Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет информационных технологий и управления

Кафедра интеллектуальных информационных технологий

Д. В. Шункевич

СЕМАНТИЧЕСКИЕ ТЕХНОЛОГИИ ПРОЕКТИРОВАНИЯ РЕШАТЕЛЕЙ ЗАДАЧ

Допущено Министерством образования Республики Беларусь
в качестве учебного пособия
для студентов учреждений высшего образования
по специальности магистратуры «Искусственный интеллект»

Минск БГУИР 2022

УДК [004.423+004.832](075.8)
ББК 32.813я7
Ш96

Рецензенты:
кафедра информационных систем управления
Белорусского государственного университета (протокол №9 от 25.03.2021);

заведующий лабораторией идентификации систем государственного научного учреждения «Объединенный институт проблем информатики Национальной академии наук Беларуси» доктор технических наук,
профессор А. А. Дудкин

Шункевич, Д. В.

Ш96 Семантические технологии проектирования решателей задач : учеб. пособие / Д. В. Шункевич. – Минск : БГУИР, 2022. – 219 с. : ил.

ISBN 978-985-543-610-3.

Сформулированы основные положения, касающиеся теории и практики разработки гибридных решателей задач в рамках учебной дисциплины «Семантические технологии проектирования решателей задач» для II ступени высшего образования, приведен подробный теоретический материал по курсу, рекомендации по выполнению лабораторных работ. Может быть полезно всем, кто интересуется актуальными вопросами разработки интеллектуальных систем и решением задач в интеллектуальных системах.

УДК [004.423+004.832](075.8)
ББК 32.813я7

ISBN 978-985-543-610-3

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

GPS (General Problem Solver) – компьютерная программа, созданная в 1959 г. и предназначенная для работы в качестве универсальной машины для решения задач, сформулированных на языке хорновских дизъюнктов.

IACPaaS (Intelligent Applications, Control and Platform as a Service) – исследовательская облачная платформа, объединяющая различные модели парадигмы облачных вычислений.

IMS (Intelligent MetaSystem) – интеллектуальная метасистема поддержки проектирования интеллектуальных систем. База знаний метасистемы IMS содержит в формализованном виде всю документацию по технологии OSTIS, и, соответственно, большую часть материала, представленного в данном учебном пособии. Метасистема реализована в web-ориентированном варианте и доступна онлайн по адресу http://ims.ostis.net, что позволяет использовать ее в учебном процессе наряду с данным пособием.

OSTIS (Open Semantic Technology for Intelligent Systems) – открытая семантическая технология проектирования интеллектуальных систем.

OWL (Web Ontology Language) – язык описания онтологий для семантической паутины.

QA3 (Question Answer system ver. 3) – вопросно-ответная дедуктивная система, созданная в 1969 г. на языке LISP.

RDF (Resource Description Framework) – разработанная Консорциумом Всемирной паутины модель для представления данных.

SCg-код (Semantic Code graphic) – графический нелинейный вариант визуализации текстов SC-кода. Алфавит SCg-кода состоит из нескольких взаимосвязанных уровней, однако в рамках данного учебного пособия используются только следующие основные графические примитивы (таблица 1.1).

Таблица 1.1 – Алфавит sc.g-элементов

Sc.g-элемент	**Пояснение**
1	2
	Константный sc.g-узел, являющийся изображением некоторого sc-элемента
	Константный sc.g-узел, являющийся изображением *связки*
	Константный sc.g-узел, являющийся изображением *структуры*
	Константный sc.g-узел, являющийся изображением *ролевого отношения*
	Константный sc.g-узел, являющийся изображением *неролевого отношения*

1	2
	Константный sc.g-узел, являющийся изображением *класса*
	Константная sc.g-дуга общего вида. Является изображением ориентированной пары sc-элементов, например, являющейся элементом некоторого бинарного отношения
	Константная позитивная стационарная sc.g-дуга принадлежности (sc.g-дуга основного вида, базовая sc.g-дуга). Является изображением пары принадлежности элемента множеству

SCn-код (Semantic Code natural) – гипертекстовый вариант визуализации текстов SC-кода, приближенный к форматированному естественному языку. В рамках SCn-кода используются идентификаторы sc-элементов, а также специальные обозначения sc-коннекторов, из которых в данном учебном пособии встречаются следующие (таблица 1.2).

Таблица 1.2 – Алфавит sc.n-коннекторов

Sc.n-коннектор	**Пояснение**
=> <=	Изображение константной sc-дуги общего вида, обозначающей направленную бинарную связь между двумя сущностями
∈ ∋	Изображение sc-дуги основного вида, обозначающей принадлежность элемента множеству

SCP (Semantic Code Programming) – графовый процедурный язык программирования, построенный на базе SC-кода.

SC-код (Semantic Code) – универсальный базовый способ смыслового представления знаний в виде семантических сетей с базовой теоретико-множественной интерпретацией.

Sc-память (семантическая память) – абстрактная графодинамическая ассоциативная память, хранящая конструкции SC-кода. Может быть физически реализована в программном (на основе традиционной памяти) или аппаратном варианте.

Sc-коннектор – связка (ребро или дуга) семантической сети, являющейся текстом SC-кода. В зависимости от ориентации, обозначаемой sc-коннектором связи, подразделяется на sc-дуги и sc-ребра.

Sc-узел – узел семантической сети, являющейся текстом SC-кода, разновидность sc-элемента.

Sc-элемент – элемент SC-кода, т. е. элемент (узел или коннектор) семантической сети, являющейся текстом SC-кода.

SPARQL (SPARQL Protocol and RDF Query Language) – язык запросов к данным (является рекомендацией консорциума W3C и одной из технологий семантической паутины), представленным по модели RDF, а также протокол для передачи этих запросов и ответов на них.

SQL (Structured Query Language – «язык структурированных запросов») – универсальный язык запросов, применяемый для создания, модификации и управления данными в реляционных базах данных.

STRIPS (Stanford Research Institute Problem Solver) – планирующая система, использующая декларативно-процедуральное представление знаний в сочетании с эвристическим поиском, создана в 1971 г.

W3C (World Wide Web Consortium) – Консорциум Всемирной паутины, организация, разрабатывающая и внедряющая технологические стандарты для Всемирной паутины.

ИСС – интеллектуальная справочная система.

Мивар (Многомерная Информационная Варьирующаяся Адаптивная Реальность) – термин, используемый для обозначения наименьшего структурного элемента (точки) дискретного информационного пространства в рамках миварного подхода.

Миварный подход – математический аппарат для разработки систем искусственного интеллекта, созданный путем комплексирования продукционного подхода и сетей Петри.

НИЦЭВТ – Научно-исследовательский центр электронной вычислительной техники (г. Москва).

ППР (программа принятия решений) – планирующая система для интеллектуального робота, созданная в 1977 г. под руководством В. П. Гладуна.

ПРИЗ (пакет прикладных инженерных задач) – система программирования, созданная под руководством Э. Х. Тыугу в 1970–1976 гг.

Рецептор – компонент (необязательно аппаратный) интеллектуальной системы, позволяющий ей воспринимать воздействие со стороны внешей среды, например, датчик или компонент интерфейса для ввода текстового сообщения.

УСК – универсальный семантический код, разработанный В. В. Марты-новым.

Эффектор – компонент (необязательно аппаратный) интеллектуальной системы, позволяющий ей осуществлять воздействие на внешнюю среду, например, манипулятор робота или компонент интерфейса для вывода текстового сообщения.

ВВЕДЕНИЕ

Говоря об интеллектуальных системах, мы, как правило, сразу же подразумеваем способность таких систем решать различные нетривиальные задачи, а также обучаться, приобретая новые знания и, что очень важно, новые навыки. Важнейшее место в составе любой интеллектуальной системы занимает решатель задач, который определяет способность системы решать различные задачи. От того, насколько гибко построен решатель задач интеллектуальной системы, зависит ее способность приобретать новые навыки, т. е. способность обучаться решению задач новых классов.

На заре развития такого направления как искусственный интеллект ученые пытались найти универсальный алгоритм, который бы позволил решить любые задачи в любой предметной области. Так появились первые проекты решателей, такие как GPS (General Problem Solver) и ряд других. Однако достаточно быстро стало очевидно, что общие механизмы решения задач не укладываются в какой-либо достаточно простой универсальный алгоритм, а если и удастся его найти, то реализовать его на современном этапе развития вычислительной техники окажется невозможным.

Исследования в области разработки моделей решения задач разделились на большое количество направлений, в каждом из которых были получены серьезные научные и практические результаты. Появились различные логические модели, от классических дедуктивных, до нечетких и темпоральных, генетические алгоритмы, искусственные нейронные сети и многие другие модели, каждая из которых позволяет решать задачи определенного класса, часто далеко не тривиальные.

В настоящее время решатель задач интеллектуальной системы, как правило, строится как монолитный (с точки зрения интеграции различных моделей решения задач) блок, который проектируется под решение задач конкретного класса. В то же время существует потребность в гибридных системах, решатель которых интегрировал бы различные модели решения задач, дополняющих достоинства и возможности друг друга. Системы такого рода существуют, однако разработка и поддержка таких систем крайне трудоемка, что делает экономически нецелесообразным их применение во многих сферах человеческой деятельности, где их применение востребовано с практической точки зрения.

Ключевым фактором, препятствующим созданию систем, в которых могли бы свободно интегрироваться различные модели решения задач, является

фактор отсутствия совместимости различных моделей решения задач, т. е. отсутствия общей формальной основы, которая бы позволила реализовать модели решения задач так, чтобы они легко могли быть интегрированы в одной системе и при необходимости дополнены новыми моделями решения задач.

В рамках данного учебного пособия подробно рассматривается подход к разработке гибридных решателей задач, предложенный автором в рамках диссертационного исследования [1] и ряда других работ, в частности [2], и являющийся частью открытой технологии проектирования семантически совместимых гибридных интеллектуальных систем (технологии OSTIS).

В учебном пособии подробно описаны требования, предъявляемые к гибридным решателям задач, существующие подходы и проблемы в области их разработки, на примерах обосновывается актуальность их применения. Подробно рассматривается иерахическая модель гибридного решателя, принципы взаимодействия ее компонентов, средства описания процесса решения задач. Далее рассматриваются методика и средства проектирования решателей в соответствии с указанной моделью.

Значительная часть учебного пособия посвящена примерам разработки конкретных решателей задач для различных сфер с учетом описанной модели гибридного решателя и соответствующих методики и средств. В практической части приведено краткое руководство по выполнению лабораторных работ по дисциплине «Семантические технологии проектирования решателей задач».

Таким образом, учебное пособие содержит актуальную информацию и призвано в доступной форме с использованием большого числа примеров донести до магистрантов новые научные результаты в области разработки решателей задач интеллектуальных систем.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1 МОДЕЛИ ОБРАБОТКИ ЗНАНИЙ И АРХИТЕКТУРА РЕШАТЕЛЕЙ ЗАДАЧ

1.1 ОБЩИЕ ПОЛОЖЕНИЯ

1.1.1 Понятие гибридного решателя задач

В настоящее время все более актуальным становится использование интеллектуальных систем в самых различных областях. Одним из ключевых компонентов интеллектуальной системы, обеспечивающим возможность решать широкий круг задач, является ***решатель задач***. Особенностью решателей задач интеллектуальных систем по сравнению с другими современными программными системами является необходимость решать задачи в условиях, когда сведения, необходимые для решения, не локализованы явно в базе знаний интеллектуальной системы и должны быть найдены в процессе решения задачи на основании каких-либо критериев.

Состав решателя задач каждой конкретной системы зависит от ее назначения, классов решаемых задач, предметной области и ряда других факторов.

В общем случае решатель задач обеспечивает возможность решения задач, связанных как с непосредственно основной функциональностью системы, так и с обеспечением эффективности работы такой системы, а также с обеспечением автоматизации развития самой этой системы. Решатель задач, обеспечивающий выполнение всех перечисленных функций, будем называть ***объединенным решателем задач*** указанной интеллектуальной системы.

Будем считать, что объединенный решатель задач, в отличие от решателя задач вообще, в общем случае решает задачи, связанные:

– с обеспечением основного функционала системы (решение явно сформулированных задач по требованию);

– обеспечением корректности и оптимизацией работы системы (перманентно на протяжении жизненного цикла системы);

– обеспечением автоматизации развития интеллектуальной системы.

Расширение областей применения интеллектуальных систем требует от таких систем возможности решения комплексных задач при помощи совместного использования в каждой из них целого ряда различных моделей представления знаний и различных моделей решения. Кроме того, решение комплексных задач предполагает использование общих информационных ресурсов (в предельном случае – всей базы знаний интеллектуальной системы) различными компонентами решателя, ориентированными на решение различных подзадач. Поскольку решатель комплексных задач осуществляет

интеграцию различных моделей решения задач, будем называть его ***гибридным решателем задач***.

Примерами комплексных задач являются:

– задачи понимания текстов естественного языка (как печатных, так и рукописных), понимания речевых сообщений, изображений: в каждом из перечисленных случаев необходимо осуществить синтаксический анализ обрабатываемого файла (сигнала), устранить незначимые фрагменты, классифицировать значимые фрагменты, соотнести их с понятиями, известными системе, выявить те фрагменты, которые система распознать не в состоянии, устранить дублирование информации и т. д.;

– задачи автоматизации адаптивного обучения школьников и студентов, предполагающие, что система может самостоятельно решать различные задачи из некоторой предметной области, а также управлять процессом обучения, самостоятельно формировать задания для учащихся и контролировать их выполнение;

– задачи планирования поведения интеллектуальных роботов, предполагающие как понимание различного рода внешней информации, так и принятие различных решений с использованием достоверных и правдоподобных методов;

– задачи комплексной и гибкой автоматизации различных предприятий.

Использование различных моделей решения задач в рамках интеллектуальной системы предполагает декомпозицию комплексной задачи на подзадачи, которые могут быть решены с помощью одной из известных интеллектуальной системе моделей решения задач. Благодаря комбинации различных моделей решения задач, множество задач, решаемых гибридным решателем, будет значительно шире, чем объединение множеств задач, решаемых по отдельности всеми решателями задач, входящими в его состав [3].

Постоянная эволюция интеллектуальных систем и технологий их разработки делает актуальной не только проблему уменьшения сроков разработки гибридных решателей задач, но и проблему снижения трудоемкости внесения изменений в состав уже разработанных решателей без необходимости изменения архитектуры всей системы в целом.

Современные подходы к построению гибридных решателей задач, как правило, предполагают совмещение разнородных моделей решения задач без какой-либо единой основы, например, посредством специализированных программных интерфейсов между разными компонентами системы, что приводит к существенным накладным расходам при разработке такой системы

и в особенности при ее модификации, в том числе при добавлении в систему новой модели решения задач.

В случае интеллектуальной системы, ориентированной не только на взаимодействие с конечным пользователем, но и способной автономно работать в условиях, имеющих высокий уровень непредсказуемости (например, в космосе или других условиях, не пригодных для работы в них человека), в ней должны быть также предусмотрены возможности принятия решений в условиях неопределенности; анализа сигналов, поступающих с различных датчиков, в том числе средства анализа изображений; анализа последствий собственной деятельности с возможностью автоматической корректировки последующих актов этой деятельности; прогнозирования состояния окружающей среды на основе собранных данных и т. д.

Современная интеллектуальная система в общем случае имеет в своем составе следующие подсистемы [4]:

– предметная (основная) подсистема;

– интеллектуальный пользовательский интерфейс;

– интеллектуальная подсистема адаптивного управления диалогом с конечным пользователем;

– интеллектуальная help-система для информационного обслуживания и обучения конечных пользователей предметной подсистемы, которые, начиная работать с системой, не обязаны иметь сразу высокую квалификацию;

– интеллектуальная подсистема управления проектированием интеллектуальной системы, которая координирует деятельность разработчиков предметной подсистемы;

– интеллектуальная подсистема управления информационной безопасностью предметной подсистемы.

Задачей всех перечисленных компонентов является не только обеспечение основной функциональности системы, т. е. возможности решать задачи, сформулированные конечным пользователем, но и обеспечение работоспособности самой системы и ее эффективности. Задачи второго типа не всегда явно сформулированы и соответствуют процессам, перманентно выполняющимся на протяжении всего времени работы интеллектуальной системы. К такого рода процессам можно отнести, например, выявление и сборку информационного мусора, оптимизацию базы знаний и т. д. Выполнение некоторых из указанных процессов необходимо для функционирования самой системы и с их прекращением работа системы также прекратится. В особенности это важно в случае автономной системы, для которой важнейшими задачами являются поиск источников энергии, избегание

опасных ситуаций во внешней среде и т. д. Таким образом, говоря в рамках данного учебного пособия о решении задач, будем подразумевать не только решение задач, инициированных пользователем, но и задач других классов, рассмотренных выше. При этом предполагается, что каждая задача может представлять собой как процедурную спецификацию соответствующего действия, так и декларативную, или смешанную. Решение перечисленных задач предполагает необходимость использования в рамках одной и той же системы самых разнообразных моделей решения задач, при этом при решении одной и той же задачи могут одновременно использоваться несколько ***моделей решения задач***.

Каждая перечисленная подсистема, в свою очередь, состоит из собственной базы знаний, решателя задач и пользовательского интерфейса.

Приведенный список подсистем отражает необходимость использования в рамках одной и той же системы самых разнообразных моделей решения задач, при этом при решении одной и той же задачи могут одновременно использоваться несколько моделей решения задач.

1.1.2 Многообразие моделей решения задач

Существующее многообразие подходов к решению задач в компьютерных системах можно разделить на два класса: решение задач с использованием хранимых программ и решение в условиях, когда программа решения неизвестна.

Решение задач с использованием хранимых программ. В данном случае предполагается, что в системе заранее присутствует программа решения задачи заданного класса и решение сводится к поиску такой программы и выполнению ее на заданных входных данных. К системам, ориентированным на такой подход в решении задач, относятся в том числе системы, использующие:

– программы, написанные на языках программирования, относящихся как к императивной, так и декларативной парадигме, в том числе на логических и функциональных языках [5];

– реализацию генетических алгоритмов [6, 7];

– нейросетевые модели обработки знаний [8–10].

Следует отметить, что даже в случае использования хранимой программы решение задачи далеко не всегда тривиально, поскольку, во-первых, требуется найти такую хранимую программу на основе некоторой спецификации, во-вторых, обеспечить ее интерпретацию.

Решение задач в условиях, когда программа решения неизвестна. В этом случае предполагается, что в системе необязательно присутствует готовая программа решения для класса задач, которому принадлежит некоторая сформулированная задача, подлежащая решению. В связи с этим необходимо применять дополнительные методы поиска путей решения задачи, не рассчитанные на какой-либо узкий класс задач (например, разбиение задачи на подзадачи, методы поиска решений в глубину и ширину, метод случайного поиска решения и метод проб и ошибок, метод деления пополам и др.), а также различные модели логического вывода (классические дедуктивные [11], индуктивные [12, 13], абдуктивные [11]; модели, основанные на нечетких логиках [14–16], логике умолчаний [17], темпоральной логике [18] и многие другие).

Подробный обзор решателей задач, разработанных в период до 1982 г., таких как GPS, STRIPS, QA3, ПРИЗ [19], ППР, приведен в [20]. Среди современных работ, исследующих вопросы применения моделей решения задач, не ориентированных на конкретную предметную область, можно выделить [21].

