «понятиями» (concepts).

R — есть множество дуг графа, называемых «отношениями» (relations), каждая дуга есть тройка $<c_i, c_j, r^m>$, где c_i — начальное понятие, из которого исходит отношение, c_j — конечное понятие, в которое входит отношение, r^m — метаотношение, по которому порождено это отношение.

A — множество активных вершин. Каждая активная вершина есть пара $<c, c^m>$, где c — понятие инфоресурса i, c^m — метапонятие некоторой метаинформации (см. ниже «Порождение инфоресурса»).

root — корень инфоресурса, пара $<c_0, c^m>$, где c_0 — некоторое понятие инфоресурса, называемое корневым, c^m — некоторое метапонятие некоторой метаинформации.

Если инфоресурс состоит из одного корневого понятия, то будем говорить, что этот инфоресурс в начальном состоянии.

2.2.4. Метаинформация

Метаинформация есть формула без переменных, начальная вершина которой имеет метку — название.

2.2.4.1. Синтаксис формул без переменных

Формула без переменных может относиться к одной из следующих групп:

1. Простая формула без переменных;
2. Простая кванторная формула без переменных;
3. Унарная формула без переменных;
4. Пропозициональная формула без переменных;
5. Структурная кванторная формула без переменных.

Далее при описании каждой группы будем сопоставлять свой графический образ.

Простая формула без переменных есть граф, состоящий из единственной вершины с меткой *c* (Рис. 5). Эту единственную вершину будем называть начальной вершиной простой формулы без переменных. Метка *c* является константой.

Простая кванторная формула без переменных есть граф, состоящий из единственной вершины с меткой, имеющей вид *QMT*, где *Q* — знак квантора, *M* — описание (конечного или бесконечного) множества, а *T* — термин (Рис. 6). Знаком квантора может быть: ∀ (для всех), ∃ (существует), ∃2 (существует не менее двух), ∃? (существует, но не для всех), ∃! (существует и единственен), ∃[] (существует подынтервал). Множество кванторов является расширяемым.

Рис. 5. Простая формула без переменных

Рис. 6. Простая кванторная формула без переменных

Описание конечного множества может иметь вид $\{c_1, \ldots, c_n\}$, где $c_1, \ldots, c_n$ — попарно различные константы, либо быть целым конечным интервалом;

в этом случае знаком квантора может быть ∀, ∃, ∃2, ∃? или ∃!. Описание бесконечного множества может быть названием сорта, неименованным множеством «*», или вещественным конечным интервалом; в этом случае знаком квантора может быть ∃, ∃2, ∃!. Если знак квантора есть ∃[], то описанием множества может быть только конечный целый или вещественный интервал.

Названием сорта может быть: «строка», «целый», «вещественный», «целый интервал», «вещественный интервал», «время», «булевый». Множество сортов является расширяемым.

Унарная формула без переменных есть граф, состоящий из начальной вершины и дуги с меткой *T* (термином), выходящей из начальной вершины и входящей в начальную вершину некоторой формулы без переменных *F* (Рис. 7).

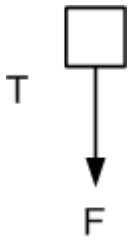

Рис. 7. Унарная формула без переменных

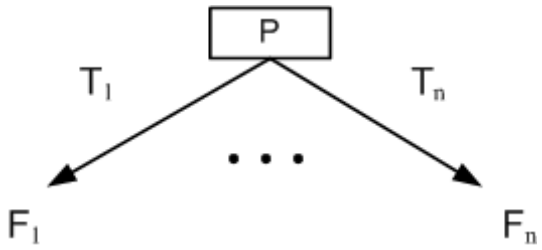

Рис. 8. Пропозициональная формула без переменных

Пропозициональная формула без переменных есть граф, состоящий из начальной вершины с пропозициональной меткой *P* и выходящих из неё *n* дуг (не менее двух), каждая из которых имеет метку T_i (термин; *i* от 1 до *n*; все эти метки должны быть попарно различны) и входит в начальную вершину некоторой формулы без переменных F_i (Рис. 8).

Пропозициональными метками P являются:

& — конъюнкция,

∨ — дизъюнкция,

| — исключающее или.

Множество пропозициональных меток является расширяемым.

Любая из дуг, выходящая из начальной вершины с пропозициональной меткой &, может иметь метку факультативности «[]».

Структурная кванторная формула без переменных состоит из начальной вершины с меткой, имеющей вид *QMT*, где *Q* — знак квантора, *M* — описание (конечного или бесконечного) множества, а *T* — термин, и формулы без переменных *F*, начальная вершина которой изображается внутри начальной вершины структурной кванторной формулы без переменных (Рис. 9). Описание конечного множества может иметь вид $\{c_1, \ldots, c_n\}$, где $c_1, \ldots, c_n$— попарно различные константы, либо быть целым конечным интервалом; в этом случае знаком квантора может быть ∀, ∃, ∃2, ∃?. Описание бесконечного множества может быть названием сорта, неименованным множеством «*», или вещественным конечным интервалом; в этом случае знаком квантора может быть ∃, ∃2.

Рис. 9. Структурная кванторная формула без переменных

Рис. 10. Семантика простой формулы без переменных

Рис. 11. Семантика простой кванторной формулы без переменных с квантором∀

Рис. 12. Семантика простой кванторной формулы без переменных с квантором ∃

2.2.4.2. Логическая семантика формул без переменных

Формула без переменных *F* истинна относительно графа *N*, если *F* истинна на корне *R* графа *N*, а метка входной вершины формулы без переменных *F* совпадает с меткой класса корня *R* графа *N*.

Простая формула без переменных F (Рис. 10) истинна на вершине *V* графа *N*, если *V* является терминальной и простой, а её меткой является константа c.

Если в ***простой кванторной формуле без переменных*** (Рис. 5) описание конечного множества имеет вид $\{c_1, \ldots, c_n\}$, где $c_1, \ldots, c_n$ — попарно различные константы, то конечное множество состоит из элементов $c_1, \ldots, c_n$; если же описание конечного множества есть конечный целый интервал, то конечное множество состоит из всех целых констант, принадлежащих этому интервалу. Если описание бесконечного множества есть название сорта, то бесконечное множество состоит из констант, представляющих все значения этого сорта; если описание бесконечного множества есть «*», то бесконечное множество состоит из всех констант специального вида. Если описание бесконечного множества есть конечный вещественный интервал, то бесконечное множество состоит из всех вещественных констант, принадлежащих этому интервалу.

Если метка входной вершины ***простой кванторной формулы без переменных*** F (Рис. 11) имеет вид $\forall$ FS T, где FS — описание конечного множества $\{d_1, \ldots, d_n\}$, а T — термин, то F истинна на вершине V графа N, если V является структурной и терминальной вершиной, содержащей n графов, каждая из которых состоит из одного корня с меткой класса — T и индивидуальными метками — элементами множества $\{d_1, \ldots, d_n\}$.

Если метка входной вершины ***простой кванторной формулы без переменных*** F (Рис. 12) имеет вид $\exists$ M T, где M — описание (конечного или бесконечного) множества, а T — термин, то F истинна на вершине V графа N,

если V является структурной и терминальной вершиной, содержащей непустое множество графов, каждая из которых состоит из одного корня с меткой класса — T и индивидуальной меткой — одним из элементов множества (индивидуальные метки попарно различны).

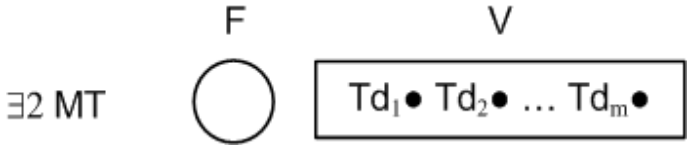

Рис. 13. Семантика простой кванторной формулы без переменных с квантором ∃2

Если метка входной вершины ***простой кванторной формулы без переменных*** F (Рис. 13) имеет вид ∃2 M T, где M — описание (конечного или бесконечного) множества, а T — термин, то F истинна на вершине V графа N, если V является структурной и терминальной вершиной, содержащей не менее двух графов, каждая из которых состоит из одного корня с меткой класса — T и индивидуальной меткой — одним из элементов множества (индивидуальные метки попарно различны).

Рис. 14. Семантика простой кванторной формулы без переменных с квантором ∃?

Если метка входной вершины ***простой кванторной формулы без переменных*** F (Рис. 14) имеет вид ∃? FS T, где FS — описание конечного множества $\{d_1, \ldots, d_n\}$, а T — термин, то F истинна на вершине V графа N, если V является структурной и терминальной вершиной, содержащей менее, чем n графов, каждая из которых состоит из одного корня с меткой класса — T и попарно различными индивидуальными метками — элементами множества $\{d_1, \ldots, d_n\}$.

Рис. 15. Семантика простой кванторной формулы без переменных с квантором ∃!

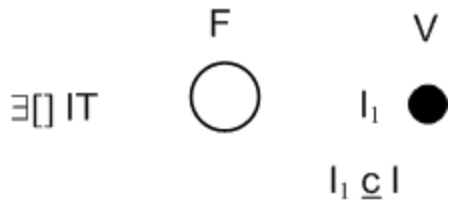

Рис. 16. Семантика простой кванторной формулы без переменных с квантором ∃[]

Если метка входной вершины ***простой кванторной формулы без переменных*** F (Рис. 15) имеет вид ∃! M T, где M — описание (конечного или бесконечного) множества, а T — термин, то F истинна на вершине V графа N, если V является простой и терминальной вершиной, а её меткой является один из элементов множества.

Если метка входной вершины ***простой кванторной формулы без переменных*** F (Рис. 16) имеет вид ∃[] I T, где I — конечный целый **(вещественный) интервал, а T — термин,** то F истинна на вершине V графа N, если V является простой и терминальной вершиной, а её меткой является целый (вещественный) интервал, являющийся подынтервалом интервала I.

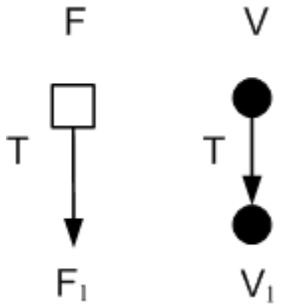

Рис. 17. Семантика унарной формулы без переменных

Унарная формула без переменных F (Рис. 17) истинна на вершине V графа N, если V является простой вершиной, из неё выходит в точности одна дуга с меткой T и эта дуга входит в вершину V_1, на которой истинна формула без переменных F_1.

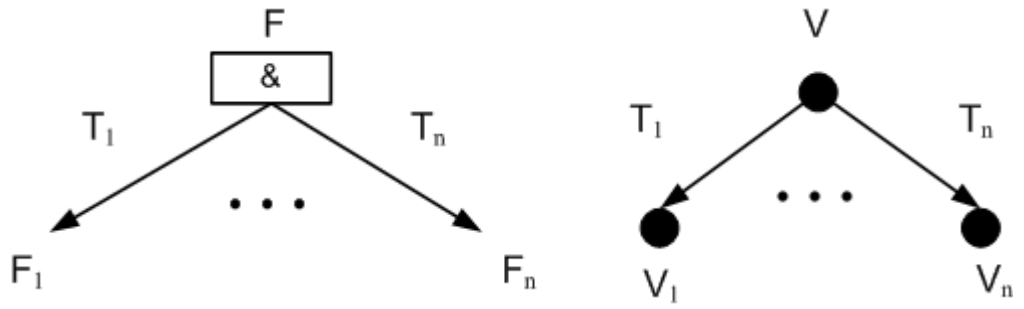

Рис. 18. Семантика конъюнкции без переменных

Если пропозициональной меткой начальной вершины W ***пропозициональной формулы без переменных*** F (Рис. 18) является &, а некоторые дуги, выходящие из W, имеют метки [], то F истинна на вершине V графа N, если V является простой вершиной, для каждой дуги с меткой T_i, выходящей из V, существует дуга с такой же меткой T_i, выходящая из W, для каждой дуги с меткой T_i, выходящей из W и не имеющей метки [], существует дуга с такой же меткой T_i, выходящая из V, а каждая формула без переменных F_i истинна на вершине, в которую входит дуга с меткой T_i, выходящая из вершины V.

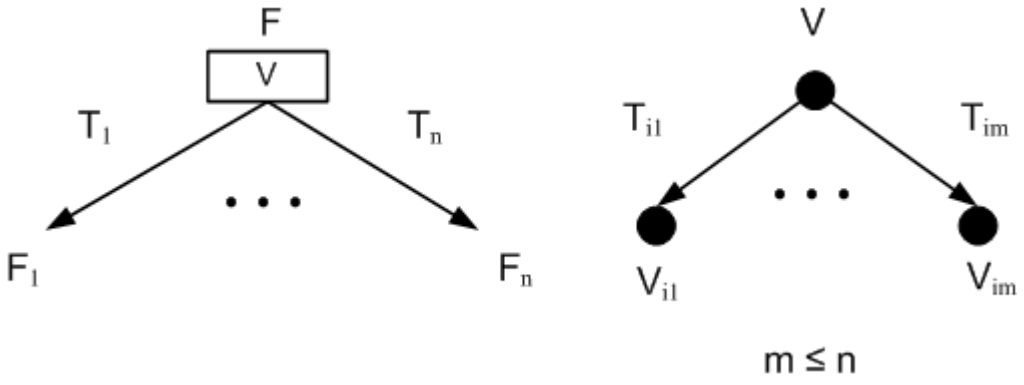

Рис. 19. Семантика дизъюнкции без переменных

Если пропозициональной меткой начальной вершины W ***пропозициональной формулы без переменных*** F (Рис. 19) является $\vee$, то F истинна на вершине V графа N, если V является простой вершиной, существует подмножество дуг, выходящих из W, между которым и множеством дуг, выходящих из V, существует взаимно-однозначное соответствие, при котором метки T_i соответствующих дуг совпадают, а формулы без переменных F_i, в

начальные вершины которых входят дуги, выходящие из W, истинны на вершинах, в которые входят соответствующие дуги с метками T_i, выходящие из V.

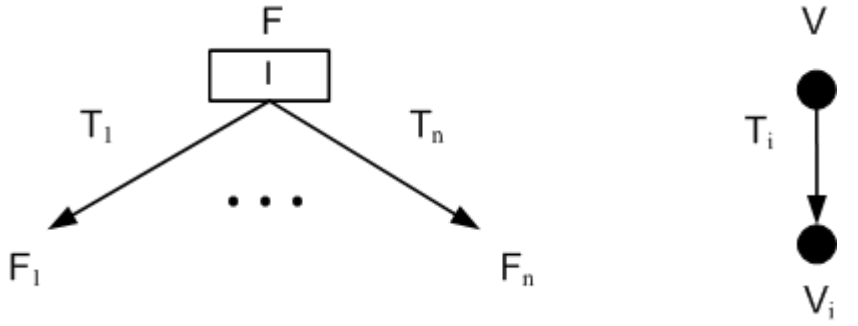

Рис. 20. Семантика исключающего ИЛИ без переменных

Если пропозициональной меткой начальной вершины W ***пропозициональной формулы без переменных*** F (Рис. 20) является |, то F истинна на вершине V графа N, если V является простой вершиной, из неё выходит единственная дуга, существует дуга, выходящая из W, такая что метки T_i этих дуг совпадают, а формула без переменных F_i, в начальную вершину которой входит дуга с меткой T_i, выходящая из W, истинна на вершине, в которую входит дуга с меткой T_i, выходящая из V.

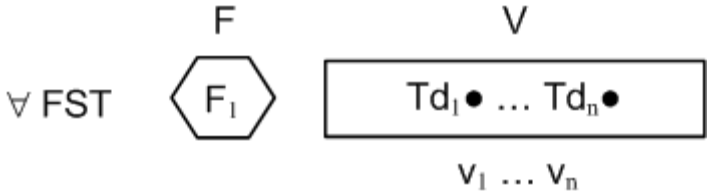

Рис. 21. Семантика структурной кванторной формулы без переменных F с квантором ∀

Если метка входной вершины ***структурной кванторной формулы без переменных*** F (Рис. 21) имеет вид ∀ FS T, где FS — описание конечного множества $\{d_1, \dots, d_n\}$, а T — термин, то F истинна на вершине V графа N, если V является структурной вершиной, содержащей n графов, каждая из которых имеет метку класса — T и индивидуальные метки — попарно различ-

ные элементы множества $\{d_1, \dots, d_n\}$, и на корне каждой из которых истинна формула без переменных F_1.

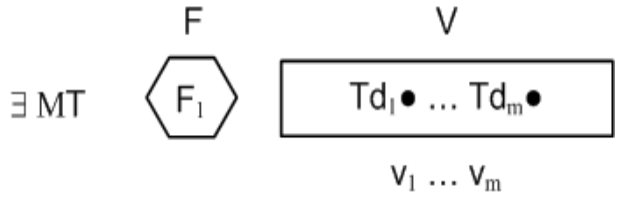

Рис. 22. Семантика структурной кванторной формулы без переменных F с квантором ∃

Если метка входной вершины ***структурной кванторной формулы без переменных*** F (Рис. 22) имеет вид ∃ M T, где M — описание (конечного или бесконечного) множества, а T — термин, то F истинна на вершине V графа N, если V является структурной вершиной, содержащей непустое множество графов, каждая из которых имеет метку класса — T и индивидуальные метки — попарно различные элементы множества, и на корне каждой из которых истинна формула без переменных F_1.

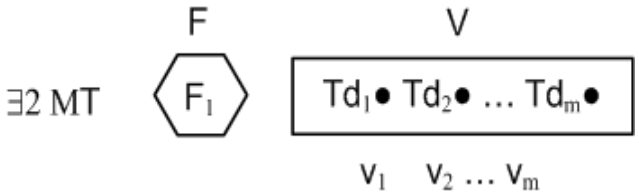

Рис. 23. Семантика структурной кванторной формулы без переменных F с квантором ∃2

Если метка входной вершины ***структурной кванторной формулы без переменных*** F (Рис. 23) имеет вид ∃2 M T, где M — описание (конечного или бесконечного) множества, а T — термин, то F истинна на вершине V графа N, если V является структурной вершиной, содержащей не менее двух графов, каждая из которых имеет метку класса — T и индивидуальные метки — попарно различные элементы множества, и на корне каждой из которых истинна формула без переменных F_1.

Рис. 24. Семантика структурной кванторной формулы без переменных F с квантором ∃?

Если метка входной вершины ***структурной кванторной формулы без переменных*** F (Рис. 24) имеет вид ∃? FS T, где FS — описание конечного множества $\{d_1, \dots, d_n\}$, а T — термин, то F истинна на вершине V графа N, если V является структурной вершиной, содержащей менее, чем n графов, каждая из которых имеет метку класса — T и индивидуальные метки — попарно различные элементы множества $\{d_1, \dots, d_n\}$, и на корне каждой из которых истинна формула без переменных F_1.

2.2.4.3. Порождающая семантика формул без переменных

Граф N порождается по формуле без переменных F в ходе следующего порождающего процесса.

На первом шаге порождается корень графа N, метка класса которого совпадает с меткой начальной вершины формулы без переменных F, а индивидуальная метка порождается. Корень графа N становится её активной вершиной, а начальная вершина формулы без переменных F ставится ей в соответствие.

Если активной вершине V графа N соответствует начальная вершина ***простой формулы без переменных*** F, то вершина V становится простой и терминальной с меткой c (Рис. 10). После этого вершина V перестаёт быть активной.

Если активной вершине V графа N соответствует начальная вершина ***простой кванторной формулы без переменных*** F (Рис. 11), метка которой

имеет вид ∀ FS T, где FS — описание конечного множества $\{d_1, \dots, d_n\}$, а T — термин, то вершина V становится структурной и терминальной вершиной, в которой, как в контейнере, порождается n графов, каждая из которых состоит из одного корня с меткой класса — T и индивидуальными метками — элементами множества $\{d_1, \dots, d_n\}$.

Если активной вершине V графа N соответствует начальная вершина ***простой кванторной формулы без переменных*** F (Рис. 12), метка которой имеет вид ∃ M T, где M — описание (конечного или бесконечного) множества, а T — термин, то вершина V становится структурной и терминальной вершиной, в которой, как в контейнере, порождается непустое конечное множество графов, каждая из которых состоит из одного корня с меткой класса — T и индивидуальной меткой — одним из элементов множества (индивидуальные метки попарно различны).

Если активной вершине V графа N соответствует начальная вершина ***простой кванторной формулы без переменных*** F (Рис. 13), метка которой имеет вид ∃2 M T, где M — описание (конечного или бесконечного) множества, а T — термин, то вершина V становится структурной и терминальной вершиной, в которой, как в контейнере, порождается не менее двух графов, каждая из которых состоит из одного корня с меткой класса — T и индивидуальной меткой — одним из элементов множества (индивидуальные метки попарно различны).

Если активной вершине V графа N соответствует начальная вершина ***простой кванторной формулы без переменных*** F (Рис. 14), метка которой имеет вид ∃? FS T, где FS — описание конечного множества $\{d_1, \dots, d_n\}$, а T — термин, то вершина V становится структурной и терминальной вершиной, в которой, как в контейнере, порождается менее, чем n графов, каждая из которых состоит из одного корня с меткой класса — T и попарно различными индивидуальными метками — элементами множества $\{d_1, \dots, d_n\}$.

Если активной вершине V графа N соответствует начальная вершина ***простой кванторной формулы без переменных*** F (Рис. 15), метка которой имеет вид ∃! M T, где M — описание (конечного или бесконечного) множества, а T — термин, то вершина V становится простой и терминальной вершиной, а её меткой — один из элементов множества.

Если активной вершине V графа N соответствует начальная вершина ***простой кванторной формулы без переменных*** F (Рис. 16), метка которой имеет ∃[] I T, где I — конечный целый (вещественный) интервал, а T — термин, то вершина V становится простой и терминальной вершиной, а её меткой — целый (вещественный) интервал, являющийся подынтервалом интервала I.

После этого эта вершина V перестаёт быть активной.

Если активной вершине V графа N соответствует начальная вершина ***унарной формулы без переменных*** F (Рис. 17), то в графа N порождается дуга с меткой T, выходящая из вершины V. Вершина V перестаёт быть активной, а вершина, в которую эта дуга входит, становится активной. Ей ставится в соответствие вершина, в которую входит дуга в унарной формуле без переменных F.

Если активной вершине V графа N соответствует начальная вершина W ***пропозициональной формулы без переменных*** F (Рис. 18), вершина W имеет пропозициональную метку &, из вершины W выходит n дуг с метками T_1, ..., T_n, то вершина V становится простой, порождается n дуг, выходящих из вершины V, с теми же метками. Вершина V перестаёт быть активной, а вершины, в которые эти дуги входят, становятся активными. Им ставятся в соответствие вершины, в которые входят дуги с теми же метками в пропозициональной формуле без переменных F. Если дуга в пропозициональной формуле без переменных F имеет метку факультативности, то дуга в графе N с той же меткой может и не порождаться. Если все дуги в пропозициональной формуле без переменных F имеют метки факультативности и в графе не по-

рождается ни одной дуги, выходящей из вершины V, то такая ситуация считается ошибочной.

Если активной вершине V графа N соответствует начальная вершина W ***пропозициональной формулы без переменных*** F (Рис. 19), вершина W имеет пропозициональную метку $\vee$, из вершины W выходит n дуг с метками T_1, ..., T_n, то в графе N порождаются дуги числом, не большим n, выходящие из вершины V, а метки этих дуг являются некоторыми попарно различными элементами множества T_1, ..., T_n. Вершина V перестает быть активной, а вершины, в которые входят порождённые дуги, становятся активными. Им ставятся в соответствие вершины, в которые входят дуги с теми же метками в пропозициональной формуле без переменных F.

Если активной вершине V графа N соответствует начальная вершина W ***пропозициональной формулы без переменных*** F (Рис. 20), вершина W имеет пропозициональную метку |, из вершины W выходит n дуг с метками T_1, ..., T_n, то в графе N порождается дуга, выходящая из вершины V с меткой — одним из элементов множества T_1, ..., T_n. Вершина V перестаёт быть активной, а вершина, в которую входит порождённая дуга, становится активной. Ей ставится в соответствие вершина, в которую входит дуга с той же меткой в пропозициональной формуле без переменных F.

Если активной вершине V графа N соответствует начальная вершина W ***структурной кванторной формулы без переменных*** F (Рис. 21), метка которой имеет вид $\forall$ FS T, где FS — описание конечного множества $\{d_1, \dots, d_n\}$, а T — термин, то вершина V становится структурной вершиной, в которой, как в контейнере, порождается n вершин, каждая из которых имеет метку класса — T и индивидуальную метку — элемент множества $\{d_1, \dots, d_n\}$ (индивидуальные метки попарно различны). Вершина V перестаёт быть активной, а вновь порождённые вершины становятся активными. Каждой из них ставится в соответствие начальная вершина формулы без переменных F_1.

Если активной вершине V графа N соответствует начальная вершина W ***структурной кванторной формулы без переменных*** F (Рис. 22), метка которой имеет вид ∃ M T, где M — описание (конечного или бесконечного) множества, а T — термин, то вершина V становится структурной вершиной, в которой, как в контейнере, порождается конечное множество вершин, каждая из которых имеет метку класса — T и индивидуальную метку — элемент множества (индивидуальные метки попарно различны). Вершина V перестаёт быть активной, а вновь порождённые вершины становятся активными. Каждой из них ставится в соответствие начальная вершина формулы без переменных F_1.

Если активной вершине V графа N соответствует начальная вершина W ***структурной кванторной формулы без переменных*** F (Рис. 23), метка которой имеет вид ∃2 M T, где M — описание (конечного или бесконечного) множества, а T — термин, то вершина V становится структурной вершиной, в которой, как в контейнере, порождается не менее двух вершин, каждая из которых имеет метку класса — T и индивидуальную метку — элемент множества (индивидуальные метки попарно различны). Вершина V перестаёт быть активной, а вновь порождённые вершины становятся активными. Каждой из них ставится в соответствие начальная вершина формулы без переменных F_1.

Если активной вершине V графа N соответствует начальная вершина W ***структурной кванторной формулы без переменных*** F (Рис. 24), метка которой имеет вид ∃? FS T, где FS — описание конечного множества $\{d_1, \ldots, d_n\}$, а T — термин, то вершина V становится структурной вершиной, в которой, как в контейнере, порождается менее n вершин, каждая из которых имеет метку класса — T и индивидуальную метку — элемент множества $\{d_1, \ldots, d_n\}$ (индивидуальные метки попарно различны). Вершина V перестаёт быть активной, а вновь порождённые вершины становятся активными. Каж-

дой из них ставится в соответствие начальная вершина формулы без переменных F_1.

Метаинформации, определённые в настоящем разделе, является контекстно-свободным в том смысле, что при порождении графа N из любой её активной вершины V, соответствующей начальной вершине формулы без переменных F, нет указаний на использование других фрагментов этой или иной графа.

Пример 1. На Рис. 25 представлена контекстно-свободная метаинформация простых баз наблюдений [33], а на Рис. 26 — граф, порождённая по этой метаинформации. Информация (простая база наблюдений) состоит из множества наблюдений, каждое из которых имеет уникальное название и возможные значения — множество названий этих значений. Граф, представляющая эту информацию, состоит из одной структурной вершины с меткой класса «База наблюдений» и индивидуальной меткой — названием конкретной базы наблюдений (в данном примере — «эндокринология»). Эта структурная вершина, как контейнер, содержит упорядоченное множество графов. Корень каждой графа является простой вершиной с меткой класса «наблюдение» и индивидуальной меткой — названием наблюдения (в примере — «аппетит», «алопеция», …). Из этого корня выходит единственная дуга с меткой «Возможные значения», входящая в структурную вершину, которая, как контейнер, содержит упорядоченное множество графов. Каждый граф состоит только из корня — простой вершины с меткой класса «значение» и индивидуальной меткой — названием значения.

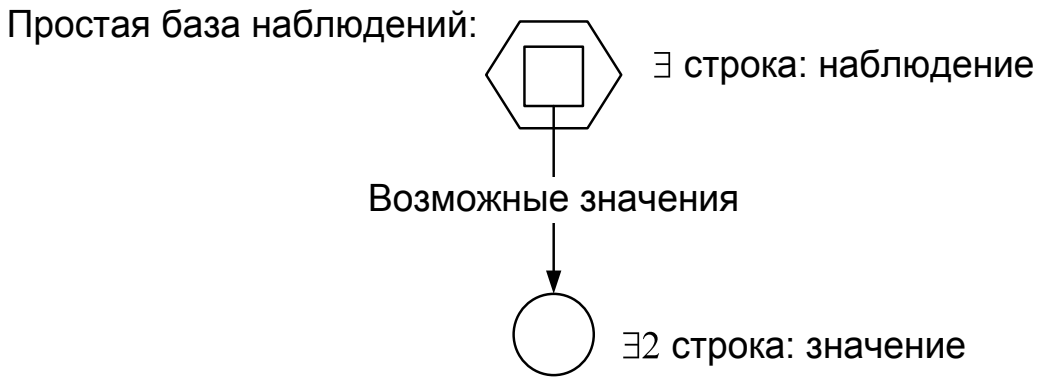

Рис. 25. Контекстно-свободная метаинформация простых баз наблюдений

Пример 2. На Рис. 27 представлена контекстно-свободная метаинформация реальных баз наблюдений [33]. Реальная база наблюдений представляет собой рекурсивную структуру. Она состоит из групп и/или наблюдений, причём каждая группа, в свою очередь, состоит из групп и/или наблюдений. Наблюдение может иметь качественные значения, количественные (целые или вещественные) значения или составные значения. Составные значения состоят из характеристик, каждая из которых может иметь качественные значения, количественные (целые или вещественные) значения или составные значения.

Простая база наблюдений: эндокринология

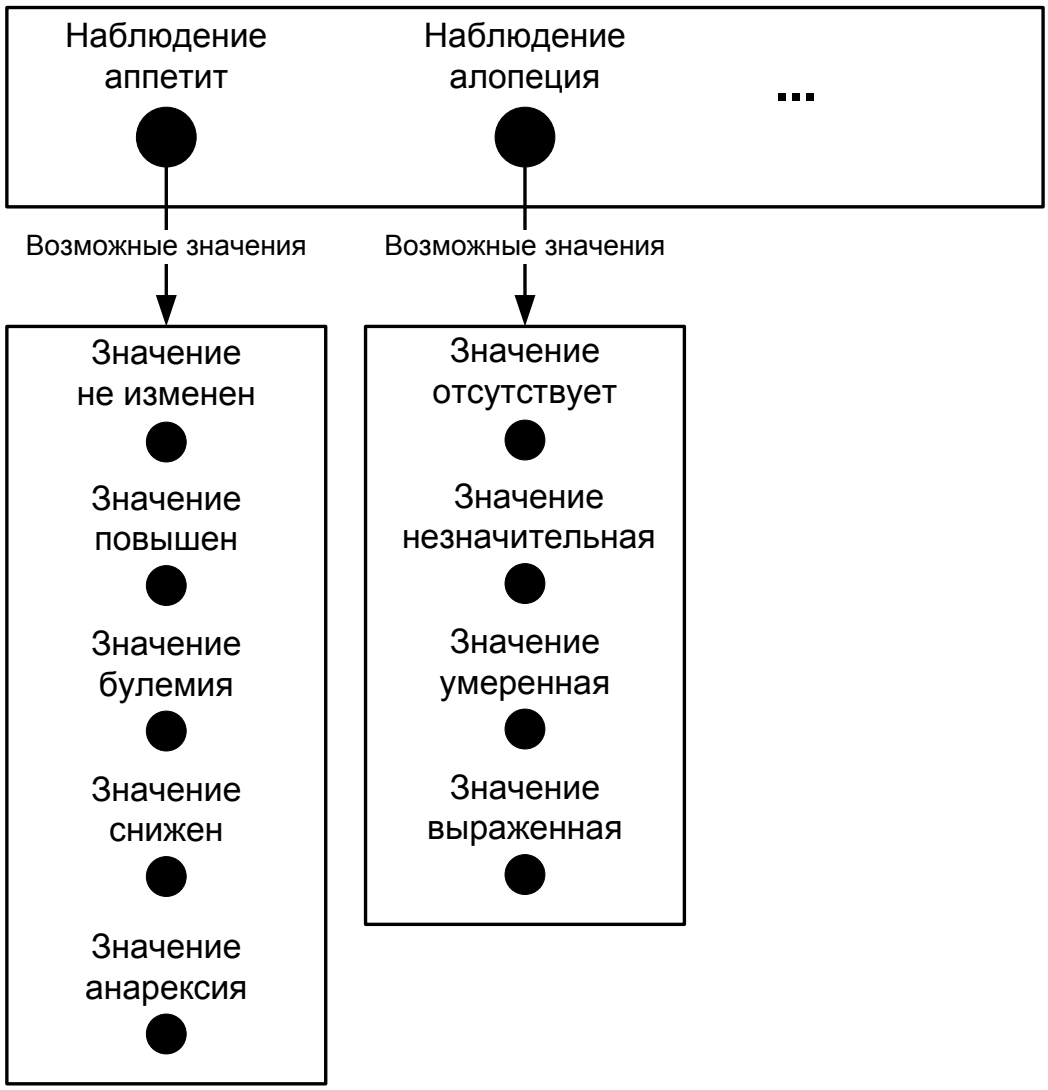

Рис. 26. Иерархическая семантическая сеть, представляющая простую базу наблюдений «эндокринология»

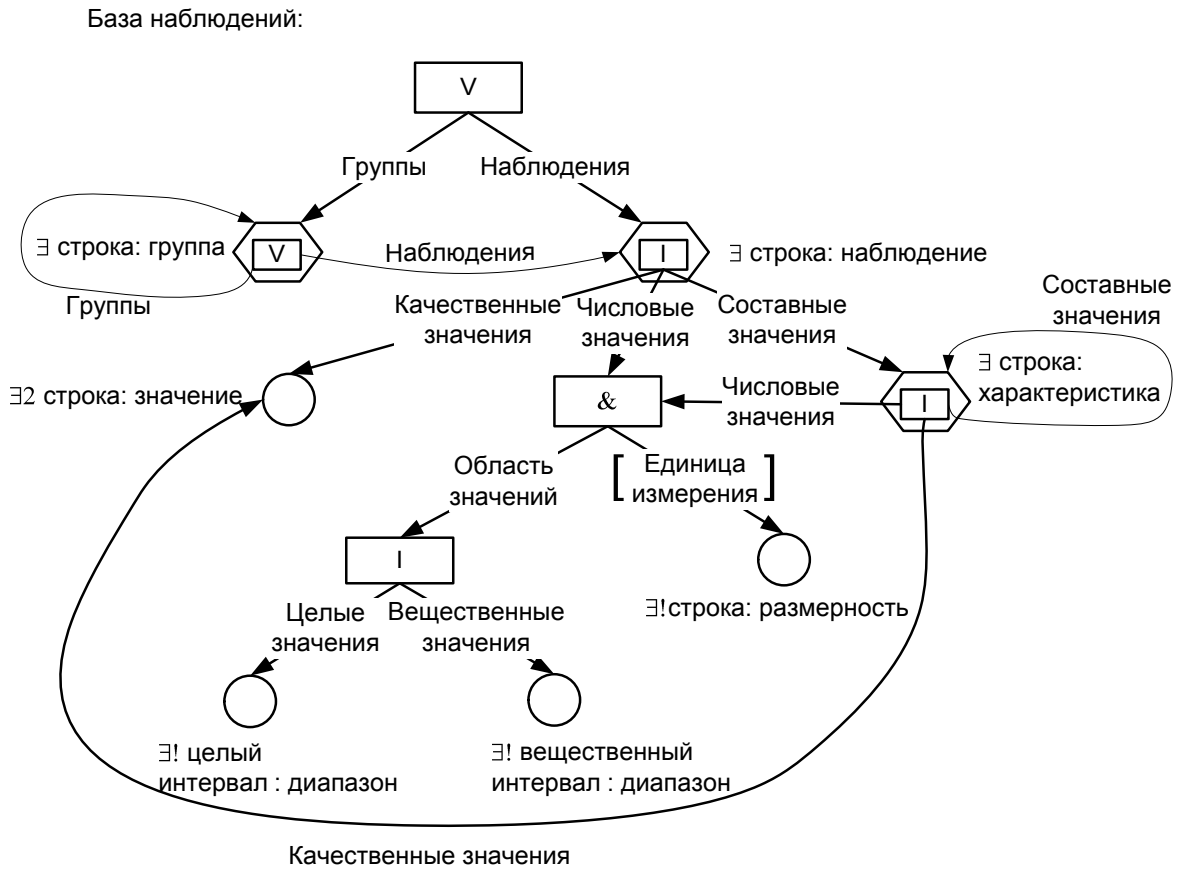

Рис. 27. Контекстно-свободная метаинформация реальных баз наблюдений

2.2.4.4. Контекстно-зависимая метаинформация

Для представления контекстно-зависимых метаинформаций (**требование 5**) введён новый тип формул — конечное множество импликаций с взаимоисключающими антецедентами, а также переменные, входящие в импликации [5, 55]. При этом контекст понимается в широком смысле: контекстом могут служить как любые фрагменты данного (порождаемого) инфоресурса, так и любые фрагменты других (ранее порождённых) инфоресурсов, принадлежащие тому же, или другим метаинформациям.

Контекстно-зависимая метаинформация есть формула общего вида метаинформации, начальная вершина которой имеет метку — название этой метаинформации. Формула общего вида есть либо формула без переменных, либо унарная формула общего вида, либо пропозициональная формула обще-

го вида, либо структурная кванторная формула общего вида либо конечное множество импликаций с взаимоисключающими антецедентами.

2.2.4.5. Синтаксис формул общего вида

Унарная формула общего вида имеет тот же вид, что и на Рис. 7, но F_1 есть формула общего вида.

Пропозициональная формула общего вида имеет тот же вид, что и на Рис. 8, но F_1, …, F_n являются формулами общего вида.

Структурная кванторная формула общего вида имеет тот же вид, что и на Рис. 9, но F есть формула общего вида.

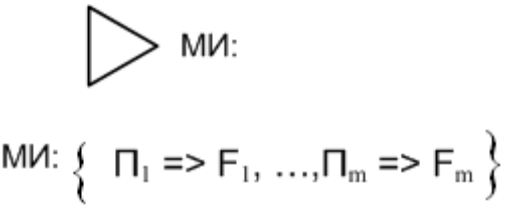

Рис. 28. Множество импликаций

Конечное множество импликаций есть граф, состоящий из одной вершины с меткой $\{П_1 \Rightarrow F_1, \ldots, П_m \Rightarrow F_m\}$, где $П_1$, …, $П_m$ —антецеденты, а F_1, …, F_m — консеквенты импликаций (Рис. 28). Антецедент импликации есть конечное множество компонент — формул с переменными, каждая из которых может иметь префикс. Префикс есть название некоторой метаинформации. Консеквентом импликации является формула с переменными общего вида. Любая переменная входящая в консеквент импликации, должна входить и в её антецедент.