Среди наиболее заметных представителей класса интеллектуальных решателей задач, разработанных в более поздний период, можно отметить:

1. Компьютерный решатель математических задач. Программный комплекс разработан А. С. Подколзиным и способен решать задачи уровня конкурсных экзаменов по математике для поступающих в вузы. Система работает с символьной записью задач и ориентируется на использование для решения задачи операций, схожих с теми, что использует человек. Решение можно оформить в протокол, пригодный для подачи прямо в соответствующую экзаменационную комиссию. По словам авторов, решатель справляется с 90 % задач из указанной предметной области, что соответствует хорошей оценке на экзаменах. Автором системы разработан специальный язык (и компилятор для него), описывающий приемы решения и последовательность их применения, систему распознавания проблемной ситуации, определяющую целесообразность применения тех или иных приемов [22].

2. Решатель задач по планиметрии НИЦЭВТ. В московском Научно-исследовательском центре электронной вычислительной техники активно ведется разработка средств автоматического решения задач по планиметрии, что отражается регулярными публикациями по данной теме, например [23]. Разработанные средства способны не только решить поставленную задачу (получить ответ), но и объяснить ход решения, а также отобразить условие

задачи и ход решения на чертеже. Для описания условий задачи и способов их решения используется онтологический подход.

3. Системы компьютерной алгебры. В настоящее время на рынке наиболее популярны такие системы компьютерной алгебры, как Maple, MathCAD, а также семейство продуктов компании Wolfram Research. Кроме указанных программных комплексов интерес представляет онлайн-сервис Wolpfram Alpha [24], также построенный на основе разработок компании Wolfram Research. Указанные программные комплексы обладают мощной функциональностью как для проведения различного рода вычислений и экспериментов, так и для построения на их основе систем различного назначения, например обучающих. Подробно возможности применения систем данного семейства рассмотрены, например, в [25].

4. Программный комплекс «УДАВ». В программном комплексе «УДАВ» реализован метод логического вывода, который базируется на миварной логической сети правил и представляет, согласно работам авторов, возможность активного обучения логического вывода, управляемого потоком данных, со снижением вычислительной сложности с N! (факториал) до линейной. «УДАВ» работает со знаниями, представленными в виде продукционных правил и процедур. Однако при всех перечисленных достоинствах указанного метода он представляет собой один из возможных методов решения задач определенных классов и сам по себе не решает проблему интеграции различных моделей решения задач в рамках одной интеллектуальной системы [26].

Однако при всем многообразии решаемых рассмотренными системами задач множество классов задач ограничивается имеющимся в системе набором жестко заданных приемов и алгоритмов решения задач, явно используемых при решении той или иной задачи.

Рассмотрим более подробно один из примеров комплексных задач, перечисленных выше, связанный с автоматизацией производства.

В рамках системы комплексной автоматизации производства условно можно выделить следующие уровни автоматизации:

– автоматизация собственно процесса производства продукта на всех этапах, от получения и оценки сырья или исходного материала до фасовки и доставки товара конечному потребителю;

– автоматизация управления процессом производства, т. е. автоматизация внесения изменений в процесс производства, например, изменение объемов партии, номенклатуры или свойств изготовляемого продукта и т. д.;

– автоматизация контроля выполнения процесса производства, предполагающая использование различных способов анализа текущей ситуации, а также механизмы выявления, классификации и устранения нештатных ситуаций вплоть до полного устранения нештатной ситуации без вмешательства оператора.

На рисунке 1.1 изображена условная схема системы автоматизации процесса производства, показывающая, применение каких из перечисленных выше моделей решения задач актуально в каждой из подсистем такой системы.

Приведенный пример показывает, что построение такой системы комплексной автоматизации практически невозможно без обеспечения согласованного использования различных видов знаний и моделей решения задач в рамках одной системы при решении одной и той же комплексной задачи. Кроме того, становится актуальной задача поддержки такой системы в состоянии, соответствующем текущему уровню развития технологий производства, дополнения ее более совершенными моделями и методами решения задач. При этом очевидно, что подобная реконфигурация системы должна осуществляться непосредственно в процессе эксплуатации системы, а не требовать каждый раз полной остановки всего производства или отдельных его частей.

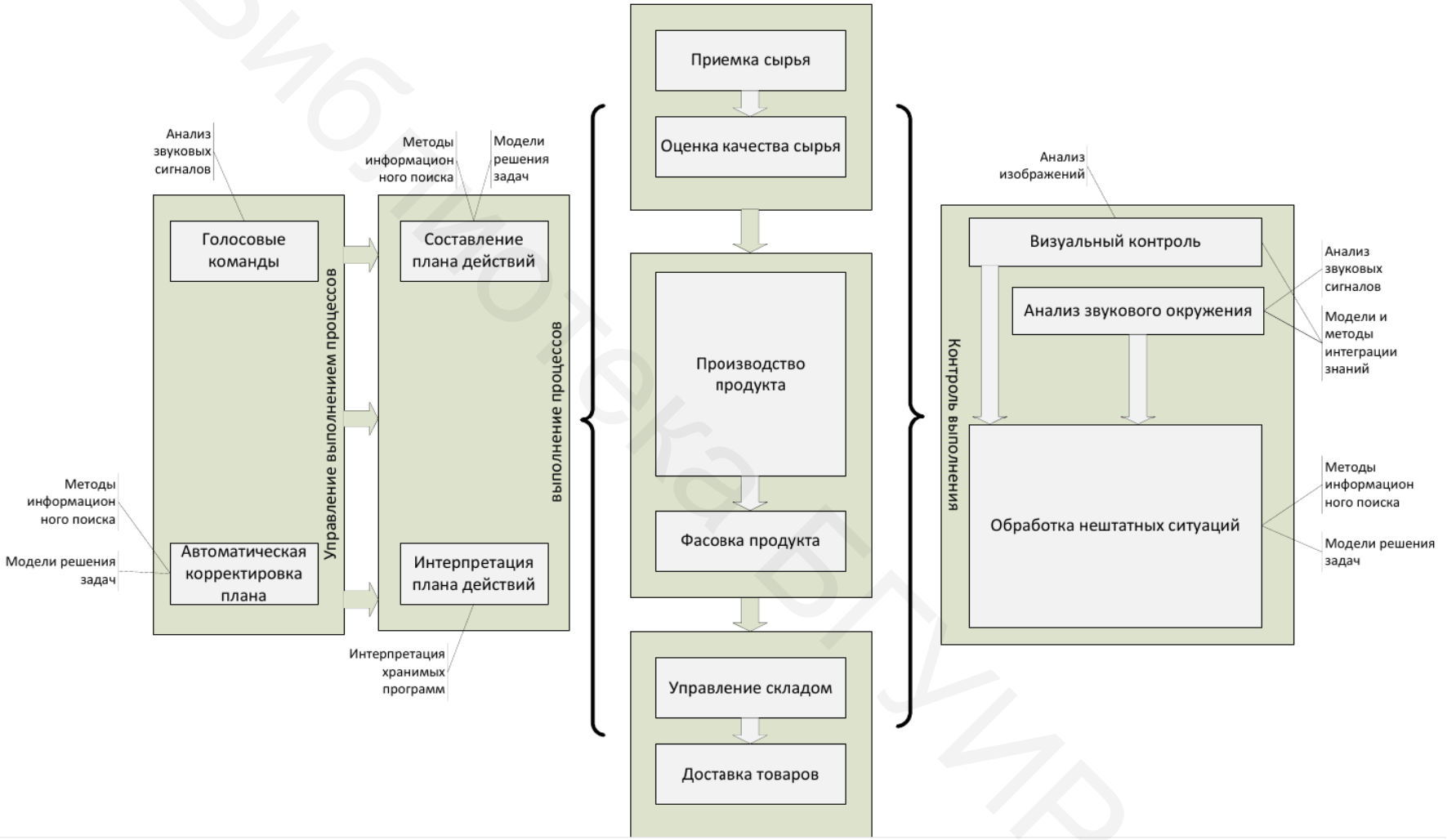

Рисунок 1.1 – Упрощенная схема системы комплексной автоматизации производства

На рисунке 1.2 более детально показана условная схема подсистемы обработки нештатных ситуаций.

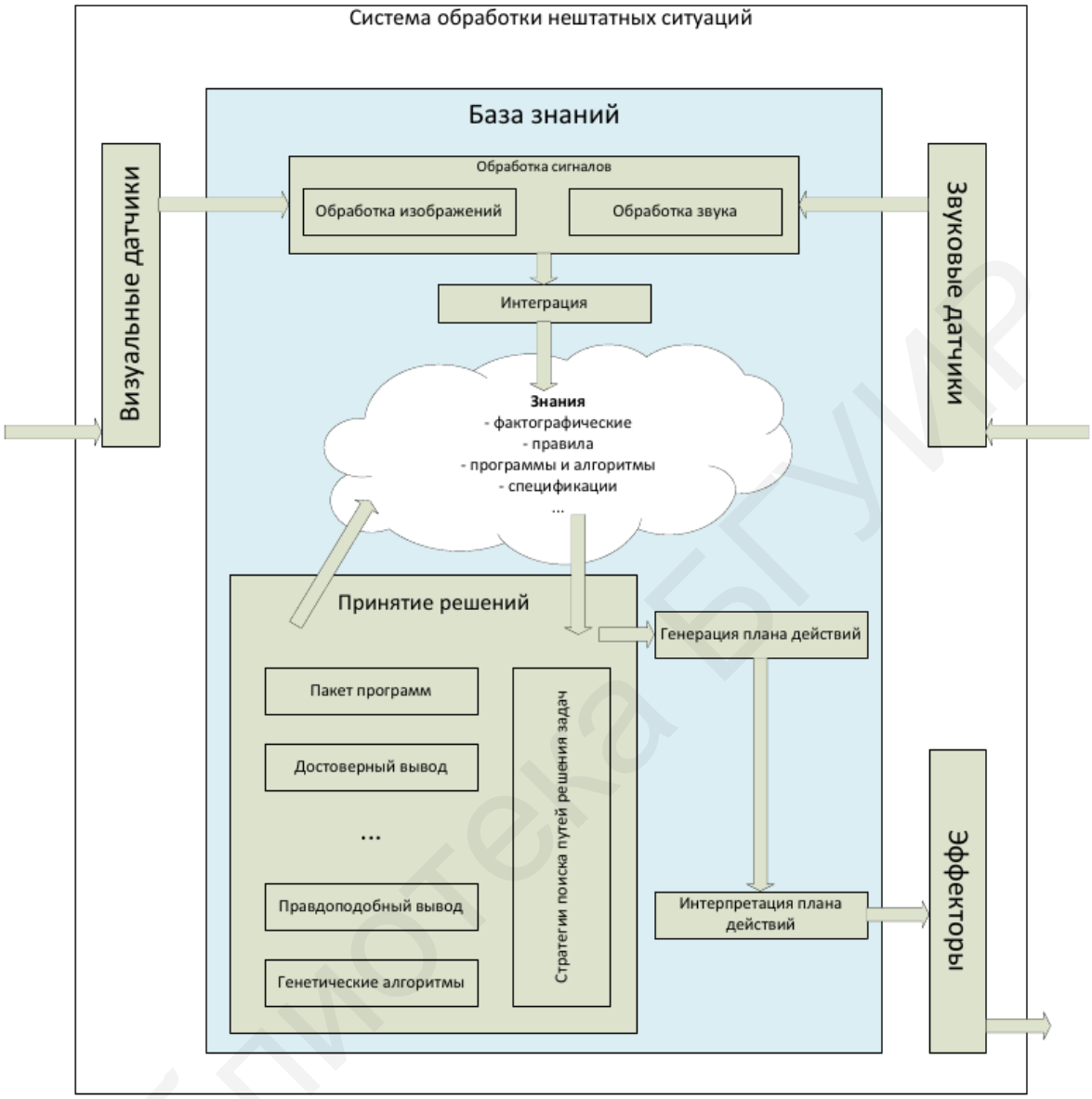

Рисунок 1.2 – Упрощенная схема подсистемы обработки нештатных ситуаций

Актуальность проблемы интеграции результатов, полученных в рамках различных научных школ и направлений в области искусственного интеллекта, в том числе обработки знаний и решения задач, подтверждается, например, работой [27], а также наличием ежегодных международных конференций Artificial General Intelligence [28], одной из целей которых является решение указанной проблемы.

1.1.3 Требования, предъявляемые к решателям задач

Рассмотренные модели и методы решения задач позволяют сформулировать требования к решателю задач интеллектуальной системы, способной решать комплексные задачи (гибридному решателю задач):

1. В каждый момент времени решатель должен обеспечивать решение задач из оговоренного класса за оговоренное время, при этом результат решения задачи должен удовлетворять некоторым известным требованиям. Другими словами, как и в случае современных компьютерных систем, корректность результатов решения задач на этапе разработки системы должна верифицироваться специальными методами, для этого могут быть использованы в том числе такие современные подходы, как unit-тестирование, тестирование методом «черного ящика» и др. [29].

2. Гибридный решатель должен обеспечивать возможность согласованного использования различных моделей решения задач при решении одной и той же комплексной задачи в случае необходимости.

3. Решатель должен быть легко **модифицируемым**, т. е. трудоемкость внесения изменений в уже разработанный решатель должна быть минимальна. Путями повышения модифицируемости решателя являются: обеспечение локальности вносимых изменений, в том числе за счет стратификации решателя на независимые уровни, обеспечение максимальной независимости компонентов решателя друг от друга, наличие готовых компонентов, которые могут быть встроены в решатель при необходимости. При этом внесение изменений должно осуществляться непосредственно в процессе эксплуатации системы.

4. Для того чтобы интеллектуальная система имела возможность анализировать и оптимизировать имеющийся решатель задач, интегрировать в его состав новые компоненты (в том числе самостоятельно), оценивать важность тех или иных компонентов и применимость их для решения той или иной задачи, спецификация решателя должна быть описана языком, понятным системе, например, при помощи тех же средств, что и обрабатываемые знания. Возможность интеллектуальной системы анализировать (верифицировать, корректировать, оптимизировать) собственные компоненты будем называть **рефлексивностью**.

5. Дополнительным требованием, предъявляемым к объединенному решателю задач по отношению к решателю задач вообще, является его **полнота** (целостность, комплексность), т. е. такой решатель должен обеспечивать всю функциональность системы, обеспечивать решение всех задач, как связанных с непосредственным назначением системы, так и обеспечивающих

эффективность ее работы. Таким образом, например, решатель, реализующий какой-либо вариант логического вывода, не может считаться объединенным, поскольку для использования системы, содержащей такой решатель, необходимо иметь хотя бы базовые средства информационного поиска, позволяющие локализовать полученный ответ, а также средства, обеспечивающие трансляцию вопроса от пользователя в систему и ответа от системы пользователю.

В свою очередь, требования, предъявляемые к технологиям разработки таких решателей, следующие:

– **комплексность и совместимость**, т. е. возможность реализовать с помощью такой технологии различные модели решения задач в унифицированном виде, позволяющем обеспечить их совместимость в рамках одного гибридного решателя при решении комплексных задач;

– **снижение сроков и трудоемкости построения решателей** за счет:

- обеспечения согласованности деятельности коллектива разработчиков;
- возможности использования готовых компонентов различной сложности и наличия библиотек таких компонентов, включающих средства спецификации компонентов и поиска компонентов на основе спецификации;
- автоматизации деятельности разработчиков;
- возможности выделения в рамках разрабатываемого решателя достаточно независимых компонентов, разработка которых может вестись отдельно друг от друга при условии, что трудоемкость согласования разработанных таким образом компонентов будет минимальной;
- наличия в технологии средств информационной поддержки разработчиков, в том числе средств обучения квалификации;

– обеспечение **модифицируемости** решателей задач, разработанных с использованием данной технологии, непосредственно в процессе эксплуатации системы, т. е. удовлетворение второго из предъявленных требований;

– быстрая **эволюционируемость самой технологии**, предполагающая в том числе наличие средств автоматизации приведения в соответствие текущему состоянию технологии уже разработанных с ее использованием решателей;

– **снижение требований к разработчикам** решателей задач, которое обеспечивается рассмотренными выше средствами снижения сроков и трудоемкости построения решателей, в частности, наличием развитых средств информационной поддержки разработчиков.

1.1.4 Проблемы в области построения гибридных решателей задач

Рассмотрим проблемы, специфичные для построения гибридных решателей задач, обусловленные их спецификой.

Несмотря на то что в настоящее время существует большое число моделей решения задач, многие из которых реализованы и успешно используются на практике в различных системах, остается актуальной проблема низкой согласованности принципов, лежащих в основе реализации таких моделей, и отсутствия единой унифицированной основы для реализации и интеграции различных моделей, что приводит к тому, что:

– затруднена возможность одновременного использования различных моделей решения задач в рамках одной системы при решении одной и той же комплексной задачи; практически невозможно комбинировать различные модели с целью решения задачи, для которой априори отсутствует алгоритм ее решения;

– практически невозможно использовать технические решения, реализованные в одной системе, в других системах, т. е. возможности использования компонентного подхода при построении решателей задач сильно ограничены. Как следствие, велико количество дублирований аналогичных решений в разных системах;

– фактически отсутствуют комплексные методики и средства построения решателей задач, которые бы обеспечивали возможность проектирования, реализации и отладки решателей различного вида.

Следствиями указанных проблем являются:

– высокая трудоемкость разработки каждого решателя, увеличение сроков их разработки, а значит, и увеличение затрат на разработку и поддержку соответствующих интеллектуальных систем;

– высокая трудоемкость внесения изменений в уже разработанные решатели, т. е. отсутствует или сильно затруднена возможность дополнения уже разработанного решателя новыми компонентами и внесения изменений в уже существующие компоненты в процессе эксплуатации системы. Таким образом, высока трудоемкость поддержки разработанных решателей;

– высокий уровень профессиональных требований к разработчикам решателей задач, что обусловлено, в частности:

- высокой сложностью существующих формализмов в области решения задач, рассчитанных на их интерпретацию компьютерной системой, а не человеком;

- отсутствием возможности рассматривать разрабатываемые решатели на разных уровнях детализации, выделения на каждом уровне достаточно независимых компонентов, что затрудняет процесс проектирования, тестирования и отладки таких решателей, а также снижает эффективность попыток объединения разработчиков решателей в коллективы по причине увеличения накладных расходов на согласование их деятельности;
- низким уровнем информационной поддержки разработчиков и автоматизации их деятельности.

Указанные проблемы решаются в рамках подхода, предлагаемого в рамках технологии OSTIS и рассмотренного в пункте 1.1.6.

1.1.5 Подходы к разработке решателей задач

Можно выделить два основных исторически сложившихся подхода к разработке решателей задач интеллектуальных систем.

Первый подход предполагает наличие в системе фиксированного решателя (например, машины логического вывода), к которому впоследствии добавляется база знаний. Наполнение базы определяется предметной областью, в которой должна работать система. Такие системы получили название «пустых» экспертных систем [30] или «оболочек» (expert system shells, [31]). Данный подход, как правило, использовался для разработки относительно несложных систем и в настоящее время не имеет широкого применения.

Второй подход, широко используемый в настоящее время, предполагает наличие программных средств доступа, совместимых с различными популярными языками программирования, к информации, хранящейся в некоторой базе. Данный подход широко используется, например, в системах, основанных на стандартах W3C [32]. Структура решателя, построенного на базе таких средств, определяется разработчиком в каждом конкретном случае и не фиксируется какими-либо стандартами. Такой подход обладает большей гибкостью, но отсутствие унификации в структуре и процессе разработки решателей приводит к несовместимости компонентов решателей, созданных разными разработчиками, большому количеству дублирований одних и тех же решений, повышению накладных расходов в процессе разработки и поддержки решателя.

Достаточно сложно выделить конкретные примеры широко используемых решателей, ориентированных на семантическое представление обрабатываемых знаний на основе стандартов W3C, поскольку отсутствуют общие принципы разработки таких решателей, и в каждом конкретном случае, как правило, разрабатывается свой частный решатель. В связи с этим

целесообразно перейти сразу к рассмотрению подходов к обработке знаний, связанных с таким представлением, выделению их достоинств и недостатков.

В настоящее время широко используется стандарт OWL2 [33] для семантического представления знаний и стандарт RDF [34] для представления знаний в виде семантических сетей. Непосредственно для хранения так называемых RDF-троек (RDF-триплетов) используются дополнительно специфицируемые форматы, например, RDF/XML, RDF/JSON, N3 и др. Существует большое количество эффективных хранилищ, обеспечивающих хранение и доступ к данным в форматах RDF.

Для осуществления доступа к данным, представленным в модели RDF, используется протокол и одноименный язык SPARQL [35], который во многом схож с языком SQL. Следующим шагом по отношению к языку SPARQL можно считать декларативный язык запросов Cypher, разработанный создателями хранилища Neo4j [36]. В запросе на языке Cypher может задаваться граф-образец (the graph pattern to match), на основе которого будет осуществляться изоморфный поиск в хранилище.

Рассмотренные языки SPARQL и Cypher обеспечивают исключительно доступ к хранимым знаниям, непосредственно обработка осуществляется уже на уровне приложения, работающего с хранилищем знаний. Кроме того, язык SPARQL не предполагает наличия средств редактирования, уже имеющихся в системе знаний.

Существует большое количество реализаций так называемых ризонеров (semantic reasoners), осуществляющих логический вывод на онтологиях, представленных в формате OWL2, а также средств разработки и редактирования таких онтологий. Полный список таких средств, признанных консорциумом W3C, можно найти на сайте www.w3.org [37]. Как видно из приведенной на нем таблицы, подавляющее большинство средств способно осуществлять только прямой логический вывод на основе отношений, описанных в онтологии.

Можно сделать вывод о том, что в настоящее время в рамках консорциума разработаны эффективные средства представления знаний, доступа к ним и механизмы логического вывода на онтологиях, представленных в таком виде. Однако отсутствуют общие принципы построения решателей задач, ориентированных на такое представление, что порождает большое количество дублирований и значительно повышает трудоемкость разработки основанных на таком представлении приложений.

Среди комплексных подходов к построению решателей задач, разрабатываемых на постсоветском пространстве, можно выделить проект

IACPaaS [38, 39], активно развивающийся в настоящее время. Целью данного проекта является разработка облачной платформы для построения на ее основе интеллектуальных сервисов различного назначения. В данном проекте активно используются библиотеки многократно используемых компонентов интеллектуальных систем. Конкретно для построения решателей задач, а также пользовательских интерфейсов таких систем используется многоагентный подход. В рамках указанного проекта осуществляется предоставление пользователю большого числа разнородных сервисов, выбор которых осуществляется самим пользователем. Используемый в рамках данного проекта подход включает общую формальную основу для интеграции различных моделей решения задач с целью их комбинирования при решении одной и той же комплексной задачи. Так, в проекте IACPaaS осуществляется построение отдельных решателей, ориентированных на решение конкретных классов задач, и предполагается возможность последующего их использования клиентами. Однако данный проект не ставит целью решение таких проблем, как разработка комплексной модели решателя, удовлетворяющей изложенным в пункте 1.1.3 требованиям, унификация различных подходов к решению задач на некоторой общей формальной основе, унификация деятельности разработчиков решателей и разработка единых принципов выделения многократно используемых компонентов решателей.

Комплексные подходы к построению интеллектуальных систем различных классов активно развивались и развиваются в Новосибирске Ю. А. Загорулько и коллегами. Ряд работ по данному направлению был опубликован еще в конце 1980-х гг., например, [40, 41], но имеются и современные разработки авторов в сфере проектирования интеллектуальных систем [42, 43]. В частности, в работе [43] высказывается тезис о том, что в процессе реализации интеллектуальных систем различных классов большую помощь разработчикам может оказать репозитарий – библиотека готовых к использованию методов принятия решений вместе с методикой и средствами их исполнения и композиции. Кроме этого, в работе [44] формулируется тезис о необходимости интеграции различных методов поддержки принятия решений для решения сложных задач.

Задачи интеграции различных подходов, в том числе связанных с решением задач, исследуются также в работе [45] и других работах тех же авторов. В данной работе предлагается подход к построению единой платформы проведения интеллектуального анализа данных, включающей в себя объектно-ориентированные, а также нечеткие модели и программные

инструментальные средства работы с ними, с использованием онтологического системного анализа.

Компонентному проектированию интеллектуальных систем, основанных на знаниях, посвящена работа [46], в которой обосновывается необходимость накопления и повторного использования различных компонентов интеллектуальных систем, предлагаются возможные решения данной проблемы с использованием онтологий.

Состояние работ англоязычных авторов, посвященных вопросам решения задач в системах, основанных на знаниях и актуальных на момент начала 1990-х гг., отражено в обзорных публикациях [47, 48]. Более поздние англоязычные работы в данной области в основном ориентированы на решение конкретных частных задач в системах, построенных на основе стандартов W3C.

Таким образом, можно сказать, что существует ряд конкретных разработок в направлении построения комплексных технологий разработки интеллектуальных систем различных классов, в том числе с использованием библиотек многократно применяемых компонентов, однако сформулированные выше проблемы не решались или в полной мере не решены ни в одном из этих подходов. Во многом это обусловлено отсутствием унифицированной формальной основы для представления любых видов знаний, в том числе различного рода программ, отсутствием строгих принципов, регламентирующих процесс построения решателей задач, а также средств поддержки разработчиков таких решателей и их компонентов.

1.1.6 Подход к разработке решателей, предлагаемый в рамках технологии OSTIS

Предлагаемые в рамках данного учебного пособия модели, методика и средства являются частью открытой семантической технологии проектирования интеллектуальных систем OSTIS. В качестве формальной основы для представления знаний в рамках данной технологии используется унифицированная семантическая сеть с теоретико-множественной интерпретацией. Такая модель представления названа SC-кодом (Semantic Computer code) и предложена в работе [49]. Элементы такой семантической сети названы sc-узлами и sc-коннекторами (sc-дугами, sc-ребрами). Модель какой-либо сущности, описанную средствами SC-кода, будем называть семантической моделью указанной сущности или просто sc-моделью.

Такого рода модель имеет два уровня:

– синтаксический, на котором выделяются классы sc-элементов с точки зрения синтаксиса (sc-узел, sc-ребро, sc-дуга общего вида, sc-дуга основного

вида) и два вида отношения инцидентности между ними (инцидентность sc-ребра sc-узлу и инцидентность конца sc-дуги sc-узлу);

– семантический, на котором выделяются классы sc-элементов с точки зрения вида обозначаемых ими сущностей (знаки множеств, знаки терминальных сущностей, знаки файлов и т. д. [50]) и далее уточняется семантика всех описываемых сущностей с использованием теоретико-множественного формализма.

Технология OSTIS ориентирована на разработку класса систем, которые названы компьютерными системами, управляемыми знаниями [51]. Компьютерные системы такого класса, разрабатываемые по технологии OSTIS, названы ostis-системами.

Каждая ostis-система состоит из независящей от платформы унифицированной логико-семантической модели этой системы (sc-модели компьютерной системы) и платформы интерпретации таких моделей. В свою очередь, каждая sc-модель компьютерной системы может быть декомпозирована на sc-модель базы знаний, sc-модель решателя задач, sc-модель интерфейса и абстрактную sc-память, в которой хранятся конструкции SC-кода (рисунок 1.3). Таким образом, в рамках данного учебого пособия будем говорить о построении sc-моделей объединенных решателей задач, обладающих свойством гибридности.

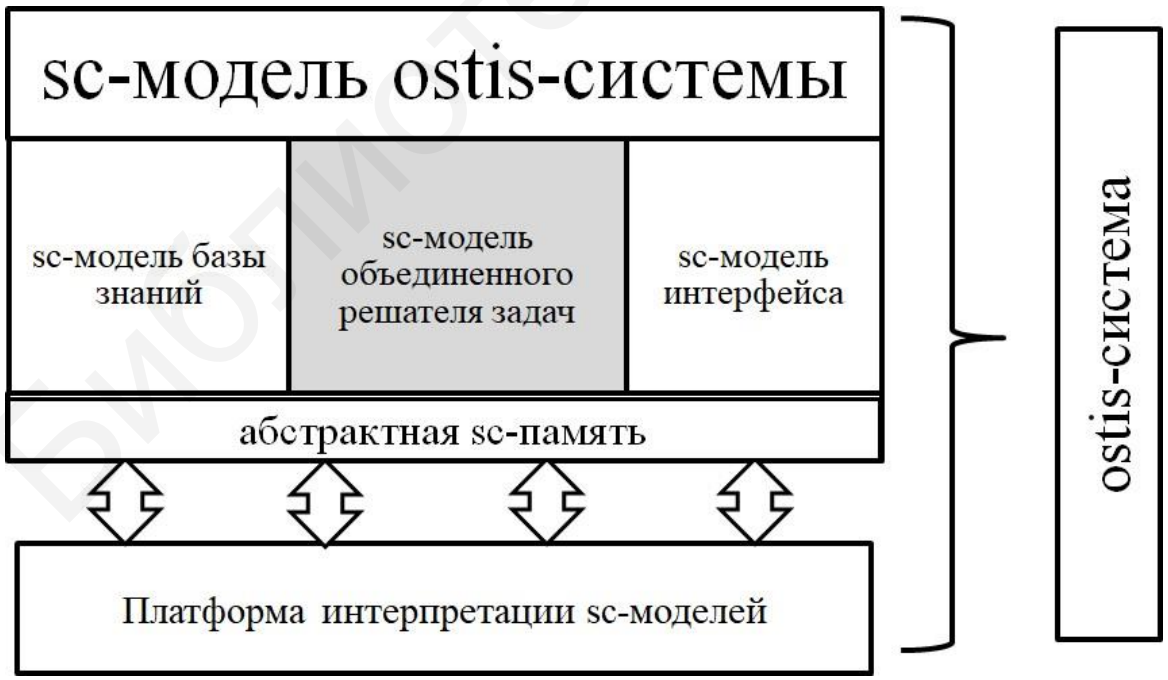

Рисунок 1.3 – Архитектура ostis-системы

При построении решателей задач используются следующие достоинства и особенности технологии OSTIS и SC-кода:

1. SC-код ориентирован на смысловое представление знаний, что позволяет обобщить модели решения задач и таким образом существенно уменьшить многообразие различных моделей, которое во многом обусловлено формами представления тех или иных видов знаний, а не их смыслом.

2. SC-код, в отличие от других широко используемых языков представления знаний, позволяет представлять в унифицированном виде любые виды знаний, в том числе логические утверждения и программы. Данный факт также позволяет унифицировать обработку знаний, представленных в таком виде, и обеспечить на основе такого формализма интеграцию различных моделей решения задач.

3. Ассоциативность и структурная перестраиваемость sc-памяти, в которой хранятся конструкции SC-кода, позволяет обеспечить модифицируемость и совместимость моделей решения задач, представленных на его основе, в частности, ассоциативность такой памяти обеспечивает возможность универсализации различных моделей решения задач.

4. Разработка предлагаемых моделей, методики и средств в рамках единой технологии позволяет задействовать другие решения, разработанные в рамках технологии, что существенно уменьшает объем специализированных решений, необходимых для реализации предлагаемого подхода, а также повышает эффективность использования предлагаемых решений при построении на их основе реальных прикладных систем. К числу таких заимствованных решений можно отнести:

- графовый процедурный язык SCP, тексты программ которого также записываются с использованием SC-кода и который будет использоваться в качестве базового языка программирования в рамках предлагаемого подхода;
- модели структуризации базы знаний и модели представления различных видов знаний, построенные на основе SC-кода, представленные в работе [52], использование указанных моделей позволит, кроме прочего, обобщить различные модели решения задач;
- модель коллективного проектирования баз знаний, представленная в работе [53], которая может быть использована для проектирования программ обработки знаний, спецификаций компонентов решателей и любых других видов знаний, имеющих отношение к построению решателей задач;
- средства автоматического редактирования и верификации различных видов знаний, также представленные в работе [53];

- вариант реализации платформы интерпретации sc-моделей компьютерных систем, рассмотренный в работе [54].

5. Рассматриваемая технология OSTIS реализуется в виде интеллектуальной метасистемы поддержки проектирования ostis-систем IMS [55, 56]. Все рассматриваемые в рамках курса модели и средства включены в состав соответствующих разделов базы знаний метасистемы, что, во-первых, обеспечивает формальную строгость полученных результатов, а во-вторых, автоматически делает их доступными широкому кругу разработчиков.

Для реализации комплекса моделей, методики и средств построения и модификации гибридных решателей задач, удовлетворяющих предъявленным выше требованиям, следует использовать перечисленные далее принципы:

1. В качестве основы для построения модели гибридного решателя задач предлагается использовать многоагентный подход. Данный подход позволяет обеспечить основу для построения параллельных асинхронных систем, имеющих распределенную архитектуру, повысить модифицируемость и производительность разработанных решателей.

2. Процесс решения любой задачи предлагается декомпозировать на логически атомарные действия, что также позволит обеспечить совместимость и модифицируемость решателей.

3. Решатель предлагается рассматривать как иерархическую систему, состоящую из нескольких взаимосвязанных уровней. Такой подход позволяет обеспечить возможность проектирования, отладки и верификации компонентов на разных уровнях независимо от других уровней, что существенно упрощает задачу построения решателя за счет снижения накладных расходов. Перечисленные принципы реализуются в виде ***модели гибридного решателя задач***.

4. Предлагается записывать всю информацию о решателе и решаемых им задачах при помощи SC-кода в той же базе знаний, что и собственно предметные знания системы. В общем случае такая информация включает:

- спецификацию агентов решателя, включая полные тексты программ агентов (например, на языке SCP);
- спецификацию всех информационных процессов, выполняемых агентами в семантической памяти, в том числе конструкции, обеспечивающие синхронизацию выполнения параллельных процессов;
- спецификацию всех задач, на решение которых направлены указанные информационные процессы.

Описание всей указанной информации в единой семантической памяти позволит, во-первых, обеспечить независимость разрабатываемых решателей от

платформы интерпретации sc-моделей, во-вторых, обеспечить возможность системы анализировать происходящие в ней процессы, оптимизировать и синхронизировать их выполнение, т. е. обеспечить **рефлексивность** проектируемых интеллектуальных систем. Данный принцип реализуется в виде ***модели взаимодействия информационных процессов в семантической памяти***.

5. При проектировании решателя как иерархической системы на каждом из уровней используется компонентный подход, что позволяет существенно снизить сроки разработки и повысить надежность решателей за счет использования ранее разработанных и отлаженных компонентов. Для реализации такого подхода в рамках метасистемы IMS разработана библиотека компонентов решателей различного уровня, а также ***методика построения и модификации решателей***, учитывающая наличие такой библиотеки.

6. ***Средства автоматизации и информационной поддержки разработчиков решателей*** строятся с использованием технологии OSTIS, т. е. в том числе с применением рассмотренных выше моделей, методик и средств. Такой подход обеспечивает высокие темпы развития указанных средств, а также существенно повышает эффективность средств информационной поддержки, т. к. позволяет строить указанные средства как часть интеллектуальной метасистемы IMS, т. е. как своего рода интеллектуальную подсистему.

1.2 СЕМАНТИЧЕСКАЯ ПАМЯТЬ

1.2.1 Понятие семантической памяти

Говоря в рамках данного учебного пособия о семантической памяти, будем иметь в виду рассмотренную ранее sc-память. Подробнее модель такой памяти (абстрактная sc-память) рассматривается в статье [55], описание программной реализации такой модели (sc-хранилище) приводится в работе [54].

Общий принцип организации обработки информации, хранимой в семантической памяти ostis-системы, представлен на рисунке 1.4.

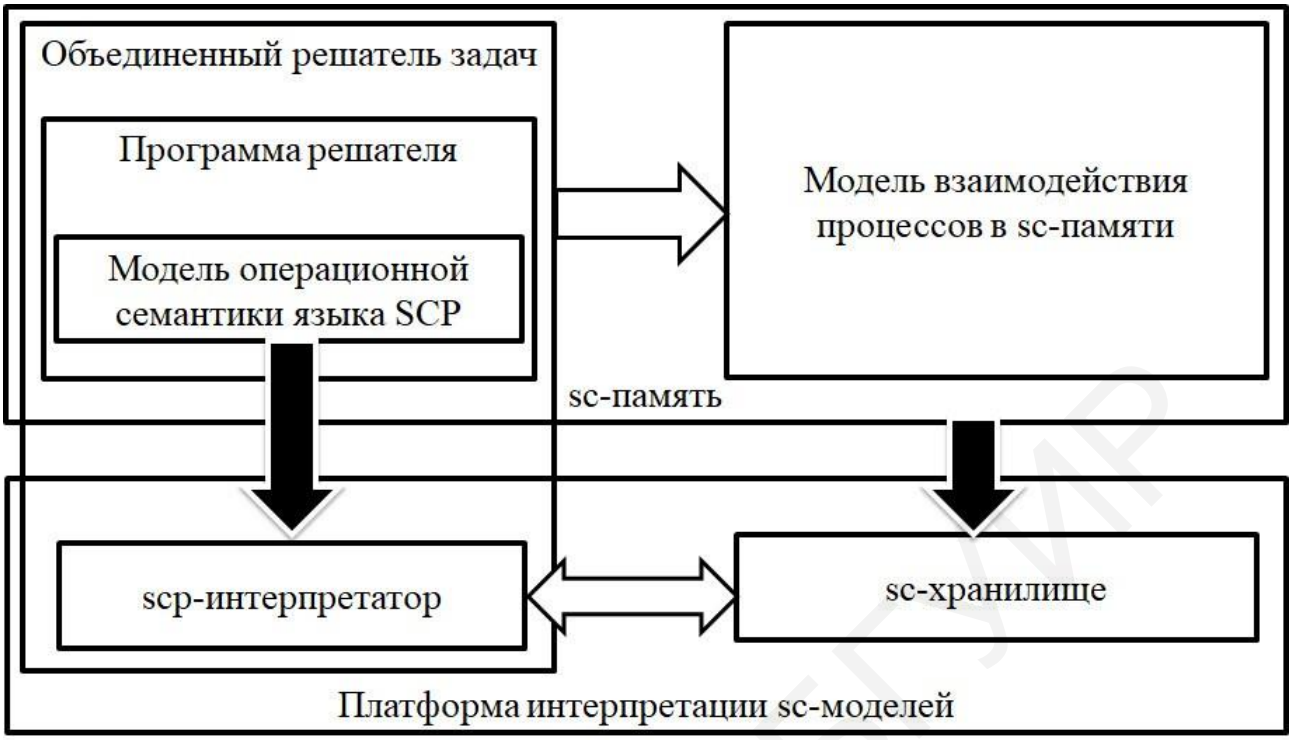

Рисунок 1.4 – Организация обработки информации в ostis-системе

В соответствии с приведенной иллюстрацией объединенный решатель задач разделяется на платформенно-зависимую часть (scp-интерпретатор) и платформенно-независимую часть – программу объединенного решателя. Программа решателя хранится в той же памяти, что и обрабатываемые знания, в той же памяти взаимодействуют и информационные процессы, выполняемые агентами решателя и направленные на решение каких-либо задач. В рамках программы решателя выделяется модель операционной семантики языка SCP, т. е. модель scp-интерпретатора, которая реализуется в рамках платформы. В свою очередь, абстрактная sc-память реализуется в виде sc-хранилища.