Формулой с переменными (общего вида) может быть простая формула с переменной, простая кванторная формула с переменной, унарная формула с переменными (общего вида), пропозициональная формула с переменными (общего вида) и структурная кванторная формула с переменными (общего вида).

Простая формула с переменной имеет тот же вид, что и на Рис. 5, но метка c может быть и переменной v, и элементом значения переменной v*.

Простая кванторная формула с переменной имеет тот же вид, что и на Рис. 6, но описание множества M может быть переменной v.

Унарная формула с переменными (общего вида) имеет тот же вид, что и на Рис. 7, но F_1 является формулой с переменными (общего вида).

Пропозициональная формула с переменными (общего вида) имеет тот же вид, что и на Рис. 8, но F_1, …, F_n являются формулами с переменными (общего вида). Если пропозициональная формула с переменными и пропозициональной меткой «&» входит в антецедент импликации, то дуги не могут иметь меток факультативности «[]», но могут иметь метки отрицания «¬».

Структурная кванторная формула с переменными (общего вида) имеет тот же вид, что и на Рис. 9, но описание множества M может быть переменной, а F — формулой с переменными (общего вида).

2.2.4.6. Логическая семантика формул общего вида

Множество импликаций с меткой $\{\Pi_1 \Rightarrow F_1, \ldots, \Pi_m \Rightarrow F_m\}$ истинно на вершине V графа N, если существует единственное $i \in [1, m]$ такое, что только импликация $\Pi_i \Rightarrow F_i$ истинна на вершине V графа N, а антецеденты всех других импликаций из этого множества не являются истинными на вершине V графа N при любых подстановках значений вместо переменных, входящих в эти антецеденты.

Импликация $\Pi \Rightarrow F$ истинна на вершине V графа N, если существует такая подстановка значений вместо переменных, входящих в импликацию, при которой на вершине V графа N истинны и антецедент, и консеквент импликации. Антецедент импликации истинен на вершине V графа N при некоторой подстановке, если при этой подстановке истинна каждая компонента этого антецедента на вершине V графа N. Если компонента антецедента импликации не имеет префикса, то эта компонента истинна на вершине V графа

N при некоторой подстановке, если в порождаемой графа N существует хотя бы одна вершина V_1, на которой истинна формула с переменными, являющаяся этой компонентой антецедента импликации, при этой подстановке. Если компонента антецедента импликации имеет префикс и этот префикс идентифицирует уже существующий другой граф N_1, то эта компонента истинна на вершине V графа N при некоторой подстановке, если в графа N_1 существует хотя бы одна вершина V_1, на которой истинна формула с переменными, являющаяся этой компонентой антецедента импликации, при этой подстановке.

Простая формула с переменной F, меткой которой является переменная v, истинна на вершине V графа N при подстановке θ, если V является терминальной и простой, а её метка есть константа c, являющаяся значением переменной v в подстановке θ. Если меткой простой формулы с переменной F является элемент значения переменной v*, а значением переменной v в подстановке θ является множество значений, то формула F истинна на вершине V графа N при подстановке θ, если V является терминальной и простой, а её метка есть константа c, являющаяся одним из элементов значения переменной v в подстановке θ.

Если метка входной вершины ***простой кванторной формулы с переменной*** F имеет вид ∀ v T или метка входной вершины обобщённой простой кванторной формулы F, входящей в антецедент импликации, имеет вид ∃ v T, где v — переменная, а T — термин, то F истинна на вершине V графа N при подстановке θ, если V является структурной и терминальной вершиной, содержащей n графов, каждая из которых состоит из одного корня с меткой класса — T и индивидуальными метками — элементами множества $\{d_1, \ldots, d_n\}$, являющегося значением переменной v в подстановке θ (заметим, что хотя логическая семантика этих формул в антецеденте импликации совпадает, их порождающая семантика различна).

Если метка входной вершины ***простой кванторной формулы с переменной*** F, входящей в консеквент импликации, имеет вид ∃ v T, где v — переменная, а T — термин, то F истинна на вершине V графа N при подстановке θ, если V является структурной и терминальной вершиной, содержащей непустое множество графов, каждая из которых состоит из одного корня с меткой класса — T и индивидуальной меткой — одним из элементов множества, являющегося значением переменной v в подстановке θ (индивидуальные метки попарно различны).

Если метка входной вершины ***простой кванторной формулы с переменной*** F, входящей в консеквент импликации, имеет вид ∃2 v T, где v — переменная, а T — термин, то F истинна на вершине V графа N при подстановке θ, если V является структурной и терминальной вершиной, содержащей не менее двух графов, каждая из которых состоит из одного корня с меткой класса — T и индивидуальной меткой — одним из элементов множества , являющегося значением переменной v в подстановке θ (индивидуальные метки попарно различны).

Если метка входной вершины ***простой кванторной формулы с переменной*** F, входящей в консеквент импликации, имеет вид ∃? v T, где v — переменная, а T — термин, то F истинна на вершине V графа N при подстановке θ, если V является структурной и терминальной вершиной, содержащей менее, чем n графов, каждая из которых состоит из одного корня с меткой класса — T и попарно различными индивидуальными метками — элементами множества $\{d_1, \dots, d_n\}$, являющегося значением переменной v в подстановке θ.

Если метка входной вершины ***простой кванторной формулы с переменной*** F, входящей в консеквент импликации, имеет вид ∃! v T, где v — переменная, а T — термин, то F истинна на вершине V графа N при подстановке θ, если V является простой и терминальной вершиной, а её меткой являет-

ся один из элементов множества, являющегося значением переменной v в подстановке θ.

Если метка входной вершины ***простой кванторной формулы с переменной*** F, входящей в консеквент импликации, имеет вид ∃[] v T, где v — переменная, а T — термин, то F истинна на вершине V графа N при подстановке θ, если V является простой и терминальной вершиной, а её меткой является целый (вещественный) интервал, являющийся подыинтервалом интервала I, являющегося значением переменной v в подстановке θ.

Унарная формула (с переменными) (общего вида) F истинна на вершине V графа N при подстановке θ, если V является простой вершиной, из неё выходит в точности одна дуга с меткой T и эта дуга входит в вершину V_1, на которой истинна формула F_1 (с переменными) (общего вида) при подстановке θ.

Если пропозициональной меткой начальной вершины W ***пропозициональной формулы (с переменными) (общего вида)*** F является &, то F истинна на вершине V графа N при подстановке θ, если V является простой вершиной, для каждой дуги с меткой T_i, выходящей из V, существует дуга с такой же меткой T_i, выходящая из W, для каждой дуги с меткой T_i, выходящей из W существует дуга с такой же меткой T_i, выходящая из V, а каждая формула (с переменными) (общего вида) F_i истинна на вершине, в которую входит дуга с меткой T_i, выходящая из вершины V при подстановке θ. Если F входит в антецедент импликации, а дуга с меткой T_i, выходящая из её начальной вершины имеет метку «¬», то в графа N дуга с меткой с меткой T_i, выходящая из V, должна отсутствовать. Если F входит в консеквент импликации, а дуга с меткой T_i, выходящая из её начальной вершины имеет метку «[]», то в графа N дуга с меткой с меткой T_i, выходящая из V, может отсутствовать.

Если пропозициональной меткой начальной вершины W ***пропозициональной формулы (с переменными) (общего вида)*** F, входящей в консеквент импликации, является $\vee$, то F истинна на вершине V графа N при подстановке θ, если V является простой вершиной, существует подмножество дуг, выходящих из W, между которым и множеством дуг, выходящих из V, существует взаимно-однозначное соответствие, при котором метки T_i соответствующих дуг совпадают, а формулы (с переменными) (общего вида) F_i, в начальные вершины которых входят дуги, выходящие из W истинны при подстановке θ на вершинах, в которые входят соответствующие дуги с метками T_i, выходящие из V.

Если пропозициональной меткой начальной вершины W ***пропозициональной формулы (с переменными) (общего вида)*** F, входящей в консеквент импликации, является |, то F истинна на вершине V графа N при подстановке θ, если V является простой вершиной, из неё выходит единственная дуга, существует дуга, выходящая из W, такая что метки T_i этих дуг совпадают, а формула (с переменными) (общего вида) F_i, в начальную вершину которой входит дуга с меткой T_i, выходящая из W, истинна при подстановке θ на вершине, в которую входит дуга с меткой T_i, выходящая из V.

Если метка входной вершины ***структурной кванторной формулы (с переменными) (общего вида)*** F имеет вид $\forall$ v T или метка входной вершины ***структурной кванторной формулы (с переменными) (общего вида)*** F, входящей в антецедент импликации, имеет вид $\exists$ v T, где v — переменная, а T — термин, то формула F истинна на вершине V графа N при подстановке θ, если V является структурной вершиной, содержащей n графов, каждая из которых имеет метку класса — T и индивидуальные метки — попарно различные элементы множества $\{d_1, \ldots, d_n\}$, являющегося значением переменной v в подстановке θ, и на корне каждой из этих графов истинна формула (с переменными) (общего вида) F_1 при подстановке θ (заметим, что хотя логическая

семантика этих формул в антецеденте импликации совпадает, их порождающая семантика различна)

Если метка входной вершины ***структурной кванторной формулы (с переменными) (общего вида)*** F, входящей в консеквент импликации, имеет вид ∃ v T, где v — переменная, а T — термин, то F истинна на вершине V графа N при подстановке θ, если V является структурной вершиной, содержащей непустое множество графов, каждая из которых имеет метку класса — T и индивидуальные метки — попарно различные элементы множества, являющегося значением переменной v в подстановке θ, и на корне каждой из этих графов истинна формула (с переменными) (общего вида) F_1 при подстановке θ.

Если метка входной вершины ***структурной кванторной формулы (с переменными) (общего вида)*** F, входящей в консеквент импликации, имеет вид ∃2 v T, где v — переменная, а T — термин, то F истинна на вершине V графа N при подстановке θ, если V является структурной вершиной, содержащей не менее двух графов, каждая из которых имеет метку класса — T и индивидуальные метки — попарно различные элементы множества, являющегося значением переменной v в подстановке θ, и на корне каждой из этих графов истинна формула (с переменными) (общего вида) F_1 при подстановке θ.

Если метка входной вершины ***структурной кванторной формулы (с переменными) (общего вида)*** F, входящей в консеквент импликации, имеет вид ∃? v T, где v — переменная, а T — термин, то F истинна на вершине V графа N при подстановке θ, если V является структурной вершиной, содержащей менее, чем n графов, каждая из которых имеет метку класса — T и индивидуальные метки — попарно различные элементы множества, являющегося значением переменной v в подстановке θ, и на корне каждой из этих

графов истинна формула (с переменными) (общего вида) F_1 при подстановке θ.

2.2.4.7. Порождающая семантика множества импликаций

Если активной вершине V графа N соответствует начальная вершина W ***множества импликаций*** с меткой $\{\Pi_1 \Rightarrow F_1, \ldots, \Pi_m \Rightarrow F_m\}$, где $\Pi_1, \ldots, \Pi_m$ — антецеденты, а $F_1, \ldots, F_m$ — консеквенты импликаций, то шаг порождения возможен лишь в случае, когда на вершине V графа N истинен антецедент только одной импликации (например, $\Pi_i \Rightarrow F_i$). В этом случае в каждой из графов, соответствующих префиксам компонент антецедента Π_i, выполняется поиск по образцу, заданному формулой этой компоненты, строится согласованная подстановка θ значений, заменяющих переменные антецедента Π_i, а шаг порождения из вершины V осуществляется в соответствии с формулой консеквента F_i при подстановке θ (т.е. при замене в этих формулах переменных их значениями из подстановки).

Пример 3. На Рис. 29 приведена контекстно-зависимая метаинформация простых историй болезни в контексте простой базы наблюдений (пример 1). Простая история болезни состоит из множества наблюдений, имеющих различные названия. Каждое наблюдение имеет некоторое множество моментов наблюдения, обозначенных константами сорта «время». С каждым моментом наблюдения связано одно значение. Контекстные зависимости состоят в том, что множество названий наблюдений в истории болезни должно быть подмножеством множества названий наблюдений в простой базе наблюдений, а значение наблюдения в любой момент наблюдения в истории болезни должно быть элементом множества значений наблюдения с тем же названием в простой базе наблюдений.

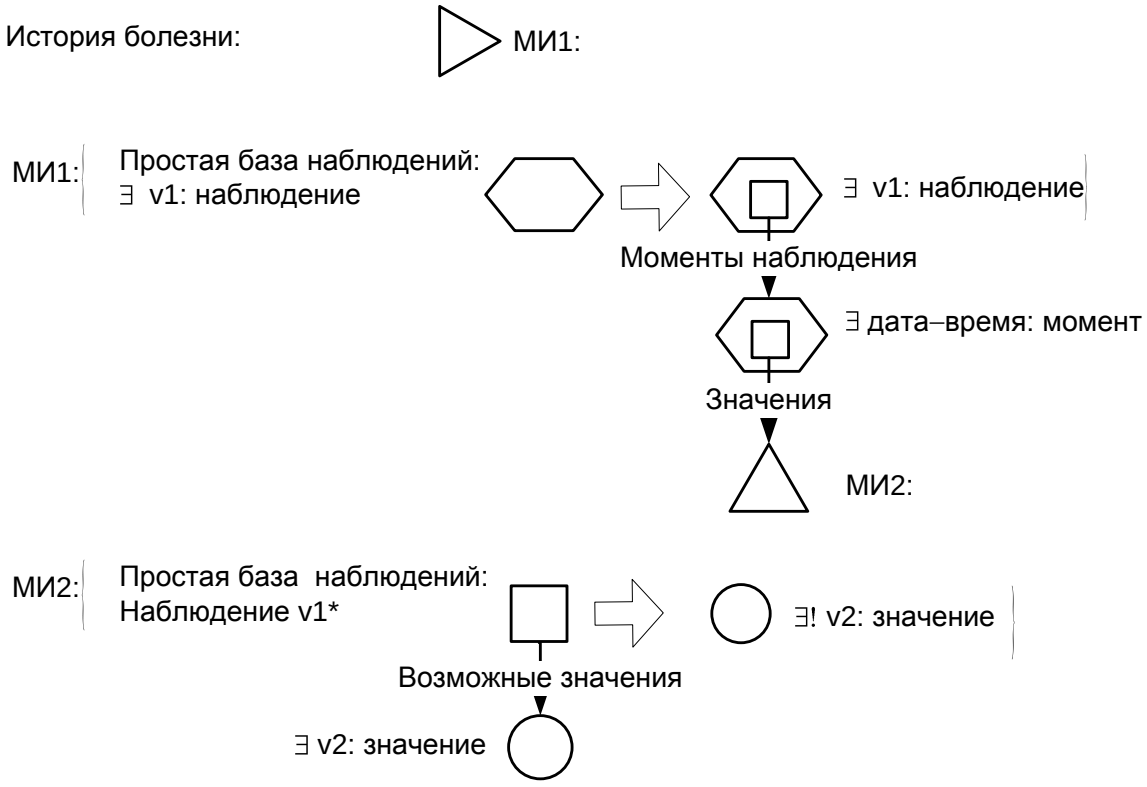

Рис. 29. Контекстно-зависимая метаинформация простых историй болезни

Пример 4. На Рис. 30 приведена контекстно-зависимая метаинформация медицинских знаний в контексте простой базы наблюдений (пример 1). Простая база медицинских знаний состоит из описания нормы и описания заболеваний. Описание нормы каждому наблюдению из простой базы наблюдений сопоставляет нормальные значения — собственное подмножество множества возможных значений этого наблюдения в простой базе наблюдений. Описание заболеваний состоит из описаний отдельных заболеваний, имеющих названия. Описание каждого заболевания состоит из его клинической картины, содержащей описания клинических проявлений, названиями которых являются названия некоторых наблюдений в простой базе наблюдений. Описание каждого клинического проявления состоит из конечного множества вариантов, не имеющих названий, а описание каждого варианта — из конечного упорядоченного множества периодов динамики, не имеющих названий. Описание каждого периода динамики состоит из области значений

(подмножество множества возможных значений наблюдения с тем же названием в простой базе наблюдений) и границ длительности (целого интервала).

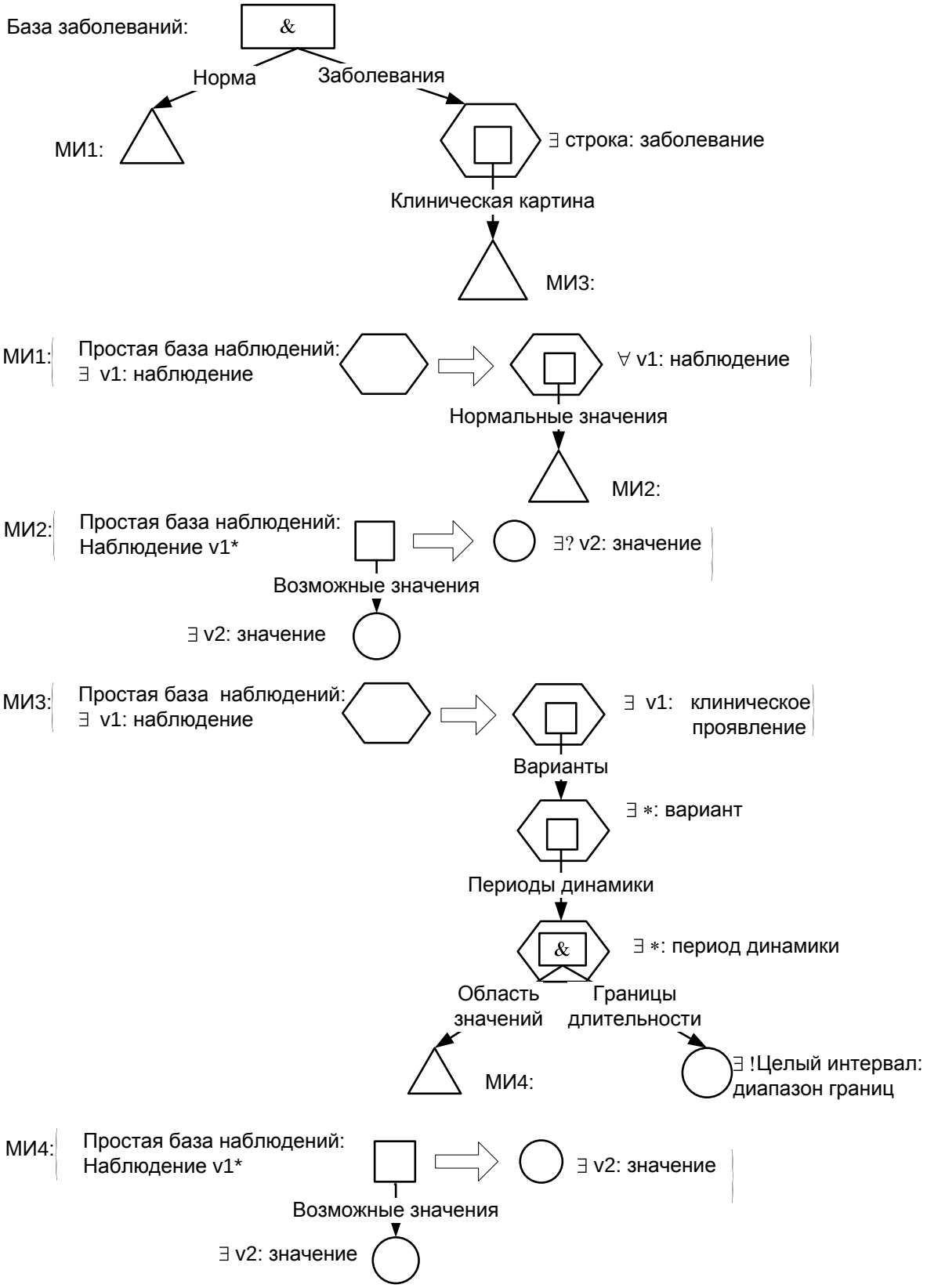

Рис. 30. Контекстно-зависимая грамматика языка представления медицинских знаний

2.3. Модель решателя задач

При разработке программных систем остро встаёт вопрос о выборе подходящих абстракций, в терминах которых будет осуществляться дальнейшая разработка. Для реализации облачной платформы интеллектуальных сервисов выбран агентно-ориентированный подход [4, 78]: программная система (интеллектуальный сервис) рассматривается как совокупность агентов, взаимодействующих между собой посредством передачи сообщений.

В данной работе агенты рассматриваются как совокупности продукций (пара «антецедент-консеквент»): когда агенту приходит сообщение, проверяются антецеденты агента и активируются один или несколько подходящих консеквентов (тело продукции). В результате работы тела продукций вычисляются новые данные, посылаются новые сообщения, приводящие к активации других агентов и т.д.

Проектирование сервиса сводится, в основном, к повторному использованию уже существующих в Фонде агентов и, при необходимости, созданию новых агентов (которые должны создаваться с учётом повторной используемости). Прикладному программисту предоставляется набор соответствующих программных интерфейсов (API), на базе которого и создаются агенты.

2.3.1. Вычислительная модель проекта IACPaaS

Обрабатываемая информация представляется инфоресурсами. Понятие инфоресурсов подробно рассмотрено в разделе 2.2, здесь же напомним, что инфоресурс I можно рассматривать как ориентированный граф $I = <C, R>$, состоящий из множества вершин (понятий) C и отношений (дуг) между R ними: $R = C \times C$. Каждая терминальная вершина $c \in C$ имеет атрибут, значение которого принадлежит множеству значений V.

Основой решателей является *агент* — это программный модуль, получающий *сообщения*, при обработке которых он может модифицировать ИР и

посылать новые сообщения другим агентам сервиса. Агент состоит из набора *блоков продукций*, обрабатывающих полученные сообщения. Сообщения одинаковых *типов* обрабатываются одними и теми же блоками продукций.

Сообщение — объект множества M, передающий информацию от агента к агенту в процессе обработки информационных ресурсов.

Блок продукций — программа, обрабатывающая сообщение некоторого типа; в процессе обработки эта программа может менять инфоресурсы, указанные в *полномочии*; на выходе этой программы — новые сообщения. Таким образом, блоком продукций p называется некоторая функция, сопоставляющая паре «состояние запущенного полномочия, сообщение» пару «новое состояние запущенного полномочия, набор выходных сообщений»: $p \in S \times M \to S \times \{\}M$.

Для решения какой-либо задачи программист объединяет несколько агентов в один сервис, Сервис есть пара: $app = \langle a_0, t \rangle$, где a_0 — корневой агент этого приложения, t — тип инфоресурса для обработки. Для использования сервиса пользователь должен получить полномочие на этот сервис. *Полномочие* есть указание сервису работать с определёнными инфоресурсами: $auth = \langle app, i_0 \rangle$, причём инфоресурс i_0 имеет тип t, указанный в сервисе app. Блокам продукций агентов доступна функция $get(\cdot)$, предоставляющая доступ к этому инфоресурсу.

Получив полномочие, пользователь может запустить его, при этом Платформой будет создан объект «запущенное полномочие» $runningApp = \langle auth, I, q \rangle$, где $I = \{i_0, i_1, \dots, i_k\}$ есть множество обрабатываемых инфоресурсов; изначально оно состоит только из одного входного инфоресурса i_0, но в процессе работы сервиса может расширятся новыми инфоресурсами (например, временными). q есть множество необработанных сообщений, $q = \{m_0, m_1, \dots, m_n\}$. Изначально это множество состоит только из одного сообщения m_0.

В процессе выполнения сервиса Платформа использует *функцию выборки следующего сообщения* для обработки. Программист сервиса приложений не должен полагаться на какой-то порядок извлечения сообщений из потока и должен считать, что функция выборки следующего сообщения не детерминирована.

Итак, процесс работы запущенного полномочия выглядит следующим образом:

1. Пользователь запрашивает запуск полномочия $auth$.

2. Платформа создаёт объект $runningApp$, который ссылается на $auth$и содержит один инфоресурс i_0 и одно начальное сообщение m_0.

3. Если множество необработанных сообщений содержит сообщение «Завершить работу», то работа сервиса завершается

4. Иначе при помощи функции выборки следующего сообщения выбирается одно из сообщений, определяется, какой агент и блок продукций будет обрабатывать его и запускается обработка этого блока продукций с этим сообщением. Если после этого были сформированы новые сообщения, они добавляются к множеству необработанных.

5. Перейти к шагу 3.

2.3.2. Недетерминизм выполнения и задача о правильности результата

Так как агенты могут обрабатывать сравнительно независимые инфоресурсы, можно распараллелить их работу. Для этого, например, достаточно запускать обработку очередного необработанного сообщения на отдельном узле компьютерного кластера.

Однако для параллельного проектирования характерен ряд ошибок, наиболее важные из них — это взаимоблокировки и состояния гонки [49, 62]. Платформа должна либо предотвращать их, либо предоставлять средства их обнаружения.

Но и в случае последовательного выполнения есть проблема определения следующего обрабатываемого сообщения. Если программист будет полагаться на определённый порядок обработки сообщений, это сделает его программу «хрупкой», неустойчивой при изменениях — при добавлении нового агента в сервис порядок выполнения будет другой. Программист при проектировании сервиса должен предполагать произвольность функции выборки очередного сообщения.

Поэтому практический интерес представляет следующая задача: как убедиться, что полученный при очередном запуске программы результат является правильным, в том числе не содержит ошибок, вызванных состоянием гонки?

Для решения этой задачи введём простую последовательную недетерминированную модель, после чего определим достаточное условие корректности, а затем покажем, как это условие может быть применимо в проекте IACPaaS [28].

2.3.3. Простая последовательная недетерминированная вычислительная модель

Рассмотрим простую вычислительную модель, состоящую из памяти, набора программ $p_1, ..., p_k$ и мультимножества необработанных сообщений *M*. *Память* есть (потенциально бесконечное) множество переменных $v_1, ..., v_n$ (своё для каждого вычислительного процесса в этой модели). В каждом состоянии памяти вычислительного процесса каждая переменная *v* имеет некоторое значение из своей области возможных значений. Каждая *программа p* есть последовательность операций чтения значения некоторой переменной, записи значения некоторой переменной, посылки сообщения некоторой программе, а также управляющих операторов. При выполнении программы *p* в вычислительном процессе, если выполняется операция посылки сообщения, то за ней могут выполняться только операции посылки сообщений. *Сообще-*

ние m есть пара: значение и имя последовательной программы, которой оно послано.

Вычислительный процесс начинается с некоторого состояния памяти, называемого *начальным состоянием памяти*, и мультимножества необработанных сообщений *M*, состоящего из одного сообщения m_0, называемого *начальным*. На очередном шаге вычислительного процесса, если мультимножество *M* не пусто, то из него произвольным образом выбирается и обрабатывается одно сообщение *m*, после чего оно исключается из мультимножества *M*. Обработка сообщения *m* состоит в выполнении последовательной программы *p*, имя которой указано в сообщении *m*. Если эта программа выполняет операции посылки сообщений, то все эти посланные сообщения добавляются к мультимножеству необработанных сообщений *M*. Вычислительный процесс заканчивается, когда мультимножество необработанных сообщений *M* становится пустым и ни одна программа p_i не работает. *Результатом* вычислительного процесса является состояние памяти после его окончания.

Сопоставим каждой переменной *v* из памяти историю её обработки. *История обработки переменной v* есть мультимножество, состоящее из пар <сообщение *m*, операция *o*> — при обработке сообщения *m* выполнилась операция *o* над некоторой переменной. На мультимножестве *S* всех сообщений, посланных в вычислительном процессе, введём отношение частичного порядка: сообщение m_1*непосредственно предшествует* сообщению m_2, если сообщение m_2 послано при обработке сообщения m_1. *Отношение частичного порядка (предшествования)* на мультимножестве сообщений *S* есть транзитивное замыкание отношения непосредственного предшествования.

Будем говорить, что история обработки некоторой переменной *v строго корректна*, если для любых двух пар этой истории $<m_1,o_1>$ и $<m_2,o_2>$ сообщения m_1 и m_2 находятся в отношении предшествования. Кроме того, будем говорить, что история обработки некоторой переменной *v корректна*, если

она либо строго корректна, либо все пары этой истории содержат только операции чтения значения этой переменной v.

В простой вычислительной модели сообщения из мультимножества необработанных сообщений M выбираются произвольно. Это значит, что существует множество разных функций выбора сообщений из мультимножества необработанных сообщений M. Каждой такой функции η соответствует мультимножество S_η всех сообщений, посланных в вычислительном процессе, и линейный порядок на этом мультимножестве S_η, совместимый с частичным порядком на S_η (линейный порядок совместим с частичным, если выполнено следующее условие: если m_1 предшествует m_2 в смысле частичного порядка, то имеет место такое же предшествование и в смысле линейного порядка).

Пусть η — некоторая функция выбора сообщений из мультимножества необработанных сообщений M. Построим дерево сообщений, представляющее частичный порядок на S_η, следующим образом:

1. Вершины дерева взаимно-однозначно соответствуют сообщениям из мультимножества S_η;

2. Корнем дерева является вершина, соответствующая сообщениюm_0;

3. Из вершины m_i выходит дуга в вершину m_j в том и только том случае, если при обработке сообщения m_i было послано сообщение m_j.

На Рис. 31 приведён пример дерева сообщений: при обработке сообщения m_0 сгенерированы сообщения m_1, m_2 и m_3. Затем, при обработке m_1 сгенерированы сообщения m_4 и m_5 и т.д. Сообщение m_3 — терминальное (при его обработке не сгенерировано ни одно сообщение).

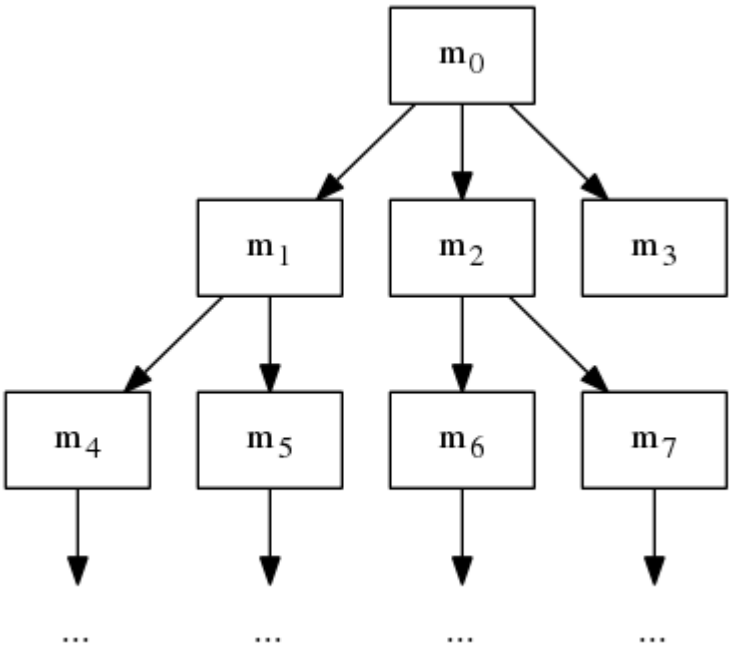

Рис. 31. Пример дерева сообщений

Один из возможных порядков обработки следующий:

1. $M = \{m_0\}$

2. m_0 извлечено и обработано, $M = \{m_1,m_2,m_3\}$

3. m_1 извлечено и обработано, $M = \{m_2,m_3,m_4,m_5\}$

4. m_3 извлечено и обработано, новых сообщений нет, $M = \{m_2, m_4,m_5\}$

5. m_2 извлечено и обработано, $M = \{m_4,m_5,m_6,m_7\}$

и т.д.

Таким образом, линейный порядок на сообщениях в данном случае есть $<m_0, m_1, m_3, m_2, \ldots>$.

На Рис. 32 показана ситуация, когда обработка сообщения m_0 приводит к изменению значений переменных v_1 и v_2, а также посылке сообщений m_1 и m_2. Далее, при обработке m_1изменяются переменные v_1 и v_3, а при обработке сообщения m_2 изменяются переменные v_1 и v_4.

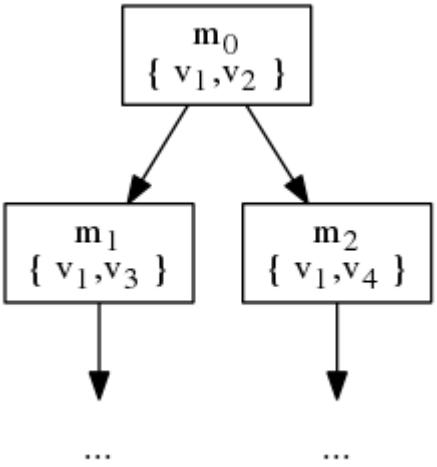

Рис. 32. Пример нарушения корректности истории переменной v_1.

В этом случае конечный результат может зависеть от порядка обработки сообщений: если сообщения будут обработаны в порядке $<m_0, m_1, m_2>$, то в переменной v_1 будет значение, записанное при обработке сообщения m_2, а при другом порядке — $<m_0, m_2, m_1>$ в переменной v_1 будет значение, записанное при обработке сообщения m_1. Поэтому история переменной v_1 не является корректной. Заметим, что история переменных v_2, v_3, v_4 — строго корректна (при условии, что при дальнейшей обработке корректность не будет нарушена).

Если при одном и том же начальном состоянии памяти вычислительного процесса получается один и тот же его результат, независимо от способа выбора сообщений из мультимножества необработанных сообщений M, то такой результат вычислительного процесса будем называть *корректным*. В следующей теореме доказывается достаточное условие корректности результата вычислительного процесса.

Теорема. Если в вычислительном процессе истории обработки всех переменных памяти корректны при некоторой функции η выбора сообщений из мультимножества необработанных сообщений M, то результат вычислительного процесса корректен.

Для доказательства теоремы сформулируем несколько очевидных лемм.

Лемма 1. Пусть сообщения m_1 и m_2 при некоторой функции ηне находятся в отношении предшествования. Обозначим w_1 — множество переменных памяти, над которыми выполняются операции записи значений, wr_1 — множество переменных памяти, над которыми выполняются операции чтения или записи значений при обработке сообщения m_1; w_2 — множество переменных памяти, над которыми выполняются операции записи значений, wr_2 — множество переменных памяти, над которыми выполняются операции чтения или записи значений при обработке сообщения m_2 (см. Рис. 33). Тогда из корректности историй обработки всех переменных памяти следуют соотношения: $w_1 \cap wr_2 = \varnothing$; $w_2 \cap wr_1 = \varnothing$.

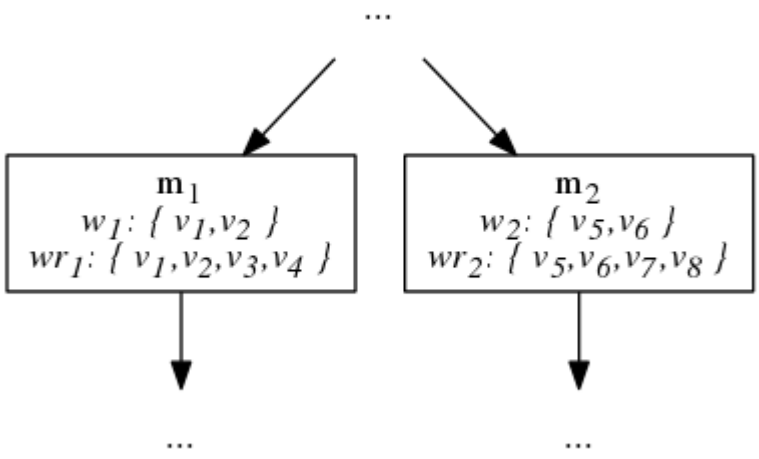

Рис. 33. Сообщения, не находящиеся в отношении предшествования (пример).

Лемма 2. Пусть m — некоторое сообщение, выбранное функцией выбора сообщений η из мультимножества M необработанных сообщений. Обозначим WR_m — множество переменных памяти, над которыми выполняются операции записи или чтения значений при обработке всех сообщений, лежащих в дереве сообщений на пути из корня в вершину m, а W^*_m — множество переменных памяти, над которыми выполняются операции записи значений при обработке всех сообщений, не находящихся с сообщением m в отноше-

нии предшествования. Тогда из корректности историй обработки всех переменных памяти следует соотношение $WR_m \cap W^*_m = \varnothing$.

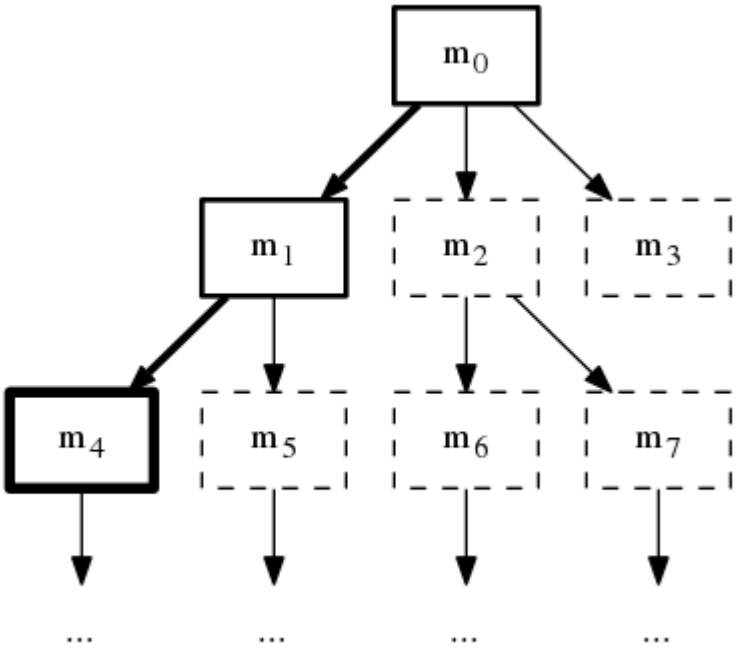

Рис. 34. Сообщение m4 и сообщения, не находящиеся в отношении предшествования (пример).

На Рис. 34 представлен пример фрагмента дерева сообщений. Рассмотрим сообщение m_4, тогда сообщения, не находящиеся с ним в отношении предшествования — m_2, m_3, m_5, m_6, m_7. Переменные, над которыми выполнялись операции записи или чтения значений при обработке сообщения m_4, образуют множество WR_{m4}.