Поскольку sc-память хранит конструкции SC-кода, которые представляют собой графовые структуры определенного рода, то любого рода события, возникающие в sc-памяти, связаны с изменением такого рода графовых конструкций. Исходя из этого можно выделить классификацию элементарных событий в sc-памяти. Рассмотрим классификацию таких элементарных событий в SCn-коде:

элементарное событие в sc-памяти*
*<= разбиение**:
{
- *событие добавления sc-дуги, выходящей из заданного sc-элемента**
- *событие добавления sc-дуги, входящей в заданный sc-элемент**

- *событие добавления sc-ребра, инцидентного заданному sc-элементу**
- *событие удаления sc-дуги, выходящей из заданного sc-элемента**
- *событие удаления sc-дуги, входящей в заданный sc-элемент**
- *событие удаления sc-ребра, инцидентного заданному sc-элементу**
- *событие удаления sc-элемента**
- *событие изменения содержимого файла**

}

Для описания различных ситуаций в sc-памяти используется понятие структуры, рассмотренное в работе [57]. Под *структурой* понимается sc-узел, являющийся знаком некоторого фрагмента базы знаний, который, в свою очередь, может быть произвольным образом специфицирован в той же базе знаний и связан с другими sc-элементами различными связями. Под ***ситуацией*** понимается структура, в которую входит хотя бы один знак временной (временно существующей) сущности, в частности, временной sc-дуги. На рисунке 1.5 приведен пример простой ситуации в базе знаний (элемент *x* временно принадлежит множеству *si*).

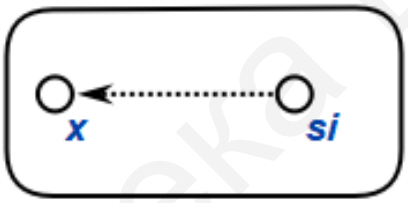

Рисунок 1.5 – Пример ситуации в sc-памяти

1.2.2 Модели параллельной обработки информации в семантической памяти

Семантическая память позволяет реализовать различные модели параллельной обработки знаний, более подробно рассмотренные, например, в работе [58]. Задача обеспечения возможности параллельного выполнения процессов обработки знаний в семантической памяти частично возлагается на вариант реализации указанной памяти, который может быть как программным, так и аппаратным, и должен включать реализацию механизма событий для агентов [55]. В частности, подходы к обеспечению параллелизма на уровне платформы, в том числе аппаратной, изложены в работах [59, 60].

Однако важной задачей является обеспечение возможности построения параллельных программ обработки знаний, описывающих алгоритм действий тех или иных агентов. Вопросы проектирования параллельных программ рассматриваются, например, в работе [61].

Базовый язык программирования SCP, предлагаемый в рамках технологии OSTIS, является языком параллельного процедурного программирования, ориентированным на обработку текстов SC-кода.

Формальное описание языка SCP включает описание:

– денотационной семантики языка SCP, т. е. построение онтологии программ данного языка (scp-программ), включающей описание классов операторов языка SCP (scp-операторов), средств спецификации scp-операторов, средств указания последовательности выполнения scp-операторов;

– операционной семантики языка SCP, т. е. построение модели интерпретатора scp-программ;

– процесса интерпретации scp-программы, т. е. построение онтологии информационных процессов, соответствующих некоторой scp-программе и выполняемых в семантической памяти, включающей описание средств синхронизации выполнения указанных процессов.

Интерпретатор языка SCP также строится как своего рода многоагентная система над общей семантической памятью, используя при этом те же принципы, что и общая модель гибридного решателя задач.

Рассмотрим ряд подходов, идеи которых использованы при разработке семантической памяти, интерпретатора языка SCP, а также при разработке общих средств спецификации информационных процессов в семантической памяти.

Вегетативная машина

Модель вегетативной машины предложена В. Б. Борщевым в работе [62]. В рассматриваемой работе сформулированы некоторые фундаментальные идеи, нашедшие применение при разработке семантической памяти:

– в предлагаемой машине элементы обрабатываемой информации и активные элементы (*процессоры*) являются частью общей структуры – некоторой вычислительной среды, называемой *конфигурацией*. Каждый процессор производит локальное изменение этой среды – совершает элементарное преобразование конфигурации. По сути такое понятие конфигурации во многом аналогично понятию общей семантической памяти;

– одни процессоры могут порождать другие процессоры (наследники) и при необходимости уничтожать их;

– обработка процессорами информации в такой системе осуществляется параллельно и независимо, но при этом проблема синхронизации должна быть решена на этапе проектирования алгоритма программистом. В этом заключается основное ограничение предлагаемой автором машины.

Очевидно, что выполнение первых двух пунктов из перечисленных возможно только при условии, что существует некоторый универсальный язык, позволяющий описывать в общей памяти не только собственно обрабатываемую информацию, но и программу обработки, и даже сами процессорные элементы. При этом при описании процессорных элементов в реально работающей системе необходимо специфицировать каждый такой элемент как с декларативной точки зрения (входные/выходные данные и т. д.), так и с операционной (описание реализуемой элементом программы действий).

В качестве такого языка может использоваться SC-код, являющийся языком кодирования информации в рамках технологии OSTIS.

A-системы

Понятие А-системы введено в статье В. Е. Котова и А. С. Нариньяни [63]. По словам самих авторов, данное понятие стоит трактовать как универсальную модель для некоторого класса параллельных систем, которая требует уточнения в случае конкретных реализаций. В частности, особый интерес в данной модели представляет механизм активации/деактивации процессорных элементов посредством так называемой спусковой функции, принимающей значения нуля и единицы. Понятно, что в конкретной реализации в качестве такой функции может быть использован любой признак, имеющий значения «истина» и «ложь», указывающий на то, что тот или иной процессорный элемент должен быть активирован в следующий момент времени. Авторами показана возможность формализации на основе данной модели любых параллельных алгоритмов, рассмотрена возможность сведения таких алгоритмов к последовательным, варианты синхронизации в рамках такой модели.

В рамках технологии OSTIS модель А-системы уточняется и развивается в контексте средств спецификации информационных процессов в семантической памяти и модели гибридного решателя задач. В частности, на каждом уровне многоуровневой модели гибридного решателя задач может быть построено однозначное соответствие компонентов данного уровня и компонентов А-системы, что позволяет говорить о возможности реализации на основе данной модели различных моделей параллельной обработки знаний.

1.3 ДЕЙСТВИЯ И ЗАДАЧИ

Семантическая теория деятельности в целом подробно освещена в работах В. В. Мартынова [64–66], а также его учеников, в частности, в статье [67]. В указанных работах рассматривается семантическая классификация действий, описываются подходы к формализации описания деятельности различного рода субъектов с использованием разработанного

В. В. Мартыновым универсального семантического кода (УСК). В этих работах детально описываются такие понятия, как воздействие, субъект воздействия (агент), объект и другие, рассматриваемые также и в данном пособии. Кроме того, в книге [68] даны такие понятия, как воздействие, действие (акт), деятель (субъект) и др.

Результаты указанных работ интегрированы на общей формальной основе, что позволяет использовать их в дальнейшем при разработке различных решателей задач.

1.3.1 Общее понятие действия

В рамках предлагаемой модели описания деятельности понятие «*действие*» является частным случаем понятия «*воздействие*», которое исследуется в предметной области временных сущностей в рамках базы знаний IMS [56]. В свою очередь, воздействие является частным случаем *процесса*. Каждый *процесс* определяется (задается) возникновением или исчезновением связей, связывающих либо заданную *временную сущность* с другими сущностями, либо части указанной *временной сущности* с другими сущностями. С точки зрения классификации фрагментов базы знаний [52] *процесс* представляет собой ситуативную структуру, в каждый момент времени описывающую текущее состояние объектов, участвующих в данном процессе. *Процесс*, описывающий изменения, происходящие исключительно в рамках sc-памяти, будем называть *процессом в sc-памяти*. Каждому *воздействию* может быть поставлена в соответствие:

– некоторая *воздействующая сущность* (субъект)*, т. е. сущность, осуществляющая воздействие (например, это может быть некоторое физическое поле);

– некоторый *объект** воздействия, т. е. сущность, на которую воздействие направлено.

Если *воздействие* связано с *материальной сущностью*, то его объектом воздействия является либо сама эта *материальная сущность*, либо некоторая ее пространственная часть. В свою очередь, каждое *действие*, выполняемое тем или иным *субъектом*, одновременно можно трактовать и как процесс решения некоторой задачи, т. е. как процесс достижения заданной цели в заданных условиях. Предполагается, что любое *действие*, выполняемое каким-либо *субъектом*, направлено на решение какой-либо задачи и выполняется целенаправленно. При этом явное указание *действия* и его связи с конкретной *задачей* может не всегда присутствовать в памяти. Некоторые задачи могут решаться определенными агентами перманентно, например, оптимизация базы

знаний, поиск некорректностей и т. д., и для подобных задач не всегда есть необходимость явно вводить *структуру* [52], являющуюся формулировкой *задачи*. Каждое *действие* обозначает некоторое преобразование, осуществляемое во внешней среде либо в памяти некоторой системы, однако в памяти явно вводятся только sc-элементы, обозначающие те *действия*, для которых есть необходимость явно хранить их спецификацию в течение некоторого времени. При выполнении действия можно выделить следующие этапы:

– построение плана деятельности, декомпозиция исходного действия;

– выполнение построенного плана действий.

Рассмотрим кратко верхний уровень классификации понятия «действие», представленный на языке SCn.

действие
= акция
= целенаправленный процесс, управляемый некоторым субъектом
*<= разбиение**:
Разбиение класса действий по отношению к памяти информационной системы
=
{
- *информационное действие*
  *=> включение**:
  действие в sc-памяти
- *поведенческое действие*
  *=> включение**:
  действие во внешней среде ostis-системы
- *эффекторное действие*
  *=> включение**:
  эффекторное действие ostis-системы
- *рецепторное действие*
  *=> включение**:
  рецепторное действие ostis-системы

}
*<= разбиение**:
Разбиение класса действий по отношению к текущему моменту времени
=
{
- *инициированное действие*
- *планируемое действие*
  = будущее действие:
- *выполненное действие*

}

Результатом выполнения *информационного действия* является в общем случае некоторое новое состояние памяти информационной системы (необязательно *sc-памяти*), достигнутое исключительно путем преобразования информации, хранящейся в памяти системы, т. е. либо посредством генерации новых знаний на основе уже имеющихся, либо посредством удаления знаний, по каким-либо причинам ставших ненужными. Следует отметить, что если речь идет об изменении состояния *sc-памяти*, то любое преобразование информации можно свести к ряду элементарных действий генерации, удаления или изменения инцидентности *sc-элементов* друг относительно друга.

В случае *поведенческого действия* результатом его выполнения будет новое состояние внешней среды. Очень важно отметить, что под внешней средой в данном случае понимаются также и компоненты системы, внешние с точки зрения памяти, т. е. не являющиеся хранимыми в ней информационными конструкциями. К таким компонентам можно отнести, например, различные манипуляторы и прочие средства воздействия системы на внешний мир, т. е. к поведенческим задачам можно отнести изменение состояния механической конечности робота или непосредственно вывод некоторой информации на экран для восприятия пользователем.

Под *эффекторным действием* понимается действие, связанное с воздействием ostis-системы на внешнюю с точки зрения системы среду.

Под *рецепторным действием* понимается действие, связанное с воздействием внешнего субъекта на ostis-систему, т. е. на ее датчики, рецепторы и т. п.

Во множество *инициированных действий* входят *действия*, выполнение которых инициировано в результате какого-либо события. По сути попадание некоторого *действия* во множество *инициированных действий* говорит о том, что спецификация данного *действия* полностью сформирована, т. е. никаких дополнительных элементов, необходимых для решения поставленной задачи, не требуется, и соответствующий *агент* (либо коллектив *агентов*, либо внешний *субъект*) может приступать к выполнению действия. Однако стоит отметить, что, с точки зрения исполнителя, такая спецификация *действия* в общем случае может оказаться недостаточной или некорректной.

Формальная спецификация понятия *инициированное действие* в SCn-коде:

инициированное действие
*=> включение**:
    выполняемое действие
    *<= разбиение**:
        {

- *активное действие*
- *отложенное действие*

}

Во множество *выполняемых действий* входят *действия*, к выполнению которых приступил какой-либо из соответствующих *субъектов*.

Попадание *действия* в данное множество говорит о следующем:

– рассматриваемое *действие* уже попало во множество *инициированных действий*;

– существует как минимум один *субъект*, способный выполнить данное *действие*.

После того как собственно процесс выполнения завершился, *действие* должно быть удалено из множества *выполняемых действий* и добавлено во множество *выполненных действий* или какое-либо из его подмножеств.

Понятие *выполняемое действие* является неосновным, т. е. носит дидактический характер, и вместо того чтобы относить конкретные *действия* к данному классу, их относят к классу *настоящих сущностей* (сущностей, существующих в данный момент времени).

Во множество *активных действий* входят *действия*, выполнение которых осуществляется непосредственно в данный момент каким-либо *субъектом*.

Во множество *отложенных действий* входят *действия*, которые уже были инициированы, однако их выполнение невозможно по каким-либо причинам, например, в случае, когда у исполнителя в данный момент есть более приоритетные задачи.

Во множество *планируемых действий* входят *действия*, начать выполнение которых запланировано на какой-либо момент в будущем.

Во множество *выполненных действий* попадают *действия*, выполнение которых завершено. В зависимости от результатов конкретного процесса выполнения рассматриваемое *действие* может стать элементом одного из подмножеств множества *выполненных действий*.

Понятие *выполненное действие* также является неосновным, и вместо того, чтобы относить конкретные *действия* к данному классу, их относят к классу *прошлых сущностей*.

Подклассы понятия *выполненное действие* в SCn-коде:

выполненное действие
= *прошлое действие*
<= *разбиение**:
{

• *успешно выполненное действие*
• *безуспешно выполненное действие*
  => *включение**:
      действие, выполненное с ошибкой
}

Во множество *успешно выполненных действий* попадают *действия*, выполнение которых успешно завершено с точки зрения *субъекта*, осуществлявшего их выполнение, т. е. достигнута поставленная цель, например, получены решение и ответ какой-либо задачи, успешно преобразована какая-либо конструкция и т. д.

Во множество *безуспешно выполненных действий* попадают *действия*, выполнение которых не было успешно завершено с точки зрения *субъекта*, осуществлявшего их выполнение, по каким-либо причинам.

Можно выделить две основные причины, по которым может сложиться указанная ситуация:

– соответствующая *задача* сформулирована некорректно;

– формулировка соответствующей *задачи* корректна и понятна системе, однако решение данной задачи в текущий момент не может быть получено в удовлетворительные с точки зрения заказчика или исполнителя сроки.

Для конкретизации факта некорректности формулировки задачи можно выделить ряд более частных классов безуспешно выполненных действий, например:

– *действие*, спецификация которого противоречит другим знаниям системы (например, не выполняется неравенство треугольника);

– *действие*, при спецификации которого использованы понятия, неизвестные системе;

– *действие*, выполнение которого невозможно из-за недостаточности данных (например, найти площадь треугольника по двум сторонам) и др.

В процессе описания в семантической памяти деятельности некоторого коллектива субъектов возникает необходимость выделять в рамках этой деятельности обособленные логически целостные фрагменты, которые могут выполняться отдельными субъектами независимо друг от друга. Спецификация понятия *класса действий* в SCn-коде:

класс действий
=множество действий, однотипных в том или ином смысле
*<= семейство подмножеств**:
      действие
*<= разбиение**:

{

- *класс логически атомарных действий =класс автономных действий*
- *класс логически неатомарных действий =класс неавтономных действий*

}

Каждое *действие*, принадлежащее некоторому конкретному *классу логически атомарных действий*, обладает двумя необходимыми свойствами:

1. Выполнение действия не зависит от того, является ли указанное действие частью декомпозиции более общего действия. При выполнении данного действия также не должен учитываться тот факт, что данное действие предшествует каким-либо другим действиям или следует за ними (что явно указывается при помощи отношения *последовательность действий**).

2. Указанное действие должно представлять собой логически целостный акт преобразования, например, в семантической памяти. Такое действие по сути является транзакцией, т. е. результатом такого преобразования становится новое состояние преобразуемой системы, а выполняемое действие должно быть либо выполнено полностью, либо не выполнено совсем, частичное выполнение не допускается.

В то же время логическая атомарность не запрещает декомпозировать выполняемое действие на более частные, каждое из которых, в свою очередь, также будет являться логически атомарным. На рисунке 1.6 с использованием языка SCg показан пример декомпозиции более сложного логически атомарного действия на более простые.

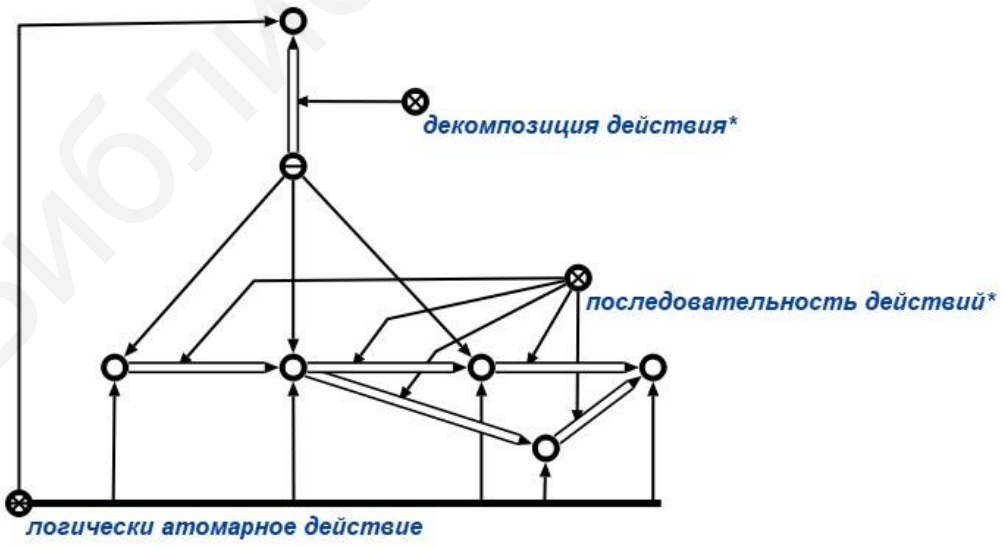

Рисунок 1.6 – Декомпозиция логически атомарного действия на поддействия

На логически атомарные действия предлагается делить всю деятельность, направленную на решение каких-либо задач ostis-системой. Соответственно решатель предлагается делить на компоненты (агенты), соответствующие таким классам логически атомарных действий, что является основой для обеспечения его **модифицируемости**.

В случае действий, выполняемых в семантической памяти, степень детализации каждого такого действия ограничивается синтаксическими особенностями используемого варианта представления знаний. В случае SC-кода можно выделить такие классы элементарных действий, как создание sc-элемента заданного типа, удаление sc-элемента, поиск sc-элементов, инцидентных указанному sc-элементу. На основе классификации таких элементарных преобразований в семантической памяти строится язык SCP, который описывается в подразделе 1.5.

Предполагается, что каждое действие выполняется целенаправленно некоторым *субъектом*. Формальная спецификация понятия *субъект* в SCn-коде (в том числе классификация субъектов):

субъект
= активная сущность
= сущность, способная самостоятельно выполнять некоторые виды действий
*=> включение**:

- *собственное «Я» ostis-системы*
- *внутренний субъект ostis-системы*
- *внешний субъект ostis-системы, с которым осуществляется взаимодействие*
- *внешний субъект ostis-системы, с которым взаимодействие не происходит*

Под *внутренним субъектом ostis-системы* понимается такой *субъект*, который выполняет некоторые действия в той же памяти, в которой хранится его знак.

К числу *внутренних субъектов ostis-системы* относятся входящие в нее *агенты*, отдельные решатели задач, целые интеллектуальные подсистемы.

К числу *внешних субъектов ostis-системы, с которыми осуществляется взаимодействие*, относятся конечные пользователи *ostis-системы*, ее разработчики, а также другие компьютерные системы (причем не только интеллектуальные).

1.3.2 Средства детализации процесса выполнения действий

Рассмотрим набор отношений, предназначенных для описания детализации процесса выполнения того или иного действия, т. е. выделения более простых частных действий.

Связки отношения *декомпозиция действия** связывают *действие* и множество более простых *действий*, на которые декомпозируется данное действие. При этом первым компонентом связки является знак указанного множества, вторым компонентом – знак более сложного *действия*.

Таким образом, *декомпозиция действия** – это *квазибинарное отношение* [56], связывающее действие со множеством действий более низкого уровня, к выполнению которых сводится выполнение исходного декомпозируемого действия.

Стоит отметить, что каждое *действие* может иметь несколько вариантов декомпозиции в зависимости от конкретного набора элементарных действий, которые способна выполнять та или иная система *субъектов*.

Принцип, по которому осуществляется такая декомпозиция в различных подходах к решению задач, будем называть ***стратегией решения задач***.

Кроме того, для детализации процесса выполнения действий используется отношение *поддействие**, связки которого связывают *действие* и некоторое более простое *действие*, выполнение которого необходимо для выполнения исходного более общего *действия*.

Для описания порядка выполнения действий используется отношение *последовательность действий**, частными видами которого являются отношения *последовательность действий при положительном результате** и *последовательность действий при отрицательном результате**.

Связки отношения *последовательность действий** связывают знаки *действий*, выполняющихся в какой-либо последовательности в процессе решения какой-либо задачи. При этом считается, что если два *действия* связаны данным отношением, то *действие*, стоящее в данной связке на втором месте, может быть выполнено только после выполнения действия, стоящего в данной связке на первом месте. Таким образом, каждое действие может быть инициировано после завершения выполнения любого из предшествующих действий.

Переход по связкам отношения *последовательность действий при положительном результате** от предшествующего действия проверки условия к последующему действию происходит при условии, если указанная проверка даст положительный результат, т. е. предшествующее действие станет *успешно выполненным действием*.

Переход по связкам отношения *последовательность действий при отрицательном результате** от предшествующего действия проверки условия к последующему действию происходит при условии, если указанная проверка

даст отрицательный результат, т. е. предшествующее действие станет *безуспешно выполненным действием*.

Для обеспечения возможности синхронизации выполнения действий используется класс действий *конъюнкция предшествующих действий*. Действия указанного класса используются в тех случаях, когда выполнение некоторого действия должно начаться только после того, как будут выполнены все предшествующие действия, а не только одно из них. Только после выполнения предшествующих действий инициируются действия, следующие за *конъюнкцией предшествующих действий*.

В некоторых случаях бывает необходимо управлять процессом выполнения какой-либо последовательности действий в зависимости от выполнения дополнительных условий. Для осуществления таких проверок вводится класс действий *проверки условия*. Действия класса *проверка условия* предполагают проверку истинности или ложности некоторого высказывания (условия), и после выполнения в зависимости от результата данной проверки становятся *успешно выполненными действиями* или *безуспешно выполненными действиями*.

Следует отметить, что предлагаемый подход к описанию переходов между действиями очень близок к решениям, описанным в статье [63]. Действительно, каждое действие независимо от его сложности можно считать аналогом процессорного элемента, рассмотренного в указанной статье, а условием инициирования какого-либо действия (спусковой функцией) является факт завершенности хотя бы одного из предыдущих действий (т. е. связанных с текущим действием связкой отношения *последовательность действий** или более частных отношений). Таким образом, использование предлагаемого языка описания деятельности различных субъектов на различных уровнях позволит не только говорить об универсальности и «понятности» такого описания за счет использования самых базовых понятий, но и о возможности реализации различных моделей параллелизма на любом уровне, начиная от параллельного выполнения операторов в рамках одной программы и заканчивая взаимодействием целых коллективов агентов в общей семантической памяти. Возможность реализации той или иной модели параллелизма в таком случае определяется исключительно особенностями решаемой задачи.

1.3.3 Спецификация действий

Важнейшим с точки зрения функционирования системы понятием является понятие задачи. Под *задачей* понимается формальная спецификация

некоторого действия (условие), достаточная для выполнения данного действия каким-либо *субъектом*. В зависимости от конкретного класса задач описываться может как внутреннее состояние самой интеллектуальной системы, так и требуемое состояние внешней среды. *Sc-элемент*, обозначающий *действие*, входит в *задачу* под атрибутом *ключевой sc-элемент'*.

Классификация задач может осуществляться по дидактическому признаку в рамках каждой предметной области, например, задачи на треугольники, задачи на системы уравнений и т. п.

Каждая *задача* может включать:

– факт принадлежности *действия* какому-либо частному классу *действий* (например, *действие. сформировать полную семантическую окрестность указываемой сущности*), в том числе состояние *действия* с точки зрения жизненного цикла (инициированное, выполняемое и т. д.);

– описание *цели* (результата*) действия*, если она точно известна;

– указание *заказчика* действия*;

– указание *исполнителя* действия* (в том числе коллективного);

– указание *аргумента(ов) действия'*;

– указание *инструмента* или *посредника действия*;

– описание *декомпозиции действия**;

– указание *последовательности действий** в рамках *декомпозиции действия**, т. е. построение плана решения задачи; другими словами, построение плана решения представляет собой декомпозицию соответствующего *действия* на систему взаимосвязанных между собой поддействий;

– указание области *действия*;

– указание *условия инициирования действия*;

– момент *начала* и *завершения действия*, в том числе планируемый и фактический, предполагаемую и/или фактическую длительность выполнения.

Некоторые задачи могут быть дополнительно уточнены контекстом – дополнительной информацией о сущностях, рассматриваемых в формулировке *задачи*, т. е. описанием того, что дано, что известно об указанных сущностях. Кроме этого, *задача* может включать любую дополнительную информацию о действии, например:

– перечень ресурсов и средств, которые предполагается использовать при решении задачи, в частности, список доступных исполнителей, временные сроки, объем имеющихся финансов и т. д.;

– ограничение области, в которой выполняется действие, например, необходимо заменить одну sc-конструкцию на другую по некоторому правилу, но только в пределах некоторого раздела базы знаний;

– ограничение знаний, которые можно использовать для решения той или иной задачи, например, необходимо решить задачу по алгебре, используя только те утверждения, которые входят в курс школьной программы до седьмого класса включительно, и не используя утверждения, изучаемые в старших классах школы.

Решаемые системой задачи можно классифицировать на *информационные задачи* и *поведенческие задачи*.

С точки зрения формулировки поставленной задачи можно выделить *декларативные формулировки задачи* и *процедурные формулировки задачи*. Следует отметить, что данные классы задач не противопоставляются и могут существовать формулировки задач, использующие оба подхода.

В *процедурной формулировке задачи* явно указываются аргументы соответствующего задаче *действия* и, в частности, вводится семантическая классификация *действий*. При этом явно не уточняется, что должно быть результатом выполнения данного действия. Заметим, что при необходимости *процедурная формулировка задачи* может быть сведена к *декларативной формулировке задачи* путем трансляции на основе некоторого правила, например, определения класса действия через более общий класс.

В случае *декларативной формулировки задачи* при описании условия задачи специфицируется *цель действия*, т. е. результат, который должен быть получен при успешном выполнении *действия*.

В свою очередь, под *вопросом* понимается задача, направленная на удовлетворение информационной потребности некоторого субъекта-заказчика.

Под *командой* (в том числе *интерфейсной командой*) понимается спецификация инициированного действия (инициированная задача).

Рассмотрим набор отношений, используемых для формальной спецификации действий в рамках *задачи*.

Связки отношения *результат** связывают *sc-элемент*, обозначающий *действие*, и *sc-конструкцию*, описывающую результат выполнения рассматриваемого действия, другими словами, цель, которая должна быть достигнута при выполнении *действия*.

Результат может специфицироваться как атомарным высказыванием, так и неатомарным, т. е. конъюнктивным, дизъюнктивным, строго дизъюнктивным и т. д. В случае когда успешное выполнение *действия* приводит к изменению какой-либо конструкции в *sc-памяти*, которое необходимо занести в историю

изменений базы знаний или использовать для демонстрации протокола решения задачи, генерируется соответствующая связка отношения *результат**, связывающая *задачу* и *sc-конструкцию*, описывающую данное изменение. Конкретный вид указанной *sc-конструкции* зависит от типа действия.

Связки отношения *исполнитель** связывают *sc-элементы*, обозначающие *действие,* и *sc-элементы*, обозначающие *субъекта*, который предположительно будет осуществлять, осуществляет или осуществлял выполнение указанного *действия*. Данное отношение может быть использовано при назначении конкретного исполнителя для проектной задачи по развитию баз знаний.

В случае когда заранее неизвестно, какой именно *субъект** будет исполнителем данного *действия*, связка отношения *исполнитель** может отсутствовать в первоначальной формулировке *задачи* и добавляться позже, уже непосредственно при исполнении.

Когда действие выполняется (является *настоящей сущностью*) или уже выполнено (является *прошлой сущностью*), то исполнитель этого действия в каждый момент времени уже определен.

Связки отношения *заказчик** связывают *sc-элементы*, обозначающие *действие*, и *sc-элементы*, обозначающие *субъекта*, который «заинтересован» в выполнении данного действия и, как правило, инициирует его выполнение. Данное отношение может быть использовано при указании того, кто поставил проектную задачу по развитию базы знаний.

Связки отношения *объект** связывают *sc-элемент*, обозначающий *действие*, и знак той сущности, над которой (по отношению к которой) осуществляется данное *действие*, например, знак *структуры*, подлежащий верификации.

Связки отношения *контекст действия** связывают *sc-элементы*, обозначающие *действие,* и *структуры*, обозначающие контекст выполнения данного *действия*, т. е. некоторую дополнительную информацию о тех сущностях, которые входят в описание *цели**. Как правило, контекст используется для указания собственно условия некоторой задачи, того, что дано, т. е. тех знаний, которые можно использовать для вывода новых знаний при решении задачи. Таким образом, контекст непосредственно влияет на то, как будет решаться та или иная задача, при этом даже задачи, соответствующие одному классу действий, могут решаться по-разному.

Контекст может быть представлен не только в виде атомарного фактографического высказывания, но и в виде высказывания более сложного вида. Это может быть, например:

– определение множества, используемого в описании *цели**;

– утверждение, учет которого может быть полезен в решении задач.

В случае процедурной формулировки задачи для указания в рамках конкретного действия тех sc-элементов, над которыми непосредственно выполняется данное действие, используется ролевое отношение *аргумент действия'*.

Для уточнения класса сущностей, которые могут быть аргументами действий, соответствующих некоторому *классу команд*, используется отношение *класс аргументов**. На рисунке 1.7 с использованием языка SCg показан пример представления задачи в sc-памяти, связанной с установлением истинности высказывания о конгруэнтности двух треугольников.

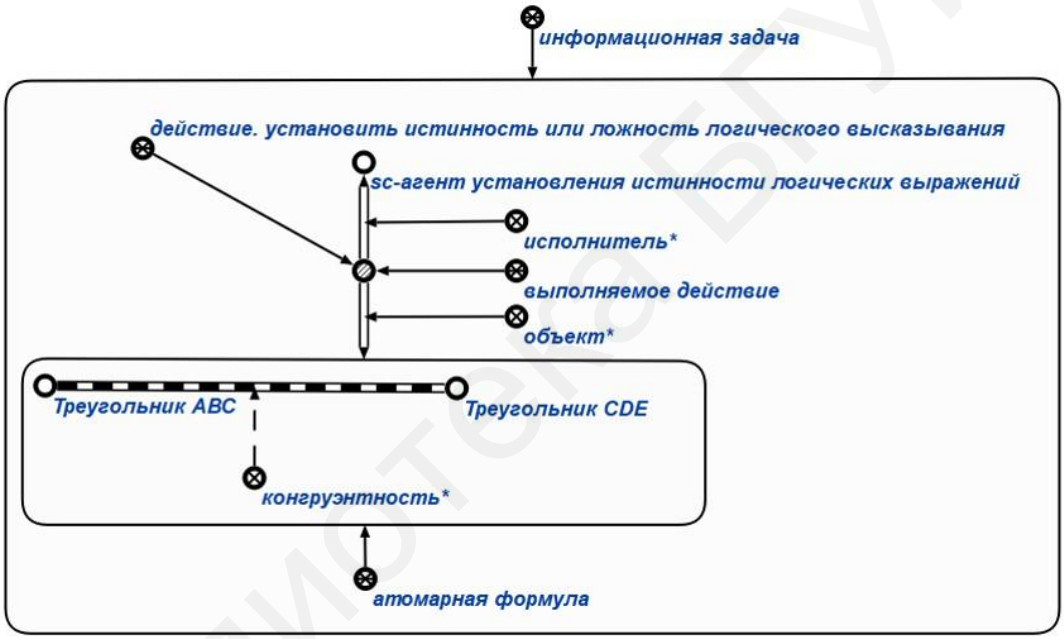

Рисунок 1.7 – Пример представления задачи

Если для некоторого *класса команд* не указан тип какого-либо из аргументов, то предполагается, что в качестве данного аргумента может выступать любой *sc-элемент*.

Кроме задачи как спецификации действия в целом необходимо выделить еще ряд *семантических окрестностей* [52], специфицирующих конкретное действие с той или иной стороны.

План выполнения действия

Каждый *план* представляет собой *семантическую окрестность*, ключевым *sc-элементом'* является *действие*, для которого дополнительно детализируется предполагаемый процесс его выполнения. Основная задача

такой детализации – локализация области базы знаний, в которой предполагается работать, а также набора агентов, необходимого для выполнения описываемого действия. При этом детализация необязательно должна быть доведена до уровня элементарных действий, цель составления плана – уточнение подхода к решению той или иной задачи, не всегда предполагающее составление подробного пошагового решения.

При описании *плана* может быть использован как процедурный, так и декларативный подход. В случае процедурного подхода для соответствующего *действия* указывается его декомпозиция на более частные поддействия, а также необходимая спецификация этих поддействий. В случае декларативного подхода указывается набор подцелей (например, при помощи логических утверждений), достижение которых необходимо для выполнения рассматриваемого *действия*. На практике оба рассмотренных подхода можно комбинировать.

В общем случае *план* может содержать и переменные, например, в случае, когда часть плана задается в виде цикла (многократного повторения некоторого набора действий). Также план может содержать константы, значения которых в настоящий момент не установлены и станут известны, например, только после выполнения предшествующих ему *действий*.

Каждый *план* может быть задан заранее как часть спецификации *действия*, т. е. задачи, а может формироваться *субъектом* уже собственно в процессе выполнения *действия*, например, в случае использования стратегии разбиения задачи на подзадачи. В первом случае *план включается** в *задачу*, соответствующую тому же действию.

Программа выполнения класса действий

Каждая *программа* представляет собой обобщенный *план* выполнения *действий*, принадлежащих некоторому классу, т. е. *семантическую окрестность*. *Ключевым sc-элементом'* является *класс действий*, для элементов которого дополнительно детализируется процесс их выполнения.

В остальном описание программы аналогично описанию *плана* выполнения конкретного *действия* из рассматриваемого *класса действий*.

Одному *классу действий* может соответствовать несколько *программ*.

Входным параметрам *программы* в традиционном понимании соответствуют аргументы, соответствующие каждому *действию* из *класса действий*, описываемого *программой*. При генерации на основе *программы плана* выполнения конкретного *действия* из данного класса эти аргументы принимают конкретные значения.

Каждая *программа* представляет собой систему описанных действий с дополнительным указанием для действия:

– либо *последовательности выполнения действий** (передачи инициирования), когда условием выполнения (инициирования) действий является завершение выполнения одного из указанных или всех указанных действий;

– либо события в базе знаний или внешней среде, являющегося условием его инициирования;

– либо ситуации в базе знаний или внешней среде, являющейся условием его инициирования.

Видно, что предложенная трактовка понятия программы как обобщенного плана выполнения некоторого класса действий никак не ограничивает область его применения, в частности, как программа в таком смысле может трактоваться рецепт приготовления какого-либо блюда, обобщенное описание какого-либо технологического процесса, инструкция по использованию какого-либо устройства и т. д.

1.3.4 Классификация действий в семантической памяти

Отдельное внимание следует уделить действиям, выполняемым в семантической памяти компьютерной системы (sc-памяти).

Каждое *действие в sc-памяти* обозначает некоторое преобразование, выполняемое некоторым *субъектом* (или *коллективом субъектов*) в рамках *sc-памяти*. Спецификация действия после его выполнения может быть включена в протокол решения некоторой задачи.

Семантика каждого *действия в sc-памяти* зависит от конкретного контекста, т. е. ориентации действия на какие-либо конкретные объекты и принадлежности действия какому-либо конкретному классу действий.

Следует четко отличать:

– каждое конкретное *действие в sc-памяти*, представляющее собой некоторый переходный процесс, переводящий sc-память из одного состояния в другое;

– каждый тип *действий в sc-памяти*, представляющий собой некоторый класс однотипных (в том или ином смысле) действий;

– sc-узел, обозначающий некоторое конкретное *действие в sc-памяти*;

– sc-узел, обозначающий структуру, которая является описанием, спецификацией, заданием, постановкой соответствующего действия.