Следствие леммы 2. Утверждение леммы 2 справедливо и для всех терминальных вершин дерева сообщений.

Доказательство теоремы. Из определения частичного порядка на мультимножестве S_η всех сообщений, посланных в вычислительном процессе, следует, что сообщения, лежащие в дереве сообщений на пути из корня к некоторой терминальной вершине m, обрабатываются последовательно, причём обработка каждого сообщения выполняется последовательной программой. Таким образом, обработка всех сообщений на пути из корня в терминальную

вершину m есть последовательный процесс. Из следствия леммы 2 следует, что множество переменных памяти, к которым в этом последовательном процессе применяются операции чтения и записи значений, не пересекается с множеством переменных, к которым применяются операции записи значений при обработке всех сообщений, не лежащих на этом пути. Поскольку это верно для пути из корня в любую терминальную вершину, то отсюда следует, что для одного и того же начального состояния памяти при любой функции η выбора сообщений из мультимножества необработанных сообщений M мультимножество сообщений S_η и результат вычислительного процесса будут одними и теми же.

2.3.4. Простая вычислительная модель с параллельными процессами

Рассмотрим более сложную вычислительную модель, состоящую из (потенциально бесконечного) множества вычислительных узлов n, n_0, n_1,..., n_r (своего для каждого последовательного вычислительного процесса), каждый из которых имеет свою собственную память, и набора программ p_1, ..., p_k. В параллельном вычислительном процессе память каждого узла есть (потенциально бесконечное) множество переменных v_1, ..., v_n, причём между множествами переменных любых двух узлов n_i и n_j имеется взаимно однозначное соответствие (будем считать, что переменные, соответствующие друг другу, имеют одинаковые имена). В вычислительном процессе переменные на разных узлах независимы, т.е. изменение значения переменной на одном узле не затрагивает значений переменных (в том числе и одноименных) на других узлах. Программы p_1, ..., p_k и сообщения определяются так же, как и в простой вычислительной модели.

Параллельный вычислительный процесс начинается с того, что начальное состояние памяти загружается на узел n_0 и доступным для обработки становится начальное сообщение m_0 (корень дерева сообщений). Его обра-

ботка состоит в том, что программа p, которой послано это сообщение, загружается на узел n_0 и выполняется на нем. Если некоторая программа p, выполняемая на узле n_k, посылает сообщение m_k, то это сообщение становится доступным для обработки. Его обработка состоит в том, что выбирается новый узел n' из множества узлов $n_1, ..., n_r$, на него загружается состояние памяти, полученное в результате выполнения программы p на узле n_k, а также программа p', которой послано сообщение m_k, и программа p' выполняется на узле n'. Если же программа p, выполняемая на узле n_t, не посылает сообщений (т.е. она обрабатывает терминальное сообщение m_t в дереве сообщений), то значения всех переменных на узле n_t, над которыми выполнялись операции записи их значений при обработке всех сообщений, лежащих на пути из корня в вершину m_t, присваиваются одноименным переменным на главном узле n. Программы, выполняемые на разных узлах, выполняются параллельно друг другу. Параллельный вычислительный процесс завершается, если вычисления на всех узлах завершились и нет сообщений, доступных для обработки. Результатом вычислительного процесса является состояние памяти на главном узле n после завершения параллельного вычислительного процесса.

В простой вычислительной модели с параллельными процессами порядок, в котором при обработке терминальных сообщений выполняются присваивания значений одноименным переменным на узле n, зависит от времени загрузки состояния памяти и программы на каждый новый узел и от времени выполнения этих программ на этих узлах. Поэтому в общем случае результат вычислительного процесса может не быть корректным. Однако, легко видеть, что следствием теоремы предыдущего раздела является следующее утверждение: если в конкретном параллельном вычислительном процессе простой вычислительной модели с параллельными процессами истории обработки всех переменных памяти корректны, то результат параллельного вычислительного процесса корректен.

2.3.5. Применение простой вычислительной модели с параллельными процессами в проекте IACPaaS

Любой сервис IACPaaS соответствует простой вычислительной модели с параллельными процессами: это набор агентов, выполняемых на компьютерном кластере. Каждый процессор, на котором выполняется агент, соответствует узлу и обладает своей независимой памятью. ИР может быть представлен в виде памяти (множества переменных): каждой вершине *c* соответствует переменная v_c, значение которой для терминальной вершины графа есть значение её атрибута, а для нетерминальных вершин — множество дуг, выходящих из этой вершины. Заметим, что агенты при выполнении могут использовать оперативную память для промежуточных расчётов, однако, её содержимое исчезает сразу после завершения работы агента и не передаётся другим агентам. Поэтому в дальнейшем будет рассматриваться только память, которой является ИР и которая используется так же, как и в простой вычислительной модели с параллельными процессами. Каждый агент есть последовательная программа, которая выполняет операции чтения и записи над ИР. В вычислительной системе может одновременно выполняться несколько агентов на разных узлах компьютерного кластера. Операции обращения к ИР соответствуют операциям чтения соответствующих переменных памяти, а операции модификации ИР соответствуют операциям записи значений в соответствующие переменные памяти.

Вычислительный процесс начинается с обработки *начального сообщения*, одного и того же для всех запусков сервиса. *Входными данными* вычислительного процесса является начальное состояние ИР. По мере того, как агенты завершают работу, состояния ИР с терминальных процессоров пересылаются на главный, где осуществляется их слияние. Вычислительный процесс заканчивается, когда все агенты завершили работу. *Результатом* вы-

числительного процесса является состояние ИР на главном узле, когда все агенты завершили работу.

2.4. Модель интерфейса

2.4.1. Требования

Платформа должна взаимодействовать с внешним миром, в частности, с пользователями. В настоящее время используются как веб-интерфейсы [58, 59], так и native-интерфейсы. Последние позволяют использовать больше ресурсов компьютеров пользователя, хотя и стремительно вытесняются веб-интерфейсами [50, 68]. Поэтому предлагаемое решение должно позволять взаимодействовать как с native- так и веб-интерфейсами.

Современные веб-интерфейсы создаются с использованием уже готовых веб-фреймворков (web framework, [88]), которые хорошо абстрагируют низкоуровневые особенности веб-программирования, и позволяют создавать веб-приложения разной направленности. В настоящее время существует как масса универсальных веб-фреймворков, например, Django [45] или Ruby on Rails [75], так и специализированные системы управления содержимым (CMS), такие как 1С-Битрикс [1], Drupal [46], WordPress [91], MediaWiki [63] и другие.

В проекте IACPaaS используется CMS MediaWiki для представления документов, инструкций, статей и т.п. Естественным было бы, если бы пользовательский интерфейс Платформы интегрировался с этой CMS для упрощения научной и практической работы пользователя.

Основной сущностью CMS MediaWiki является «страница» на которой могут быть смешаны визуальные представления разного рода — текст с гиперссылками, картинки и диаграммы, таблицы, а также интерфейсные элементы дополнительных подключаемых модулей (плагинов). Поэтому предла-

гаемое интерфейсное решение должно встраиваться в соответствующую модульную wiki-среду.

2.4.2. Компоненты модели интерфейса

За основу модели интерфейса взята концепция «MVC» (Модель-представление-контроллер) [73], в которой выделены следующие сущности:

1. **Модель** предоставляет данные и методы работы с этими данными, реагирует на запросы, изменяя своё состояние. Не содержит информации, как эти данные можно визуализировать. Модель представляется инфоресурсом специального вида.

2. **Представление,** используемое для представления информации. Этот компонент представлен по-разному, в зависимости от того, какой тип интерфейса используется, в частности, для веб-интерфейса используется компонент, формирующий фрагмент html-текста. Представлен различными агентами.

3. **Контроллер**. Этот компонент обеспечивает связь между пользователем и сервисом, и тоже представлен агентами специального вида.

2.4.2.1. Компонент «Модель»

Модель разработана с учётом:

1. Абстрагирования существующих WIMP-интерфейсов,

2. Расширяемости набора интерфейсных элементов,

3. Рекурсивности интерфейса (одни элементы вложены в другие).

На Рис. 35 показана метаинформация для основы интерфейсных элементов.

Путём порождения по метаинформации «Интерфейсный элемент» получается описание конкретного интерфейса. Это описание интерпретируется выбранным для взаимодействия с системой представлением: если выбрано веб-представление, то в результате получается HTML-текст, передаваемый

далее браузеру пользователя, если native-представление, то результатом является native-интерфейс.

Прикладной программист может задавать дополнительные подсказки для компонента «представление» для того, чтобы улучшить дизайн интерфейса. Например, для веб-представления программист может задавать классы отдельных элементов и этот класс будет преобразован в атрибут «class» созданного html-элемента. Такие подсказки создаются при порождении метаотношения «атрибуты».

Кроме того, в метаинформации выделен специальный атрибут «надпись», который используется для визуальной идентификации элемента пользователем.

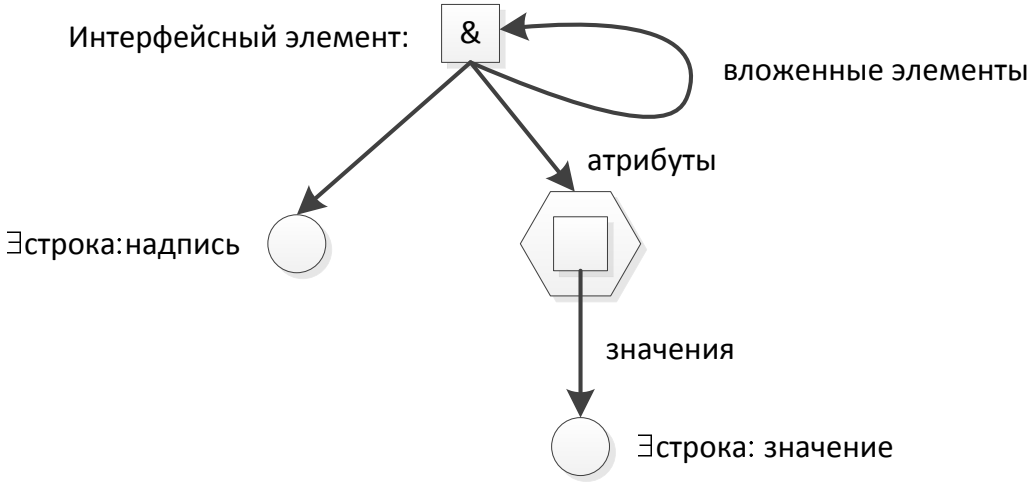

Рис. 35. Метаинформация «Интерфейсный элемент»

Метаинформации для описания конкретных интерфейсных элементов получаются расширением метаинформации «Интерфейсный элемент». На Рис. 36 показаны метаинформации для основных, поддерживаемых Платформой, интерфейсных элементов. Для описания простых элементов, таких как «Кнопка», «Раздел», хватает описательных возможностей метаинформации «Интерфейсный элемент»; другие же (например, «Ссылка», «Поле ввода», «Выбор из списка»), требуют описания дополнительных свойств.

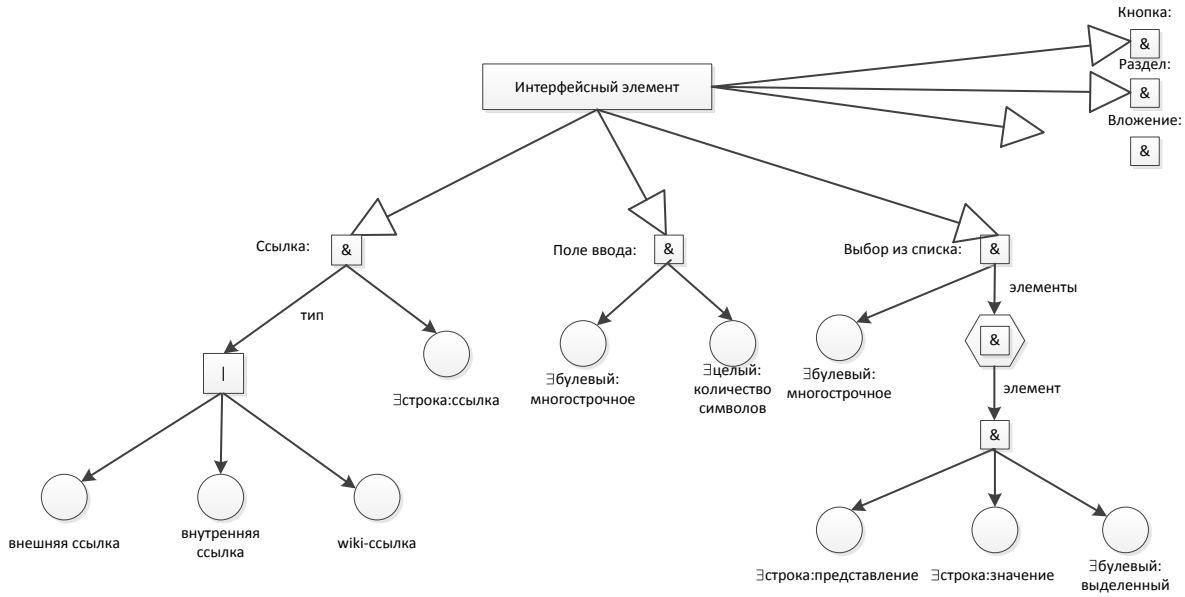

Рис. 36. Основные интерфейсные элементы

На Рис. 37 представлен пример инфоресурса для интерфейса «Калькулятор», состоящего из двух полей ввода, поля выбора операции и кнопки «Вычислить».

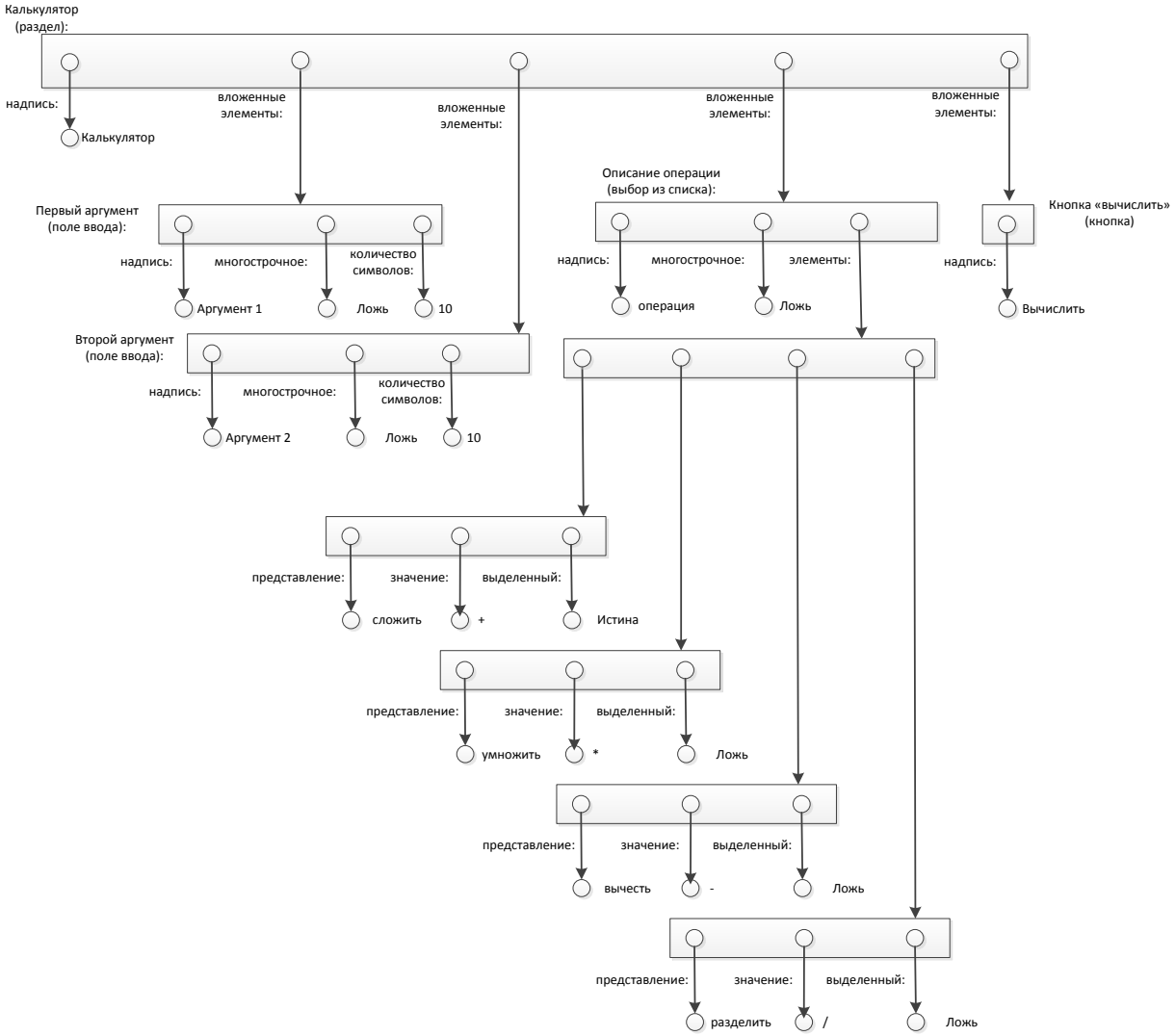

Рис. 37. Пример инфоресурса для интерфейса «Калькулятор»

Компонент «Веб-представление» преобразует этот инфоресурс в следующий HTML-текст:

```
<div id=”калькулятор”>
  <label for=”первый аргумент”>Аргумент 1</label>
  <input id=”первый аргумент” type=”text” maxlength=”10”/>
  <label for=”второй аргумент”>Аргумент 2</label>
  <input id=”второй аргумент” type=”text” maxlength=”10”/>
  <label for=”Описание операции”>операция</label>
  <select id=”Описание операции”>
        <option value=”+” selected>сложить</option>
```

```
        <option value=”*”>умножить</option>
        <option value=”-”>вычесть</option>
        <option value=”/”>разделить</option>
  </select>
  <button id=”Кнопка «Вычислить»”>Вычислить</button>
</div>
```

2.4.2.2. Компонент «Вид»

Компонент «Вид» паттерна MVC представлен агентом (далее — «Интерфейсный агент»), который должен обрабатывать по крайней мере одно сообщение «Сформировать представление». Внутри этого сообщения должна быть ссылка на инфоресурс модели, которую надо визуализировать. Приняв это сообщение, Интерфейсный агент должен изменить соответствующую визуализацию.

Допускаются разные агенты подобного типа; нужный агент (или агенты) выбирается проектировщиком пользовательского интерфейса.

В частности, для создания веб-интерфейса, используется агент «Формирователь HTML-представления». Этот агент формирует соответствующее HTML-представление, которое будет передано браузеру.

Учитывая, что Интерфейсный агент «знает» об особенностях соответствующего пользовательского интерфейса, он используется и для обработки запросов, перерабатывая события в интерфейсе в сообщения для остальных компонентов.

2.4.2.3. Компонент «Контроллер»

Компонент «контроллер» паттерна MVC представлен агентом (далее — «Интерфейсный контроллер»), который должен обрабатывать следующие сообщения:

1. Запрос от пользователя. Это сообщение содержит описания действий пользователя в соответствующем типу интерфейса виде. Например, для веб-интерфейса это набор параметров GET- или POST-запроса [47].

2. Результат обработки запроса. Интерфейсный контроллер посылает сервису сообщение на обработку поступившего запроса; после обработки результата Интерфейсному контроллеру поступает сообщение «Результат обработки запроса» со ссылкой на инфоресурс модели. Интерфейсный контроллер передаёт её Интерфейсному агенту для дальнейшей визуализации.

2.4.2.4. Взаимодействие Интерфейсного агента и Интерфейсного контроллера

На Рис. 38 показано взаимодействие компонентов Платформы. В случае, когда используется взаимодействие через браузер, модуль «Интерфейс» распадается на три подмодуля: браузер, Интернет (как среда передачи), веб-сервер.

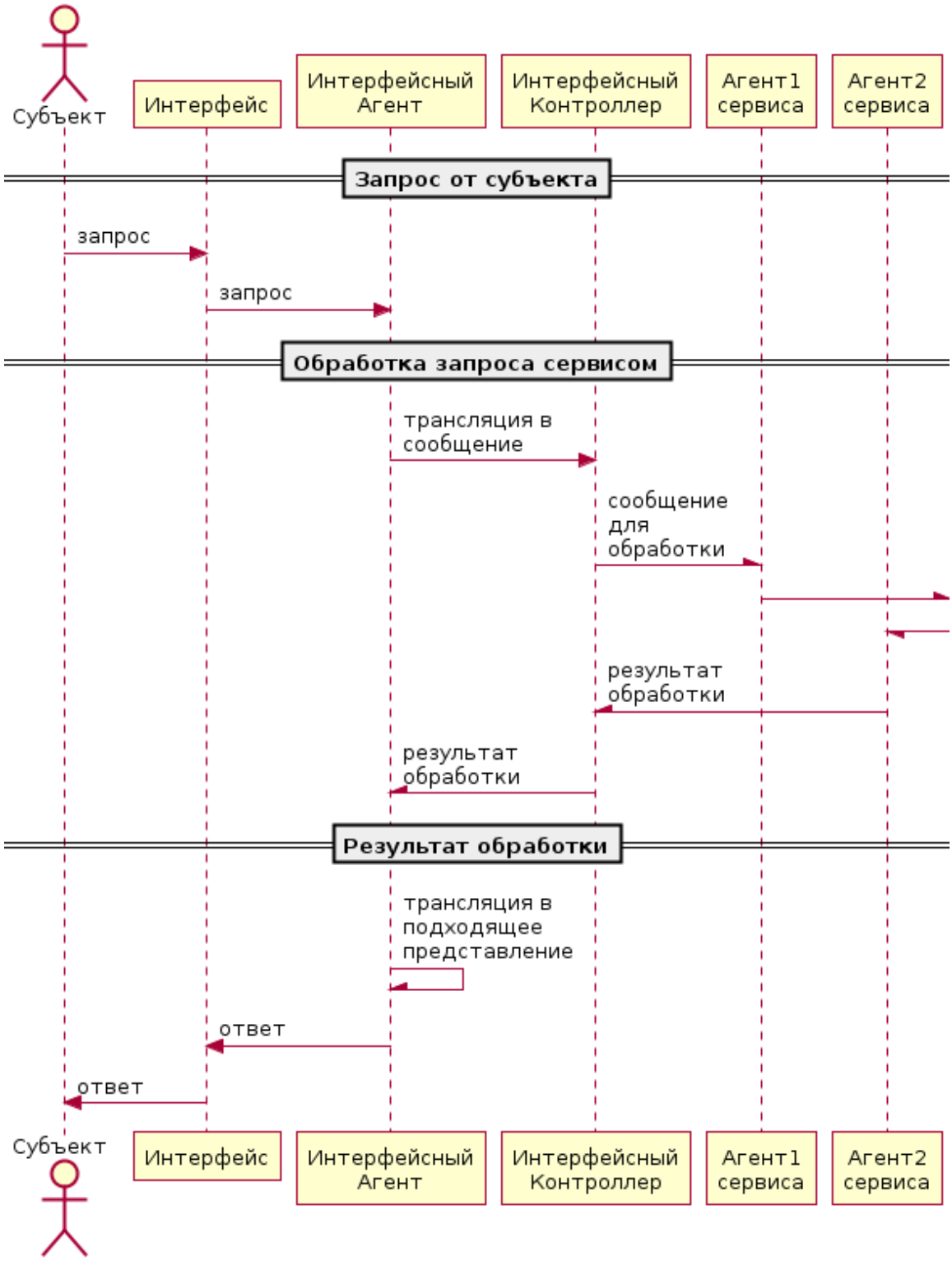

Рис. 38. Взаимодействие «Интерфейсного агента» и «Интерфейсного контроллера»

2.5. Выводы по главе

В настоящей главе:

1. Разработаны основные требования к проекту IACPaaS как непрерывному процессу совершенствования Фонда, поддержанного Платформой. Приведены концептуальная архитектура проекта IACPaaS.

2. Предложена концепция Платформы, как сущности, состоящей из Фонда, как хранилища инфоресурсов, сервисов и агентов, сайта как средства доступа к ресурсам Платформы и виртуальной машины для выполнения функциональности Платформы и сервисов.

3. Предложена модель информационных ресурсов как особого вида семантических сетей, структура которых задаётся метаинформацией, которая, в свою очередь, также является семантической сетью особого вида.

4. Для описания множества метаинформаций введено исчисление формул метаинформаций. Формулы имеют логическую семантику: формула истинна для тех и только тех инфоресурсов, которые принадлежат классу, задаваемому этой метаинформацией. Формулы также имеют порождающую семантику, которая задаёт процесс порождения инфоресурсов как исчисление.

5. Введена модель решателя задач как набора взаимодействующих с помощью недетерминированного обмена сообщениями агентов.

6. Для определения неконфлюентности результата, порождаемой проблемой недетерминизма, в главе введено понятие корректности результата работы, введено понятие истории обработки сообщений и последовательная модель решателя. Показано, что нарушение корректности результата следует из некорректности истории обработки сообщений. Следовательно, некорректность прогона решателя можно отследить по некорректности его истории обработки сообщений. Введена параллельная модель решателя и показано, что данное доказательство применимо так же и к параллельной модели решателя.

7. Разработана модель интерфейса как взаимодействие трёх модулей: интерфейсного агента, интерфейсного контролёра и агентов сервиса. Предложено декларативное представление абстрактного интерфейса.

3. МЕТОДЫ РЕАЛИЗАЦИИ ОБЛАЧНОЙ ПЛАТФОРМЫ ДЛЯ СОЗДАНИЯ И ИСПОЛЬЗОВАНИЯ ИНТЕЛЛЕКТУАЛЬНЫХ СЕРВИСОВ

В данной главе решается задача разработки методов реализации облачной платформы для создания и использования интеллектуальных сервисов.

В разделе 3.1 рассмотрена общая архитектура Платформы IACPaaS, в следующих разделах рассмотрены методы реализации основополагающих модулей Платформы: процессора инфоресурсов (раздел 3.2), процессора решателей задач (раздел 3.3), процессора пользовательских интерфейсов (раздел 3.4).

В настоящее время на основе этих методов реализована версия Платформы с последовательной обработкой сообщений (далее просто «последовательная версия») и разработан проект версии с распределённой обработкой сообщений (далее просто «распределённая версия»). Последовательная версия позволяет запускать несколько сервисов в режиме разделения времени между сервисами по методу карусельной диспетчеризации (round robin), эмулируя параллельное выполнение сервисов. Распределённая версия позволяет распределять вычислительную нагрузку между компьютерами кластера.

В данной главе описано устройство обеих версий, отличия версий оговариваются специально.

3.1. Общая архитектура платформы IACPaaS

3.1.1. Проект

В Платформе выделяются четыре уровня (см. Рис. 39):

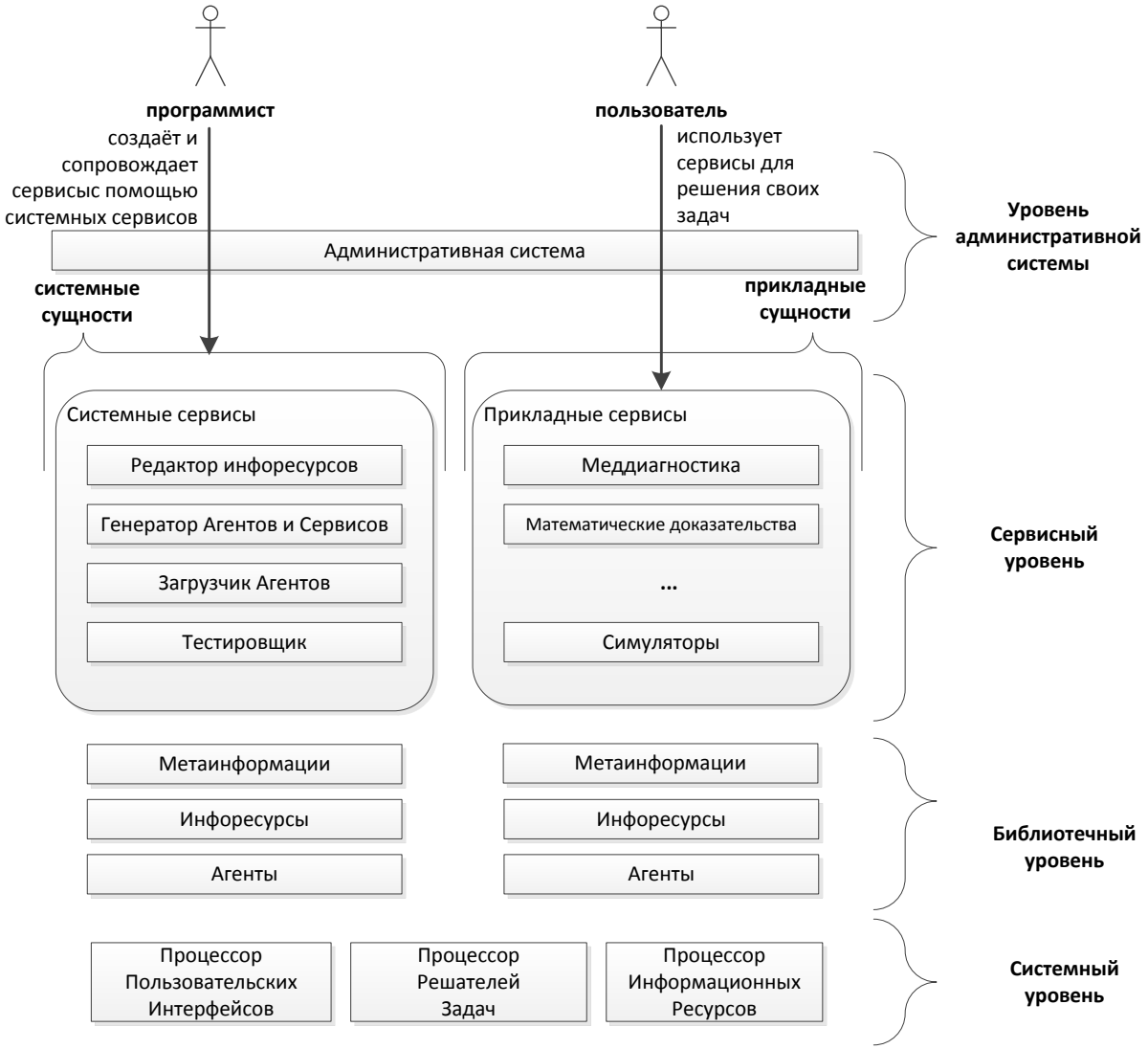

Рис. 39. Концептуальная архитектура Платформы IACPaaS

1. Системный уровень. Это самый низкий уровень Платформы, при помощи которого поддержаны хранение и доступ к инфоресурсам, запуск и работа сервисов, а также взаимодействие с пользователем. Низкоуровневый функционал с этого уровня недоступен пользователям напрямую, но используется для осуществления функционирования сервисов. Кроме того, этот функционал используется для утилит сопровождения.

2. Библиотечный уровень. На этом уровне располагаются компоненты сервисов — агенты и инфоресурсы.

3. Сервисный уровень. На этом уровне вводятся сервисы как совокупности взаимодействующих агентов. Выделяются прикладные сервисы, создава-

емые прикладными разработчиками для решения задач пользователей, и системные сервисы, необходимые для функционирования Платформы и развития Фонда. Среди них можно выделить:

3.1. Редактор Инфоресурсов (см. раздел 4.3.2). Используется для создания и редактирования инфоресурсов Фонда.

3.2. Генератор Агентов, генератор Сервисов (см. раздел 4.3.5). Оба этих сервиса используются для создания агентов и сервисов соответственно. Эти Генераторы используются прикладными программистами.

3.3. Загрузчик Агентов (см. раздел 4.3.6). Используется для обновления кода существующих в Фонде агентов.

3.4. Прогонщик тестов Агентов и Сервисов (см. раздел 4.3.4). Используется прикладными программистами в процессе сопровождения и отладки агентов и сервисов.

5. Административная система (см. раздел 4.3.1) позволяет пользователю запустить прикладной сервис или создать новый сервис (прикладной или системный) посредством запуска системных сервисов.

Платформа осуществляет выполнение сервиса в целом посредством трёх программных компонентов (см. Рис. 40): процессора пользовательского интерфейса, процессора решателей задач и процессора инфоресурсов.

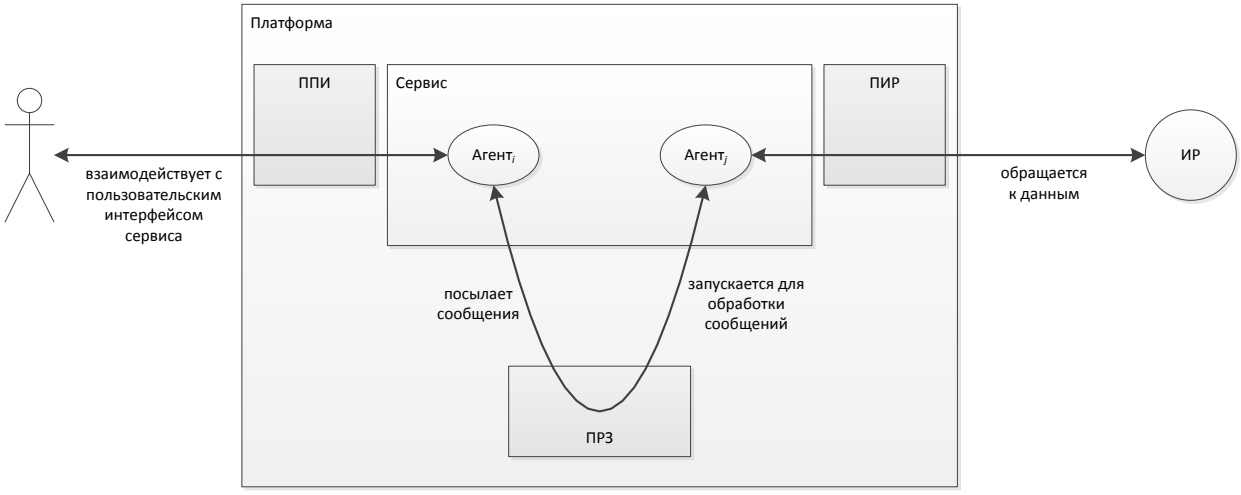

Рис. 40. Архитектурно-контекстная диаграмма сервисов

1. Пользователь взаимодействует с сервисом через процессор пользовательского интерфейса (ППИ). ППИ переводит запросы пользователя в соответствующие сообщения для агентов сервиса.

2. Агенты взаимодействуют друг с другом посредством процессора решателей задач (ПРЗ), который предоставляет возможности посылки сообщений агентами и их обработки.

3. В процессе работы агент может получать доступ к данным в инфоресурсах (получение из них информации и модификация); эта возможность предоставляется процессором инфоресурсов (ПИР).

Далее в этой главе будет рассмотрены методы реализации всех процессоров Платформы: в разделе 3.2 этой главы будет рассмотрен процессор информационных ресурсов, в разделе 3.3 — процессор решателей задач, а в разделе 3.4 — процессор пользовательского интерфейса.

3.2. Методы реализации процессора информационных ресурсов

3.2.1. Постановка задачи и требования к процессору информационных ресурсов

Основное назначение ПИР — долговременное хранение Фонда и предоставление удобного программного интерфейса к нему. Поэтому требования разбиваются на две группы: требования к хранению, требования к программному интерфейсу.

3.2.1.1. Требования к хранению

Требование 1. Модуль ПИР должен предоставлять возможность хранения инфоресурсов Фонда.

Одним из способов осуществления надёжного хранения данных и неконфликтующего доступа к ним в настоящее время являются системы OLTP (Online Transaction Processing, [43]). Данные системы вводят понятие

«транзакции», как небольшой группы действий, выполняемых атомарно и независимо от других таких транзакций. Такие системы характеризуются большим числом изменений, одновременным доступом множества пользователей к одним и тем же данным для выполнения операций чтения, удаления или модификации. Стандартный набор требований к таким системам называется ACID-требования [51].

Платформа предполагает одновременную работу большого количества агентов, которые могут получать доступ к одним и тем же инфоресурсам, могут некорректно обрабатывать инфоресурсы, нарушая их соответствие метаинформации, и в работе которых могут быть сбои. Поэтому к ПИР выдвигаются следующие требования:

Требование 2. ПИР должен быть транзакционным; все операции с инфоресурсами должны совершаться в рамках транзакций (группа последовательных операций над инфоресурсами; ПРЗ начинает транзакцию вызовом соответствующей функции, затем выполняет действия над инфоресурсами и, в завершение, либо завершает транзакцию, либо отменяет транзакцию).

Требование 3. Поддержка транзакций, предоставляемая ПИР, должны удовлетворять ACID-требованиям:

1. Атомарность (atomicity): никакие изменения не должны фиксироваться частично. Транзакция или полностью применяется, или не применяется вообще;

2. Согласованность (consistency): после завершения транзакций инфоресурсы должны находиться в согласованном состоянии (т.е. не противоречить метаинформации, по которой они порождены);

3. Изолированность (isolation): во время выполнения транзакции другие параллельные транзакции не должны влиять на результат;

4. Надёжность (durability): Изменения, сделанные успешно завершённой транзакцией, должны остаться сохранёнными после возвращения системы в работу, независимо от проблем на нижних уровнях (к примеру, обесточива-

ние системы или сбои в оборудовании). Другими словами, если получено подтверждение от системы, что транзакция выполнена, то сделанные в транзакции изменения не отменяются из-за какого-либо сбоя.

3.2.1.2. Требования к программному интерфейсу

ПИР должен предоставлять программный интерфейс, удовлетворяющий следующим требованиям:

Требование 4. Естественность взаимодействия. Так как основным языком прикладного программирования Платформы является Java, то программный интерфейс должен использовать основные концепции этого объектно-ориентированного языка, в частности, должен быть объектно-ориентированным. В сигнатуре программного интерфейса должны быть явно выделены абстракции для «Инфоресурса», «Понятия», «Отношения», эти абстракции должны быть представлены с помощью интерфейсов (interface) Java.

Требование 5. Как показывает опыт использования системы [34, 35], самой часто используемой операцией является навигация по инфоресурсу: получение доступа к некоторому понятию, отношению и т.д. Поэтому должен быть предоставлен удобный механизм навигации внутри инфоресурса. Вторая по популярности операция — получение значения атрибутов понятий; соответственно, программный интерфейс должен позволять легко получать значения атрибутов.

Требование 6. Должно поддерживаться соответствие метаинформации и порождаемой по ней информации (это требование почти полностью пересекается с требованием согласованности хранения). Перед каждой операцией модификации должны быть осуществлены все проверки возможности этой операции. В случае невозможности выполнить операцию должно быть сгенерировано исключение (exception) Java.