Формальная спецификация понятия *действие в sc-памяти* на языке SCn:

действие в sc-памяти
= внутреннее действие ostis-системы
= действие, выполняемое в sc-памяти
= действие, выполняемое решателем задач ostis-системы
<= включение:*
процесс в sc-памяти
=> включение:*

- *действие в sc-памяти, инициируемое вопросом*
- *действие редактирования базы знаний ostis-системы*
- *действие установки режима ostis-системы*
- *действие редактирования файла, хранимого в sc-памяти*
- *действие интерпретации программы, хранимой в sc-памяти*

Формальная спецификация понятия *действие в sc-памяти, инициируемое вопросом* на языке SCn:

действие в sc-памяти, инициируемое вопросом
= действие, направленное на формирование ответа на поставленный вопрос
=> включение:*

- *действие. сформировать заданный файл*
- *действие. сформировать заданную структуру*
  => включение:*
  - *действие. верифицировать заданную структуру*
    => включение:*
    - *действие. установить истинность или ложность указываемого логического высказывания*
    - *действие. установить корректность или некорректность указываемой структуры*
    - *действие. сформировать структуру, описывающую некорректности, имеющиеся в указываемой структуре*
  - *действие. установить тип заданного sc-элемента*
    => включение:*
    - *действие. установить позитивность/негативность указываемой sc-дуги принадлежности или непринадлежности*
  - *действие. сформировать семантическую окрестность*
    => включение:*
    - *действие. сформировать полную семантическую окрестность указываемой сущности*
    - *действие. сформировать базовую семантическую окрестность указываемой сущности*
    - *действие. сформировать частную семантическую окрестность указываемой сущности*
  - *действие. сформировать структуру, описывающую связи между указываемыми сущностями*
    => включение:*

- *действие. сформировать структуру, описывающую сходства указываемых сущностей*
- *действие. сформировать структуру, описывающую различия указываемых сущностей*

- *действие. сформировать структуру, описывающую способ решения указываемой задачи*
- *действие. сформировать план генерации ответа на указанный вопрос*
- *действие. сформировать протокол выполнения указываемого действия*
- *действие. сформировать обоснование корректности указываемого решения*
- *действие. верифицировать обоснование корректности указываемого решения*
- *действие, одним из аргументов которого является некоторая обобщенная структура*
- *действие, направленное на установление темпоральных характеристик указываемой сущности*
  действие, направленное на установление пространственных характеристик указываемой сущности

Формальная спецификация понятия *действие редактирования базы знаний* на языке SCn:

действие редактирования базы знаний
*=> включение**:
- *действие. исправить ошибки в заданной структуре*
- *действие. преобразовать указанную структуру в соответствии с указанным правилом*
- *действие. отождествить два указанных sc-элемента*
- *действие. включить множество*
  = сделать все элементы множества si явно принадлежащими множеству sj, т. е. сгенерировать соответствующие sc-дуги принадлежности
- *действие генерации sc-элементов*
  *=> включение**:
  - *действие генерации, одним из аргументов которого является некоторая обобщенная структура*
    *=> включение**:
    - *действие. сгенерировать структуру, изоморфную указываемому образцу*
  - *действие. сгенерировать sc-элемент указанного типа*
    *=> включение**:
    - *действие. сгенерировать sc-коннектор указанного типа*
    - *действие. сгенерировать sc-узел указанного типа*
  - *действие. сгенерировать структуру, содержащую указанные sc-элементы*
  - *действие. сгенерировать файл с заданным содержимым*
  - *действие. обновить понятия*
    = действие. заменить неиспользуемое понятие на его определение через используемое понятие
  - *действие. установить указанный файл в качестве основного идентификатора указанного sc-элемента*

- *действие. протранслировать содержимое указываемого файла в sc-память*
- *действие. интегрировать указанную структуру в текущее состояние базы знаний*
- *действие. сгенерировать структуру, описывающую историю эволюции ostis-системы*
- *действие. сгенерировать структуру, описывающую историю эксплуатации ostis-системы*

- *действие удаления sc-элементов*
  *=> включение**:
  - *действие. удалить указанные sc-элементы*
    *=> включение**:
    - *действие. удалить указанный sc-элемент*
    - *действие. удалить sc-элементы, входящие в состав указанной структуры и не являющиеся ключевыми узлами каких-либо sc-агентов*
  - *действие. исключить указанные sc-элементы из клиентской части базы знаний*

1.4 АРХИТЕКТУРА РЕШАТЕЛЯ ЗАДАЧ

1.4.1 Принципы реализации многоагентного подхода в семантической памяти

В рамках технологии OSTIS гибридный решатель задач любой компьютерной системы трактуется как система агентов, работающих над общей семантической памятью. Ориентация на многоагентный подход как основу для построения модицифируемых решателей обусловлена следующими основными преимуществами такого подхода [69]:

– автономность (независимость) агентов в рамках такой системы, что позволяет локализовать изменения, вносимые в решатель при его эволюции, и снизить соответствующие трудозатраты, а также обеспечить устойчивость такой системы к отказам некоторых агентов;

– децентрализация обработки, т. е. отсутствие единого контролирующего центра, что также позволяет локализовать вносимые в решатель изменения;

– возможность параллельной работы разных информационных процессов, соответствующих как одному агенту, так и разным, как следствие, – возможность распределенного решения задач;

– активность агентов и многоагентной системы в целом, дающая возможность при общении с такой системой не указывать явно способ решения поставленной задачи, а формулировать задачу в **декларативном ключе**.

Наиболее общее и широко используемое определение понятия интеллектуального агента дается в [70].

Состояние разработок в области многоагентных систем подробно отражено в монографии [71], обзорах [72] и [73]. Работы в данной области с 1998 г. регулярно публикуются в специализированном англоязычном журнале Autonomous Agents and Multi-Agent Systems [74].

Эффективность применения многоагентных систем подтверждается использованием таких систем в самых различных сферах, в частности, наиболее активно развивается направление моделирования различных коллективных процессов, в том числе социальных [75] и промышленных [76]. Существуют и активно развиваются среды для агентно-ориентированного моделирования и имитации [77, 78], в том числе свободно распространяемые [79, 80]. Подробный обзор таких сред приведен в работах [81, 82].

Однако, как отмечается в работе [83], в настоящее время разработка многоагентных систем затруднена не только сложностью процесса, обусловленного динамической и распределенной природой проектируемых систем, но и отсутствием общепринятой методологии и достаточного количества хороших инструментальных средств, позволяющих строить на базе многоагентного подхода эффективные прикладные системы.

С учетом всех перечисленных достоинств такого подхода задача построения модели гибридного решателя задач, удовлетворяющего всем требованиям, предъявленным в подразделе 1.3, сводится к построению модели многоагентной системы над общей семантической памятью.

Для того чтобы построить такую модель многоагентной системы, необходимо уточнить модель каждого компонента, входящего в ее состав, а именно:

– **модель агента**, входящего в состав такой системы, включая классификацию таких агентов и набор понятий, характеризующих каждый агент в рамках системы; в настоящее время наиболее популярной является модель BDI (belief-desire-intention), в рамках которой предполагается описывать на соответствующих языках «убеждения», «желания» и «намерения» каждого агента системы;

– **модель среды**, в рамках которой находятся агенты, на события в которой они реагируют и в рамках которой могут осуществлять некоторые преобразования;

– **модель коммуникации агентов**, в рамках которой уточняется язык взаимодействия агентов (структура и классификация сообщений) и способ передачи сообщений между агентами.

К основным недостаткам большинства популярных современных средств построения многоагентных систем [79, 80, 84–89] можно отнести следующие:

1. Жесткая ориентация большинства средств на модель BDI приводит к существенным накладным расходам, связанным с необходимостью выражения конкретной практической задачи в системе понятий BDI. В то же время ориентация на модель BDI неявно провоцирует искусственное разделение языков, описывающих собственно компоненты BDI и знания агента о внешней среде, что приводит к отсутствию унификации представления и, соответственно, дополнительным накладным расходам.

2. Большинство современных средств построения многоагентных систем ориентированы на представление знаний агента при помощи узкоспециализированных языков, зачастую не предназначенных для представления знаний в широком смысле. Речь при этом идет как о знаниях агента о себе самом, так и знаниях о внешней среде. В некоторых подходах вначале строится онтология, для создания которой, однако, часто используются средства с низкой выразительной способностью, не предназначенные для построения онтологий [87, 88]. В конечном итоге такой подход приводит к сильной ограниченности возможностей построенных многоагентных систем и их несовместимости.

3. Абсолютное большинство современных средств предполагает, что взаимодействие агентов осуществляется путем обмена сообщениями непосредственно от агента к агенту или посредством специальных коммуникационных центров [90], например, в случае взаимодействия агентов в глобальной сети. Такой подход обладает существенным недостатком, связанным с тем, что в этом случае каждый агент системы должен иметь достаточно полную информацию о других агентах в системе, что приводит к дополнительным затратам ресурсов, кроме того, добавление или удаление одного или нескольких агентов приводит к необходимости оповещения об этом других агентов. Данная проблема решается путем организации общения агентов по принципу «доски объявлений» [91], предполагающему, что сообщения помещаются в некоторую общую для всех агентов область, при этом каждый агент в общем случае может не знать, какому из агентов адресовано сообщение и от какого из агентов получено то или иное сообщение. Кроме того, в построенной таким образом системе легче обеспечивается параллельное решение несвязанных друг с другом задач, поскольку сообщения, относящиеся к одной задаче, будут игнорироваться агентами, решающими другую задачу. Однако данный подход не исключает проблему, связанную с необходимостью разработки специализированного языка взаимодействия агентов, который в общем случае не связан с языком, на котором описываются знания агента о решаемых задачах и окружающей среде.

4. Многие средства построения многоагентных систем построены таким образом, что логический уровень взаимодействия агентов жестко привязан к физическому уровню реализации многоагентной системы. Например, при передаче сообщений от агента к агенту разработчику многоагентной системы необходимо помимо семантически значимой информации указывать ip-адрес компьютера, на котором расположен агент-получатель, кодировку, с помощью которой закодирован текст сообщения, и другую техническую информацию, обусловленную исключительно особенностями текущей реализации средств.

5. В большинстве подходов среда, с которой взаимодействуют агенты, уточняется отдельно разработчиком для каждой многоагентной системы, что с одной стороны, расширяет возможности применения соответствующих средств, но, с другой стороны, приводит к существенным накладным расходам и несовместимости таких многоагентных систем. Кроме того, в ряде случаев разработчик также обязан учитывать особенности технической реализации средств разработки в плане их стыковки с предполагаемой средой, в роли которой может выступать, например, локальная или глобальная сеть.

Перечисленные недостатки можно устранить за счет использования следующих принципов:

1. Коммуникацию агентов предлагается осуществлять по принципу «доски объявлений», однако в отличие от классического подхода в роли сообщений выступают спецификации в общей семантической памяти выполняемых агентами действий (процессов), направленных на решение каких-либо задач, а в роли среды коммуникации выступает сама эта семантическая память. Такой подход позволяет:

- исключить необходимость разработки специализированного языка для обмена сообщениями;
- обеспечить «обезличенность» общения, т. е. каждый из агентов в общем случае не знает, какие еще агенты есть в системе, кем сформулирован и кому адресован тот или иной запрос; таким образом, добавление или удаление агентов в систему не приводит к изменениям в других агентах, что обеспечивает модифицируемость всей системы;
- агентам, в том числе конечному пользователю, формулировать задачи в *декларативном ключе*, т. е. не указывать для каждой задачи способ ее решения; таким образом, агенту заранее не нужно знать, каким образом система решит ту или иную задачу, достаточно лишь специфицировать конечный результат;
- сделать средства коммуникации агентов и синхронизации их деятельности более понятными разработчику и пользователю системы, не требующими изучения специальных низкоуровневых типов данных и форматов

сообщений; таким образом повышается доступность предлагаемых решений широкому кругу разработчиков.

Следует отметить, что такой подход позволяет при необходимости организовать обмен сообщениями между агентами напрямую, а значит, может являться основой для моделирования многоагентных систем, предполагающих другие способы взаимодействия между агентами.

2. В роли внешней среды для агентов выступает та же семантическая память, в которой формулируются задачи и посредством которой осуществляется взаимодействие агентов. Такой подход обеспечивает унификацию среды для различных систем агентов, что, в свою очередь, обеспечивает их совместимость.

3. Спецификация каждого агента описывается средствами SC-кода в той же семантической памяти, что позволяет:

- минимизировать число специализированных средств, необходимых для спецификации агентов, как языковых, так и инструментальных;
- с одной стороны – минимизировать необходимую в общем случае спецификацию агента, которая включает условие его инициирования и программу, описывающую алгоритм работы агента, с другой стороны – обеспечить возможность произвольного расширения спецификации для каждого конкретного случая, в том числе возможность реализации модели BDI и др.

4. Синхронизацию деятельности агентов предполагается осуществлять на уровне выполняемых ими процессов, направленных на решение тех или иных задач в семантической памяти. Таким образом, каждый агент трактуется как некий абстрактный процессор, способный решать задачи определенного класса. При таком подходе необходимо решить задачу обеспечения взаимодействия параллельных асинхронных процессов в общей семантической памяти, для решения которой можно заимствовать и адаптировать решения, применяемые в традиционной линейной памяти. При этом вводится дополнительный класс агентов – метаагенты, задачей которых является решение возникающих проблемных ситуаций, таких как взаимоблокировки.

5. Каждый информационный процесс в любой момент времени имеет ассоциативный доступ к необходимым фрагментам базы знаний, хранящейся в семантической памяти, за исключением фрагментов, заблокированных другими процессами в соответствии с механизмом синхронизации выполнения параллельных процессов. Таким образом, с одной стороны, исключается необходимость хранения каждым агентом информации о внешней среде, с

другой стороны, каждый агент, как и в классических многоагентных системах, обладает только частью всей информации, необходимой для решения задачи.

Важно отметить, что в общем случае невозможно априори предсказать, какие именно знания, модели и способы решения задач понадобятся системе для решения конкретной задачи. В связи с этим необходимо, с одной стороны, обеспечить возможность доступа ко всем необходимым фрагментам базы знаний (в пределе – ко всей базе знаний), с другой стороны, иметь возможность локализовать область поиска пути решения задачи, например, рамками одной предметной области [53].

Каждый из агентов обладает набором ключевых элементов (как правило, понятий), которые он использует в качестве отправных точек при ассоциативном поиске в рамках базы знаний. Набор таких элементов для каждого агента уточняется на этапах проектирования многоагентной системы в соответствии с методикой, рассматриваемой в подразделе 2.1. Уменьшение числа ключевых элементов агента делает его более универсальным, однако снижает эффективность его работы за счет необходимости выполнения дополнительных операций.

Таким образом, модель гибридного решателя задач, построенная на основе многоагентного подхода, представляет собой коллектив агентов обработки знаний, которые обмениваются сообщениями исключительно путем спецификации информационных процессов в рамках той же семантической памяти, в которой хранятся обрабатываемые ими знания. В свою очередь, агенты, входящие в состав гибридного решателя, также могут являться решателями частного вида, которые далее декомпозируются по тем же принципам.

Кроме перечисленных в начале данного пункта достоинств, такой подход позволяет рассматривать гибридный решатель задач как иерархическую систему. Некий целостный коллектив агентов, реализующий какую-либо подсистему гибридного решателя (например, машину дедуктивного вывода, подсистему верификации базы знаний и т. д.), может рассматриваться как единый неатомарный агент, поскольку коллективы агентов и отдельные агенты работают в соответствии с одними и теми же принципами.

1.4.2 Понятие абстрактного sc-агента и его семантическая классификация

Рассмотрим более детально реализацию многоагентного подхода, на основе которого строится рассматриваемая модель гибридного решателя. Единственным видом *субъектов*, выполняющих преобразования в *sc-памяти*,

будем считать *sc-агенты*. Для формального определения понятия *sc-агента* воспользуемся введенным ранее понятием *класса логически атомарных действий*. Будем называть *sc-агентом* некоторый *субъект*, способный выполнять *действия в sc-памяти*, принадлежащие некоторому определенному *классу логически атомарных действий*.

Логическая атомарность выполняемых sc-агентом действий предполагает, что каждый sc-агент реагирует на соответствующий ему класс ситуаций и/или событий, происходящих в sc-памяти, и осуществляет определенное преобразование sc-текста (текста SC-кода), находящегося в семантической окрестности обрабатываемой ситуации и/или события. При этом каждый sc-агент в общем случае не имеет информации о том, какие еще sc-агенты в данный момент присутствуют в системе, и осуществляет взаимодействие с другими sc-агентами исключительно посредством формирования некоторых конструкций (как правило, спецификаций действий) в общей sc-памяти. Таким сообщением может быть, например, вопрос, адресованный другим sc-агентам в системе (заранее неизвестно, каким конкретно), или ответ на поставленный другими sc-агентами вопрос (опять же, неизвестно, каким конкретно). Таким образом, каждый sc-агент в каждый момент времени контролирует только фрагмент базы знаний в контексте решаемой данным агентом задачи, состояние всей остальной базы знаний в общем случае непредсказуемо для sc-агента.

Важно отметить, что конечный пользователь ostis-системы с логической точки зрения также выступает в процессе обработки знаний как sc-агент, формирующий в sc-памяти сообщения путем выполнения элементарных действий, предусмотренных пользовательским интерфейсом. Аналогичным образом осуществляется взаимодействие ostis-системы с другими системами и окружающей средой в целом. Вся информация поступает в ostis-систему и из нее исключительно посредством соответствующих sc-агентов интерфейса.

Перечислим некоторые достоинства предлагаемого подхода к организации обработки знаний в sc-памяти:

– поскольку обработка осуществляется агентами, которые обмениваются сообщениями только через общую память, добавление нового агента или исключение (деактивация) одного или нескольких существующих агентов, как правило, не приводит к изменениям в других агентах, поскольку агенты не обмениваются сообщениями напрямую;

– инициирование агентов осуществляется децентрализованно и чаще всего независимо друг от друга; таким образом, даже существенное расширение числа агентов в рамках одной системы не приводит к ухудшению ее производительности;

– спецификации агентов и, как будет показано далее, их программы могут быть записаны на том же языке, что и обрабатываемые знания, что существенно сокращает перечень специализированных средств, предназначенных для проектирования таких агентов и их коллективов, а также их анализа, верификации и оптимизации, и упрощает разработку системы за счет использования более универсальных компонентов.

Поскольку предполагается, что копии одного и того же *sc-агента* или функционально эквивалентные *sc-агенты* могут работать в разных ostis-системах, будучи при этом физически разными sc-агентами, то целесообразно рассматривать свойства и классификацию не sc-агентов, а классов функционально эквивалентных sc-агентов, которые будем называть *абстрактными sc-агентами*.

Таким образом, под *абстрактным sc-агентом* понимается некоторый класс функционально эквивалентных *sc-агентов*, разные экземпляры (т. е. представители) которого могут быть реализованы по-разному.

Каждый *абстрактный sc-агент* имеет соответствующую ему спецификацию. В спецификацию каждого *абстрактного sc-агента* входит:

– указание ключевых *sc-элементов sc-агента*, т. е. тех *sc-элементов*, хранимых в *sc-памяти*, которые для него являются «точками опоры»;

– формальное описание условий инициирования *sc-агента*, т. е. тех *ситуаций* в *sc-памяти*, которые инициируют деятельность данного *sc-агента*;

– формальное описание первичного условия инициирования *sc-агента*, т. е. такой ситуации в *sc-памяти*, которая побуждает *sc-агента* перейти в активное состояние и начать проверку наличия своего полного условия инициирования;

– строгое, полное, однозначно понимаемое описание деятельности данного *sc-агента*, оформленное при помощи каких-либо понятных, общепринятых средств, не требующих специального изучения, например на естественном языке;

– описание результатов выполнения работы соответствующими *sc-агентами*.

Первичным условием инициирования некоторого sc-агента может быть некоторое элементарное изменение (событие) в семантической памяти из числа рассмотренных в пункте 1.2.1.