Требование 7. ПИР должен предоставлять высокоуровневый программный интерфейс для поддержки транзакций, который был бы совместим с механизмами обработки исключений Java (тройкой try-catch-finally).

3.2.1.3. Прочие требования

Требование 8. Данное требование относится к распределённой версии платформы. Должны быть предоставлены средства низкоуровневой синхронизации процессов ПРЗ: передача полной копии инфоресурса от одного процесса ПРЗ на другой, передача изменений, слияние инфоресурса и изменений, для поддержки модели параллельного выполнения из раздела 2.2.

Требование 9. Сборка мусора. Все временные инфоресурсы должны удаляться в процессе работы Платформы. Временными являются те инфоресурсы, которые после завершения работы блока продукций не достижимы из постоянных инфоресурсов по ссылкам.

3.2.2. Проект

Процессор инфоресурсов предназначен для осуществления взаимодействия с инфоресурсами. Процессор состоит из двух модулей: модуль хранения и модуль представления (Рис. 41).

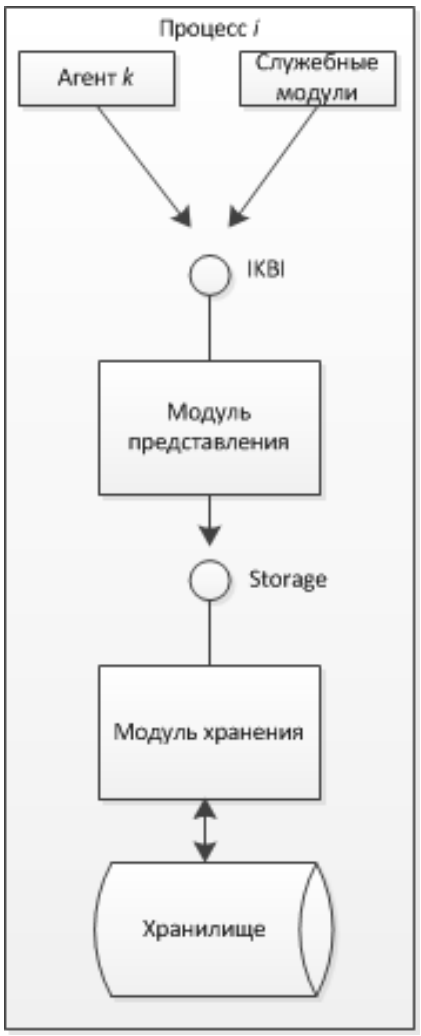

Рис. 41. Процессор инфоресурсов в последовательной версии.

В проекте распределённой версии модуль хранения расширен до модуля хранения и синхронизации, для того, чтобы организовать взаимодействие между несколькими процессами: см. Рис. 42

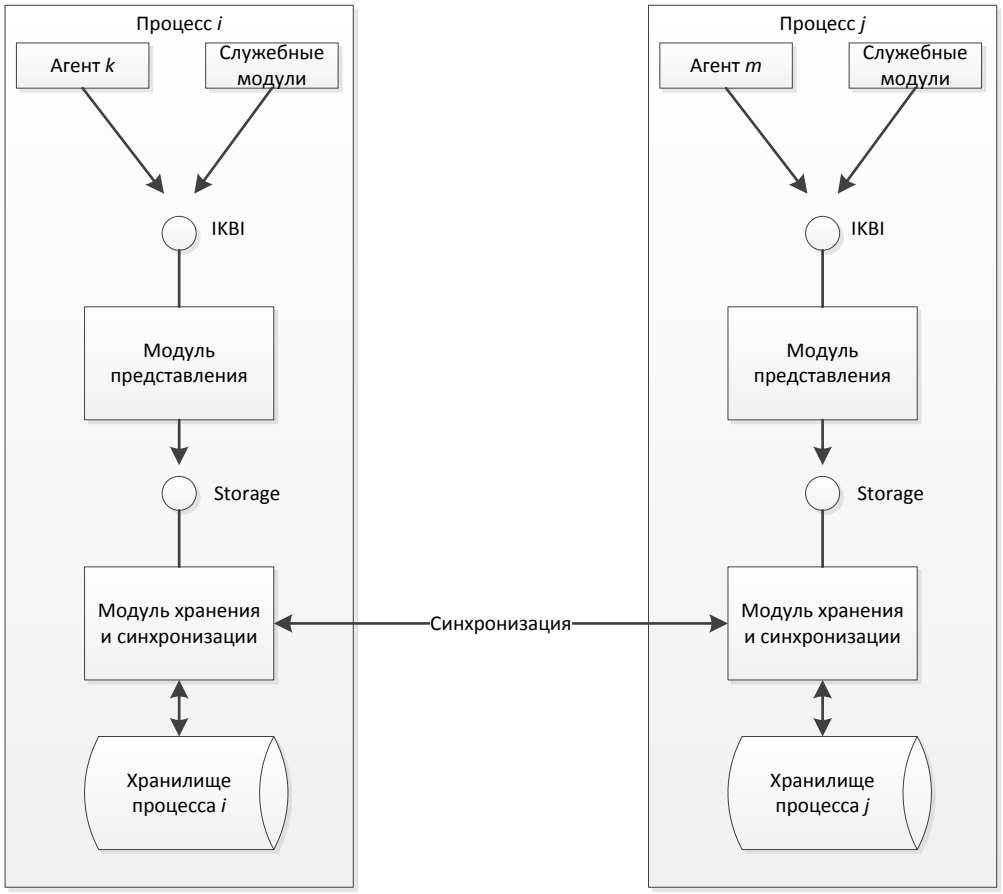

Рис. 42. Процессор инфоресурсов в распределённой версии

Как ясно из названия, модуль хранения используется для сохранения данных Фонда во внешней памяти (диске) и последующего доступа к сохранённым данным. Данные хранятся в виде файлов с определённой структурой (см. ниже), модуль осуществляет чтение и интерпретацию этих файлов. Также модуль хранения осуществляет проверку целостности файлов и восстановление после сбоев. Кроме того, модуль предоставляет низкоуровневые элементарные функции по доступу к содержимому инфоресурсов, такие как «создать вершину-понятие», «связать две вершины-понятия дугой-отношением», «изменить атрибут вершины или дуги» и т.д.

Модуль представления предоставляет высокоуровневые программные интерфейсы для остальных модулей Платформы, такие как «инфоресурс», «метаинформация», «понятия», «метапонятия», «отношение». Кроме того, модуль представления поддерживает соответствие между метаинформацией

и порождёнными инфоресурсами таким образом, чтобы любой порождённый инфоресурс синтаксически соответствовал метаинформации.

3.2.2.1. Модуль хранения

Основное назначение модуля — предоставить интерфейс для обработки инфоресурсов как графов и скрыть (инкапсулировать) детали взаимодействия с ними, такие как хранение инфоресурсов во внешней памяти, транзакции и восстановление после сбоев, синхронизация с другими процессами Платформы. Вторая задача модуля — получить более эффективный доступ к аппаратным возможностям кластера, по сравнению с основным языком разработки платформы (Java).

Будем рассматривать функциональность модуля снизу вверх, начиная с отдельных элементов хранения и заканчивая предоставляемыми программными интерфейсами.

Для реализации **требования 1**, все данные Фонда разбиты на инфоресурсы-графы. Каждый инфоресурс хранятся во внешней памяти в наборе файлов, весь конгломерат файлов называется *Хранилищем*.

Инфоресурс, как граф, состоит из вершин, связей между ними и атрибутов вершин и дуг. Заметим, что метаинформация представляется также в виде инфоресурсов; интерпретация инфоресурса как метаинформации осуществляется в следующем модуле — модуле представления. Таким образом, модуль хранения оперирует только с концепцией инфоресурса.

Модуль предоставляет программный интерфейс `Storage`, подробно рассмотренный в Приложении 1, который предназначен для низкоуровневой обработки инфоресурсов и синхронизации с другими процессами Платформы. Вышележащий модуль представления использует его в своей работе.

Сборка мусора. Для реализации **требования 9**, модуль хранения предоставляет функционал для удаления ненужных инфоресурсов. Данный механизм реализуется следующим образом:

1. В Фонде выделяются несколько инфоресурсов, которые объявляются «постоянными»;

2. Если в некоторый инфоресурс существует хотя бы одно отношение из постоянного инфоресурса в другой, то этот другой инфоресурс объявляется *достижимым из постоянного инфоресурса* (или просто *достижимым*). Если в некоторый инфоресурс ведёт отношение из достижимого, он также считается достижимым;

3. Все остальные инфоресурсы считаются *недостижимыми*.

Модуль хранения предоставляет функцию `collectGarbage()`, которая, обходя все инфоресурсы, находит достижимые и удаляет оставшиеся недостижимые инфоресурсы.

Поддержка транзакций. Для реализации **требований 2** и **3** модуль хранения предоставляет три функции: запуска транзакции, фиксации транзакции и её отмены. Любое взаимодействие с Фондом осуществляется в рамках какой-либо транзакции, т.е. между вызовами функции начала транзакции и функцией фиксации или отмены транзакции (**требование 7**).

В соответствии с моделью (см. раздел 2.3), память процессов Платформы разделена, и модифицированные данные объединяются путём слияния, поэтому реализация механизма транзакций оказывается простой: каждый раз при вызове функций модификации инфоресурса создаётся его копия, и дальше изменяется именно она. В случае удачной фиксации транзакции оригинал уничтожается, и заменяется на копию. В случае отката транзакции просто уничтожается копия. Если произошёл сбой в работе Платформы, то при следующем запуске Платформы все копии уничтожаются.

Поддержка синхронизации. В данном разделе представлен проект поддержки синхронизации ПИР.

Рассмотрим, как реализуется **требование 8**.

По аналогии с простой вычислительной моделью с параллельными процессами в вычислительной модели проекта IACPaaS при выполнении сервиса

каждая посылка сообщения m приводит: к созданию нового процесса n_2 для запуска на нем агента p_2; пересылке на него состояния ИР I_1, полученного после выполнения в процессе n_1 агента p_1, пославшего это сообщение; пересылке на него истории обработки всех переменных, полученной к этому моменту. В случае, если процессы расположены на разных компьютерах, то пересылка с компьютера на компьютер такого количества информации может свести на нет все плюсы от распараллеливания процесса выполнения сервиса. В этом разделе рассматриваются возможности сокращения объёма пересылаемой информации.

Если на каждом компьютере хранить начальное состояние ИР I_0, то пересылке может подлежать не все состояние ИР I_1, а только его изменение ΔI_1, т.е. значения тех переменных, которые были изменены при выполнении агента p_1 на первом компьютере.

Текущая история обработки всех элементов инфоресурса также может пересылаться в более экономном виде, достаточном для проверки корректности результата вычислений. При обработке сообщения m агентом p_1 в процессе n_1 для некоторого элемента v либо последовательно выполнялись только операции чтения этого элемента, либо выполнялись операции записи новых элементов и их значений и, возможно, чтения, либо по отношению к элементу v не выполнялось никаких операций вообще.

Если для элемента v при обработке сообщения m агентом p_1 в процессе n_1 выполнялись только операции чтения, а при обработке всех сообщений на пути из корня в дереве сообщений к m не выполнялось операций записи для этого элемента, то текущая история обработки этого элемента для сообщения m состоит из уникального идентификатора этого сообщения m и информации о том, что имело место только чтение.

Если для элемента v при обработке сообщения m агентом p_1 в процессе n_1 выполнялись только операции чтения, а m' есть ближайшая к m вершина на пути от корня в дереве сообщений к вершине m, при обработке которой

выполнялись операции записи элемента *v*, то текущая история обработки этого элемента для сообщения *m* представляет собой пару, первый элемент которой состоит из уникального идентификатора сообщения *m'* и информации о том, что имела место запись, а второй элемент пары состоит из уникального идентификатора сообщения *m* и информации о том, что имело место чтение.

Если для этого элемента при обработке сообщения *m* агентом p_1 в процессе n_1 выполнялись операции записи, то текущая история обработки этого элемента для сообщения *m* состоит из уникального идентификатора этого сообщения *m* и информации о том, что имела место запись.

Если для этого элемента при обработке сообщения *m* агентом p_1 в процессе n_1 выполнялись операции создания и удаления, то такой элемент в текущую историю обработки всех элементов не входит.

Если же для этого элемента не выполнялось никаких операций обработки на всем пути от корня дерева сообщений до вершины *m*, то такой элемент в текущую историю обработки всех элементов не входит.

Текущая история обработки всех элементов пересылается из процесса n_1 в процесс n_2. Далее история обработки всех элементов инфоресурса будет называться *историей инфоресурса*, или просто *историей*, если ясно, о каком инфоресурсе идёт речь.

Таким образом, история есть множество троек:

1. Элемент (понятие, отношение или атрибут);

2. Уникальный идентификатор сообщения, при обработке которого осуществлена *запись* этого элемента. Если записи элемента ещё не было, то 0;

3. Уникальный идентификатор сообщения, при обработке которого осуществлено *чтение* этого элемента. Если чтения элемента ещё не было, то 0.

В ходе последовательных обработок инфоресурсов его история дополняется. Например, если некоторый элемент *v* до обработки сообщения m_p только читался, то его сокращённая история будет иметь вид $<v, 0, ID(m_k)>$

(где $ID(m_k)$ есть уникальный идентификатор сообщения m_k), а в процессе обработки m_p была осуществлена запись, то его новая сокращённая история будет иметь вид $<v, ID(m_p), 0>$.

Некорректность результата вычислений может быть выявлена по текущей истории обработки всех элементов по его окончании при объединении текущих историй для всех терминальных вершин в дереве сообщений в одну общую, имеющую тот же вид.

3.2.2.2. Модуль представления

Основное назначение этого модуля — предоставить высокоуровневый программный интерфейс доступа к инфоресурсам и метаинформации для остальных модулей системы, прежде всего для агентов (**требование 4**). Вторая задача этого модуля — контролировать правильность порождения и редактирования инфоресурсов (**требование 6**), причём делать это «прозрачно», т.е. так, чтобы прикладной программист не создавал дополнительный код для проверки инфоресурсов (**требование 4**).

Модуль представления предоставляет программный интерфейс для других модулей Платформы и агентов из Фонда, и подробно рассмотрен в Приложении 2.

Рассмотрим ключевые идеи этого программного интерфейса:

1. Выделяются три главных программных интерфейса (**требование 5**): IInforesource, IConcept, IRelation, которые служат для доступа к инфоресурсам, понятиям и отношениям, соответственно. Эти интерфейсы предоставляют средства для навигации по инфоресурсам: получение доступа к нужным вершинам по их путевым именам, по их идентификаторам и пр.;

2. Для редактирования инфоресурсов предоставляются интерфейсы IInforesourceEditor, IConceptEditor, IRelationEditor. Эти интерфейсы расширяют предыдущие (IInforesource, IConcept, IRelation) дополнительными методами по модификации инфоресурсов: удалению понятий и отношений, изме-

нению их атрибутов. Перед каждой такой правкой осуществляется проверка семантической допустимости этой операции и, если она недопустима, операция отвергается. Таким образом, все проверки правильности операций встроены в соответствующие функции и прикладному программисту нет необходимости писать дополнительный код по проверке правильности, в этом и заключается «прозрачность» контроля;

3. Для порождения (или допорождения) инфоресурсов предоставляются интерфейсы IInforesourceGenerator, IConceptGenerator, IRelationGenerator, расширяющие интерфейсы редактирования операциями порождения. Так же, как операции редактирования, перед каждым изменением инфоресурса проверяется семантическая допустимость этой операции;

4. Реализация этих интерфейсов расположена в классах Inforesource, Concept, Relation, InforesourceEditor, ConceptEditor, RelationEditor, InforesourceGenerator, ConceptGenerator, RelationGenerator, все операции осуществляются по одной схеме:

4.1. Проверить допустимость операции;

4.2. Вызвать соответствующие методы низкоуровневого интерфейса Storage для доступа (чтения, модификации, создания) к инфоресурсу;

4.3. «Обернуть» возвращаемый результат, если он есть, в один из классов Inforesource, Concept, Relation, для дальнейшего высокоуровневого взаимодействия.

3.3. Методы реализации процессора решателей задач

3.3.1. Постановка задачи и требования к процессору решателей задач

Основное назначение ПРЗ — осуществлять работу сервисов, эта задача сводится к осуществлению запуска и работы агентов, а также осуществлению обмена сообщениями между ними.

3.3.1.1. Основные требования

Требование 1. ПРЗ должен предоставлять способ реализации агентов, соответствующий модели раздела 2.3. В частности, требуется определить, каким будет программное представление агентов и их блоков продукций, как будет осуществляться запуск, работа и остановка блоков продукций агентов.

Требование 2. ПРЗ должен предоставлять способ реализации сервисов, соответствующий модели раздела 2.3. В частности, требуется определить, как будет осуществляться запуск, работа и остановка сервисов.

Требование 3. В целях отладки сервиса должен быть предоставлен программный интерфейс для вывода отладочных сообщений в журнал работы сервиса. Каждому отдельному запуску сервиса должен соответствовать отдельный журнал. Разработчик сервисов должен иметь возможность просматривать отдельные журналы работы сервисов.

Требование 4. В случае аварийной ситуации при работе какого-либо блока продукций сервиса, работа этого сервиса должна быть полностью остановлена. Аварийными считаются следующие ситуации:

1. Возник сбой (например, исключительная ситуация) при работе блока продукций;

2. Послано сообщение агенту по шаблону, которому не соответствует никакой блок продукций в этом агенте;

3. При выполнении системного сервиса выполнена команда завершения не в блоке продукций инициализирующего агента этого сервиса.

3.3.1.2. Требования к распределению нагрузки

Требования этого раздела относятся к распределённой версии Платформы.

Требование 5. **Единица распределения.** Минимальной единицей распределения является БП. Это означает следующее:

1. Если началось выполнение некоторого БП, ПРЗ не может разделить её поток выполнения на несколько, однако допускается запуск в одно и то же время одного и того же БП на разных процессах;

2. Если ПРЗ необходимо выполнить очередной БП, он должен быть выполнен на любом свободном процессе.

Требование 6. Выбор процесса для выполнения БП. Сформированные в процессе работы БП сообщения должны приводить к запуску соответствующих БП, которые обработают эти сообщения. Выбор процесса для выполнения этих БП должен определяться при помощи алгоритма планировщика. Исключение: посылка финализирующего сообщения системному агенту. В случае, если такое сообщение послано, ПРЗ переводит приложение в фазу «завершение приложения».

Требование 7. (Требование относится к распределённой версии Платформы) Работа ПРЗ должна начинаться с запуска ПО ЦП. Работа любого сервиса становится возможна при подключении хотя бы одного РП. Для повышения производительности ПРЗ к нему подключаются один или несколько РП, на которые распределяется вычислительная нагрузка.

Все следующие требования относятся к распределённой версии Платформы.

Планировщик (scheduler) ПРЗ осуществляет распределение вычислительной нагрузки по процессам Платформы.

Требование 8. Два способа распределения. Планировщик осуществляет распределение нагрузки на двух уровнях:

1. Планирование внутри приложений: распределение работы БП приложения по процессам Платформы, управляемое схемой распределения приложения, с целью повысить эффективность работы приложения;

2. Общее планирование: общее распределение работы всех активных сервисов по процессам Платформы с целью повысить эффективность работы всех сервисов в целом.

Введение этих двух уровней необходимо из-за схемы распределения приложений (в противном случае всё планирование сводилось бы ко второму случаю).

Требование 9. Поддержка схемы распределения. На уровне внутри приложения планировщик должен распределять задания по процессам Платформы согласно схеме распределения приложения.

Требование 10. Приложения и схема распределения. С каждым приложением должна быть связана схема распределения этого приложения.

Схема распределения приложения (далее — СРП) это информационный ресурс, который указывает способ распределения вычислительной нагрузки приложения по процессам Платформы для повышения эффективности вычислений. СРП задаётся программистом приложения и меняется (отлаживается) им для повышения эффективности приложения.

Требование 11. Представление СРП. Схема распределения — это информационный ресурс.

Требование 12. Содержимое СРП. СРП должен определять для каждого агента, запускается ли обработка посланных агентом сообщений в том же процессе, или на других процессах Платформы (в первом случае получается экономия на передаче информации между процессами, за счёт последовательной обработки, во втором случае — появляется возможность параллельной обработки, за счёт потерь при передаче информации между процессами).

Требование 13. Последовательная обработка. Если агент посылает одно или несколько сообщений для последовательной обработки, планировщик должен добавить эти сообщения в множество необработанных сообщений в процессе и, извлекая сообщения из этого множества, обработать их. Порядок извлечения сообщений из множества не регламентируется (другими словами, прикладной программист приложения не должен рассчитывать что посланные сообщения обработаются в каком-то определенном порядке). Планировщик должен последовательно обработать все сообщения из этого

множества: запустить обработку первого извлеченного сообщения, дождаться завершения обработки (или сбоя), запустить обработку второго сообщения и т.д. Если при обработке таких сообщений вновь посланы сообщения для последовательной обработки, они должны быть добавлены в то же множество (без регламентации порядка извлечения сообщений из этого множества).

Требование 14. Параллельная обработка. Если агент посылает одно или несколько сообщений для параллельной обработки, планировщик должен обработать каждое из сообщений в отдельном процессе (на существующем свободном или вновь созданном). В случае, если все посланные агентом сообщения для параллельной обработки, то одно из этих сообщений допускается обработать в том же процессе.

3.3.2. Проект

3.3.2.1. Предпосылки к архитектуре

ПРЗ должен решать ряд противоречивых задач, выделим их:

1. ПРЗ должен быть тесно интегрирован с ПИР, но ПИР не должен быть частью ПРЗ; их объединение существенно усложнит отладку, тестирование и дальнейшее сопровождение;

2. Разработка распределённой версии значительно сложнее, чем разработка последовательной; реализация именно распределённой версии замедлило бы разработки на Платформе. Поэтому необходимо было реализовать сначала последовательную версию, а после — распределённую;

3. ПРЗ должен быть кроссплатформенным но, в то же время, эффективно поддерживать специфичную аппаратуру, в частности, сеть Myrinet кластера ИАПУ ДВО РАН;

4. ПРЗ должен быть лёгок в сопровождении, следовательно, ключевые модули должны быть созданы на простых языковых платформах (например, Java); в то же время необходимы специфичные возможности управления, что

требует привлечения языковых платформ с низкоуровневыми возможностями;

5. Архитектура ПРЗ должна быть интегрированной, но в то же время лёгкой для тестирования (высокая интегрированность затрудняет модульное тестирование); для разработчиков Платформы важны быстро выполнимые локальные тесты, но для полноценного тестирования нужны сетевые тесты.

Для преодоления этих конфликтов предлагаются следующие решения.

1. БП агентов выполняется внутри процессов ОС. Каждый процесс при этом выполняет только один поток управления агента. Тем самым, в каждый момент времени внутри процесса работает не более одного блока продукции. Таким образом реализуется **требование 5**. Распределение достигается за счёт множества запущенных процессов. Такой подход удобен по следующим причинам:

1.1. Унификация программного обеспечения и распределение одним способом. Хотя современные ОС и языковые платформы позволяют создавать многопоточные приложения, но выбор этого способа приводит к необходимости проектировать распределение на трёх уровнях: на низком уровне распределения потоков внутри процесса, на среднем уровне распределения процессов внутри многопроцессорного компьютера и на высоком уровне распределения вычислительной нагрузки по компьютерам ВФ. Проектное решение об отказе от потоков упрощает проектирование за счёт устранения низкого уровня управления потоками. Кроме того, процессы универсальны для всех современных ОС, а реализации потоков различаются в разных ОС, что может привести к усложнению сопровождения ПРЗ.

1.2. Изоляция процессов. Сбои в БП могут быть очень серьёзными, вплоть до распространения сбоя на весь процесс. Такой сбой не повлияет на остальные работающие БП, т.к. они выполняются в других процессах, а сбойный процесс будет завершён и вместо него будет запущен другой.

1.3. Особенности ОС, связанные с управлением памятью. Транзакции могут быть легко реализованы низкоуровневыми средствами самой ОС; но такие возможности доступны на уровне процессов ОС, а не на уровне потоков.

1.4. Упрощение поддержки транзакций на уровне языка. В случае потоковой реализации пришлось бы предоставлять прикладному программисту специальные средства синхронизации, что становится источником ошибок. В случае однопотоковой реализации возможно создать эту поддержку на нижних уровнях, снижая ответственность прикладного программиста;

2. Выделяются два уровня системы: Java-уровень и Native-уровень:

2.1. Native-уровень служит для доступа к низкоуровневым функциям системы, за счёт которых осуществляется следующее: быстрая загрузка и сохранение, поддержка транзакций средствами ОС; тонкая адаптация к оборудованию (в частности, к аппаратуре Myrinet); быстрая операция синхронизации процессов;

2.2. Java-уровень создаётся на Java и содержит высокоуровневый функционал;

2.3. В целом, Native-уровень выполняет простейшие операции, типа «переслать сообщения с одного компьютера на другой». Java-уровень выполняет более продвинутую маршрутизацию, например, определением наименее загруженного компьютера и пересылкой сообщения туда. Основной мотив — упрощение дальнейшего сопровождения: сопровождение кода на Java потребует меньших ресурсов.

3. Транзакции реализуются поверх подходящих средств ОС: разделяемой памяти (shared memory), отображаемых в память файлов (memory mapped files). Эти средства имеют в настоящее время эффективную поддержку как со стороны аппаратуры, так и со стороны ОС, поэтому предполагается высокая эффективность транзакционного механизма.

4. Создаётся специальный протокол передачи сообщений с соответствующей программной поддержкой между процессами. Цель — достичь минимальных накладных расходов на протоколах: TCP/IP (и, с некоторыми ограничениями, UDP), MPI, GM.

5. Native-уровень ПРЗ создаётся как расширение Native-уровня ПИР путём добавления соответствующей функциональности. Тем самым достигается интегрированность модулей. ПРЗ и ПИР работают совместно для поддержки транзакций и организации сетевого взаимодействия. В то же время Native-части модулей ПРЗ/ПИР чётко разделены; ожидается, что это снизит накладные расходы на отладку и дальнейшее сопровождение и развитие.

3.3.2.2. Уровни ПРЗ

Выделим в Платформе несколько уровней (см. Рис. 43):

Рис. 43. Уровни ПРЗ

1. Общесистемный уровень. Этот уровень представлен контентом и решателями задач при поддержке Административной системы и CMS «MediaWiki».

2. Сервисный уровень. На этом уровне рассматриваются сервисы, полномочия запуска сервисов, а также детали поддержки Административной системы.

3. Агентный уровень. На этом уровне рассматривается функционирование агентов в контексте сервиса и окружения вообще, в частности, рассмотрен вопрос взаимодействия с другими системами.

4. Блокопродукционный уровень. На этом уровне рассматривается функционирование агента при обработке сообщения, а также его внутреннее устройство.

5. Уровень процессов. На этом уровне рассматриваются функционирование процессов: подготовка всей системы к запуску, связывание процессов друг с другом, запуск агентов внутри процессов и т.п.

6. Уровень ПИР. ПРЗ в своём функционировании опирается на возможности, предоставляемые ПИР. ПИР используется для хранения инфоресурсов, в том числе системных (инфоресурсы агентов, сервисов и т.п.), а также для синхронизации процессов.

3.3.2.3. Общесистемный уровень

На этом уровне Платформа рассматривается как набор сервисов, решающих разные прикладные и системные задачи. Все эти сервисы содержатся в Фонде.

Выделяется специальный системный сервис — Административная система (см. раздел 4.3.1), которая используется для управления доступом пользователей к Платформе.

Пользователь взаимодействует с Платформой с помощью Административной Системы следующим образом (см. Рис. 44):

1. Пользователь может взаимодействовать с Административной Системой для получения какой-либо информации, например, списка информаци-

онных ресурсов, доступных полномочий запуска, либо для управления своим доступом;

2. Пользователь может запустить некоторый сервис для решения своих задач, для этого он пользуется соответствующей функцией Административной Системы для запуска сервиса, та, проверяет права пользователя и, если те позволяют запустить сервис, обращается к нижележащим компонентам Платформы для запуска сервиса, после чего пользователь начинает взаимодействовать с запущенным сервисом.

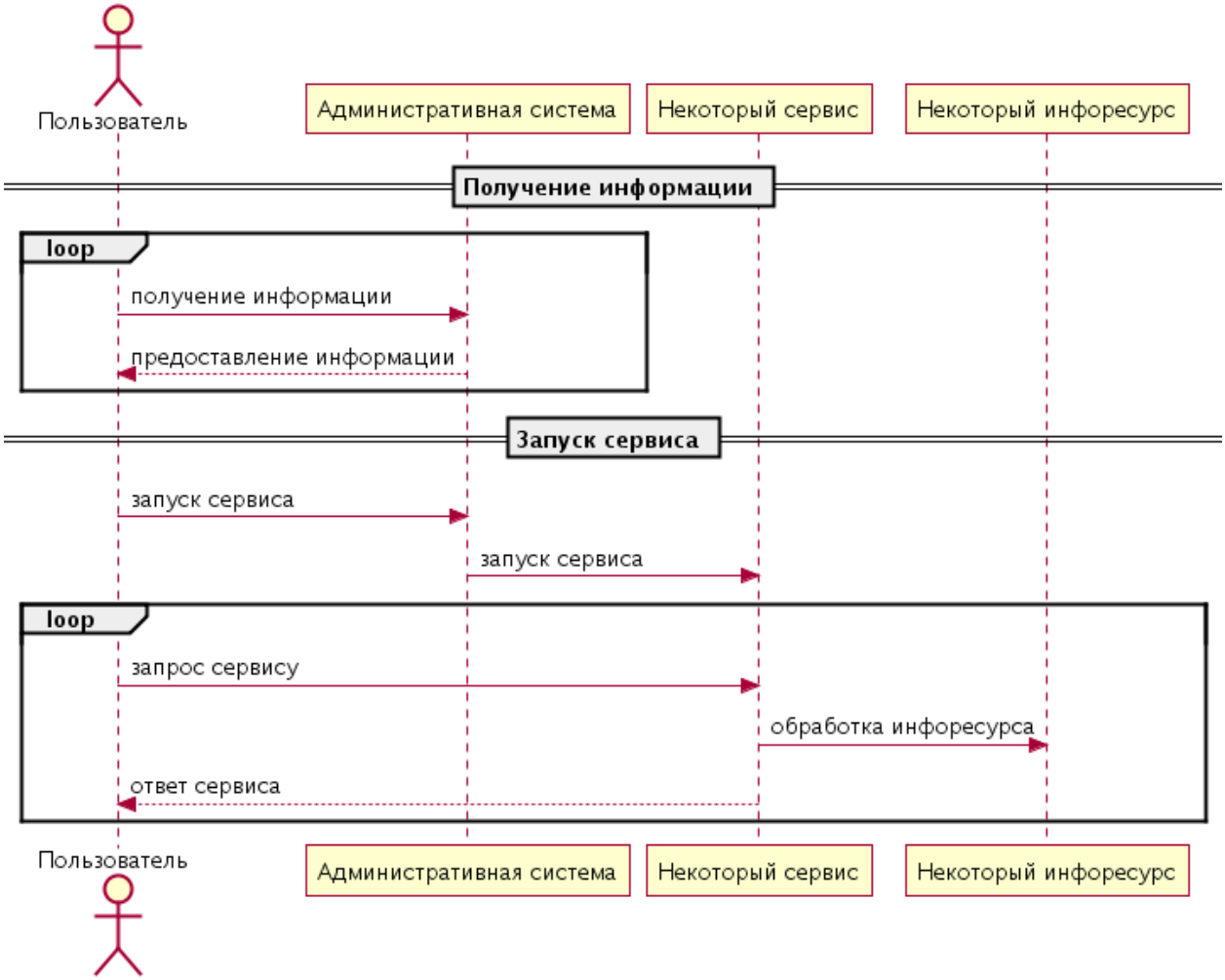

Рис. 44. Диаграмма последовательности «работа с сервисом»

3.3.2.4. Сервисный уровень

Структура. Рассмотрим внутреннее устройство сервиса и его контекст (Рис. 45).

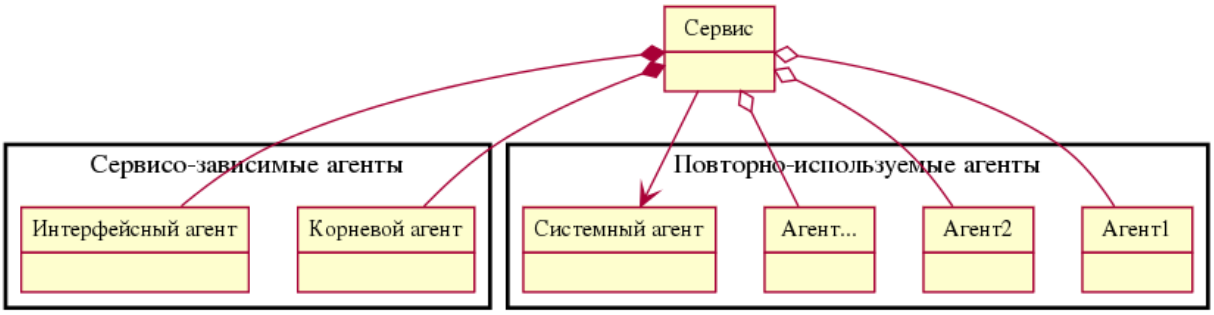

Рис. 45. Схематичное устройство сервиса

Сервис есть совокупность агентов, часть из которых — повторно-используемые. Среди агентов сервиса выделяются несколько особенных:

1. Корневой агент. Выполнение сервиса начинается с передачи этому агенту сообщения. Корневой агент создаётся разработчиком сервиса.

2. Интерфейсный агент. Агент, используемый для взаимодействия с пользователем. Интерфейсный агент также создаётся прикладным программистом; но он менее специфичен, чем корневой агент, поэтому возможно его более широкое повторное использование.

3. Системный агент. Используется для взаимодействия с нижележащими модулями системы. Например, Административная Система обращается к Системному Агенту для запуска сервиса, Прогонщик тестов (см. раздел 4.3.4) обращается к Системному Агенту для запуска тестируемого агента в некотором контексте и т.п. Системный агент является повторно используемым агентом и создаётся системными разработчиками.

Сервис включает в себя по крайней мере одного агента — корневого агента сервиса.

С каждым сервисом связаны три типа инфоресурсов (см. Рис. 46):

1. Входные инфоресурсы, в которых содержится входные данные для работы сервиса,

2. Выходные инфоресурсы, в которые сервис записывает результаты своей работы,

3. Собственные инфоресурсы, используемые для автоматического управления сервисом.

Рис. 46. Классы инфоресурсов сервиса

Каждое описание сервиса есть инфоресурс, порождённый по метаинформации «Структура сервиса» (см. Рис. 47). Таким образом выполняется **требование 2**.

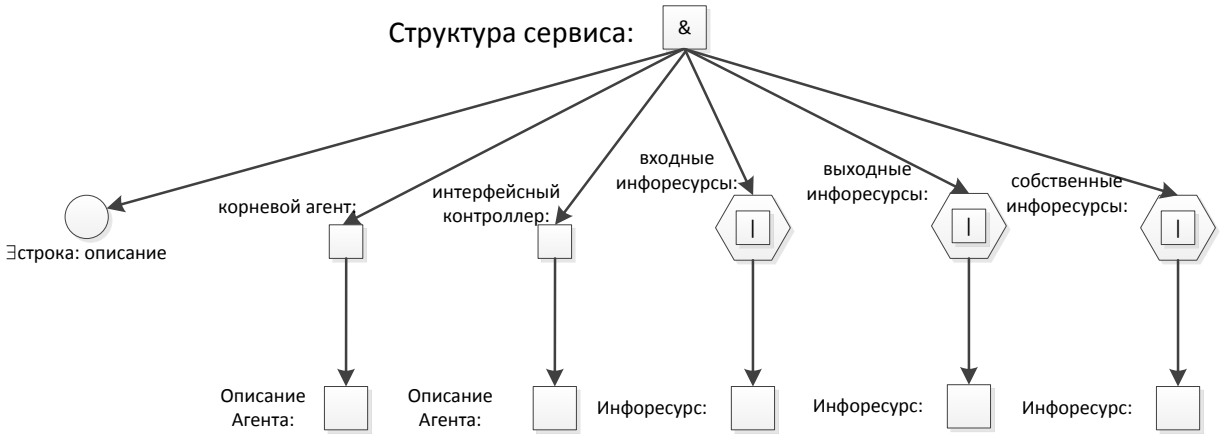

Рис. 47. Метаинформация «Структура сервиса»

В описании сервиса присутствует:

1. Текстовое описание сервиса (для пользователей системы);

2. Указание корневого агента;

3. Необязательное указание агента-интерфейсного контроллера;

4. Указание типов обрабатываемых инфоресурсов: ссылки на метаинформации для допустимых входных, выходных и собственных инфоресурсов.

Поведение. Для запуска сервиса Административная Система обращается к Системному агенту с соответствующим запросом (вообще говоря, запустить сервис может любой другой сервис, например, Прогонщик тестов, при наличии соответствующих прав). На Рис. 48 показано поведение Платформы при запуске сервиса, его работе и остановке.

На схеме показано, что запуск основных агентов осуществляется «Системным агентом». Однако, это эмулируемое поведение: просто в очередь сервиса помещается сообщение с обратным адресом «Системный агент».

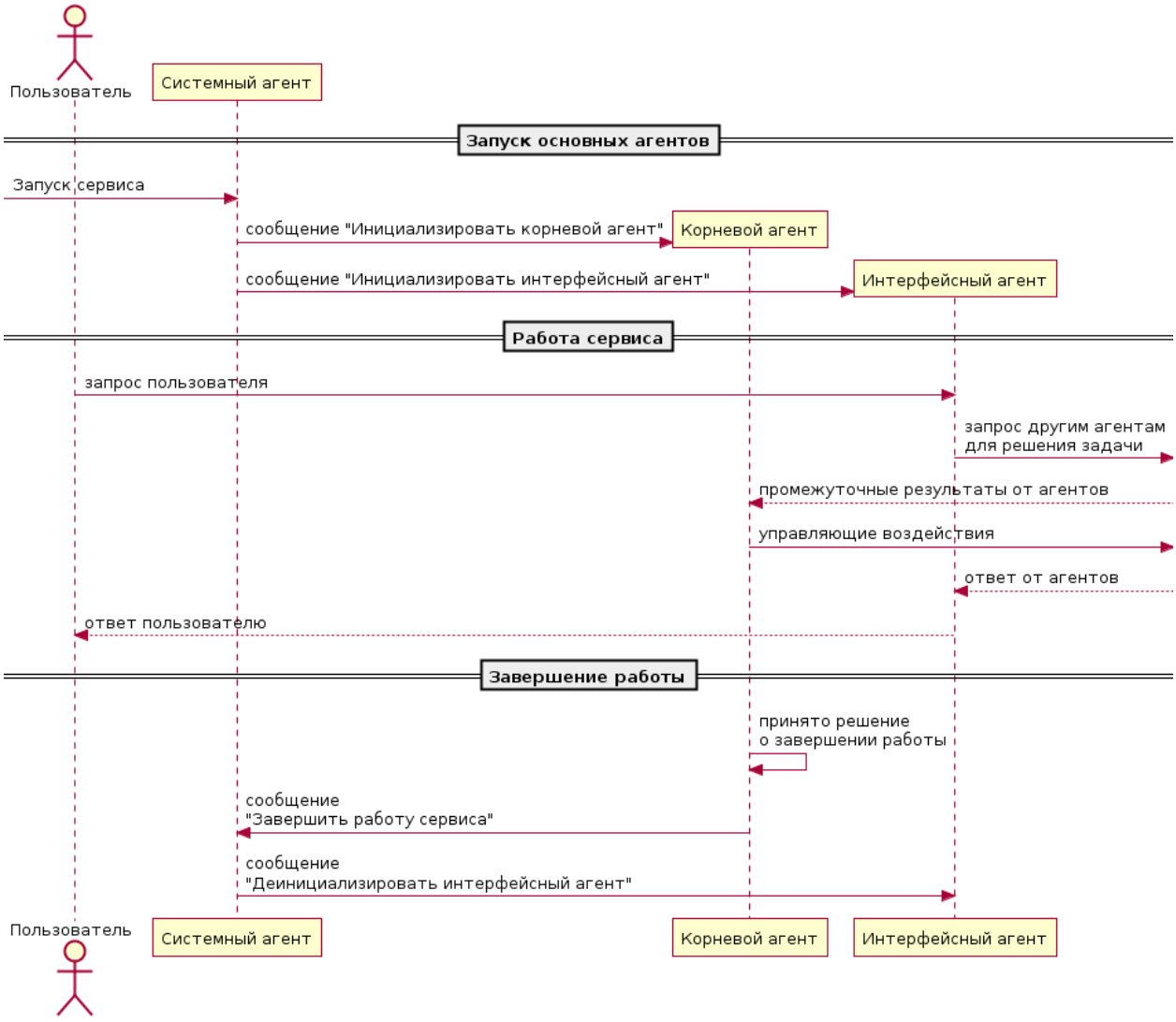

Рис. 48. Диаграмма последовательности «Запуск и работа сервиса»

На Рис. 49 более подробно рассмотрен процесс работы сервиса. Внешняя система должна взаимодействовать с сервисом посредством агентов ввода-вывода (обозначенный как «ВВ-агент» на рисунке). Агент ввода-вывода определяет, какой из агентов будет обрабатывать поступивший запрос и отправляет ему соответствующее сообщение. После обработки агенту ввода-вывода поступает результирующее сообщение (возможно, от какого-то другого агента системы), которое он транслирует внешней системе.

Способ обработки запросов от внешней системы (т.е. поведение агентов ввода-вывода и прочих агентов системы) определяется разработчиком сервиса.

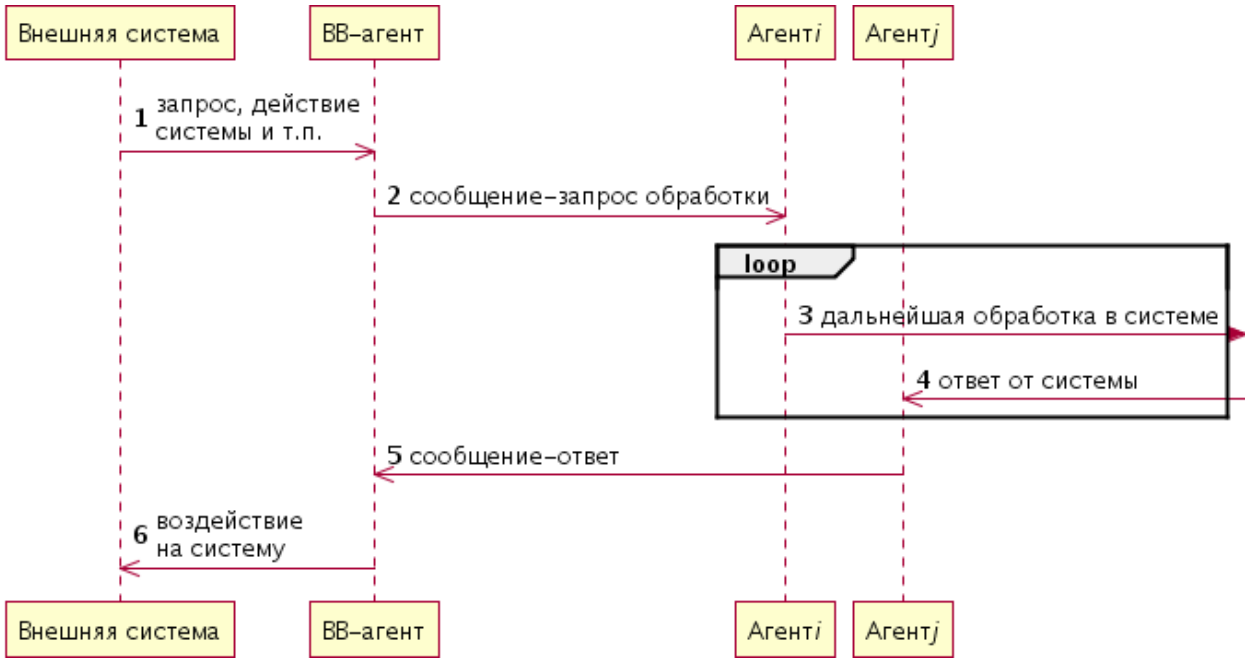

Рис. 49. Диаграмма последовательности «Работа сервиса»

Запуск сервиса осуществляется следующим образом:

1. Запускающий сервис посылает сообщение Системному агенту о необходимости запустить сервис.

2. Системный агент подготавливает новый контекст для запуска.

3. Системный агент формирует очередь сообщений, состоящую из одного сообщения «Инициализировать сервис».

4. После этого Система обрабатывает это сообщение, в новом, только что созданном контексте: вызывает корневой агент и передаёт ему сообщение «Инициализировать сервис». Далее работа сервиса определяется работой корневого агента.

Рассмотрим теперь, как происходит работа сервиса. Платформа циклически читает новые сообщения из очереди и обрабатывает их согласно диаграмме на Рис. 50.

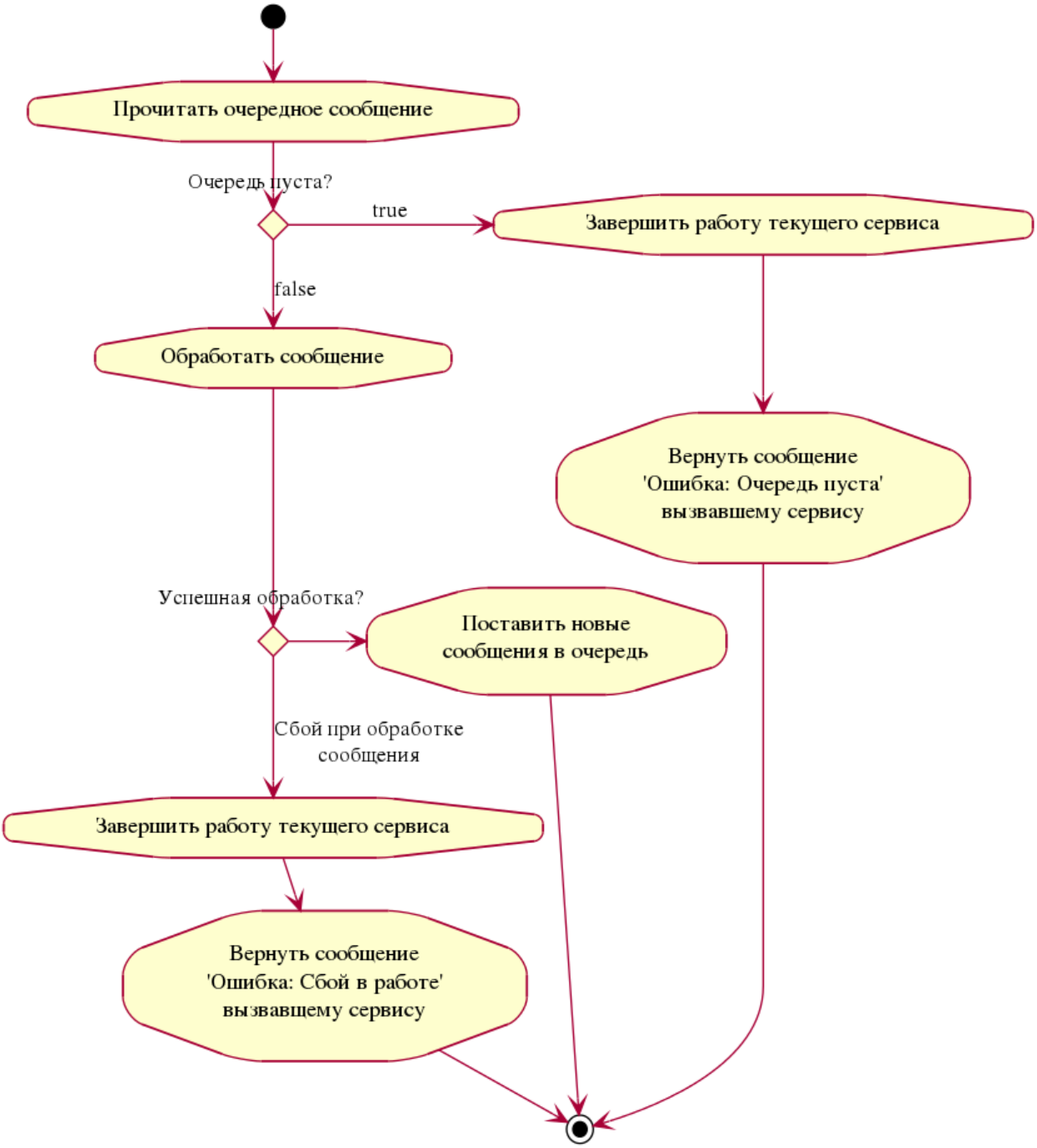

Рис. 50. Диаграмма деятельности «Работа сервиса»

Обработка сообщения рассмотрена в разделах 3.3.2.5 и 3.3.2.6. В случае сбоев (**требование 4**) при обработке сообщения ПРЗ останавливает работу сервиса и посылает сообщение-ошибку вызвавшему сервису.

3.3.2.5. Агентный уровень

Структура. Агенты представляются инфоресурсами с метаинформацией, представленной на Рис. 51. Тем самым выполняется **требование 1**. Метаинформация агента является, таким образом, его спецификацией.

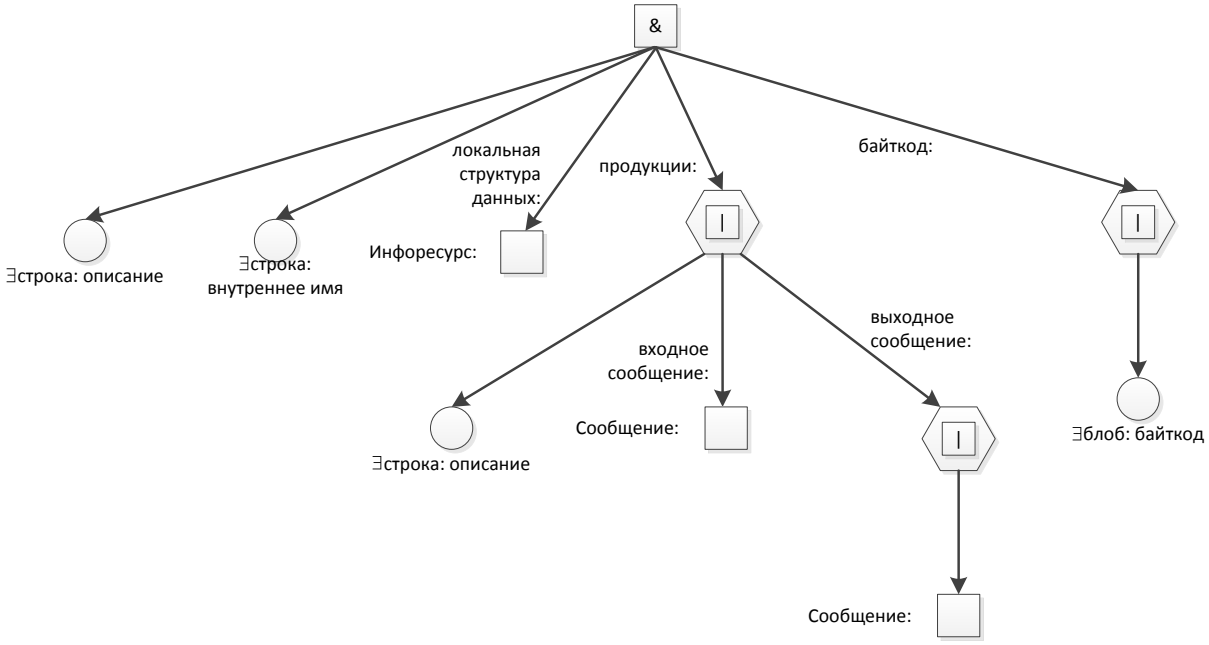

Рис. 51. Метаинформация «Структура агента»

Инфоресурсы агентов используются:

1. Прежде всего, самим процессором решателей задач — для поиска и передачи управления нужному методу Java-класса агента;

2. Генератором Агента (см. раздел 4.3.5) для создания «заглушки» кода, по которому прикладной программист агентов создаст код реализации;

3. Программистом сервиса как спецификации агентов для выбора агентов, используемых сервисом.

В инфоресурсе агента представлено название агента, название Java-класса (терминал «внутреннее имя») и соответствующий байткод (нетерминал «код») реализации. Реализация может состоять из нескольких Java-классов, но только один из них является классом агента, остальные классы вспомогательные (прикладной программист агента сам определяет разбиение

реализации на агенты). Код агента должен удовлетворять требованиям, приведённым в разделе 4.2.4.

Все блоки продукций агента представлены поддеревом «множество блоков продукций». Для каждого блока продукций задаётся его описание, ссылка на обрабатываемое сообщение и перечисляются допустимые выходные сообщения.

«Локальная структура данных» задаёт описание структуры данных, которые может сохранить агент между запусками блоков продукций.

Сообщения представляются инфоресурсом с метаинформацией, представленной на Рис. 52. Сообщение имеет имя и собственную структуру для данных, передаваемых в сообщении.

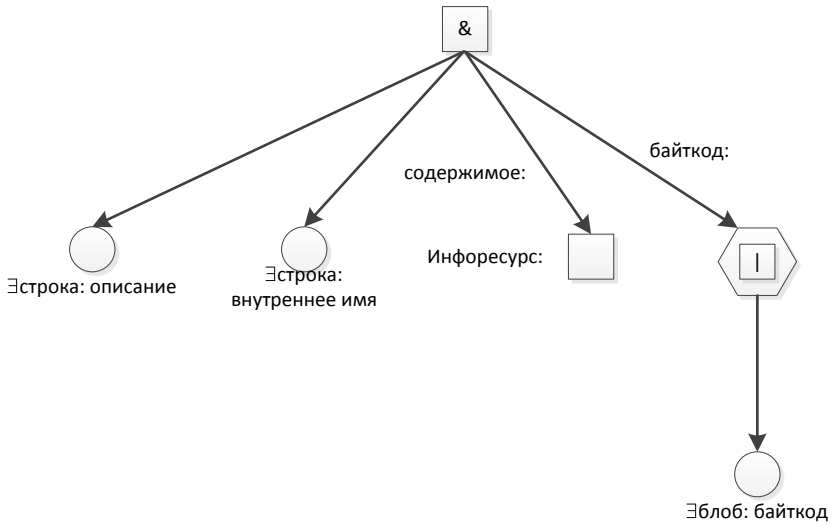

Рис. 52. Метаинформация «Структура сообщения»

Для удобства обращения к данным из агентов, прикладной программист может дополнить сообщения дополнительным Java-кодом (который размещается в нетерминале «код»); этот код будет доступен при выполнении соответствующего блока продукций.

Реализация агента задаётся Java-классом, наследующим от класса Agent. Базовый класс предоставляет ряд методов для доступа к инфоресурсам: входным (метод getInputs()), выходным (метод getOutputs()), собственным (getOwns()), а также локальным данным агента (метод getLocal()).

Каждому блоку продукций агента должен соответствовать метод runProductionBlock() с соответствующей сигнатурой. Разработчику агента предоставляется утилита «Генератор кода агента» для автоматизации создания этих методов (см. раздел 4.3.5).

Поведение. Агенты активируются, когда им приходит некоторое сообщение. Это сообщение запускает соответствующий обработчик (блок продукций). На Рис. 53 показана схема работы ПРЗ при обработке сообщения.

Для реализации **требования 3** разработчику сервисов предоставляется специальные функции вывода логов — trace(), debug(), info(), warning(), error() — которые используются для вывода диагностических сообщений в журнал работы сервиса. После неудачных запусков сервиса разработчик может просмотреть журнал и изучить работу сервиса по оставшимся отладочным сообщениям.

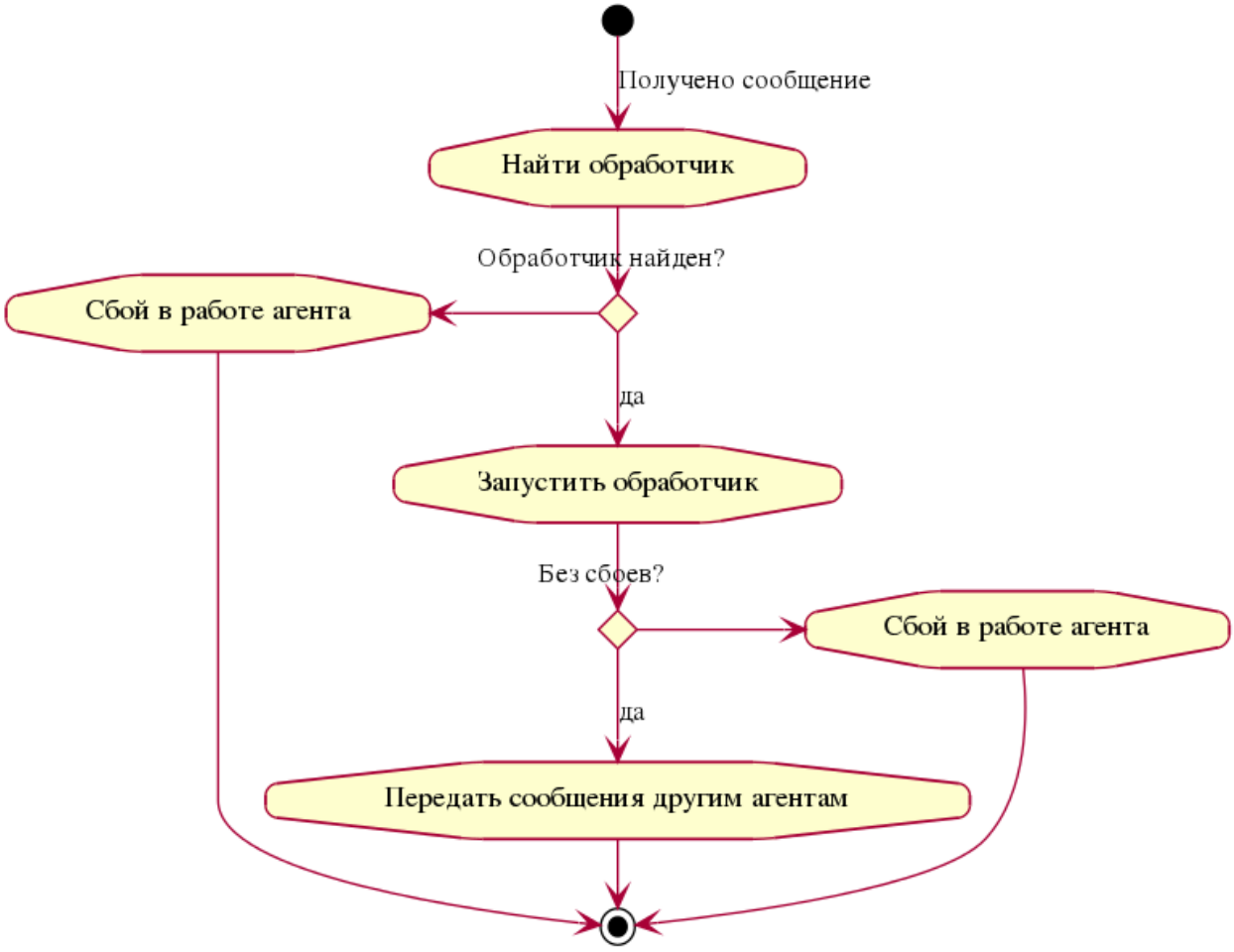

Рис. 53. Процесс обработки сообщения агентом

Специальные виды агентов. Существуют несколько специальных видов агентов:

1. Корневые агенты. С этого агента начинается работа каждого сервиса. Ничем не отличается от остальных агентов, за исключением следующих особенностей: Корневые агенты создаются отдельно для каждого сервиса, они обязательно должны обрабатывать сообщение «Инициализировать сервис».

2. Агенты ввода-вывода. Эти агенты используются для связи с внешними системами. Разновидностями агентов ввода-вывода являются интерфейсные агенты;

3. Системный агент, который предоставляет доступ к функциям Платформы из сервиса.

По своей структуре агенты ввода-вывода — «гибридные», т.е. состоят из двух частей: чисто-агентной и внешне-системной частей.

Гибридный агент должен удовлетворять требованиям к байткоду агентов (т.е. у него должны быть обработчики сообщений и т.д.). Требований к гибридной части не выдвигается: она может быть произвольной, сообразно требованиям разработчика этого агента.

Агенты ввода-вывода создаются как обычные агенты (например, сервисом «Генератор агентов»), но дополняются следующими сущностями:

1. Конструктор класса с параметром Configuration, вызываемый при создании агента, и используемый для его настройки,

2. Метод clean(), в котором агент ввода-вывода может содержать операции финализации агента, и который вызывается при штатном завершении процесса,

3. Методами взаимодействия с внешней системой (определяются разработчиком агента).

3.3.2.6. Блокопродукционный уровень

Отличия последовательной и распределённой версий на уровне процессов. В настоящее время Платформа поддерживает запуск трёх процессов: центрального процесса, рабочего процесса и процесса ввода-вывода для поддержки пользовательского интерфейса. Однако, в отсутствии модуля синхронизации в ПИР (который планируется реализовать в распределённой версии), невозможна полноценная распределённая обработка сообщений: при обработке блока продукций осуществляется монопольный доступ к Хранилищу, поэтому параллельная обработка нескольких блоков продукций невозможна.

Однако возможна поддержка многозадачности за счёт разделения времени. Платформа использует невытесняющую многозадачность и карусельную (round robin) диспетчеризацию: первый в очереди блок продукций вы-

полняется целиком, затем выполняется блок продукций другого запущенного сервиса и т.д. Тем самым обеспечивается равномерная работа сервисов.

В режиме невытесняющей многозадачности поддержка работы сервисов становятся чувствительной к «зависаниям» в работе отдельных продукций. Для этого в поддержку выполнения блоков продукций введены таймауты: если блок продукций выполняется больше 1 минуты, это детектируется как сбой в сервисе; работа этого блока продукций приостанавливается и начинается работа следующего.

Поведение. На блокопродукционном уровне происходит обработка сообщений конкретным агентом. На агентном уровне ПРЗ выбирает обрабатывающий сообщение Java-класс агента, а на блокопродукционном ПРЗ выбирает конкретный метод этого Java-класса для обработки сообщения (см. Рис. 54). Тем самым реализуются **требования 5 и 6.**

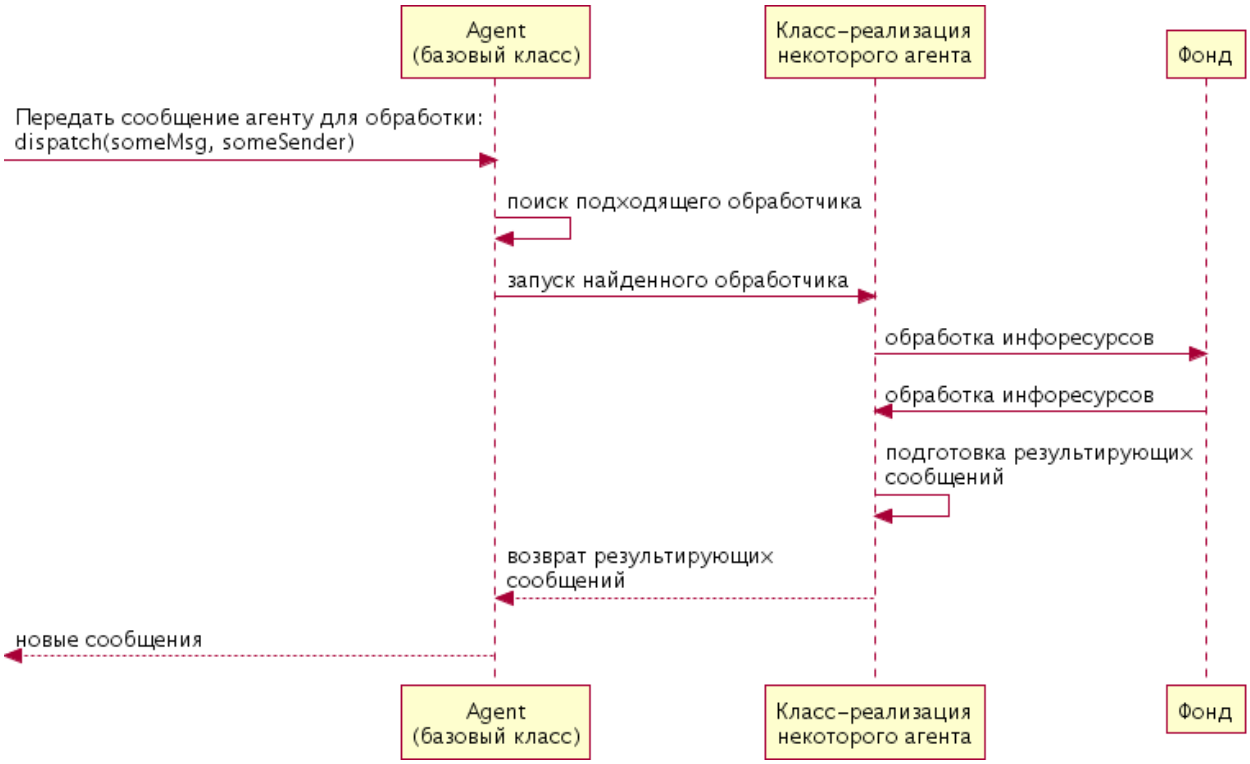

Рис. 54. Обработка сообщения (на блокопродукционном уровне).

3.3.2.7. Уровень процессов

На этом уровне система рассматривается как набор связанных друг с другом процессов Платформы, на которые распределяется вычислительная нагрузка (см. Рис. 55).

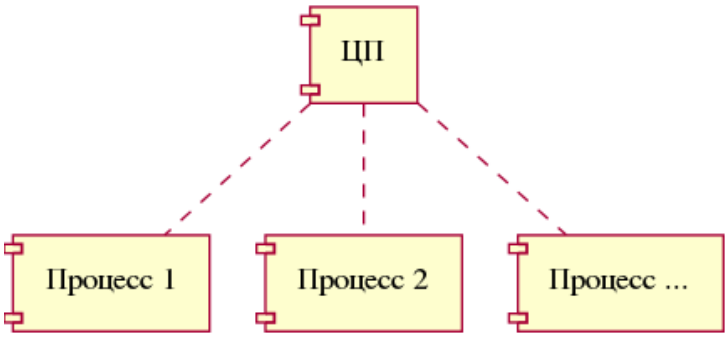

Рис. 55. Структура процессов Платформы

Процессы связаны друг с другом с помощью некоторого транспортного сетевого уровня операционной системы, например, TCP [23].

Выделяются следующие типы процессов:

1. Центральный процесс (далее — «ЦП»). Во всей сети представлен единственным экземпляром и служит для организации взаимодействия всех процессов и хранения Фонда.

2. Рабочий процесс (далее — «РП»). Модули, на которых выполняются блоки продукций агентов.

3. Процесс ввода-вывода (далее — «ВВП»). Разновидность РП, на которых выполняются блоки продукций агентов ввода вывода.

Поведение. Рассмотрим взаимодействие всех процессов в целом (см. Рис. 56):

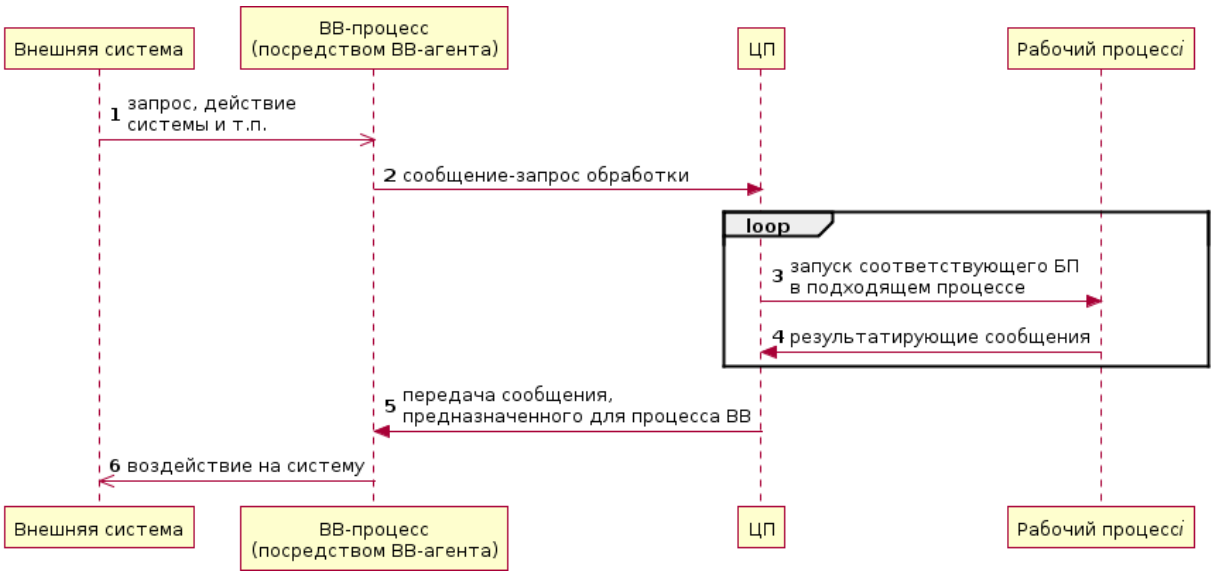

Рис. 56. Взаимодействие на уровне процессов.

1. Внешняя система (или пользователь посредством некоторого программного обеспечения) взаимодействует с процессом ввода-вывода.

2. Агент ввода-вывода (запущенный в рамках процесса ввода-вывода) преобразует воздействие от внешней системы в сообщение для сервиса для обработки и посылает соответствующим (определяется разработчиком сервиса) агентам.

3. ЦП определяет, каким процессом надо обработать это сообщение, передаёт на этот процесс команду обработки сообщения.

4. Блок продукций, запущенный на некотором РП, обрабатывает сообщение и, возможно, формирует новые. Они передаются на ЦП для дальнейшего распределения.

5. Шаги 3 и 4 повторяются в цикле до тех пор, пока для агента ввода-вывода не будет сформировано результирующее сообщение. ЦП передаёт это сообщение процессу, на котором запущен агент ввода-вывода, который обрабатывает его (т.е. запускается соответствующий блок продукций);

6. Агент ввода-вывода выдаёт ответ внешней системе.

Работа платформы начинается с запуска трёх процессов (**требования 7-14**): ЦП, РП и ВВП «Веб-процесс». В такой минимальной конфигурации на

единственном РП выполняются все блоки продукций всех агентов всех сервисов, Веб-процесс служит для взаимодействия с внешним миром, а ЦП распределяет нагрузку между ними. Все процессы запускаются либо вручную обслуживающим персоналом платформы, либо соответствующими утилитами автоматического запуска операционной системы. На Рис. 57 показана последовательность ручного запуска процессов.

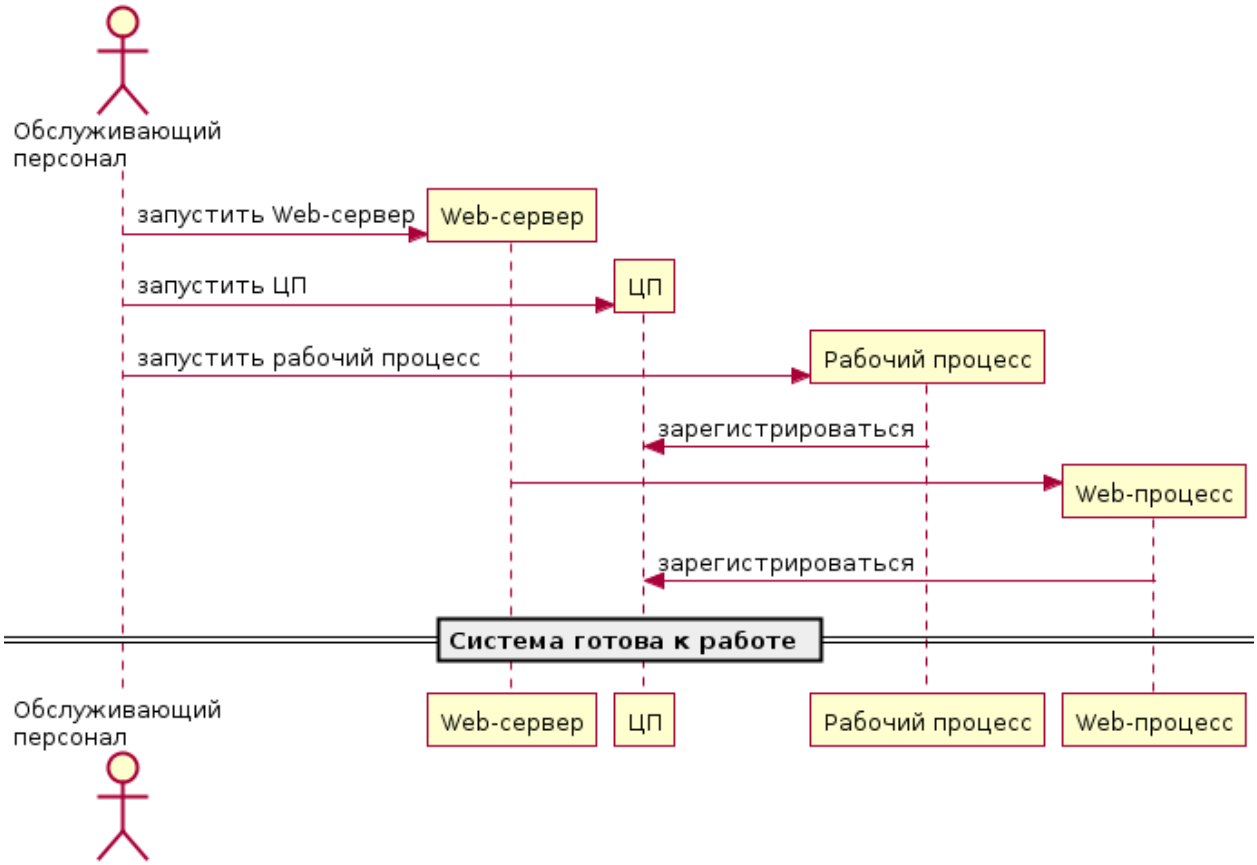

Рис. 57. Запуск процессов

Структура. Модуль ЦП устроен следующим образом (Рис. 58):

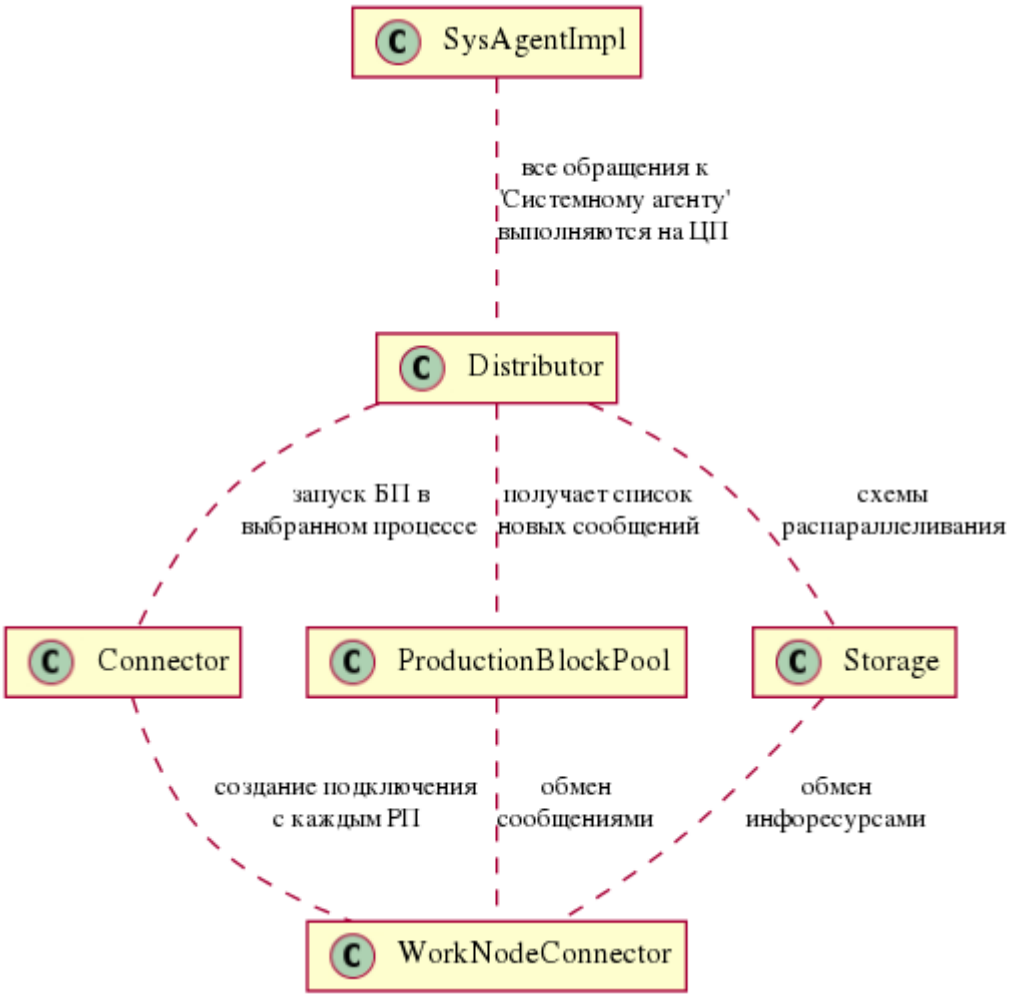

Рис. 58. Структура ЦП, нижний уровень

Модуль РП устроен так (см. Рис. 59):

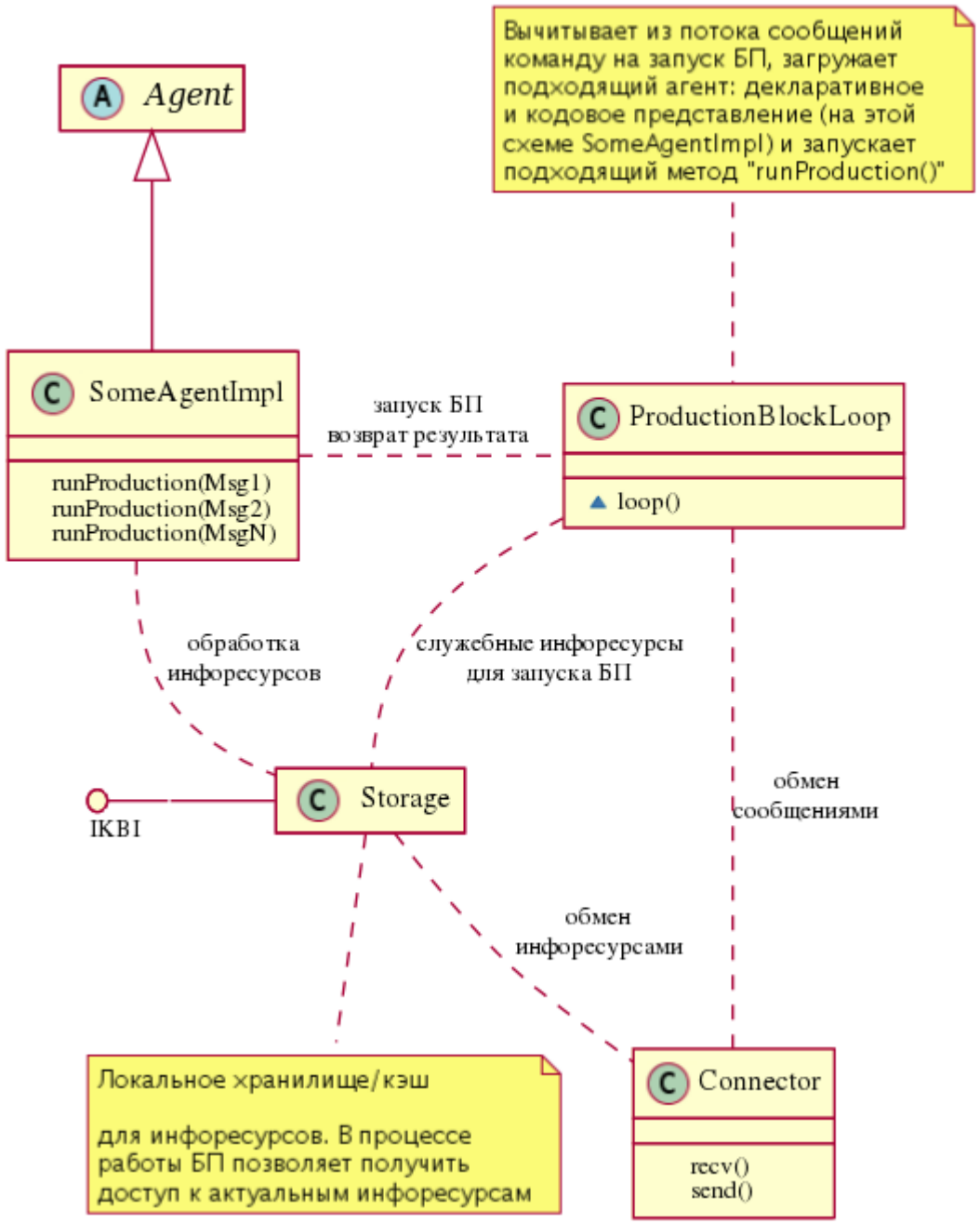

Рис. 59. Структура РП, нижний уровень.