Классификация *абстрактных sc-агентов* в SCn-коде:

абстрактный sc-агент
*<= разбиение**:
{

- *неатомарный абстрактный sc-агент*
- *атомарный абстрактный sc-агент*

}

*<= разбиение**:

{

- *внутренний абстрактный sc-агент*
- *эффекторный абстрактный sc-агент*
- *рецепторный абстрактный sc-агент*

}

*<= разбиение**:

{

- *абстрактный sc-агент, не реализуемый на языке SCP*
- *абстрактный sc-агент, реализуемый на языке SCP*

}

*<= разбиение**:

{

- *абстрактный sc-агент интерпретации scp-программ*
- *абстрактный программный sc-агент*
- *абстрактный sc-метаагент*

}

*<= разбиение**:

{

- *платформенно-зависимый абстрактный sc-агент*
  *=> включение**:
  абстрактный sc-агент, не реализуемый на языке SCP
- *платформенно-независимый абстрактный sc-агент*

}

абстрактный sc-агент, не реализуемый на языке SCP

= абстрактный sc-агент, который не может быть реализован на платформенно-независимом уровне

*<= разбиение**:

{

- *эффекторный абстрактный sc-агент*
- *рецепторный абстрактный sc-агент*
- *абстрактный sc-агент интерпретации scp-программ*

}

абстрактный sc-агент, реализуемый на языке SCP

= абстрактный sc-агент, который может быть реализован на платформенно-независимом уровне

*<= разбиение**:

{

- *абстрактный sc-метаагент*
- *абстрактный программный sc-агент, реализуемый на языке SCP*

}

абстрактный программный sc-агент
*<= разбиение**:
{
- *эффекторный абстрактный sc-агент*
- *рецепторный абстрактный sc-агент*
- *абстрактный программный sc-агент, реализуемый на языке SCP*

}

С точки зрения реализации можно выделить два класса *абстрактных sc-агентов*:

– под *неатомарным абстрактным sc-агентом* понимается *абстрактный sc-агент*, который декомпозируется на коллектив более простых *абстрактных sc-агентов*, каждый из которых, в свою очередь, может быть как *атомарным абстрактным sc-агентом*, так и *неатомарным абстрактным sc-агентом;* при этом в каком-либо варианте *декомпозиции абстрактного sc-агента** дочерний *неатомарный абстрактный sc-агент* может стать *атомарным абстрактным sc-агентом* и реализовываться соответствующим образом;

– под *атомарным абстрактным sc-агентом* понимается абстрактный sc-агент, для которого уточняется платформа его реализации, т. е. существует соответствующая связка отношения *программа sc-агента**.

В свою очередь, *атомарные абстрактные sc-агенты* подразделяются на *платформенно-независимые абстрактные sc-агенты* и *платформенно-зависимые абстрактные sc-агенты*.

К *платформенно-независимым абстрактным sc-агентам* относят *атомарные абстрактные sc-агенты*, реализованные на *языке SCP*. При описании *платформенно-независимых абстрактных sc-агентов* платформенная независимость понимается с точки зрения технологии OSTIS, т. е. реализации на языке SCP, поскольку *атомарные sc-агенты*, реализованные на указанном языке, могут свободно переноситься с одной платформы интерпретации sc-моделей на другую.

К *платформенно-зависимым абстрактным sc-агентам* относят *атомарные абстрактные sc-агенты*, реализованные ниже уровня sc-моделей, т. е. не на *языке SCP*, а на каком-либо другом языке описания программ. Существуют *sc-агенты*, которые принципиально должны быть реализованы на платформенно-зависимом уровне, например, собственно *sc-агенты* интерпретатора *языка SCP* или рецепторные и эффекторные *sc-агенты*, обеспечивающие взаимодействие с внешней средой.

По отношению к внешней среде *абстрактные sc-агенты* могут быть *внутренними абстрактными sc-агентами, эффекторными абстрактными sc-агентами, рецепторными абстрактными sc-агентами*.

Каждый *внутренний абстрактный sc-агент* обозначает класс *sc-агентов*, которые реагируют на события в *sc-памяти* и осуществляют преобразования исключительно в рамках этой же *sc-памяти*.

Каждый *эффекторный абстрактный sc-агент* обозначает класс *sc-агентов*, которые реагируют на события в *sc-памяти* и осуществляют преобразования во внешней относительно данной *ostis-системы* среде.

Каждый *рецепторный абстрактный sc-агент* обозначает класс *sc-агентов*, которые реагируют на события во внешней относительно данной *ostis-системы* среде и осуществляют преобразования в памяти данной системы.

Существуют *абстрактные sc-агенты*, которые принципиально не могут быть реализованы на языке SCP. С данной точки зрения можно выделить два соответствующих класса *абстрактных sc-агентов*:

– каждый *абстрактный sc-агент, не реализуемый на языке SCP*, должен быть реализован на уровне платформы интерпретации sc-моделей, в том числе аппаратной. К таким *абстрактным sc-агентам* относятся абстрактные sc-агенты интерпретации scp-программ, а также эффекторные и рецепторные абстрактные sc-агенты;

– каждый *абстрактный sc-агент, реализуемый на языке SCP*, может быть реализован на платформенно-независимом уровне, но при необходимости может реализовываться и на уровне платформы, например, с целью повышения производительности.

Отдельное внимание следует уделить классификации абстрактных sc-агентов с точки зрения используемых средств синхронизации деятельности sc-агентов того или иного класса. По данному критерию выделяются три класса абстрактных sc-агентов:

1. К *абстрактным sc-агентам интерпретации scp-программ* относятся нереализуемые на платформенно-независимом уровне *абстрактные sc-агенты*, обеспечивающие интерпретацию *scp-программ* и *scp-метапрограмм*, в том числе создание *scp-процессов*, собственно интерпретацию *scp-операторов*, а также другие вспомогательные действия. По сути, агенты данного класса обеспечивают работу sc-агентов более высоких уровней (программных sc-агентов и sc-метаагентов), реализованных на языке SCP, в частности, обеспечивают соблюдение указанными агентами общих принципов синхронизации, которые будут изложены далее в подразделах 1.6 и 1.7.

2. К *абстрактным программным sc-агентам* относятся все *абстрактные sc-агенты*, обеспечивающие основную функциональность системы, т. е. ее возможность решать те или иные задачи. Агенты данного класса должны работать в соответствии с общими принципами синхронизации деятельности субъектов в sc-памяти.

3. Задачей *абстрактных sc-метаагентов* является координация деятельности *абстрактных программных sc-агентов* (точнее, их элементов, т. е. конкретных *программных sc-агентов*), в частности, решение проблемы взаимоблокировок [92]. Агенты данного класса могут быть реализованы на *языке SCP*, однако для синхронизации их деятельности используются другие принципы, соответственно, для реализации таких агентов требуется *язык SCP* другого уровня, классификация операторов которого полностью аналогична классификации *scp-операторов*, однако эти операторы имеют другую операционную семантику, учитывающую отличия в принципах синхронизации (работы с *блокировками**). Программы такого языка будем называть *scp-метапрограммами*, соответствующие им *процессы в sc-памяти* – *scp-метапроцессами*, операторы – *scp-метаоператорами*.

Классификация *sc-агентов* по отношению к платформе интерпретации sc-моделей, внешней среде ostis-системы и средствам синхронизации их деятельности полностью аналогична соответствующей классификации *абстрактных sc-агентов*, в связи с чем не будем отдельно рассматривать такую классификацию.

Дополнительно вводится класс *активных sc-агентов*. Под *активным sc-агентом* понимается *sc-агент* ostis-системы, который реагирует на события, соответствующие его условию инициирования и, как следствие, его *первичному условию инициирования**. Не входящие во множество *активных sc-агентов sc-агенты* не реагируют ни на какие события в *sc-памяти*.

Для описания деятельности sc-агентов в sc-памяти вводится понятие *процесса* и *процесса в sc-памяти*, которые будут рассмотрены ниже в подразделе 1.6.

Классификация sc-агентов по семантике выполняемых ими преобразований в sc-памяти будет представлена в пункте 1.4.4, посвященном классификации решателей задач.

1.4.3 Формальные средства спецификации абстрактных sc-агентов

Каждый абстрактный sc-агент должен быть специфицирован соответствующим образом, в противном случае функционирование

соответствующих sc-агентов станет невозможным. Рассмотрим ряд отношений, используемых для спецификации абстрактных sc-агентов.

Связки отношения *ключевые sc-элементы sc-агента** связывают между собой *sc-узел*, обозначающий *абстрактный sc-агент*, и *sc-узел*, обозначающий множество *sc-элементов*, которые являются ключевыми для данного *абстрактного sc-агента*, т. е. данные *sc-элементы* являются элементами программ, реализующих данный *абстрактный sc-агент*.

Связки отношения *программа sc-агента** связывают между собой *sc-узел*, обозначающий *атомарный абстрактный sc-агент*, и *sc-узел*, обозначающий множество программ, реализующих указанный *атомарный абстрактный sc-агент*.

В случае *платформенно-независимого абстрактного sc-агента* каждая связка отношения *программа sc-агента** связывает *sc-узел*, обозначающий указанный *абстрактный sc-агент*, с множеством *scp-программ*, описывающих деятельность данного *абстрактного sc-агента*. Данное множество содержит одну *агентную scp-программу* и произвольное количество (может быть и ни одной) *scp-программ*, которые необходимы для выполнения указанной *агентной scp-программы*. Понятие *агентной scp-программы* будет подробнее рассмотрено в пункте 1.5.1.

В случае *платформенно-зависимого абстрактного sc-агента* каждая связка отношения *программа sc-агента** связывает *sc-узел*, обозначающий указанный *абстрактный sc-агент*, с множеством файлов, содержащих исходные тексты программы на некотором внешнем языке программирования, реализующей деятельность данного *абстрактного sc-агента*.

Связки отношения *первичное условие инициирования** связывают между собой *sc-узел*, обозначающий *абстрактный sc-агент*, и бинарную ориентированную пару, описывающую первичное условие инициирования данного *абстрактного sc-агента*, т. е. такую спецификацию *ситуации* в *sc-памяти*, возникновение которой побуждает *sc-агента* перейти в активное состояние и начать проверку наличия своего полного условия инициирования.

Связки отношения *условие инициирования и результат** связывают между собой *sc-узел*, обозначающий *абстрактный sc-агент*, и бинарную ориентированную пару, связывающую условие инициирования данного *абстрактного sc-агента* и результаты выполнения экземпляров данного sc-агента в какой-либо конкретной системе.

Кроме описанных отношений спецификация *абстрактного sc-агента* также включает *описание поведения sc-агента*, которое представляет собой

семантическую окрестность [52], описывающую деятельность sc-агента до какой-либо степени детализации.

Строго говоря, для того чтобы элементы некоторого *абстрактного sc-агента* (конкретные *sc-агенты*) могли работать (реагировать на соответствующие события в sc-памяти и выполнять соответствующую программу), достаточно описать *первичное условие инициирования** и *программу sc-агента** для данного абстрактного sc-агента. Остальные фрагменты спецификации могут быть использованы для дополнительного анализа соответствующего агента другими агентами, для дидактических целей и т. д.

Образец формальной спецификации *абстрактного sc-агента* на примере *абстрактного sc-агента поиска структуры, изоморфной заданному образцу*, показан на рисунке 1.8

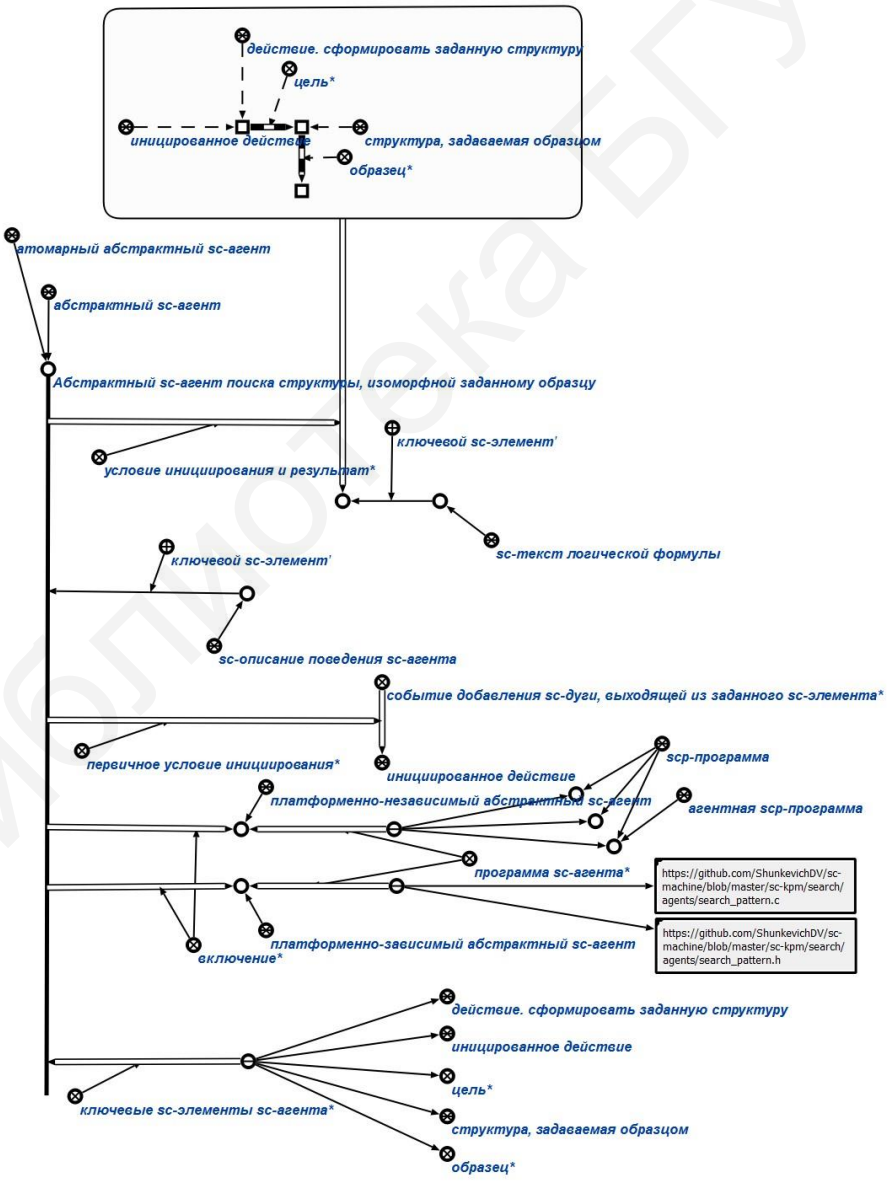

Рисунок 1.8 – Пример спецификации абстрактного sc-агента

1.4.4 Агентно-ориентированная модель гибридного решателя задач

Семантическая агентно-ориентированная модель гибридного решателя задач (sc-модель гибридного решателя задач) строится на основании всех принципов, рассмотренных ранее.

Пользуясь введенной терминологией, будем считать, что *sc-модель гибридного решателя задач* представляет собой неатомарный абстрактный sc-агент, являющийся результатом объединения всех абстрактных sc-агентов, необходимых для решения задач некоторого класса (в том числе комплексных), в один. Другими словами, под *sc-моделью гибридного решателя задач* понимается коллектив всех sc-агентов, необходимых для решения задач данного класса, воспринимаемый как единое целое.

Таким образом, графически принцип обработки информации в ostis-системе можно проиллюстрировать рисунком 1.9. В левой части рисунка показан пример решателя, который декомпозируется на sc-агенты (атомарные и неатомарный), каждому из которых в конечном итоге соответствует некоторая scp-программа. В свою очередь, каждой из программ может соответствовать несколько процессов в sc-памяти, которые показаны в правой части рисунка. Каждый из процессов специфицируется в той же памяти, в частности, указываются соответствующие процессу блокировки.

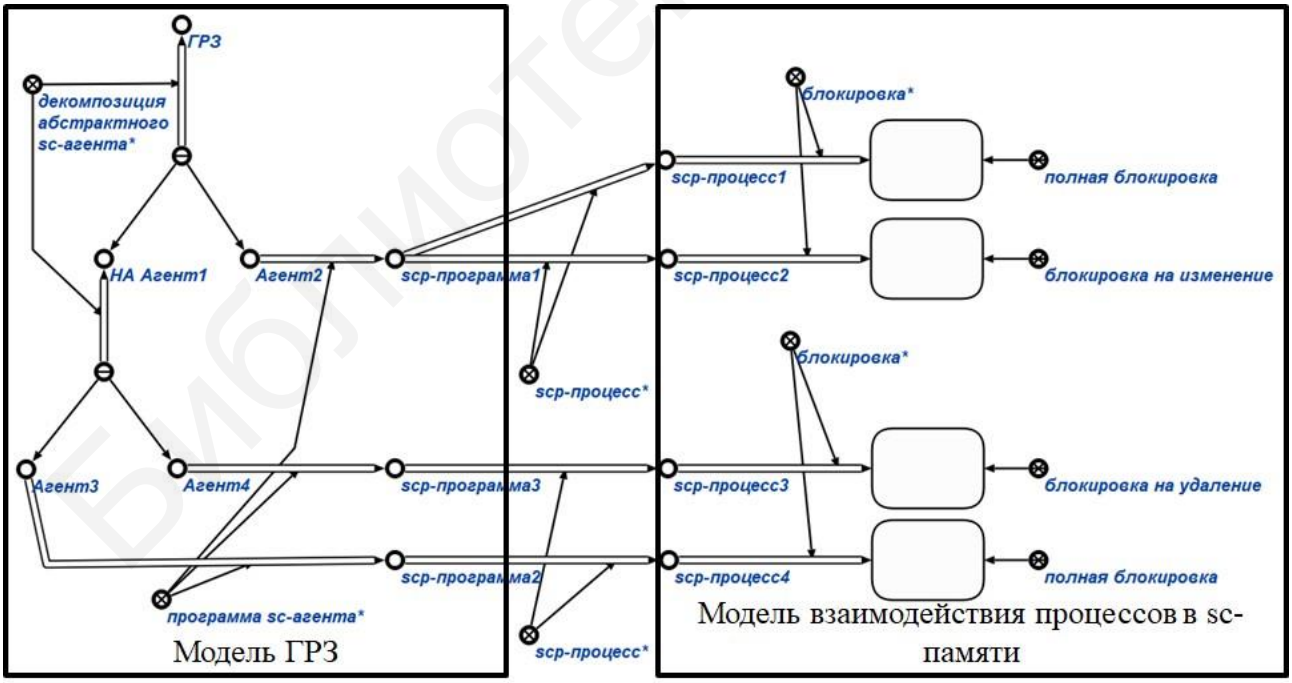

Рисунок 1.9 – Детализированная схема обработки информации в ostis-системе

Выделяют несколько уровней детализации гибридного решателя задач: уровень агентов scp-интерпретатора, реализуемых на уровне платформы; уровень атомарных программных sc-агентов, реализуемых на уровне платформы; уровень атомарных программных sc-агентов, реализуемых на языке SCP; уровень неатомарных программных sc-агентов, иерархия которых в общем случае может иметь произвольное число уровней, самым верхним из которых является уровень самого гибридного решателя, который также трактуется как неатомарный sc-агент. При этом некоторые sc-агенты могут входить в состав нескольких неатомарных sc-агентов, в том числе расположенных на разных уровнях иерархии. Благодаря этому устраняется дублирование функционально эквивалентных агентов в составе решателя.

Графически такая иерархия может быть проиллюстрирована рисунком 1.10. Как видно из рисунка, предложенный подход позволяет абстрагировать принципы построения решателя как системы sc-агентов от того, каким образом реализована программа того или иного атомарного абстрактного sc-агента в такой системе.