Структура ВВП такая же, как и у РП, разница только в конфигурировании: при запуске ВВП на нём запускается соответствующий агент ввода-вывода.

Цикл работы агента ввода-вывода следующий:

1. Запускается процесс на одном из компьютеров кластера (таким образом, чтобы было удобнее взаимодействовать с внешней системой);

2. Этот компьютер сконфигурирован так, что после полной инициализации процесса создаёт соответствующий ВВ-агент;

3. ВВ-агент настраивается на взаимодействие с внешней системой;

4. Процесс запускает агента как экземпляр агента (следовательно, привязанный к этому процессу);

5. ВВ-агент может работать реактивно: как агент, принимая и обрабатывая сообщения от ПРЗ;

6. ВВ-агент может работать проактивно: принимая сообщения от внешней системы, и переправляя их в ПРЗ, либо порождая события самостоятельно.

3.4. Методы реализации процессора пользовательских интерфейсов

3.4.1. Постановка задачи и требования к процессору пользовательских интерфейсов

Основное назначение ППИ — предоставить средства для организации взаимодействия пользователей и сервисов Платформы.

В разделе 2.4.1 предложены требования к модели. Реализация должна удовлетворять этим требованиям.

Кроме того, к реализации выдвигаются также следующие требования:

1. Должно быть предоставлено API для создания абстрактного интерфейса. Модель, описанная в главе 2, предполагает, что абстрактный интерфейс представляется инфоресурсом; однако создание такого инфоресурса средствами IKBI может быть неудобным, т.к. может потребовать указания массы малозначительных деталей. Использование специализированного интерфейса инкапсулирует эти детали, что повышает сопровождаемость кода.

2. ППИ должен поддерживать взаимодействие с браузером пользователя по технологии AJAX [87], что позволяет значительно ускорить взаимодействие пользователя с приложением: для обновления данных на странице требуется загрузить только новую часть, а не всю страницу.

3.4.2. Проект

3.4.2.1. Интерфейсный Агент

Как показано в разделе 3.3.2.5, для взаимодействия с внешними системами используются агенты ввода-вывода. Поэтому для взаимодействия с веб-сервером (как внешней системой) вводится Интерфейсный Агент, который является посредником во взаимодействии веб-сервера и сервиса. Этот агент преобразует запросы от веб-сервера в сообщения агентной системы, а также преобразует ответ от сервиса в HTML для передачи веб-серверу (который передаст его пользователю в браузер). Этот агент является универсальным для всей платформы и повторно используется в каждом сервисе, в котором необходимо взаимодействие с пользователем.

Однако, Интерфейсный Агент осуществляет только трансляцию сообщений. Сценарий интерфейса (логику) определяет другой агент — Интерфейсный Контроллер сервиса. Этот агент уникален для каждого сервиса (хотя может повторно использоваться если у сервисов одинаковые интерфейсы).

Как уже упоминалось в разделе 2.4, в проекте IACPaaS используется CMS «MediaWiki» и поэтому взаимодействие с пользователем осуществляется через эту CMS: пользователь запрашивает нужную страничку сайта, запрос отправляется веб-серверу, веб-сервер передаёт запрос «MediaWiki», а «MediaWiki» передаёт запрос сервису.

CMS «MediaWiki» предлагает следующую абстракцию информационного устройства:

1. Вся информация представлена страницами, которые связаны ссылками,

2. Вводится упрощенный, но расширяемый синтаксис для написания страниц,

3. Предоставляются средства упорядочивания и поиска нужных страниц.

Предлагаемый механизм интеграции с сервисами использует эти особенности: в CMS «MediaWiki» вводится специальное расширение — тег {{ui}}. CMS «MediaWiki», встретив этот тег на какой-либо странице, формирует запрос Интерфейсному Агенту, ожидает от него ответ и размещает его в этом месте.

У этого расширения могут быть параметры, в частности:

1. Идентификатор сервиса, к которому относится это окно,

2. Идентификатор раздела ВИР, которое визуализирует это окно,

3. Размеры окна,

4. Прочие атрибуты .

Пример тега:

{{ui|service="Медицинская диагностика"|section="Выбор пользователя для диагностики"|width=800|height=600}}.

Этот тег встраивает в wiki-страницу окно с размерами 800x600 пикселей, и передает Интерфейсному Агенту параметры service (равный "Медицинская диагностика") и section ("Выбор пользователя для диагностики").

Такой подход позволяет комбинировать обычную информацию из Wiki и информацию из агентной системы, как показано на Рис. 60: среди общего текста есть два тега {{ui}}, на основе которых MediaWiki формирует «окно» в агентную среду.

Для взаимодействия сервиса используется технология AJAX: если пользователь взаимодействует с каким-либо интерфейсным элементом в окне (нажимает кнопку, щёлкает по ссылке и т.п.), запрос отправляется агентной системе и та изменяет содержимое этого окна или его части.

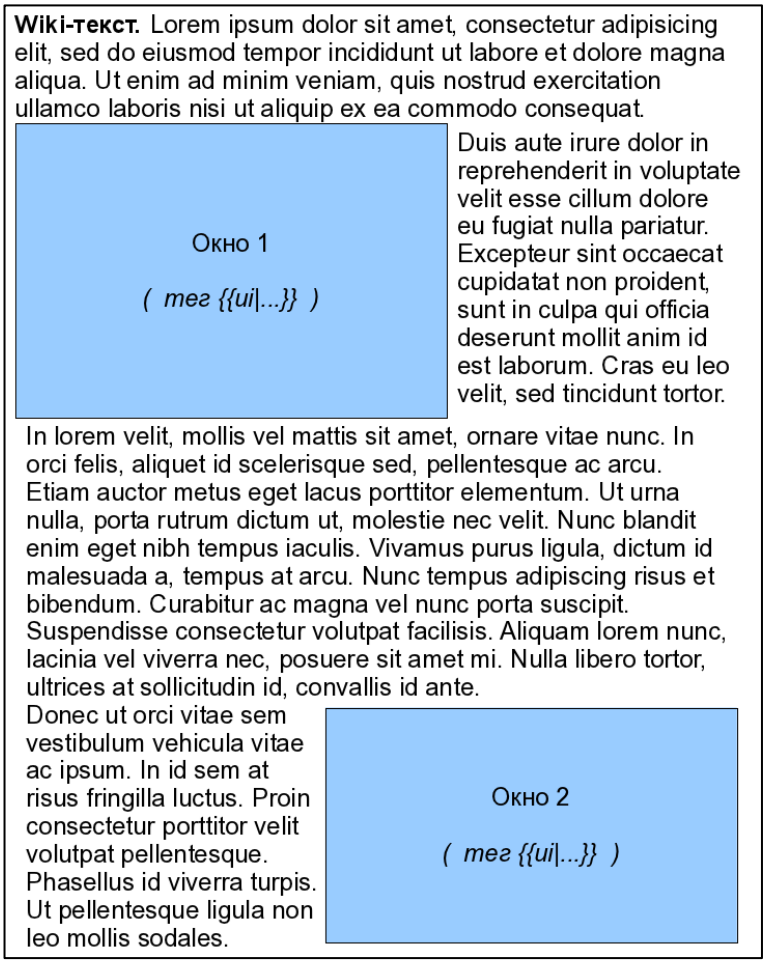

Рис. 60. Пример расположения wiki-текста и окон интерфейса.

Обработка тега {{ui}} реализована в виде расширения (plugin) для CMS «MediaWiki». При начальной загрузки всей wiki-страницы или каких-то действиях пользователя в отрисованном окне это расширение посылает запрос Интерфейсному Агенту в виде набора «параметр», «значение параметра». Набор параметров и их значения задаёт разработчик сервиса при программировании интерфейса. Интерфейсный Агент преобразует этот набор в сообщение для Интерфейсного Контроллера и передает ему (агент-контроллер может запустить дальнейшую обработку сервисом или обработать запрос самостоятельно). В результате чего формируется новый абстрактный интерфейс, контроллер передаёт его Интерфейсному Агенту, который преобразует его в HTML-текст, который затем передаётся расширению MediaWiki; переданный HTML-текст вставляется в нужное место страницы.

Для того, чтобы организовать взаимодействие с JavaScript- или Flash-приложениями, Интерфейсный агент также предоставляет следующую функциональность:

1. Запуск Flash-скриптов;

2. Передача инфоресурса в виде JSON-представления. Приложение с веб-странички может получать весь инфоресурс целиком, например, для обновления интерфейса;

3. Загрузка и выгрузка бинарных данных в и из скриптов.

3.4.2.2. Интерфейсный Контроллер

Каждый сервис, которому требуется взаимодействие с пользователем, должен предоставлять агента-контроллера, с которым взаимодействует интерфейсный агент. Контроллер должен обрабатывать сообщение «Запрос от интерфейсного агента», и посылать обратно сообщение «Результат обработки».

Программист сервиса определяет логику обработки «Запроса от интерфейсного агента» — т.е. как будет обрабатываться те или иные сообщения.

3.4.2.3. Программный интерфейс для создания абстрактных интерфейсов

Абстрактный интерфейс есть инфоресурс, порождённый по метаинформации из раздела 2.4.2.1. Разработчик сервиса может использовать средства IKBI (см. раздел 3.2.2.2) для порождения интерфейса.

Однако платформа IACPaaS предоставляет высокоуровневые функции для создания абстрактных интерфейсов. Они расположены в модуле UiBuildHelper; для каждого типа интерфейсного элемента введён функции создания соответствующего фрагмента абстрактного интерфейса, такие как button(), checkbox(), form(), text() и т.д. Эти функции упрощают создание фрагментов интерфейса, за одним вызовом скрывая построение соответствующего поддерева инфоресурса и заполнение его свойств. Тем самым упро-

щается создания абстрактного интерфейса, код создания интерфейса становится более понятным и, следовательно, более сопровождаемым.

3.5. Выводы по главе

В настоящей главе:

1. Разработана концептуальная архитектура платформы IACPaaS, которая состоит из четырёх уровней: системного, библиотечного, сервисного уровней и административной системы.

2. Предложен способ поддержки работы сервисов в контексте трёх процессоров: процессора пользовательских интерфейсов (обеспечивающего взаимодействие сервиса с пользователем), процессора информационных ресурсов (обеспечивающего взаимодействие сервиса с инфоресурсами), процессора решателей задач (обеспечивающего обмен сообщениями и саму работу агентов, составляющих сервис).

3. Выдвинуты требования для процессора инфоресурсов, в частности, требования о соответствии модели, разработанной во второй главе, требования к надёжности, требования к программному интерфейсу.

4. Разработан проект последовательной версии и распределенной версии процессоров инфоресурсов. Предложен способ реализации модели решателя из главы 2 в данном проекте. Разработан проект модуля представления (предоставляющий программный интерфейс для прикладных программистов), который поддерживает логическую семантику метаинформации/информации, предложенную в главе 2.

5. Выдвинуты требования для процессора решателей задач. В частности, выдвинуты требования о единице распараллеливания: ей должен быть блок продукций отдельного агента; приведены требования к планировщику.

6. Разработан проект процессора решателя задач как многоуровеневой структуры, где нижние уровни являются основой для верхних. Выделено

пять уровней: общесистемный уровень, сервисный уровень, агентный уровень, блокопродукционный уровень, уровень процессов. Последний уровень опирается на процессор инфоресурсов для осуществления синхронизации между отдельными процессами. Разработана структура уровней и их сущностей и поведение соответствующих сущностей.

7. Выдвинуты требования для процессора пользовательских интерфейсов, в частности, требование интеграции с CMS «MediaWiki» и удобства программирования интерфейса.

8. Разработан проект процессора пользовательских интерфейсов. Введено понятие интерфейсного агента, как преобразователя абстрактного интерфейса в конкретный, интерфейсного контролёра, как диспетчера сообщений. Предложен проект программного интерфейса.

4. ТЕХНОЛОГИЯ РАЗРАБОТКИ СИСТЕМНЫХ И ПРИКЛАДНЫХ СЕРВИСОВ С ПОМОЩЬЮ ПЛАТФОРМЫ IACPAAS И ЕЁ ИНСТРУМЕНТАЛЬНАЯ ПОДДЕРЖКА

В главе решается задача разработки технологии создания облачных ИС с использованием облачной платформы.

4.1. Технология разработки сервисов

Разработка сервиса состоит из шести традиционных для разработки программных средств [39] этапов:

1. Разработка требований к сервису;
2. Проектирование сервиса, с учётом того, что сервис есть набор взаимодействующих агентов и информационных ресурсов;
3. Специфицирование;
4. Разработка недостающих компонентов (инфоресурсов и агентов сервиса) или модификация существующих для повторного использования;
5. Тестирование и отладка;
6. Ввод в эксплуатацию.

4.1.1. Разработка требований к сервису

На этом этапе разработчик и заказчик сервиса должны описать функциональность сервиса, форматы входных и выходных данных (с учётом того, что сервис обрабатывает только инфоресурсы).

Если предполагается, что сервис будет взаимодействовать с внешними системами, разработчик и заказчик должны сформулировать требования к такому взаимодействию.

В частности, если сервис должен обладать интерфейсом пользователя, то разработчик и заказчик должны сформулировать требования к пользовательскому интерфейсу.

Платформа IACPaaS не предоставляет инструментальных средств для поддержки разработки требований, поэтому выбор технологии создания требований остаётся на усмотрение разработчика.

4.1.2. Проектирование сервиса

На этом этапе разработчик сервиса проектирует устройство и поведение сервиса.

Платформа IACPaaS не предоставляет инструментальных средств для поддержки проектирования, поэтому разработчик может использовать инструментарий и технологии проектирования на свой выбор.

Однако, разработчик сервиса должен учитывать, что платформа навязывает собственный способ решения задач через декомпозицию — агентный подход, в соответствии с которым:

1. В Фонде уже присутствует некоторый набор агентов и инфоресурсов, рекомендуемых для повторного использования;

2. Естественный (для Платформы IACPaaS) способ декомпозиции — это декомпозиция на отдельные агенты и инфоресурсы, отдельные компоненты системы рекомендуется оформлять как и инфоресурсы и агенты;

3. Разработчик должен стремиться к тому, чтобы отдельные компоненты (агенты и инфоресурсы) сервиса были повторно-используемыми.

Второе требование платформы IACPaaS состоит в том, что обрабатываемая информация должна быть представлена инфоресурсами (с метаинформацией как структурой). Поэтому разработчик сервиса должен спроектировать необходимые метаинформации.

4.1.3. Специфицирование сервиса

На этом этапе необходимо создать инфоресурс-описание сервиса. Это описание ссылается на используемые сервисом метаинформации для входных, выходных и собственных инфоресурсов, поэтому предварительно необходимо подготовить эти метаинформации.

Создание инфоресурса-описания сервиса сводится к порождению инфоресурса по метаинформации «Структура сервиса», в процессе порождения надо указать в порождаемом инфоресурсе: название сервиса, описание сервиса, ссылки на метаинформации входных, выходных и собственных инфоресурсов.

Останутся непорождёнными ссылки на корневой агент и интерфейсный контроллер, которые заполняются позже.

4.1.4. Разработка метаинформаций и инфоресурсов сервиса

Для создания метаинформации предназначен системный сервис «Редактор» (см. раздел 4.3.2). С его помощью разработчик сервиса создаёт необходимые сервису метаинформации.

Как уже было отмечено выше, сервису необходима по крайней мере одна метаинформация для входного или выходного инфоресурсов. Разработчик сервиса может создать их самостоятельно, либо повторно использовать метаинформации, существующие в Фонде.

Редактор также может использоваться для создания информаций по метаинформации.

4.1.5. Разработка модулей сервиса

В рамках Платформы IACPaaS разработка компонентов сервиса есть разработка агентов, она рассмотрена в разделе 4.2.

Среди всех разрабатываемых для сервисов агентов должен быть корневой агент. Именно с него начинается работа сервиса (для этого Платформа первым делом передаёт сообщение «Инициализировать сервис» этому агенту). Допускается повторное использование уже существующего корневого агента.

Разработчик сервиса должен сослаться из инфоресурса сервиса на эти агенты. После этого инфоресурс сервиса полон и сервис готов к тестированию и отладке.

4.1.5.1. Разработка интерфейса

Если сервис предполагает взаимодействие с пользователем, требуется разработать (или повторно использовать) агент-интерфейсный контроллер и набор страничек на wiki, с помощью которых осуществляется взаимодействие с пользователем.

Сервис взаимодействует с пользователем с помощью интерпретации тега {{ui}}, размещённого на wiki-страницах. Поэтому разработчик должен создать соответствующие страницы в проекте. Каждая такая страница может включать справку по отображаемому окну, ссылки на другие странички сервиса и т.п.

Каждый раз, когда происходят какие-либо операции со страницей: загрузка этой страницы, что требует отрисовки тега {{ui}}, взаимодействие пользователя с интерфейсными элементами, запускается интерфейсный агент, который преобразует запросы от сервиса в сообщение агентной системе и передаёт его агенту-интерфейсному контроллеру. Программист сервиса, таким образом, должен предоставить подходящую реализацию агента-интерфейсного контроллера. Этот агент должен обрабатывать сообщения с шаблоном «Запрос от интерфейсного агента», в котором содержатся параметры запроса от сервиса.

Логику обработки этого сообщения программист определяет самостоятельно: простые запросы (например, перерисовка окна) контроллер может обработать самостоятельно, а обработку сложных может поручить другим агентам.

После обработки агент-контролёр должен ответить интерфейсному агенту сообщением с шаблоном «Отобразить окно», в котором описано новое состояние интерфейсного окна. Кроме того, используется шаблон «Вернуть инфоресурс в окно», чтобы передать инфоресурс из Фонда в окно в виде JSON-текста, и шаблон «Вернуть строку в окно», когда полная перерисовка интерфейса не требуется.

Интерфейс есть инфоресурс, порождённый по метаинформации «Структура интерфейса». Методы порождения инфоресурсов, предлагаемые IKBI, слишком низкоуровневые, поэтому программисту предлагается программный интерфейс (в модуле UiBuildHelper) для более удобного создания интерфейсного инфоресурса.

4.1.6. Тестирование и отладка

Платформа IACPaaS предоставляет возможность ведения журналов работы сервиса, они используются для поиска ошибок в работе сервисов. Для этого разработчику предоставляется системный сервис Просмотрщик Журнала Работы Сервиса.

Для модульного тестирования отдельных агентов используется Прогонщик тестов агента, его использование рассмотрено в 4.3.4.

4.1.7. Ввод в эксплуатацию

После завершения тестирования и отладки разработчик сервиса должен сообщить локальному администратору о готовности сервиса для публичного использования. После этого администратор может предоставлять полномочия на использование сервиса пользователям.

Для того, чтобы предоставить такое полномочие, администратор выбирает входные, выходные и собственные инфоресурсы, порождённые из соответствующих метаинформаций сервиса.

После этого пользователь может запускать сервис с этими инфоресурсами.

Заметим, что в процессе работы сервиса всё так же осуществляется журналирование его работы; журналы могут использоваться разработчиком для отладки сервиса.

4.2. Технология разработки агентов

Разработка агента состоит их следующих традиционных этапов:

1. Разработка требований к агенту;
2. Проектирование агента;
3. Специфицирование агента;
4. Кодирование агента;
5. Тестирование и отладка;
6. Ввод в эксплуатацию.

4.2.1. Разработка требований к агенту

На этом этапе разработчик агента должен согласовать с разработчиком сервиса функциональность агента, структуру входных и выходных данных (учитывая, что взаимодействие между агентами осуществляется посредством сообщений, здесь требуется разработать или использовать уже существующие шаблоны подходящих входных и выходных сообщений).

Если предполагается разработка агента ввода-вывода, разработчик агента и сервиса должны сформулировать требования к такому взаимодействию.

Платформа IACPaaS не предоставляет инструментальных средств для поддержки разработки требований к агентам, поэтому выбор технологии создания требований остаётся на усмотрение разработчика.

4.2.2. Проектирование агента

На этом этапе разработчик сервиса проектирует устройство и поведение агента (как совокупности блоков продукций, обрабатывающих сообщения).

Платформа IACPaaS не предоставляет инструментальных средств для поддержки проектирования, поэтому разработчик может использовать инструментарий и технологии проектирования на свой выбор.

4.2.3. Специфицирование агента

После того, как проект агента готов и стало ясно, какие сообщения он будет обрабатывать, надо специфицировать его, чтобы Платформа могла правильно использовать этот агент.

Спецификация агента есть инфоресурс с метаинформацией «Структура агента», поэтому для описания агента надо породить эту метаинформацию. В инфоресурсе-описании указывается:

1. Название агента в Фонде;

2. Внутреннее имя агента (название класса в байткоде);

3. Описание агента;

4. Перечисление блоков продукций с сообщениями, которые эти блоки продукций обрабатывают;

5. Реализация блоков продукций (байткод агента).

Байткод агента задаётся после его кодирования.

Спецификация сообщения есть инфоресурс с метаинформацией «Структура сообщения». В этом инфоресурсе указывается: название сообщения, структура передаваемых данных.

4.2.4. Кодирование агента

Байткод агента должен удовлетворять следующим правилам:

1. Весь агент должен размещаться в Java-классе с названием <внутреннее имя агента>Impl,

2. Этот класс должен наследовать от класса Agent,

3. Для каждого блока продукций класс должен содержать метод runProduction() с двумя аргументами: классом-обёрткой обрабатываемого сообщения, классом-обёрткой результата обработки сообщения.

Байткод сообщений, обрабатываемых агентом, должен удовлетворять следующим правилам:

1. Весь агент должен размещаться в Java-классе с названием <внутреннее имя сообщения>Impl,

2. Этот класс должен наследовать от класса Message,

3. Класс может содержать дополнительные методы для доступа к нужным элементам инфоресурса сообщения.

Чтобы упростить кодирование агента, Платформа предоставляет системный сервис «Генератор Агента», который формирует шаблон агента по его спецификации (инфоресурсу-описанию агента). Разработчику остаётся только наполнить этот шаблон содержимым: закодировать блоки продукций агента.

Шаблон представляется классом с именем <внутреннее имя агента> (в файле <внутреннее имя агента>.java), в котором описаны все методы, соответствующие блокам продукций данного агента. Разработчик должен создать класс-наследник с именем <внутреннее имя агента>Impl, в котором должен переопределить (override) все методы, соответствующие блокам продукций. Такая схема необходима для того, чтобы пользователь мог выбирать произвольный язык программирования реализации агента, не ограничиваясь только языком программирования Java.

Платформа IACPaaS не предоставляет никаких инструментальных средств для поддержки кодирования, оставляя выбор за разработчиком.

Разработчик может создавать более удобный (по сравнению с IKBI) способ доступа к данным сообщения, для этого ему надо разработать программную обёртку для сообщения. Помимо шаблона агента, Генератор Агентов также формирует пустые шаблоны сообщений, которые могут быть заполнены разработчиком.

После того, как агент закодирован, разработчик может поместить его в Фонд, используя для этого системный сервис «Загрузчик Агентов». Разработчик компилирует код реализации агента и сообщений, объединяет скомпилированный код в JAR-архив (ссылка на определение), после чего запускает сервис «Загрузчик Агентов», указывает инфоресурс агента (его спецификацию) и передаёт этот JAR-архив загрузчику. Загрузчик Агентов загружает class-файлы из JAR-архива в соответствующий байткод агента и сообщений.

После этого агент готов к использованию.

4.2.5. Тестирование и отладка

Платформа предоставляет системный сервис «Прогонщик тестов агента» (см. раздел 4.3.4), при помощи которого разработчик может создавать модульные тесты для проверки агента. Каждый такой тест является парой «сообщение для агента», «ожидаемое сообщение». Такие наборы могут быть подготовлены заранее (например, тестером проекта) и использоваться для тестирования только что закодированного агента. Кроме того, эти наборы могут использоваться для регрессионного тестирования в процессе совершенствования агента.

Кроме того, агент может записывать сообщения в журнал работы; после тестовых прогонов этот журнал доступен разработчику, он может использовать его для поиска неисправностей.

4.2.6. Ввод в эксплуатацию

Сразу после загрузки байткода агент готов к эксплуатации, поэтому специальных действий по вводу агента в эксплуатацию не требуется.

4.3. Системные сервисы

В этом разделе рассмотрены системные сервисы, реализованные в настоящее время в рамках Проекта IACPaaS.

Прежде всего, это Информационно-Административная Система, которая обеспечивает контролируемый доступ пользователей к сервисам.

Далее рассмотрен Редактор Инфоресурсов (и его сужение — Визуализатор Инфоресурсов) — универсальный сервис, используемый повсеместно во всём Проекте для редактирования и просмотра инфоресурсов.

Для разработки новых сервисов используются Прогонщик тестов, Генератор кода агентов и Загрузчик байткода агентов.

Архитектура сервисов представлена с помощью обозначений, приведённых на Рис. 61.

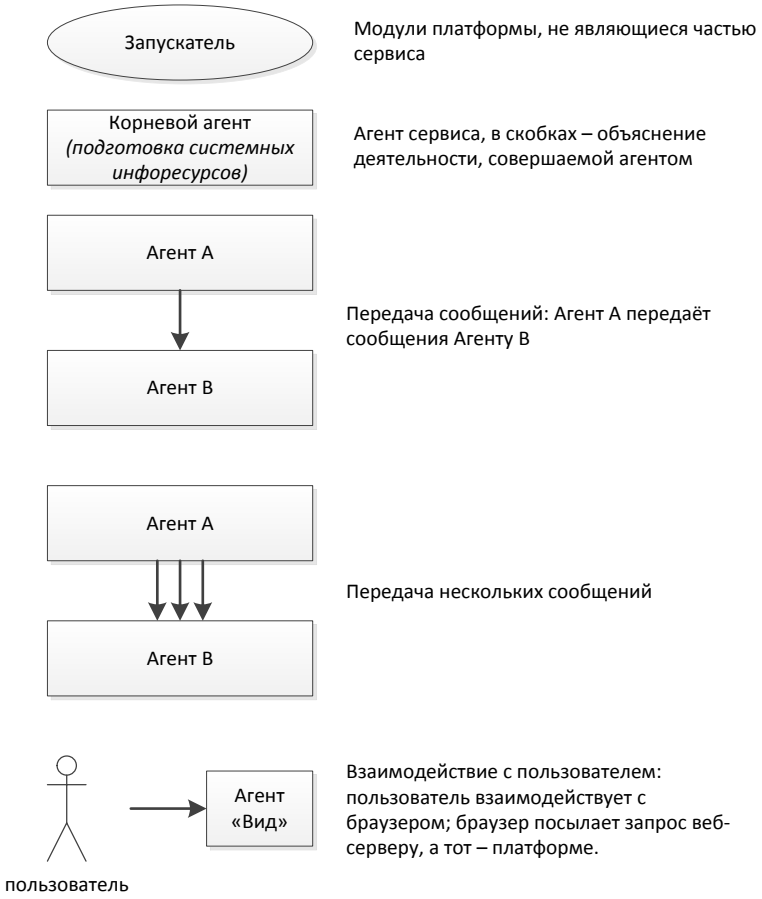

Рис. 61. Условные обозначения, используемые для представления архитектуры сервисов

4.3.1. Административная система

Назначение: Административная Система (далее просто АС) контролирует доступ пользователей к сервисам Фонда. Она позволяет пользователям получить базовый (гостевой) доступ к Фонду или идентифицироваться и получить расширенный доступ (в зависимости от прав, выданных пользователю администратором). Кроме того, АС предоставляет информацию по разным аспектам системы.

Общая функциональность: Функции АС можно разбить на три группы:

1. Контролируемый запуск сервисов.

2. Управление запросами пользователей, (в рамках проекта АС называемый «заявкооборот»).

3. Предоставление информации по разным аспектам системы.

Рассмотрим первую группу функций. «Полномочием» будем называть четвёрку «пользователь, сервис, входные инфоресурсы, выходные инфоресурсы». Запуск полномочия есть запуск данного сервиса с указанными входными и выходными инфоресурсами от имени указанного пользователя.

Пользователь может подавать заявку (см. ниже) на запуск полномочия. Если такая заявка одобрена, пользователь может запускать это полномочие.

Вторая группа функций — это заявкооборот — процесс обработки и использования заявок для контроля действий пользователя. Заявка есть запрос пользователя, который может быть одобрен (и тогда пользователю предоставляются соответствующие права), отменён (и тогда пользователь теряет соответствующие права), отклонён (например, из-за необоснованности заявки). Выделяются следующие типы заявок:

1. Заявка на регистрацию — запрос пользователя-гостя о предоставлении расширенных прав. Для предоставления расширенных прав пользователь обязан зарегистрироваться (указать идентифицирующую информацию о себе) и обосновать запрос на расширение прав.

2. Заявка на получение полномочия (см. ниже). Заявка на модификацию Фонда — запрос пользователя на редактирования инфоресурсов.

Гости и зарегистрировавшиеся пользователи имеют возможность подачи заявок. Пользователи со специальными правами — администраторы — имеют возможность просматривать заявки, одобрять, отменять или отклонять их.

Если заявка пользователя одобрена, он может выполнять соответствующее действие: редактировать Фонд, запускать полномочия и пр.

Третья группа функций — информационная. АС предоставляет информацию об инфоресурсах Фонда, о пользователях Фонда, о полномочиях пользователей. Кроме того, предоставляется функционал для обмена сооб-

щениями (внутренняя почта), публикации новостей (как по всему Фонду, так и по отдельным разделам), редактирования и чтения справочных материалов («Частые вопросы и ответы»).

Общая архитектура. АС состоит из 14-и агентов (Рис. 62):

1. Корневого агента, который подготавливает системные инфоресурсы для работы АС. Кроме того, он используется для сохранения диагностической информации в случае сбоев АС (для последующей отладки);

2. Повторно-используемого интерфейсного агента, с помощью которого осуществляется взаимодействие с пользователем через браузер и веб-сервер;

3. Интерфейсного контроллера, который выступает в этом сервисе как диспетчер, просто распределяя работу между остальными агентами;

4. Агенты-обработчики запросов, которые реализуют логику всех групп функций АС: информационную, заявкооборот, запуск полномочий;

5. Повторно-используемый Системный Агент, который используется для запуска полномочий.

Запускатель формирует Инициализирующее сообщение, получив которое, Корневой агент подготавливает системные инфоресурсы. После этого сервис ожидает новых действий пользователя.

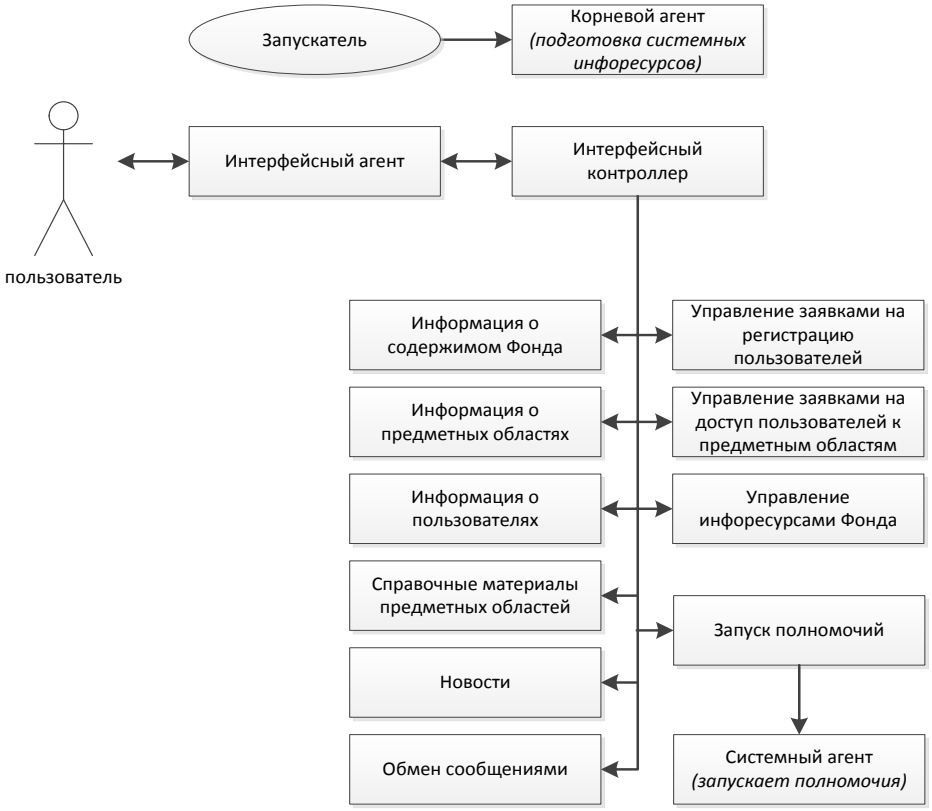

Рис. 62. Архитектура АС

Средства платформы, используемые при реализации. АС есть сервис, поэтому она использует базовые средства Платформы: запуск агентных сервисов. Как сервис с пользовательским интерфейсом, АС использует Интерфейсный агент для организации пользовательского веб-интерфейса. Доступ пользователей к разным аспектам АС реализован через страницы Wiki (такие как страница полномочий пользователя, страница просмотра Фонда, страница управления заявками и т.д.). Для запуска других сервисов АС использует Системный Агент: АС формирует контекст запуска сервиса (входные, выходные инфоресурсы, идентификатор пользователя) и передаёт его в сообщении этому агенту. АС — первый сервис, запускаемый при запуске Платформы, для чего также требуются особая поддержка в виде Запускателя сервисов (отличного от Системного Агента, который запускает сервисы по сообщению).

Разработчики. Концепция АС (заявкооборот, запуск полномочий, взаимодействие с пользователем с помощью Wiki) разработана автором диссертации. Проектирование, кодирование и сопровождение осуществлялось Ф.М. Москаленко и В.А. Тимченко под руководством А.С. Клещёва и В.В. Грибовой [31, 32, 65].

4.3.2. Редактор

Назначение: Редактор предназначен для просмотра, изменения инфоресурсов и метаинформаций, а также для порождения инфоресурсов по метаинформациям.

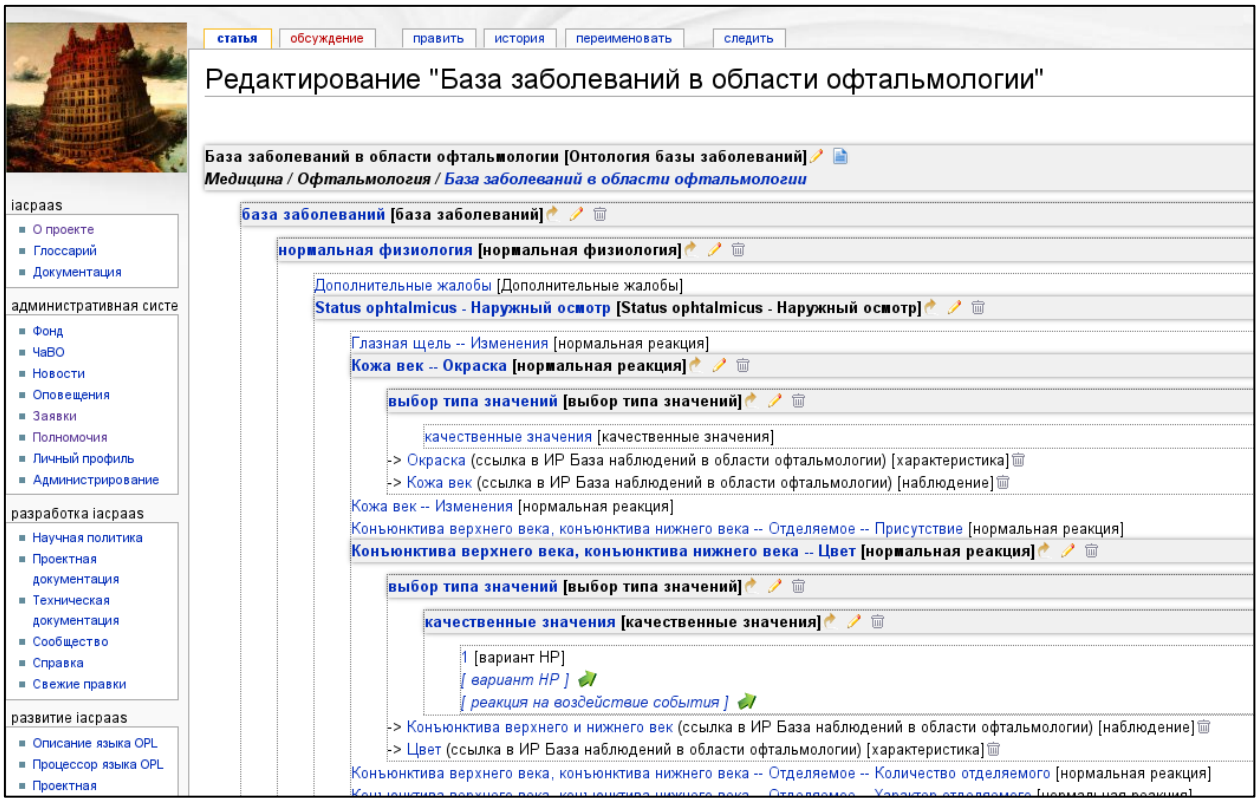

рис. 63. Пример окна Редактора

Общая функциональность: Редактор предлагает веб-интерфейс, в котором в отображён инфоресурс (рис. 63). Инфоресурс отображается как дерево, начиная от корневой вершины. Пользователь может:

1. Просматривать инфоресурс, используя следующие действия (с помощью соответствующих интерфейсных элементов):

1.1. «Развернуть» часть инфоресурса, чтобы увидеть содержимое этой части, «свернуть» ненужные части;

1.2. Перейти к некоторой части инфоресурса: остальные части инфоресурса скрываются, корнем отображаемого дерева становится выбранный фрагмент; либо вернуться к предыдущему фрагменту.

2. Редактировать инфоресурс, используя следующие действия:

2.1. Изменить атрибуты некоторого видимого понятия (возможно, сначала, «раскрыв» фрагмент дерева, содержащий понятие);

2.2. Изменить атрибуты некоторого отношения;

2.3. Удалить отношение или понятие;

2.4. Породить новое отношение (ссылающееся на существующее понятие или на новое понятие — в последнем случае пользователю будет предоставлен диалог для указания свойств понятия).

Такой же функционал используются и для редактирования метаинформаций.

Сопровождающему персоналу Платформы предоставляется возможность аварийного запуска Редактора для устранения проблем в Фонде (например, для исправления сбоев при модификации системных инфоресурсов).

Общая архитектура. Редактор состоит из четырёх агентов (см. Рис. 64):

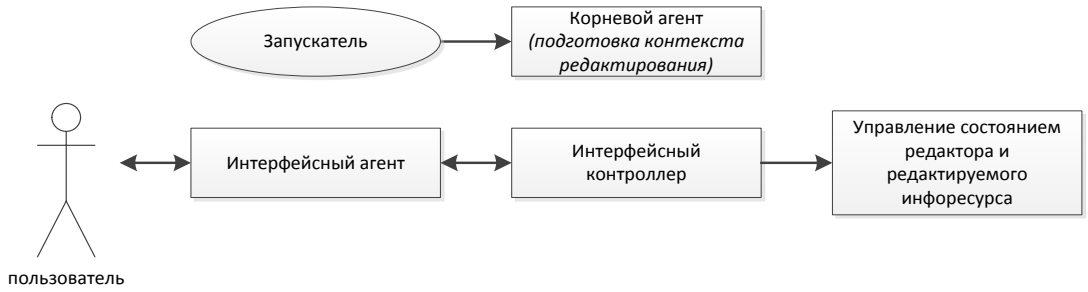

Рис. 64. Архитектура Редактора

1. Корневого, который подготавливает контекст для редактирования. «Контекст редактирования» есть инфоресурс, в котором указан редактируемый инфоресурс, состояние дерева отображения (открытые и скрытые части дерева), история редактирования, состояние процесса редактирования элементов инфоресурса;

2. Повторно-используемого Интерфейсного Агента;

3. Интерфейсного Контроллера, который просто передаёт сообщения от интерфейсного агента к следующему агенту;

4. Агент редактирования, который обновляет контекст редактирования, а также редактируемый инфоресурс по действиям пользователя и отрисовывает соответствующий этому дереву и инфоресурсу абстрактный интерфейс.

Запускатель формирует Инициализирующее сообщение, получив которое, Корневой агент подготавливает контекст редактирования. После этого сервис ожидает действий пользователя.

Средства платформы, используемые при реализации. Так как Редактор сделан как сервис из агентов, то он использует базовые средства платформы: запуск агентных сервисов. Как сервис с пользовательским интерфейсом, Редактор использует Интерфейсный Агент для организации пользовательского веб-интерфейса. Для запуска в аварийном режиме используется механизм начального запуска сервиса (тот же, что используется для запуска административной системы).

Разработчики. Первая версия сервиса разработана автором диссертации; отладка и дальнейшее сопровождение осуществляется Ф.М. Москаленко и В.А. Тимченко.

4.3.3. Визуализатор инфоресурсов

Визуализатор инфоресурса есть Редактор, запущенный в режиме просмотра: в этом режиме пользователю не показываются элементы управления

для редактирования (изменения, порождения, удаления) элементов инфоресурса.

4.3.4. Прогонщик тестов агента

Назначение: Прогонщик тестов агента предназначен для тестирования агентов Фонда, которое заключается в запуске тестируемого агента с заранее заданными сообщениями (и инфоресурсами, которые требуются для обработки этого сообщения агентом) и сравнении полученных сообщений с ожидаемыми. Каждая такая пара — 1) входное сообщение и инфоресурсы в начальном состоянии и 2) ожидаемые выходные сообщения и инфоресурсы в конечном состоянии — называется «тестовой ситуацией». Тестовые ситуации объединяются в «наборы тестов». Наборы тестов подготавливает разработчик агента или тестировщик. На основе запуска всех наборов тестов прогонщик формирует отчёт о прогоне, который используется для отладки агента или в регрессионном тестировании.

Общая функциональность: Сервис прогонщика запускается с указанием некоторого набора тестов и тестируемого агента. Прогонщик последовательно выбирает тестовые ситуации из набора и запускает агент, передавая ему тестовое входное сообщение.

Запуск агента в тестовом режиме имеет две особенности: он должен обращаться только к тестовым инфоресурсам, выходные сообщения должны попасть к Прогонщику (и не передаваться агентам-получателям). Эти две особенности реализует Системный Агент: Прогонщик посылает ему сообщение с шаблоном «Запустить Агент» и Системный Агент запускает указанный в сообщении агент в тестовом режиме. В тестовом режиме агент, обращаясь к инфоресурсам, получает доступ только к тестовым инфоресурсам. Кроме того, все выходные сообщения перехватываются Системным Агентом и возвращаются Прогонщику.

Получив ответ от Системного Агента, Прогонщик сравнивает полученный результат с ожидаемым и формирует отчёт о прогоне тестовой ситуации.

После того, как все тестовые ситуации прогнаны, Прогонщик формирует полный отчёт (в виде соответствующего инфоресурса) о прогоне всего тестового набора. После этого пользователь Прогонщика может посмотреть отчёт при помощи Визуализатора инфоресурсов.

Общая архитектура. Прогонщик тестов состоит из шести агентов (Рис. 65):

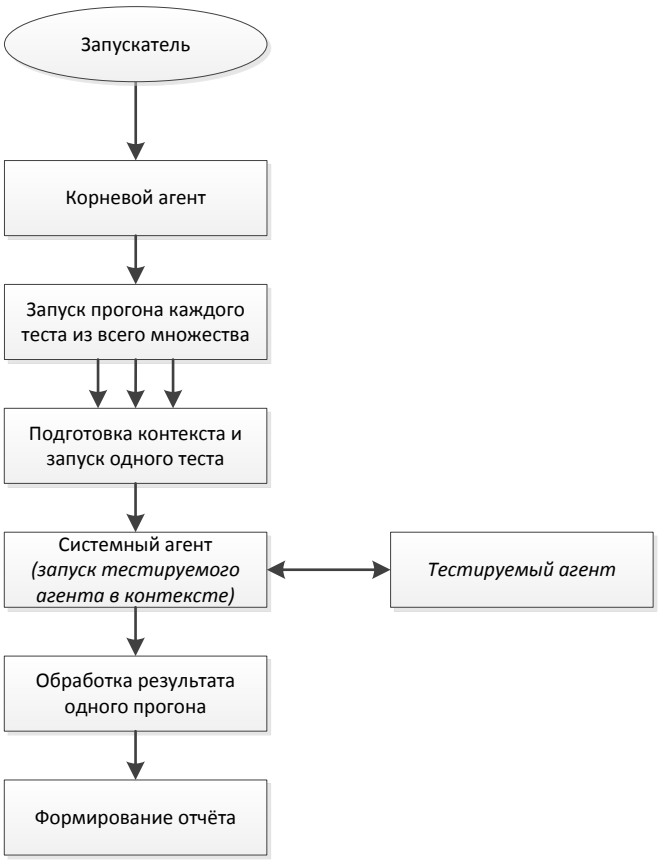

Рис. 65. Архитектура Прогонщика тестов

1. Корневого, который необходим из-за требований к сервисам; он сразу передаёт сообщение следующему агенту;

2. Агента «Запуск прогона каждого теста». Для каждого теста из входного инфоресурса «Множество тестов» этот агент посылает сообщение следу-

ющему агенту. Обработка каждого такого сообщения независима, поэтому может осуществляться параллельно;

3. Агента «Подготовка контекста и запуск одного теста». Подготавливает подходящие сообщения для Системного агента;

4. Повторно используемого агента «Системный агент», который запускает тестируемый агент в указанном контексте;

5. Агента «Обработка результата одного прогона», который получает сообщения от Системного агента о результатах запуска тестируемого агента, сравнивает с ожидаемым результатом и передаёт результат сравнения следующему агенту;

6. Агента «Формирование отчёта», который формирует итоговый инфоресурс с отчётом о прогоне тестов.

Запускатель формирует Инициализирующее сообщение, получив которое, корневой агент запускает прогон тестов.

Средства платформы, используемые при реализации. Прогонщик тестов сделан как сервис из агентов, поэтому он использует базовые средства платформы: запуск агентных сервисов. Для запуска агентов в тестовом контексте используется Системный Агент.

Разработчики. Прогонщик тестов разработан М.Б. Тютюнником под руководством Е.А. Шалфеевой. Отладка и дальнейшее сопровождение осуществляется Ф.М. Москаленко и В.А. Тимченко под руководством Е.А. Шалфеевой.

4.3.5. Генератор кода агента

Назначение: Как указано в главе 3, код реализации агента должен обладать определённой структурой; генератор агента автоматизирует создание кода с этой структурой, предоставляя разработчику агента «заготовки» JAVA-кода реализации агентов и сообщений. Такие «заготовки» есть готовый код агента, но с пустыми телами методов-обработчиков сообщений. Для

получения полноценного агента программисту остаётся только заполнить эти методы соответствующим кодом.

Общая функциональность: Разработка агента начинается с его специфицирования в виде инфоресурса с соответствующей структурой. После этого разработчик агента запускает генератор кода агента, который проходит по этому инфоресурсу и:

1. Создаёт заголовок класса (фрагмент «public abstract class SomeAgent inherited Agent», где «SomeAgent» — внутреннее имя агента), конструктор класса (фрагмент «public SomeAgent(IRunnigAuthority, IInforesource) { … }»);

2. Для каждого блока продукций создаёт соответствующий метод (фрагмент «public void runProduction(SomeMessage, SomeMessage.ResultCreator)», где «SomeMessage» — внутреннее имя обрабатываемого блоком продукций сообщения).

Для всех обрабатываемых сообщений генератор кода агента также создаёт заготовки кода (вида «public class SomeMessage extends Message { … }»). Разработчик агента может дополнить эти заготовки кода. Это может потребоваться, например, для упрощения доступа к частям инфоресурса.

После этого генератор упаковывает полученные файлы в JAR-архив, который может быть загружен разработчиком агентов для кодирования реализации агента.

Общая архитектура. Генератор состоит из четырёх агентов (Рис. 66):

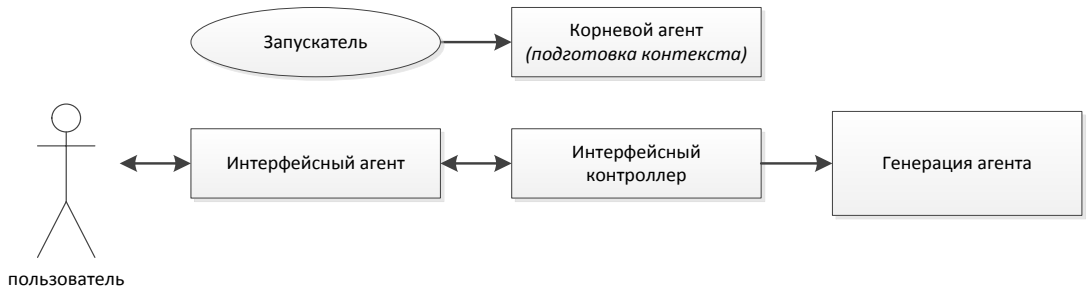

Рис. 66. Архитектура Генератора кода агента

1. Корневого агента. В инициализирующем сообщении для него указан инфоресурс агента, для которого надо сгенерировать код. Корневой агент генерирует этот код;

2. Повторно используемого Интерфейсного Агента, для организации пользовательского интерфейса;

3. Интерфейсного контроллера: он предоставляет веб-интерфейс, с помощью которого можно скачать сгенерированный корневым агентом код;

4. Собственно агента «Генератор агентов», который осуществляет создаёт нужные файлы.

Средства платформы, используемые при реализации. Генератор кода агента сделан как сервис из агентов, поэтому он использует базовые средства платформы: запуск агентных сервисов. Генератор использует Интерфейсный Агент для организации пользовательского веб-интерфейса.

Разработчики. Код для генерации кода агента по инфоресурсу разработан автором диссертации. Набор агентов и собственно сервис разработаны Ф.М. Москаленко и В.А. Тимченко.

4.3.6. Загрузчик байткода агента

Назначение: Загрузчик байткода агента предназначен для размещения скомпилированного кода реализации агентов и сообщений (байткода) в Фонде.

Общая функциональность: Загрузчик предоставляет разработчику агентов веб-интерфейс для загрузки байткода агента. Разработчик компилирует JAVA-код агентов и сообщений, собирает их в JAR-архив и передаёт Загрузчику. Загрузчик распаковывает JAR-архив, находит инфоресурсы агента и сообщений в Фонде и прикрепляет к понятиям «байткод» в этих инфоресурсах скомпилированные class-файлы из JAR-архива. После этого агент становится доступным в Фонде и может обрабатывать сообщения

Общая архитектура. Загрузчик Кода Агентов состоит из пяти агентов (Рис. 67):

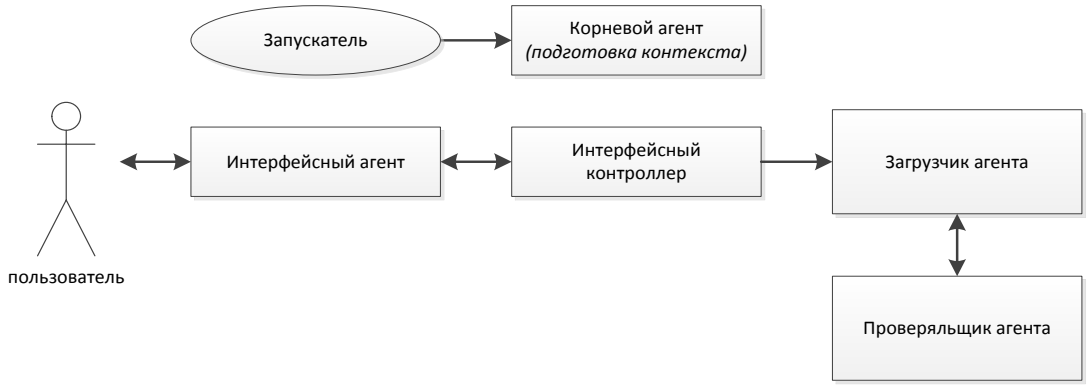

Рис. 67. Архитектура Загрузчика агентов

1. Корневого агента. В инициализирующем сообщении для него указан инфоресурс агента, для которого надо загрузить код. Корневой агент ожидает сообщения от интерфейсного контроллера о загрузке JAR-архива, распаковывает его и прикрепляет к понятиям «байткод» соответствующие class-файлы из архива;

2. Повторно-используемого Интерфейсного Агента для организации пользовательского интерфейса;

3. Интерфейсного контроллера, предоставляющего веб-интерфейс для загрузки JAR-архива;

4. Собственно агента-загрузчика, который загружает код агента в соответствующий инфоресурс. Перед загрузкой осуществляется проверка кода агентом-проверяльщиком.

5. Агента-проверяльщика, который используется агентом-загрузчиком для проверки кода. Этот агент является повторно-используемым: он используется для проверки агентов перед запуском.

Средства платформы, используемые при реализации. Загрузчик агента сделан как сервис из агентов, поэтому он использует базовые средства платформы: запуск агентных сервисов.

Разработчики. Код для загрузки кода агента разработан автором диссертации. Набор агентов и собственно сервис разработаны Ф.М. Москаленко и В.А. Тимченко.

4.4. Прикладные сервисы

Перечисленные ниже прикладные сервисы разработаны в рамках проекта «Модели, методы и инструментальные сервисы для создания профессиональных виртуальных облачных сред» [18].

4.4.1. Графический редактор трёхмерных сцен

Назначение: Графический редактор предназначен для создания, изменения, просмотра визуального отображения декларативной модели виртуальной среды, представленной в виде инфоресурса. Концепция и архитектура описаны в [16, 17, 18, 19, 20, 53, 54].

Общая функциональность: Графический редактор предлагает веб-интерфейс, в котором отображается трёхмерная сцена виртуальной среды на основе загруженного инфоресурса декларативной модели (Рис. 68). Декларативная модель виртуальной среды определяет необходимое логическое наполнение сцены в виде множества объектов и их атрибутов. Редактор позволяет визуально изменять презентационные атрибуты объектов — их форму, отображение, положение в пространстве, анимацию. Пользователь может:

1. Просматривать виртуальную среду, используя перемещение в трёхмерном пространстве, или меняя обзор камеры;

2. Редактировать виртуальную среду, используя следующие действия:

2.1. Перемещать, вращать и масштабировать объекты сцены;

2.2. Изменить форму отображения объекта сцены выбором нужной трёхмерной модели и текстуры;

2.3. Выбрать анимацию объекта сцены;

2.4. Сохранить виртуальную сцену или отдельные объекты в инфоресурс декларативной модели;

3. Создавать графические элементы, применяемые в редактировании объектов сцены, используя следующие действия:

3.1. Создать библиотеку трёхмерных моделей;

3.2. Создать трёхмерные модели, включая статичные и анимационные, на основе полигональных сеток или спрайтов;

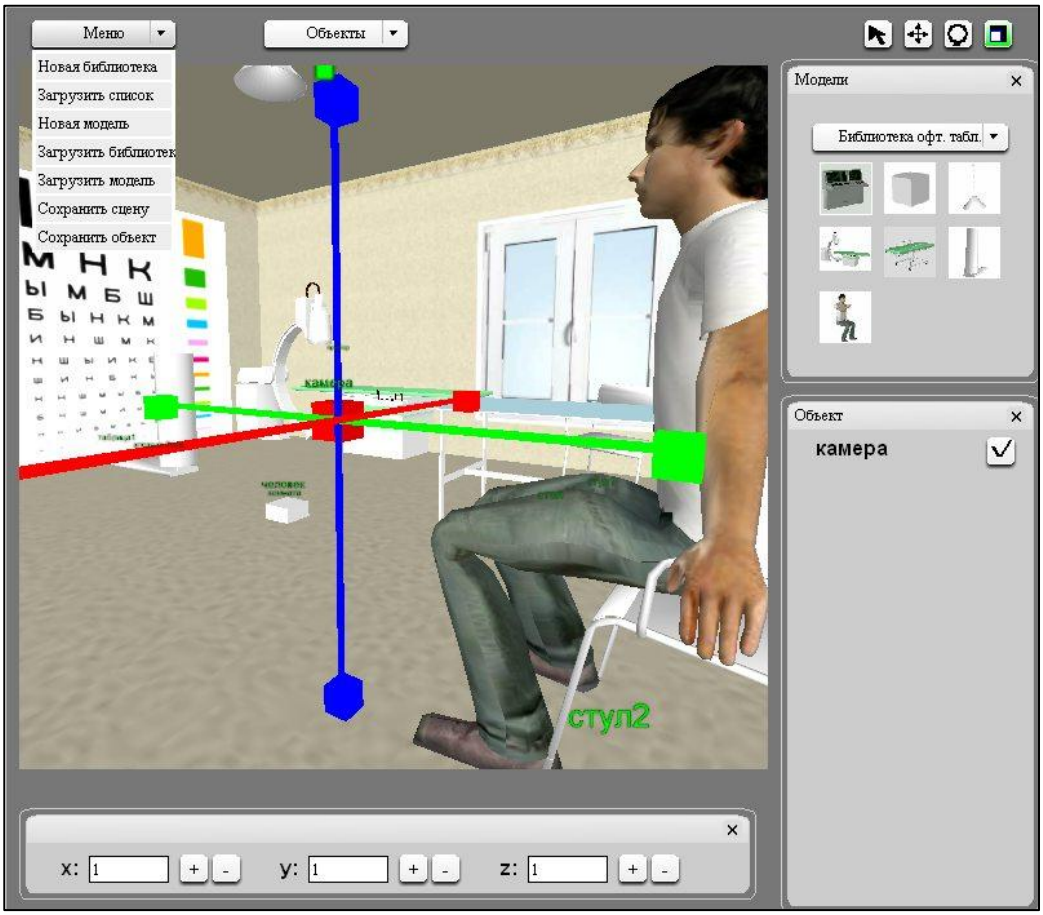

Рис. 68. Графический редактор

Общая архитектура. Графический редактор состоит из трёх агентов (Рис. 69):

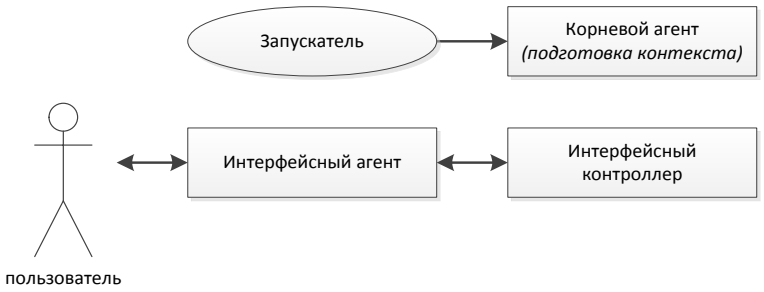

Рис. 69. Архитектура Графического редактора

1. Корневого агента;

2. Повторно используемого Интерфейсного Агента;

3. Интерфейсного Контроллера, который отображает веб-клиент графического редактора (реализованного через плагин Flash) и обрабатывает его сообщения о действиях пользователя и об изменении виртуальной среды.

Средства платформы, используемые при реализации. Так как Графический редактор сделан как сервис из агентов, то он использует базовые средства платформы: запуск агентных сервисов. Как сервис с пользовательским интерфейсом, Графический редактор использует: Интерфейсный агент для организации пользовательского веб-интерфейса, который включает возможность отображения и работы встроенного Flash-плагина; средство платформы для преобразования декларативной модели в формат JSON; средства платформы для асинхронной загрузки с неё и на неё различных ресурсов (файлы моделей, текстур и др.).

Разработчики. В. В. Грибова, Л.А. Федорищев.

4.4.2. Интерпретатор виртуальных сред

Назначение: Интерпретатор предназначен для отображения и функционирования виртуальной среды в соответствии с декларативной моделью, представленной в виде инфоресурса.

Общая функциональность: Интерпретатор предлагает веб-интерфейс, в котором отображается и функционирует виртуальная среда (Рис. 70).

Рис. 70. Интерпретатор виртуальных сред. Показан пример отрисовки сцены, созданной в Графическом Редакторе.

Основные функции интерпретатора виртуальных сред:

1. Создание, отображение и управление виртуальной среды по ее декларативной модели;

2. Загрузка ресурсов виртуальной сцены: трёхмерных моделей объектов, текстур, иконок, анимаций;

3. Интерактивное взаимодействие с объектами сцены (с помощью манипулятора «мышь»);

4. Инициализация логических параметров виртуальной среды;

5. Сохранение и загрузка анимационных роликов;

6. Отображение объяснений и результатов действий пользователя.

Общая архитектура. Интерпретатор состоит из трёх агентов (Рис. 71):

1. Корневого агента;

2. Повторно используемого Интерфейсного Агента;

3. Интерфейсного Контроллера, который отображает веб-клиент интерпретатора (реализованного через плагин Flash) и обрабатывает его сообщения о действиях пользователя и об изменении виртуальной среды,

Интерпретатор также может отправлять сообщения другим (произвольным) агентам, указанным в декларативной модели для реализации дополнительной функциональности.

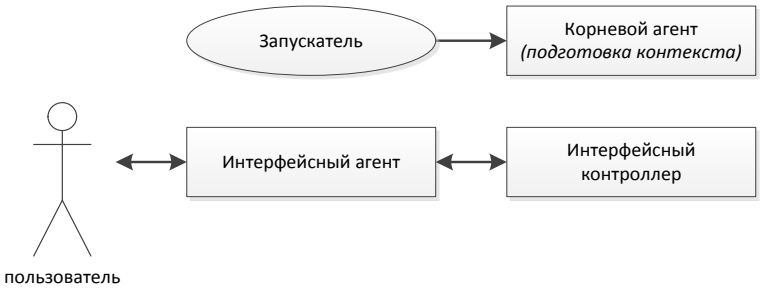

Рис. 71. Архитектура Интерпретатора

Средства платформы, используемые при реализации. Так как Интерпретатор сделан как сервис из агентов, то он использует базовые средства платформы: запуск агентных сервисов. Как сервис с пользовательским интерфейсом, Интерпретатор использует Интерфейсный Агент для организации пользовательского веб-интерфейса, который включает возможность отображения и работы встроенного Flash-плагина. Интерпретатор также использует: средство платформы для преобразования декларативной модели в формат JSON и средства платформы для асинхронной загрузки с неё и на неё различных ресурсов (файлы моделей, текстур и др.).

Разработчики. В. В. Грибова, Л.А. Федорищев.

4.4.3. Компьютерный обучающий тренажёр по офтальмологии

Назначение: Компьютерный обучающий тренажёр включает обучающие задания по классическим методам исследования в офтальмологии: обу-

чение студентов определению остроты зрения по таблицам, определению остроты зрения по оптотипам Б.Л. Поляка. Модели рассмотрены в [9, 10, 11, 12, 13, 36].

Общая функциональность:

1. Проведение классических методов исследования в офтальмологии;

2. Интерактивное взаимодействие с виртуальным медицинским оборудованием и виртуальным пациентом;

3. Генерация упражнений на основе случайных значений требуемых логических атрибутов на основе декларативной модели;

4. Объяснение ошибочных действий обучающегося.

Общая архитектура. Компьютерный обучающий тренажёр использует Интерпретатор виртуальных сред, поэтому включает дополнительные агенты для реализации специфических функций каждого из методов исследования, таких как «Агент обработки офтальмологической таблицы», «Агент обработки оптотипов».

Разработчики. В.В. Грибова, М.В. Петряева, Л.А. Федорищев, М.Ю. Черняховская.

4.4.4. Виртуальная химическая лаборатория

Назначение: Виртуальная химическая лаборатория предназначена для визуализации проведения химических опытов и экспериментов из школьной практики, описанных в декларативной модели, в программной виртуальной среде (Рис. 72).

Общая функциональность:

1. Проведение школьных химических экспериментов;

2. Интерактивное взаимодействие с виртуальным химическим оборудованием;

3. Генерация упражнений на основе случайных значений требуемых логических атрибутов на основе декларативной модели;

4. Объяснение ошибочных действий обучающегося.

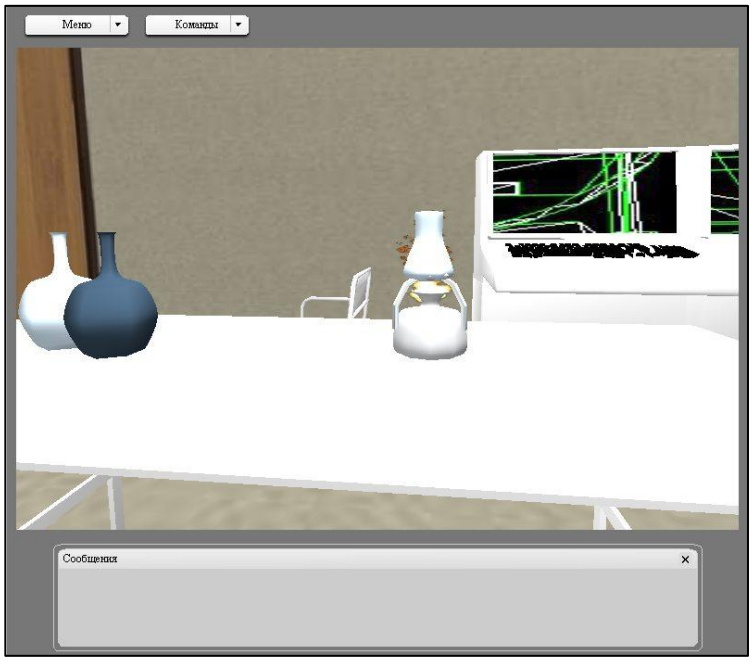

Рис. 72. Виртуальная химическая лаборатория

Общая архитектура. Сервис «Виртуальная химическая лаборатория» использует Интерпретатор виртуальных сред, и включает только один дополнительный агент для реализации смешивания химических реагентов и получения результатов реакций.

Разработчики. Л.А. Федорищев.

4.4.5. Модель городского района

Назначение: Демонстрационный проект модели городского района предназначен для визуальной интерактивной демонстрации объектов запланированного под строительство нового жилого комплекса или существующего района города и позволяет показать клиентам все преимущества покупки недвижимости в данном районе (Рис. 73).

Общая функциональность: Интерактивный просмотр и исследование модели.

Рис. 73. Модель городского района

Общая архитектура. Модель городского района основывается на сервисе интерпретатора виртуальных сред.

Средства платформы, используемые при реализации. Так как Модель городского района сделана как сервис из агентов, то она использует базовые средства платформы: запуск агентных сервисов. Как сервис с пользовательским интерфейсом, он использует Интерфейсный агент для организации пользовательского веб-интерфейса, который включает возможность отображения и работы встроенного Flash-плагина.

Разработчики. Л.А. Федорищев.

4.5. Выводы по главе

В данной главе:

1. Предложена технология разработки сервисов, состоящая из шести этапов, которые должен пройти разработчик сервисов. В частности, показано, что для разработки сервиса нужно разработать (или повторно использовать) подходящие агенты, инфоресурсы и метаинформацию.

2. Предложена технология разработки агентов.

3. Описаны системные инструментальные сервисы, которые используются для поддержания разработки прикладных сервисов: редактор, используемый как для создания прикладных метаинформаций и инфоресурсов, прогонщик тестов, используемый для тестирования агентов, генератор и загрузчик кода агентов.

4. Описан ряд прикладных сервисов, разработанных с использованием Платформы: два инструментальных сервиса: графический редактор трёхмерных сцен, построенный на его основе интерпретатор виртуальных сред, три прикладных сервиса: компьютерный обучающий тренажёр по офтальмологии, виртуальная химическая лаборатория, модель городского района.

ЗАКЛЮЧЕНИЕ

В настоящей работе получены следующие результаты:

1. Разработаны основные требования к проекту IACPaaS как непрерывному процессу совершенствования Фонда, поддержанного Платформой. Приведены концептуальная архитектура проекта IACPaaS. Предложена концепция Платформы, как сущности, состоящей из Фонда, как хранилища инфоресурсов, сервисов и агентов, сайта как средства доступа к ресурсам Платформы и виртуальной машины для выполнения функциональности Платформы и сервисов.

2. Предложена модель информационных ресурсов как особого вида семантических сетей, структура которых задаётся метаинформацией, которая, в свою очередь, также является семантической сетью особого вида. Для описания множества метаинформаций введено исчисление формул метаинформаций. Формулы имеют логическую семантику: формула истинна для тех и только тех инфоресурсов, которые принадлежат классу, задаваемому этой метаинформацией. Формулы также имеют порождающую семантику, которая задаёт процесс порождения инфоресурсов как исчисление. Введена модель решателя задач как набора взаимодействующих с помощью недетерминированного обмена сообщениями агентов. Для разрешения неконфлюентности результата, порождаемой проблемой недетерминизма, в главе введено понятие корректности результата работы, введено понятие истории обработки сообщений и последовательная модель решателя. Показано, что для недетерминированной и параллельной модели посылки сообщений нарушение корректности результата следует из некорректности истории обработки сообщений. Разработана модель интерфейса как взаимодействие трёх модулей: интерфейсного агента, интерфейсного контролёра и агентов сервиса. Предложено декларативное представление абстрактного интерфейса.

3. Разработана концептуальная архитектура платформы IACPaaS, которая состоит из четырёх уровней: системного, библиотечного, сервисного уровней и административной системы. Предложен способ поддержки работы сервисов в контексте трёх процессоров: процессора пользовательских интерфейсов (обеспечивающего взаимодействие сервиса с пользователем), процессора информационных ресурсов (обеспечивающего взаимодействие сервиса с инфоресурсами), процессора решателей задач (обеспечивающего обмен сообщениями и саму работу агентов, составляющих сервис).

4. Выдвинуты требования для процессора инфоресурсов, в частности, требования о соответствии модели, разработанной во второй главе, требования к надёжности, требования к программному интерфейсу. Выдвинуты требования для процессора решателей задач, в частности, о единице распараллеливания; приведены требования к планировщику. Выдвинуты требования для процессора пользовательских интерфейсов, в частности, требование интеграции с CMS «MediaWiki» и удобства программирования интерфейса. Разработаны проекты последовательной и параллельной версий процессоров инфоресурсов. Разработан проект модуля представления (предоставляющий программный интерфейс для прикладных программистов), который поддерживает логическую семантику метаинформации. Разработан проект процессора решателя задач как многоуровеневой структуры, где нижние уровни являются основой для верхних. Выделено пять уровней: общесистемный уровень, сервисный уровень, агентный уровень, блокопродукционный уровень, уровень процессов. Последний уровень опирается на процессор инфоресурсов для осуществления синхронизации между отдельными процессами. Разработана структура уровней, их сущностей и поведение соответствующих сущностей. Разработан проект процессора пользовательских интерфейсов. Введено понятие интерфейсного агента, как преобразователя абстрактного интерфейса в конкретный, интерфейсного контролёра, как диспетчера сообщений.

5. Предложена технология разработки сервисов, состоящая из шести этапов. В частности, показано, что для разработки сервиса нужно разработать (или повторно использовать) подходящие агенты, инфоресурсы и метаинформацию. Предложена технология разработки агентов. Описаны системные инструментальные сервисы, которые используются для поддержания разработки прикладных сервисов: редактор, используемый для создания прикладных метаинформаций и инфоресурсов, прогонщик тестов, используемый для тестирования агентов, генератор и загрузчик кода агентов.

ЛИТЕРАТУРА

1. Басыров Р. 1С-Битрикс: Корпоративный портал. Руководство разработчика. — М.: Рид Груп, 2012. — 352 с.

2. Бобков В.А., Голенков Е.А., Клещев А.С., Нурминский Е.А., Исследования в области информатики в ИАПУ ДВО РАН // Вестник ДВО РАН. 2006. № 4. С. 51-63

3. Гаврилова Т.А., Хорошевский В.Ф., Базы знаний интеллектуальных систем. — СПб.: Питер, 2000. — 384 с.

4. Грибова В.В., Агентный подход к разработке интеллектуальных Интернет-сервисов / А.С. Клещев, Д.А. Крылов, Ф.М. Москаленко, В.А. Тимченко, Е.А. Шалфеева // Труды конгресса по интеллектуальным системам и информационным технологиям «IS&IT'12». — М.: Физматлит, 2012. т.1. — С. 218-223.

5. Грибова В.В., Клещев А.С., Крылов Д.А. Контекстно-зависимые грамматики искусственных языков // Научно-техническая информация. Сер.2. 2013. № 6. С. 1-9.

6. Грибова В.В., Клещев А.С., Крылов Д.А. Контекстно-свободные грамматики искусственных языков // Научно-техническая информация. Сер.2. 2013. №4. С.9-17.

7. Грибова В.В., Клещев А.С., Шалфеева Е.А. Управление интеллектуальными системами // Известия РАН. Теории и системы управления. 2010. № 6. С. 122-137.

8. Грибова В.В., Облачная платформа для разработки и управления интеллектуальными системами / А.С. Клещев, Д.А. Крылов, Ф.М. Москаленко, С.В. Смагин, В.А. Тимченко, М.Б. Тютюнник, Е.А. Шалфеева // Международная научно-техническая конференция «Открытые семантические технологии проектирования интеллектуальных систем» (OSTIS-2011). - Минск: БГУИР. 2011. С. 5-14.

9. Грибова В.В., Петряева М.В., Федорищев Л.А. Компьютерный обучающий тренажер с виртуальной реальностью для офтальмологии // Открытое образование, 2013, №6

10. Грибова В.В., Петряева М.В., Федорищев Л.А. Разработка виртуального мира медицинского компьютерного обучающего тренажера // Дистанционное и Виртуальное Обучение, 2011- № 9.- С.56-66;).

11. Грибова В.В., Петряева М.В., Федорищев Л.А., Черняховская М.Ю. Модель виртуального мира мультимедиа тренажера для медицинского образования // Varna, Bulgaria, 2011, №22, P. 140-148

12. Грибова В.В., Петряева М.В., Федорищев Л.А., Черняховская М.Ю. Структура формального представления объектов в компьютерных диагностических тренажерах // PhysioMedi, 2011

13. Грибова В.В., Петряева М.В., Федорищев Л.А., Черняховская М.Ю. Формализация методов исследования в офтальмологии для компьютерных диагностических тренажеров // PhysioMedi, 2012

14. Грибова В.В., Платформа для разработки интеллектуальных мульти-агентных интернет-сервисов / А.С. Клещёв, Д.А. Крылов, Ф.М. Москаленко, В.А. Тимченко, Е.А. Шалфеева, А.В. Созыкин // Материалы всероссийской науч.-практ. конф. «Информационные технологии и высокопроизводительные вычисления». - Хабаровск. – С. 95-104.

15. Грибова В.В., Проект IACPaaS. Комплекс для интеллектуальных систем на основе облачных вычислений / А.С. Клещев, Д.А. Крылов, Ф.М. Москаленко, С.В. Смагин, В.А. Тимченко, М.Б. Тютюнник, Е.А. Шалфеева // Искусственный интеллект и принятие решений. 2011. № 1. С. 27-35.

16. Грибова В.В., Федорищев Л.А. Виртуальная реальность в образовании: система разработки интернет-проектов // НИТО, 2012

17. Грибова В.В., Федорищев Л.А. Интернет-комплекс для создания обучающих систем с виртуальной реальностью // Дистанционное и Виртуальное Обучение, 2012 — № 7

18. Грибова В.В., Федорищев Л.А. Облачный сервис для разработки виртуальных интерактивных сред // Varna, Bulgaria, 2013

19. Грибова В.В., Федорищев Л.А. Обучающие виртуальные системы и средства их создания // Вестник компьютерных и информационных технологий, 2012, №3, с. 48-51

20. Грибова В.В., Федорищев Л.А. Обучающие виртуальные системы на основе онтологий и трехмерной компьютерной графики // Образование и Виртуальность, Ялта. – 2011

21. Джарратано Дж., Райли Г. Экспертные системы. Принципы разработки и программирование. — СПб: Вильямс. 2007. — 1152 с.

22. Джексон П. Введение в экспертные системы: Учебное пособие / Пер. с англ. — М.: Вильямс, 2001. — 624 с.: ил.

23. Камер Д. Сети TCP/IP, том 1. Принципы, протоколы и структура. Пер. с англ. — М. : «Вильямс», 2003. — 880 с.

24. Клещев А. С., Орлов В. А. Компьютерные банки знаний. Требования к многоцелевому банку знаний. // Информационные технологии. — 2006. — № 4. — с.21-28.

25. Клещев А. С., Орлов В.А. Многоцелевой банк знания. Часть 1. Концепция и политика. — Владивосток: ИАПУ ДВО РАН, 2003. — 40 с.

26. Клещев А. С., Орлов В.А. Многоцелевой банк знания. Часть 3. Концепция универсального редактора ИРУО. — Владивосток: ИАПУ ДВО РАН, 2003.— 40 с.

27. Клещев А.С., Использование технологиии облачных вычислений для разработки и управления интеллектуальными системами / В.В. Грибова, Е.А. Шалфеева, Д.А. Крылов, С.В. Смагин, Ф.М. Москаленко, В.А.Тимченко, М.Б. Тютюнник — ИАПУ ДВО РАН, Владивосток, 2010, 18 с.

28. Клещев А.С., Крылов Д.А. Корректность результата вычислений в условиях состояния гонки // Информатика и системы управления. 2013. №3(37). С. 99-109.

29. Клещев А.С., Орлов В.А. Компьютерные банки знаний. Универсальный подход к решению проблемы редактирования информации // Информационные технологии. — 2006. — №5. — с. 25-31.

30. Крылов Д.А. Облачная платформа для создания и управления интеллектуальными интернет-сервисами // Инфокоммуникационные и вычислительные технологии и системы: материалы III Международной конференции. — Улан-Удэ: изд-во Бурятского госуниверситета, 2010. С. 180-183.

31. Москаленко Ф.М., Тимченко В.А., Шалфеева Е.А. Административная система (версия 1.0) интернет-комплекса IACPaaS. Зарегистрировано в Реестре программ для ЭВМ 28 сентября 2012 г. Свидетельство Федеральной службы по интеллектуальной собственности о государственной регистрации программы для ЭВМ № 2012618861.

32. Москаленко Ф.М., Тимченко В.А., Шалфеева Е.А. Административная система облачной платформы для разработки и использования интеллектуальных интернет-сервисов // Искусственный интеллект. Интеллектуальные системы: Материалы XIII Международной научно-технической конференции (пос. Кацивели, АР Крым, 23-27 сентября 2013 г.). — Донецк: ИПИИ «Наука і освіта». — 2013. — С. 25-28.

33. Никифорова Н.Ю., Черняховская М.Ю. Базы наблюдений в компьютерном банке медицинского знания // Материалы Седьмой Международной научно-технической конференции «Искусственный интеллект. Интеллектуальные и многопроцессорные системы», Кацивели, 24-29 сентября 2006. Таганрог: Изд-во ТРТУ, 2006, Т.1, С.76-80

34. Орлов В.А. Многоцелевой банк знания. Часть 6. Особенности реализации. — Владивосток: ИАПУ ДВО РАН, 2003. — 40 с.

35. Орлов В.А. Многоцелевой банк знания. Часть 7. Примеры редактирования информации различных уровней общности. — Владивосток: ИАПУ ДВО РАН, 2003. — 43 с.

36. Петряева М.В., Федорищев Л.А. Формализация методов исследования в неврологии для медицинских интеллектуальных систем // Современные системы искусственного интеллекта и их приложения в науке, 2013

37. Рыбина Г.В. Основы построения интеллектуальных систем. — М.: Финансы и статистика; ИНФРА-М, 2010. — 432 с.

38. Табаков В.В. Облачные вычисления — технологическая инновация // Сборник материалов Второй международной научно-практической конференции «Проблемы развития инновационно-креативной экономики». URL : http://econference.ru/blog/conf06/227.html Дата обращения : 12.12.2013.

39. Шафер Д., Фатрелл Р., Шафер Л. Управление программными проектами. Достижение оптимального качества при минимуме затрат : Пер. с англ. — М.: Вильямс, 2003. — 1136 стр.

40. Bachant J. McDermott J. R1 revisited : four years in the trenches // The AI Magazine. 1984. Vol. 5. No. 3. Pp.21-32.

41. Chang W., Abu-Amara H., Sanford J. Transforming enterprise cloud services. — London, New York: Springer, 2010. — 525 p.

42. Chang, et al. Bigtable : A Distributed Storage System for Structured Data // In proc. of the 7th USENIX Symposium on Operating Systems Design and Implementation. 2006. Vol. 7. Pp.205-218.

43. Claybrook B. OLTP: Online Transaction Processing Systems. — Willey, 1992. — 384 p.

44. Differential Ontology Editor. URL : http://www.eurecom.fr/~troncy/DOE/. Дата обращения : 12.12.2013.

45. Django. The Web framework for perfectionists with deadlines. URL : https://www.djangoproject.com/. Дата обращения : 12.12.2013.

46. Drupal. Come for software, stay for the community. URL : https://drupal.org/. Дата обращения : 12.12.2013

47. Fielding et al. Hypertext Transfer Protocol — HTTP/1.1. — RFC Editor, 1999. URL : https://tools.ietf.org/html/rfc2616. Дата обращения : 12.12.2013

48. Gennari, J.H. et al. The evolution of Protégé: An environment for knowledge-based systems development // International Journal of Human-Computer Studies. 2003. No. 58(1). Pp. 89-123

49. Glass R.L. Facts and Fallacies of Software Engineering. — Boston: Addison-Wesley Professional, 2003. — 224 p.

50. Graham Paul. The Other Road Ahead. 2001. URL : http://www.paulgraham.com/road.html. Дата обращения : 12.12.2013.

51. Gray J. The Transaction Concept : Virtues and Limitations. Proceedings of the 7th International Conference on Very Large Databases. — Cupertino: Tandem Computers. pp. 144-154.

52. Gribova V. et al. A software platform for the development of intelligent multi-agent internet-services // Proceedings of the Distributed Intelligent Systems and Technologies Workshop (DIST'2013). — Petersburg, Russia. — Pp. 29-36.

53. Gribova V.V. , Fedorischev L.A. The architecture of Internet software environment for creating teachware with virtual reality // Emerging Intelligent Computing Technology and Applications. — Springer Berlin Heidelberg, 2012. Vol. 304, No. 11. — Pp. 394-399

54. Gribova V.V., Fedorischev L.A. Internet Software Environment for Creating Teachware with Virtual Reality // CISME, World Academic Publishing Company, Hong Kong. 2012, Vol. 2, No 8, pp. 25-29

55. Gribova V.V., Kleshchev A.S., Krylov D.A. Context-Dependent Grammars of Artificial Languages // Automatic Documentation and Mathematical Linguistics. 2013. Vol. 47. N 3. Pp. 93-101.

56. Gribova V.V., Kleshchev A.S., Krylov D.A. The context-free grammars of artificial languages // Automatic Documentation and Mathematical Linguistics. 2013. Vol. 47. N 2. Pp. 59-67.

57. How Heroku Works. URL : https://devcenter.heroku.com/articles/how-heroku-works. Дата обращения : 12.12.2013

58. Internet 2011 in numbers. Royal Pingdom. URL : http://royal.pingdom.com/2012/01/17/internet-2011-in-numbers/. Дата обращения : 12.12.2013

59. Internet 2012 in numbers. Royal Pingdom. URL : http://royal.pingdom.com/2013/01/16/internet-2012-in-numbers/. Дата обращения : 12.12.2013

60. Kendal, S.L.; Creen, M. An introduction to knowledge engineering. — London: Springer, 2007. — 287 p.

61. Knoodle. URL : http://knoodl.com. Дата обращения : 12.12.2013

62. McConnel S. Code Complete : A Practical Handbook of Software Construction, 2nd ed. – Redmond, Wa.: Microsoft Press, 2004. — 960 p.

63. MediaWiki. URL : https://www.mediawiki.org/. Дата обращения : 12.12.2013.

64. MoKi: the Enterprise Modelling Wiki, URL : https://moki.fbk.eu/. Дата обращения : 12.12.2013.

65. Moskalenko P. , Shalfeyeva E., Timchenko V. An administrative system of a software platform for cloud computing and development of intellectual services // In proc. of 2013 2nd International Symposium on Computer, Communication, Control and Automation (3CA 2013). 1-2 Dec. 2013. pp.155-156.

66. Musen M. Technology for Building Intelligent Systems: From Psychology to Engineering. — Modelling Complex Systems. / ed. B. Shuart. — Nebraska: University of Nebraska Press, 2007. Vol. 52. — pp. 145-184.

67. Neologism — Easy Vocabulary Publishing. URL : http://neologism.deri.ie/. Дата обращения : 12.12.2013.

68. Nielson Jakob. Mobile Sites vs. Apps: The Coming Strategy Shift. 2012, URL : http://www.nngroup.com/articles/mobile-sites-vs-apps-strategy-shift/. Дата обращения : 12.12.2013.

69. Norvig P., Cohn D. Adaptive software. URL : http://norvig.com/adapaper-pcai.html. Дата обращения : 12.12.2013.

70. OntoWiki — Agile Knowledge Management and Semantic Web, URL : http://aksw.org/Projects/OntoWiki.html. Дата обращения : 12.12.2013.

71. Open Knowledge Base Connectivity Home Page, URL : http://www.ai.sri.com/~okbc/. Дата обращения : 12.12.2013.

72. PoolParty Semantic Suite, URL : http://www.poolparty.biz/ Дата обращения : 12.12.2013.

73. Reenskaug Trygve. MVC. Xerox PARC 1978-79. http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html Дата обращения : 12.12.2013.

74. Reese George, Cloud Application Architectures. Building Applications and Infrastructure in the Cloud. — O'Reilly Media, 2009. — 208 p.

75. Ruby on Rails. Web development that doesn’t hurt. URL : http://rubyonrails.org/. Дата обращения : 12.12.2013.

76. Sanderson D. Programming Google App Engine: Build and Run Scalable Web Apps on Google's Infrastructure. — Sebastopol, California : O'Reilly Media, 2009. — 394 p.

77. Segaran T., Evans C., Taylor J. Programming the Semantic Web. — Sebastopol, CA : O’Reilly Media. — 302 p.

78. Shoham Y. Agent Oriented Programming (Technical Report STAN-CS-90-1335). — Stanford University: Computer Science Department, 1990.

79. Soboleo, URL : http://www.soboleo.com/ Дата обращения : 12.12.2013.

80. Software as a Service: Strategic Backgrounder. — 2001. — URL : http://www.siia.net/estore/ssb-01.pdf. Дата обращения : 12.12.2013.

81. Sowa J. Semantic Networks. Encyclopedia of Artificial Intelligence. — 1987. — URL : http://www.jfsowa.com/pubs/semnet.htm Дата обращения : 12.12.2013.

82. SPIN — SPARQL Interferencing Notation. URL : http://spinrdf.org/ Дата обращения : 12.12.2013.

83. The Protégé Ontology Editor and Knowledge Acquisition System. URL : http://protege.stanford.edu/ Дата обращения : 12.12.2013.

84. TopQuadrant : TopBraid Live. URL : http://www.topquadrant.com/products/TB_Live.html Дата обращения : 12.12.2013.

85. TopQuadrant, URL : http://www.topquadrant.com/index.html Дата обращения : 12.12.2013.

86. Tudorache T., Nyulas C., Noy N., Musen M. WebProtégé : A Collaborative Ontology Editor and Knowledge Acquisition Tool for the Web. Semantic Web. // Semantic Web Journal. IOS Press, 2013. Vol. 4 No. 1/2013, pp.89-99

87. Ulman Ch., Dykes L. Beginning Ajax. — New York : Wiley, 2007. — 498 p.

88. Web application framework. URL : http://docforge.com/wiki/Web_application_framework Дата обращения : 12.12.2013.

89. What is Cloud Computing by Amazon Web Services. — URL : http://aws.amazon.com/what-is-cloud-computing/. Дата обращения : 12.12.2013

90. Windows Azure General Availability — The Official Microsoft Blog. — URL : http://blogs.technet.com/b/microsoft_blog/archive/2010/02/01/windows-azure-general-availability.aspx. Дата обращения : 12.12.2013.

91. WordPress. Blog Tool, Publishing Platform, and CMS. URL : https://wordpress.org/. Дата обращения : 12.12.2013.

ПРИЛОЖЕНИЯ

Приложение 1. Структура программного интерфейса Storage

Как указано в разделе 3.2.2.1, модуль хранения предоставляет низкоуровневый программный интерфейс Storage. Этот программный интерфейс представлен Java-классом с одноимённым названием. Модуль представления использует этот интерфейс для предоставления высокоуровневого интерфейса IKBI, рассмотренного в следующем приложении.

Программный интерфейс Storage предоставляет возможность управления объектами Фонда (Инфоресурсами, понятиями, отношениями. Метаинформация представляется так же в виде инфоресурсов; интерпретация инфоресурса как метаинформации осуществляется в другом модуле — модуле представления). Каждый объект имеет собственный уникальный идентификатор. Поэтому в сигнатуру всех методов для обращения с каким-либо объектом входит обязательный параметр — идентификатор объекта.

Рассматриваемый программный интерфейс реализован на языке программирования C++ для повышения эффективности взаимодействия с файлами Хранилища. Доступ к модулю на C++ осуществляется с помощью технологии JNI (Java Native Interface), при котором в Java-части создаётся класс, с методами, объявленными как native, для которых JNI-компилятор создаёт соответствующие заглушки на языке C. Ниже будет рассмотрена именно Java-часть этого программного интерфейса. *Примечание:* Каждый метод может сгенерировать исключение CacheException, которое для простоты в описаниях методов не показано.

Все действия с объектами должны осуществляться в рамках транзакции, открываемой и фиксируемой (или откатываемой) соответствующими методами.

Поддержка транзакций. Транзакция запускается методом

```
public native long startTransaction();
```

Этот метод возвращает идентификатор транзакции, который следует использовать во всех остальных низкоуровневых операциях с хранилищем.

Для фиксации транзакции используется метод

```
public native void commitTransaction(long trid);
```

которому передаётся идентификатор открытой транзакции. Для отката транзакции предназначен метод

```
public native void rollbackTransaction(long trid);
```

Общие для всех объектов хранилища методы. Первый — удаление объекта:

```
public native void deleteObject(
    long transactionId,
    long id);
```

Вторым параметром передаётся идентификатор удаляемого объекта. Второй метод — проверка типа объекта:

```
public native void checkType(
    long trid,
    long objectId,
    byte requiredType);
```

Второй параметр — идентификатор проверяемого объекта, третий — нужный тип. Если объект имеет нужный тип, выполнение метода завершается успешно, в противном случае гененрируется исключение.

Управление инфоресурсами. Для создания нового инфоресурса используется метод

```
public native long createInforesource(
    long transactionId,
    long metaInforesourceId,
    long metaAxiomId);
```

Вторым параметром является идентификатор метаинформации, а третьим — идентификатор понятия в этой метаинформации, которое будет использовано как корневая вершина при порождении. Чтобы продолжить порождение инфоресурса, необходимо определить его метаинформацию, для этого используется метод

```
public native long getMetaInforesourceId(
    long transactionId,
    long inforesourceId);
```

Управление понятиями инфоресурсов. Следующая группа методов используется для управления понятиями. Создание понятия:

```
public native long createConcept(
    long transactionId,
    long inforesourceId,
    byte conceptType);
```

После создания понятия можно установить его свойства: установить имя понятия:

```
public native void setConceptName(
    long trid,
    long conceptId,
    String name);
```

установить тип терминала-сорта:

```
public native void setConceptTerminalSortType(
    long trid,
    long conceptId,
    byte type);
```

установить значение терминала-значения:

```
public native void setConceptTerminalValueValue(
    long transactionId,
    long conceptId,
```

```
    Object value);
```

добавить комментарий к понятию

```
public native void setConceptComment(
    long transactionId,
    long conceptId,
    String comment);
```

У существующего понятия можно получить его свойства: имя:

```
public native String getConceptName(
    long transactionId,
    long conceptId);
```

тип (нетреминал, терминал-сорт или терминал-значение):

```
public native byte getConceptType(
    long transactionId,
    long conceptId);
```

тип терминала-сорта (строка, целое число, вещественное число, дата, логическое значение, блоб):

```
public native byte getConceptTerminalSortType(
    long transactionId,
    long conceptId);
```

значение терминала-значения:

```
public native Object getConceptTerminalValue(
    long transactionId,
    long conceptId);
```

тип значения терминала-значения:

```
public native byte getConceptTerminalValueType(
    long transactionId,
    long conceptId);
```

комментарий у понятия:

```
public native String getConceptComment(
```

```
    long transactionId,
    long conceptId);
```

Для получения идентификаторов инфоресурсов используется метод:

```
public native long[] getConcepts(
    long transactionId,
    long inforesourceId);
```

Для порождения и навигации по инфоресурсу используется следующие методы:

Получить идентификатор аксиомы инфоресурса:

```
public native long getAxiom(
    long transactionId,
    long inforesourceId);
```

Получить идентификатор метааксиомы инфоресурса (т.е. понятия, с которого начинается порождение):

```
public native long getMetaAxiom(
    long transactionId,
    long inforesourceId);
```

Получить идентификаторы отношений, которые исходят из понятия:

```
public native long[] getOutcomingRelations(
    long transactionId,
    long conceptId);
```

Получить идентификаторы отношений, которые входят в понятие:

```
public native long[] getIncomingRelations(
    long transactionId,
    long conceptId);
```

Получить идентификатор инфоресурса, содержащей понятие:

```
public native long getInforesourceId(
    long transactionId,
    long conceptId);
```

Управление отношениями. Следующий метод создаёт отношение между двумя понятиями:

```
public native long createRelation(
    long transactionId,
    long fromConceptId,
    long toConceptId,
    long metainforesourceId,
    long protoRelationId,
    byte endSp);
```

Для получения свойств отношения используется методы

```
public native RelationData getRelationData(
    long transactionId,
    long relationId);
public native long getRelationEnd(
    long transactionId,
    long relationId);
public native long getRelationMeta(
    long trid,
    long relationId);
```

Управление Хранилищем. Для управления и настройки работы Хранилища используются следующие методы:

```
Осуществить сборку мусора:
public native void collectGarbage(
    long trid,
    boolean isFull);
```

Установленный параметр `isFull` указывает, что надо осуществить более полный анализ используемых инфоресурсов, при котором в Хранилище остаются только те инфоресурсы, до которых есть ссылка от постоянных инфоресурсов (и сами постоянные инфоресурсы). Если параметр `isFull` не

установлен, используется упрощённая сборка мусора, при котором используется счётчик ссылок на инфоресурсы: удаляются те инфоресурсы, у которых счётчик достиг нуля. Упрощённая сборка мусора быстрее, но не удаляет циклически связанные инфоресурсы.

Запуск модуля хранения начинается с вызова метода:

```
native public long init(
    String cacheFilename);
```

Методу передаётся указание на расположение файлов Хранилища.

Приложение 2. Структура программного интерфейса IKBI

Модуль представления предоставляет API для других модулей Системы и агентов из Фонда. Модуль предоставляет три основных интерфейса (сущностей вида interface языка программирования Java): IInforesource (для управления инфоресурсами), IConcept (для управления понятиями) и IRelation (для управления отношениями).

Эти интерфейсы инкапсулируют низкоуровневый интерфейс Storage.

Ссылки на необходимые объекты (инфоресурсы или понятия) передаются блоку продукций вместе с сообщением, уже «обёрнутые» в один из этих программных интерфейсов. Соответственно, программист агента использует именно их для взаимодействия с объектами Фонда.

Основой трёх вышеперечисленных программных интерфейсов является интерфейс IFundObject, приведём его определение. *Примечание:* все методы всех рассматриваемых далее классов могут сгенерировать исключение IkbiException; его указание в листинге для простоты опущено).

```
public interface IFundObject
{
    /** Получить идентификатор объекта Фонда */
    long getId();
```

```
    /** Уничтожить объект Фонда (вызывая метод
     *  соответствующего объекта фонда):
     *  1. Удаление ИР: удаляется, если нет ссылок
     *  внутрь из другого ИР, иначе - не удаляется.
     *  2. Удаление отношения: удаляется, если у
     *  понятия-конца оно не единственное входящее,
     *  иначе - см. ниже.
     *  3. Удаление понятия: если имеет одно
     *  входящее отношение, то удаляется вместе с
     *  этим входящим отношением и рекурсивно
     *  удаляются потомки, иначе - не удаляется.
     */
    void delete();
}
```

Теперь рассмотрим программный интерфейс IInforesource:

```
public interface IInforesource
    extends IFundObject
{

    /** Получить понятие-аксиому инфоресурса */
    IConcept getAxiom();

    /** Получить метаинфоресурс */
    IInforesource getMetaInforesource();

    /** Получить имя инфоресурса */
    String getName();
```

```
/** Получить все понятия инфоресурса */
IConcept[] getAllConcepts();

/** Получить генератор инфоресурса, используя
 * этот инфоресурс как метаинформацию */
IMetaInforesourceGenerator getGeneratorByMeta();

/** Получить генератор данного инфоресурса для
 * продолжения порождения */
IInforesourceGenerator getGenerator();

/** Найти понятие в текущем инфоресурсе одним из
 * двух способов: 1) по пути к нему от аксиомы,
 * 2) среди всех понятий инфоресурса.
 * @param pathToConcept путь к понятию от аксиомы
 * нужно начинать с символа "/" и указывать имена
 * промежуточных понятий, разделяя их символом "/"
 * при поиске среди всех понятий инфоресурса
 * задается только имя искомого понятия
 */
IConcept findConcept(String pathToConcept);

/**
 * Найти понятие в текущем инфоресурсе по пути к
 * нему от понятия-аксиомы поиск нужно начинать
 * с символа "/"
 * Синоним для getAxiom().goTo(path)
 */
IConcept goTo(String path);
```

```
    /**
     * Найти понятие в текущем инфоресурсе по пути к
     * метапонятию в метаинформации.
     * поиск нужно начинать с символа "/"
     * Синоним для getAxiom().metaGoTo(metaPath)
     */
    IConcept gotoByMeta(String metaPath);

    /** Проверить полноту инфоресурса
     * @param traverser позволяет описать условия
     * проверок на полноту ИРов, на которые
     * сделаны ссылки
     */
    void checkCompleteness(ITraverser traverser);
}
```

В этом интерфейсе можно выделить операции получения отдельных элементов инфоресурса (имени инфоресурса, его понятий, корневого понятия и т.д.) — это методы с префиксом «get», методы навигации по инфоресурсу, которые позволяют обратиться к понятиям инфоресурса по их путевым именам (методы с префиксом «goto»). Кроме того, есть несколько методов для копирования и сравнения инфоресурсов.

Инфоресурсы состоят из понятий, для доступа к ним используется интерфейс IConcept:

```
public interface IConcept
    extends IFundObject
{
    /**
     * Получить инфоресурс, которому принадлежит
```

```
 * понятие
 */
IInforesource getInforesource();

/* Управление отношениями понятия */

/** Получить исходящие из понятия отношения */
IRelation[] getOutcomingRelations();

/** Получить входящие в понятие отношения */
IRelation[] getIncomingRelations();

/* Управление основными атрибутами */

/**
 * Получить имя понятия (только для нетерминалов
 * и терминалов, описывающих сорта)
 */
String getName();

/** Получить комментарий понятия */
String getComment();

/**
 * Получить значение понятия (для терминалов,
 * описывающих значения)
 */
<T> T getValue();
```

```
/**
 * Получить тип значения понятия (для терминалов,
 * описывающих сорта и значения)
 */
ValueType getValueType();

/**
 * Получить тип понятия (возможные значения –
 * аксиома, нетерминал, терминал, описывающий сорт,
 * терминал, описывающий значение)
 */
ConceptType getType();

/* Перемещение по инфоресурсу */

/**
 * Перейти к понятию-потомку (прямому) с именем
 * nextConceptName
 */
IConcept next(
    String nextConceptNameOrStringedValue);

/**
 * Перейти к понятию-потомку (прямому), которому
 * соответствует заданное метапонятие
 */
IConcept nextByMeta(
    String nextMetaConceptNameOrStringedValue);
```

```
/**
 * Перейти к прямому понятию-предку, которому
 * соответствует заданное метапонятие
 */
IConcept prevByMeta(
    String nextMetaConceptNameOrStringedValue);

/**
 * Перейти к понятию-потомку по пути
 * (id1/id2/../idN)
 *
 * @param path путь к понятию-потомку в текущем
 * инфоресурсе (имена понятий разделяются
 * символом "/")
 *
 * @return искомое понятие
 */
IConcept goTo(String path);

/** Перейти к понятию-потомку по пути в
 * метаинформации
 * @param metaPath путь к понятию в метаинформации,
 * являющемуся метапонятием для искомого (имена
 * понятий разделяются символом "/")
 */
IConcept gotoByMeta(String metaPath);

/** Получить набор (прямых) понятий-потомков */
IConcept[] getChildren();
```

```
    /* Генератор и редактор */

    /**
     * Получить интерфейс для редактирования
     * атрибутов понятия
     */
    IConceptEditor getEditor();

    /**
     * Получить прямой потомок данного понятия по
     * метаотношению
     */
    IConcept next(IRelation metaRelation);
}
```

Понятия связываются отношениями, доступ к которым осуществляется через программный интерфейс IRelation:

```
public interface IRelation extends IFundObject
{
    /** Получить понятие,
     * являющееся началом отношения */
    IConcept getBegin();

    /** Получить понятие,
      * являющееся концом отношения */
    IConcept getEnd();

    /** Получить метаотношение данного отношения */
    IRelation getProtoRelation()
```

```
;

    /** Получить спецификатор конца отношения */
    RelationSpecifierType getEndSp();

    /** Получить понятие, являющееся концом
     * метаотношения */
    IConcept getProtoEnd();

    /**
     * Получить интерфейс для редактирования
     * атрибута отношения */
    IRelationEditor getEditor();
}
```

Приложение 3. Структура программного интерфейса UiBuildHelper

Для случаев, когда прикладному программисту необходимо построить фрагмент интерфейса (или весь интерфейс окна) вручную, ему предлагается программный интерфейс UiBuildHelper, который предлагает методы, упрощающие создание элементов абстрактного интерфейса.

Для каждого типа интерфейсного элемента (например, кнопки), существует несколько одноимённых методов, различающихся сигнатурами: от большей детализации, с большим числом параметров, когда прикладной программист указывает все свойства интерфейсного элемента, до минимальной детализации, когда большинство параметров принимают значения по умолчанию. Для добавления новых свойств используются «декораторы».

В листинге ниже приведён сокращённый пример, с параметрами по умолчанию. *Примечание:* все методы всех рассматриваемых далее классов могут сгенерировать исключение IkbiException; его указание в листинге для простоты опущено).

```
public class UiBuildHelper
    extends UiBuildHelperBase
{
    /* blob /
       Методы для формирования интерфейсного элемента
       «Блоб (загружаемый файл)» */

    public static IConcept blob(
        String paramName,
        String title,
        Map<String, Object> params);

    /* button /
      Методы для формирования интерфейсного элемента
      «Обычная кнопка» */

    /** @param label надпись на кнопке */
    public static IConcept button(String label);

    /** @param label надпись на кнопке
      * @param actionParamValue значение параметра
      *        action, которое будет установлено при
      *        нажатии на кнопку
     */
    public static IConcept button(
```

```
    String label,
    String actionParamValue);

/* checkbox /
  Методы для формирования интерфейсного элемента
  «Пометка (Флажок)» */

/** @param label имя переключателя
  * @param checked признак, означающий будет ли
                  переключатель отмечен */
public static IConcept checkbox(
    String label,
    boolean checked,
    boolean disabled);

/* flash /
  Методы для формирования интерфейсного элемента
  «Флеш-скрипт» */

/** @param filename имя файла со скриптом
  * @param width ширина окна для скрипта
  * @param height высота окна для скрипта
  * @param flashVersion требуемая версия Flash
  * @param params дополнительные параметры для
                  скрипта */
public static IConcept flash(
    String filename,
    int width,
    int height,
```

```
    String flashVersion,
    Map<String, Object> params);

/* form /
  Методы для формирования интерфейсного элемента
  «Форма» */

/** @param label метка формы
  * @param elements элементы внутри формы */
public static IConcept form(
    String label,
    IConcept... elements);

/* goToPage /
  Методы для формирования интерфейсного элемента
  «Перенаправление на другую страницу» */

/** @param name   название страницы
  * @param params дополнительные параметры, которые
          передаются контроллеру этой страницы */
public static IConcept goToPage(
    String name,
    Map<String, Object> params);

/* iconButton /
  Методы для формирования интерфейсного элемента
  «Иконка-кнопка» */

/** @param klass class-параметр html-объекта для
```

```
            связи с CSS-скриптами
    * @param actionOnClick действие, вызываемое при
            нажатии на кнопку
    * @param params дополнительные параметры,
            Передаваемые в контроллер */
public static IConcept iconButton(
    String klass,
    String actionOnClick,
    Map<String, Object> params);

/* link /
  Методы для формирования интерфейсного элемента
  «Ссылка» */

/** @param label метка элемента
  * @param params дополнительные параметры,
            передаваемые при нажатии на ссылку */
public static IConcept link(
    String label, Map<String, Object> params);

/* list /
  Методы для формирования интерфейсного элемента
  «Маркированный список элементов» */

/** @param elements список вложенных элементов */
public static IConcept list(IConcept... elements);

/* listbox /
  Методы для формирования интерфейсного элемента
```

```
  «Список с множественным выбором» */

/** @param label метка элемента
  * @param values множество элементов списка */
public static IConcept listbox(
    String label, String... values);

/* radiobutton /
  Методы для формирования интерфейсного элемента
  «Радиокнопка» */

/**
 * Радиокнопка
 * @param label метка элемента
 * @param checked начальное состояние радиокнопки
 * @param comment подпись
 */
public static IConcept radiobutton(
    String label, boolean checked, String comment);

/* sec
  Методы для формирования интерфейсного элемента
  «Раздел» */

/** @param elements вложенные элементы */
public static IConcept sec(IConcept... elements);

/* select /
  Методы для формирования интерфейсного элемента
```

```
      «Выбор одного элемента из списка» */
    public static IConcept select(
        String label, String... values);

    /* text
      Методы для формирования интерфейсного элемента
      «Текст» */

    /** @param text текст надписи */
    public static IConcept text(String text);

    /* Декораторы */
    /**  Добавляет класс к элементу интерфейса */
    public static IConcept klass(
        String klass, IConcept element);

    /** Добавляет идентификатор к элементу интерфейса*/
    public static IConcept id(
        String id, IConcept element);

    /** Добавляет атрибут «ширина» к элементу */
    public static IConcept width(
        int widthInPixel, IConcept element);

    /** Добавляет атрибут «высота» к элементу */
    public static IConcept height(
        int heightInPixel,
        IConcept element);
}
```

Пример использования интерфейса:

```
resMsg.setInterface(
    form(
        "Калькулятор",
        sec(
            select("Действие",
                new String[]{
                    "сложить",
                    "вычесть",
                    "умножить",
                    "разделить"}),
            textfield("a", "0"),
            textfield("b", "0"),
            button("Вычислить")),
        sec(
            text("Результат: \"" + result + "\"")
        )));
```

(здесь resMsg — объект-обёртка результирующего сообщения, метод setInterface() используется для установления ссылки из этого сообщения на созданный интерфейс).

Данный код приведёт к созданию следующего интерфейса:

Рис. 74. Пример порождённого интерфейса

Приложение 4. Акт об использовании результатов кандидатской диссертационной работы

Федеральное государственное бюджетное учреждение науки
ИНСТИТУТ АВТОМАТИКИ И ПРОЦЕССОВ УПРАВЛЕНИЯ
Дальневосточного отделения Российской академии наук
(ИАПУ ДВО РАН)
Радио ул., д. 5, Владивосток, 690041
Тел./факс (4232) 310439, 310452 E-mail: director@iacp.dvo.ru http: www.iacp.dvo.ru
ОКПО 02698217, ОГРН 1022502127878 ИНН/КПП 2539007627/253901001

«УТВЕРЖДАЮ»
И.О. директора ИАПУ ДВО РАН,
член-корреспондент РАН А.А. Сорокин, д. ф.-м. н
«24» декабря 2013 г.

АКТ № 6 /71
от «17» декабря 2013 г.
об использовании результатов
кандидатской диссертационной работы

КРЫЛОВА ДМИТРИЯ АЛЕКСАНДРОВИЧА

Комиссия в составе:
председатель: В.В. Грибова, д.т.н., с.н.с., зав. лабораторией №71,
члены комиссии: А.С. Клещёв, д.ф.-м.н., профессор, г.н.с. лаб. №71,
Н.В. Киншт, д.т.н., профессор, г.н.с. лаб. технической диагностики
Е.А. Шалфеева, к.т.н., доцент, с.н.с. лаб. №71,

составили настоящий акт о том, что научные результаты диссертационной работы Крылова Д.А. «Модели и методы реализации облачной платформы для разработки и использования интеллектуальных сервисов», представляемой на соискание учёной степени кандидата технических наук, использованы в научных исследованиях лаборатории интеллектуальных систем, проводимых по следующим программам/проектам:

- облачная платформа для создания и использования интеллектуальных сервисов (РФФИ 13-07-00024-а)
- интеллектуальные многоагентные системы для управления распределенной обработкой онтологий, знаний и данных (РФФИ 10-07-00090-а),
- управление концептуальными метаонтологиями, онтологиями, знаниями и данными в интеллектуальных системах (РФФИ 10-07-00089-а),
- облачная платформа для разработки и использования пакетов прикладных программ и интеллектуальных систем (ДВО РАН 12-II-УО-01И-001, по интеграционному проекту с Уральским отделением РАН)
- программа №14 Президиума РАН "Интеллектуальные информационные технологии, математическое моделирование, системный анализ и автоматизация", проект «Развитие систем управления базами знаний с коллективным доступом» (ДВО РАН 09-I-П2-04)
- программа №15 фундаментальных исследований ОЭММПУ "Управление движением, теория сложных информационно-управляющих систем", проект «Модели мультиагентных систем для управления распределенной обработкой информации» (ДВО РАН 09-I-ОЭМПУ-02)

К настоящему времени на платформе IACPaaS выполнена реализация диссертационной работы Л.А. Федорищева «Модели, методы и инструментальные сервисы для создания профессиональных виртуальных облачных сред», в том числе: Графический редактор трёхмерных сцен, Интерпретатор виртуальных сред, Компьютерный обучающий тренажёр по офтальмологии, Виртуальная химическая лаборатория.

Кроме того, на платформе реализована иерархическая сетевая облачная база данных по сбору информации для диагностики силовых трансформаторов на подстанциях Дальневосточного региона.

Председатель комиссии _____ В.В. Грибова

Члены комиссии: _____ А.С. Клещёв

_____ Н.В. Киншт

_____ Е.А. Шалфеева