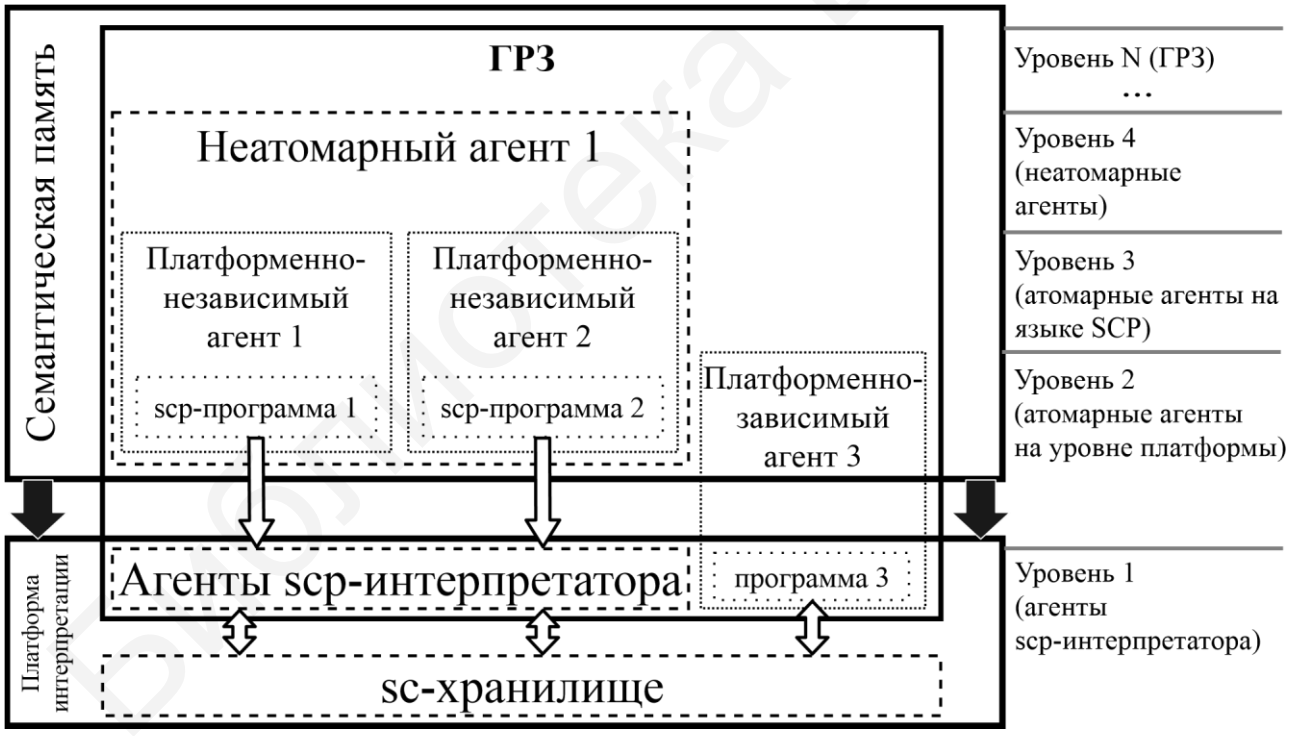

Рисунок 1.10 – Иерархическая агентно-ориентированная модель решателя

Кроме того, такая иерархия уровней, во-первых, обеспечивает возможность поэтапного проектирования гибридного решателя задач с постепенным повышением степени детализации от верхнего уровня к нижнему, а во-вторых, – возможность проектирования, отладки и верификации

компонентов на разных уровнях независимо от других уровней, что существенно упрощает задачу построения и модификации гибридного решателя за счет снижения накладных расходов. Заметим, что в соответствии с рассмотренной моделью частные решатели в составе гибридного строятся на основе той же модели и по тем же принципам.

Кроме того, на каждом из уровней существует вероятность того, что часть или даже все необходимые компоненты уже были реализованы кем-либо ранее и могут быть использованы повторно в разрабатываемом решателе. Более подробно методика компонентного проектирования решателей будет рассмотрена далее в подразделе 2.1.

Классификацию решателей задач можно осуществлять по нескольким критериям. Рассмотрим несколько вариантов такой классификации, записанных в SCn-коде.

По типу соответствующей компьютерной системы:

решатель задач
*=> включение**:
- *решатель задач IMS*
- *решатель задач вспомогательной компьютерной системы*
  *=> включение**:
  - *решатель задач интерфейса компьютерной системы*
    *=> включение**:
    - *решатель задач пользовательского интерфейса компьютерной системы*
    - *решатель задач интерфейса компьютерной системы с другими компьютерными системами*
    - *решатель задач интерфейса компьютерной системы с окружающей средой*
  - *решатель задач подсистемы поддержки проектирования компонентов определенного класса*
    *=> включение**:
    - *решатель задач подсистемы поддержки проектирования баз знаний*
      *=> включение**:
      машина повышения качества базы знаний
      *=> включение**:
      - *машина верификации базы знаний*
        *=> включение**:
        - *машина поиска и устранения некорректностей*
        - *машина поиска и устранения неполноты*
      - *машина оптимизации базы знаний*
      - *машина выявления и устранения информационного мусора*
    - *решатель задач подсистемы поддержки проектирования решателей*
      *=> включение**:
      - *решатель задач подсистемы поддержки проектирования программ обработки знаний*

- *решатель задач подсистемы поддержки проектирования агентов обработки знаний*

- *решатель задач подсистемы управления проектирования компьютерных систем и их компонентов*

- *решатель задач самостоятельной компьютерной системы*

По типу интерпретируемой модели решения задач:

решатель задач
*=> включение**:

- *машина информационного поиска*
  *=> включение**:
  - *машина информационного поиска информации, удовлетворяющей заданной спецификации*
  - *машина информационного поиска информации, не удовлетворяющей заданной спецификации*
  - *машина, выявляющая отсутствие информации, удовлетворяющей заданной спецификации*
- *решатель задач с использованием хранимых программ*
  *=> включение**:
  - *интерпретатор нейросетевых моделей*
  - *интерпретатор генетических алгоритмов*
  - *интерпретатор императивных программ*
    *=> включение**:
    - *интерпретатор процедурных программ*
    - *интерпретатор объектно-ориентированных программ*
  - *интерпретатор декларативных программ*
    *=> включение**:
    - *интерпретатор логических программ*
    - *интерпретатор функциональных программ*
- *решатель задач в условиях, когда программа решения не известна*
  *=> включение**:
  - *решатель, реализующий поиск решения задачи в глубину*
  - *решатель, реализующий поиск решения задачи в ширину*
  - *решатель, реализующий метод проб и ошибок*
  - *решатель, реализующий метод разбиения задачи на подзадачи*
  - *решатель, реализующий метод решения задач по аналогии*
  - *решатель, реализующий метод сведения условия задачи к языку логики предикатов первого порядка*
  - *машина логического вывода*
    *=> включение**:
    - *машина дедуктивного вывода*
      *=> включение**:
      - *машина прямого дедуктивного вывода*
      - *машина обратного дедуктивного вывода*

- *машина индуктивного вывода*
- *машина абдуктивного вывода*
- *машина нечеткого вывода*
- *машина вывода на основе логики умолчаний*
- *машина темпорального логического вывода*

По объекту обработки (цели решения задачи):

решатель задач
*=> включение**:
- *решатель явно сформулированных задач*
  *=> включение**:
  - *машина поиска значений заданного множества величин*
  - *машина установления истинности заданного логического высказывания в рамках заданной формальной теории*
  - *машина формирования способа решения указанной задачи*
    *=> включение**:
    - *машина формирования доказательства заданного высказывания в рамках заданной формальной теории*
  - *машина верификации ответа на указанную задачу*
  - *машина верификации способа решения указанной задачи*
    *=> включение**:
    - *машина верификации доказательства заданного высказывания в рамках заданной формальной теории*
- *машина классификации сущностей*
  *=> включение**:
  - *машина соотнесения сущности с одним из заданного множества классов*
  - *машина разделения множества сущностей на классы по заданному множеству признаков*
- *машина синтеза естественно-языковых текстов*
- *машина анализа естественно-языковых текстов*
- *машина синтеза сигналов*
- *машина анализа сигналов*
- *машина обработки мультимедийных данных*

Поскольку каждый решатель согласно приведенной выше модели представляет собой sc-агент (как правило, неатомарный), то приведенная классификация решателей задач является одновременно и классификацией sc-агентов по семантике выполняемых ими преобразований в семантической памяти.

1.5 БАЗОВАЯ МАШИНА ОБРАБОТКИ ЗНАНИЙ

В качестве базового языка для описания программ, реализующих деятельность sc-агентов в рамках *sc-памяти*, предлагается *язык SCP*, разрабатываемый в рамках технологии OSTIS. *Язык SCP* ориентирован на обработку унифицированных семантических сетей, представленных в *SC-коде*.

Базовая модель обработки текстов *SC-кода* включает в себя:

– модель предметной области *scp-программ*, в которую включаются все тексты *scp-программ* и в которой исследуется классификация операторов этих программ и средства их спецификации;

– модель предметной области агентов интерпретатора *scp-программ (Абстрактной scp-машины)*, который является частью платформы интерпретации sc-моделей (термин *абстрактная* в данном случае, как и в случае с *абстрактным sc-агентом*, показывает, что разрабатывается семантическая модель scp-интерпретатора, включающая спецификацию каждого из агентов в его составе, которая может в дальнейшем быть реализована в рамках какой-либо платформы интерпретации sc-моделей, в том числе аппаратной).

Онтология языка SCP построена на основе общей онтологии описания деятельности различных субъектов. Такой подход, во-первых, позволяет сократить количество ключевых sc-элементов, уникальных для онтологии языка SCP, и, соответственно, упрощает его изучение, во-вторых, обеспечивает возможность построения на базе языка SCP языков, операторы которых описывают более сложные преобразования в sc-памяти, при этом общие принципы интерпретации остаются неизменными. Кроме того, при таком подходе при каждой новой попытке интерпретации scp-программы будет создаваться отдельный независимый scp-процесс, являющийся, по сути, копией scp-программы с учетом известных входных данных. Такой подход к интерпретации имеет ряд достоинств, о которых будет сказано далее в пункте 1.5.3.

Операторы языка SCP (*scp-операторы*) трактуются как знаки элементарных действий в sc-памяти, которые должны интерпретироваться интерпретатором scp-программ, который является частью *платформы интерпретации sc-моделей*. В свою очередь, *scp-процесс* трактуется как некоторое *действие в sc-памяти*, в декомпозиции которого могут присутствовать только знаки scp-операторов. Данный факт гарантирует, что каждый *scp-процесс* может быть интерпретирован интерпретатором *scp-программ*.

Кроме того, выделены два уровня языка SCP, первый из которых обеспечивает возможность реализации программных sc-агентов (т. е. уровень, использующий рассматриваемый в разделе 1.7 механизм блокировок), второй – sc-метаагентов (т. е. уровень, не использующий механизм блокировок). Операторы языка второго уровня во многом аналогичны операторам языка первого уровня, однако они имеют другую операционную семантику, которая не учитывает блокировки и в рамках которой они рассматриваются как обычные конструкции SC-кода.

В рамках базовой модели обработки текстов *SC-кода* выделяются дополнительные классы *структур* (sc-конструкций) [52], на работу с которыми ориентированы те или другие классы *scp-операторов*.

Представим в виде SCn-кода описание классификации структур с точки зрения базовой модели их обработки:

структура
*<= разбиение**:
{
- *sc-конструкция нестандартного вида*
- *sc-конструкция стандартного вида*
  *<= разбиение**:
  {
  - *одноэлементная sc-конструкция*
  - *трехэлементная sc-конструкция*
  - *пятиэлементная sc-конструкция*

  }

}

На рисунках 1.11–1.14 приведены примеры конструкций каждого типа в SCg-коде.

Каждая *sc-конструкция нестандартного вида* состоит из произвольного количества *sc-элементов* произвольного типа.

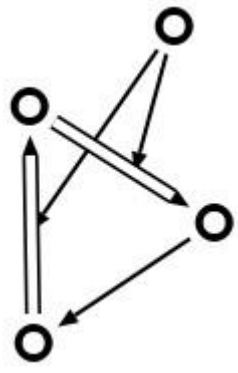

Рисунок 1.11 – Пример sc-конструкции нестандартного вида в SCg-коде

В свою очередь, каждый элемент *sc-конструкции стандартного вида* имеет свою условную строго фиксированную позицию в рамках этой sc-конструкции (первый элемент, второй элемент и т. д.). В зависимости от указанной позиции вводятся дополнительные ограничения на тип соответствующего *sc-элемента*.

Каждая *одноэлементная sc-конструкция* состоит из одного *sc-элемента* произвольного типа.

Рисунок 1.12 – Пример одноэлементных sc-конструкций в SCg-коде

Каждая *трехэлементная sc-конструкция* состоит из трех *sc-элементов*. Второй элемент всегда является *sc-коннектором*, остальные элементы могут быть произвольного типа.

Рисунок 1.13 – Пример трехэлементной sc-конструкции в SCg-коде

Каждая *пятиэлементная sc-конструкция* состоит из пяти *sc-элементов*. Второй и четвертый элементы обязательно являются *sc-коннекторами*, остальные элементы могут быть произвольного типа.

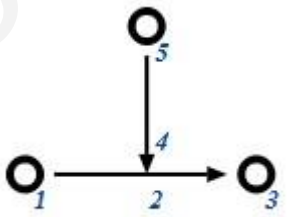

Рисунок 1.14 – Пример пятиэлементной sc-конструкции в SCg-коде

1.5.1 Понятие scp-программы и scp-процесса

Каждая *scp-программа* представляет собой *обобщенную структуру*, описывающую один из вариантов декомпозиции действий некоторого класса, выполняемых в sc-памяти. Знак *sc-переменной*, соответствующей конкретному декомпозируемому действию, является в рамках *scp-программы ключевым sc-элементом'*. Также явно указывается принадлежность данного знака множеству *scp-процессов*.

Таким образом, каждая *scp-программа* описывает в обобщенном виде декомпозицию некоторого *scp-процесса* на взаимосвязанные *scp-операторы* с указанием аргументов для данного *scp-процесса* при их наличии.

По сути каждая *scp-программа* представляет собой описание последовательности элементарных операций, которые необходимо выполнить над семантической сетью, чтобы выполнить более сложное действие некоторого класса.

Частным случаем *scp-программ* являются *агентные scp-программы. Scp-программы* данного класса представляют собой реализации программ агентов обработки знаний и имеют жестко фиксированный набор параметров. Каждая такая программа имеет ровно два *in-параметра'* (данное понятие описано в пункте 1.5.2).

Значение первого параметра является знаком бинарной ориентированной пары – второго компонента связки отношения *первичное условие инициирования** для абстрактного *sc-агента*. В множество *программ sc-агента** входит рассматриваемая *агентная scp-программа*.

Значением второго параметра является *sc-элемент*, с которым непосредственно связано событие, в результате возникновения которого был инициирован соответствующий *sc-агент*, т. е., например, сгенерированная либо удаляемая *sc-дуга* или *sc-ребро*.

В свою очередь, под *scp-процессом* понимается некоторое *действие в sc-памяти*, однозначно описывающее конкретный акт выполнения некоторой *scp-программы* для заданных исходных данных. Если *scp-программа* описывает алгоритм решения какой-либо задачи в общем виде, то *scp-процесс* обозначает конкретное действие, реализующее данный алгоритм для заданных входных параметров.

По сути *scp-процесс* представляет собой копию, созданную на основе *scp-программы*, в которой каждой *sc-переменной*, за исключением *scp-переменных'*, соответствует сгенерированная *sc-константа*.

Принадлежность некоторого действия множеству *scp-процессов* по определению гарантирует тот факт, что в декомпозиции данного действия будут присутствовать только знаки элементарных действий (*scp-операторов*), которые может интерпретировать scp-интерпретатор.

Пример выполнения scp-процесса, соответствующего scp-программе добавления элемента в канторовское множество, представлен на рисунках 1.15–1.19. В рассмотренном примере считается, что заданный элемент ранее не содержался в заданном множестве и будет добавлен в него после выполнения scp-процесса.

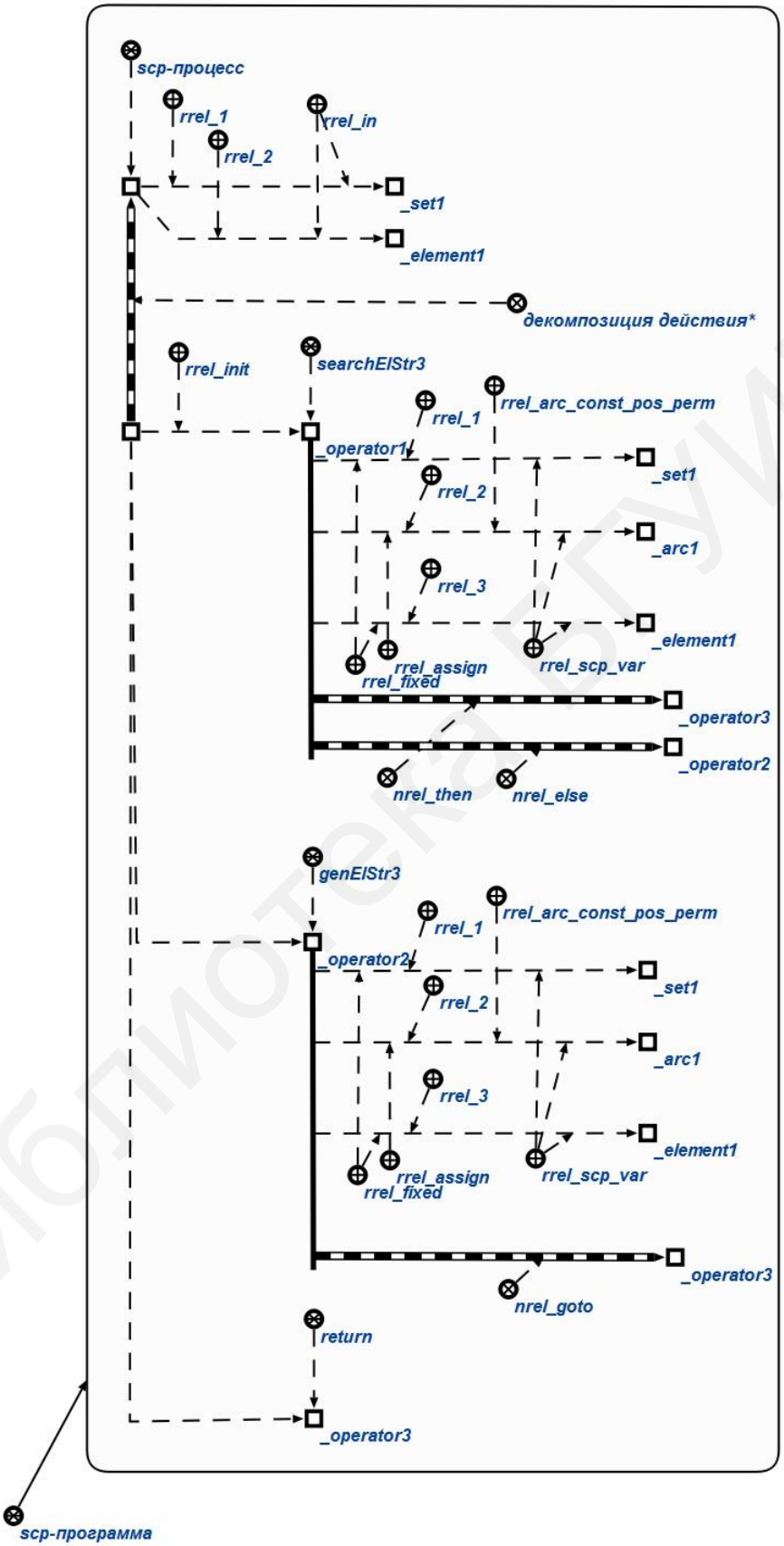

Рисунок 1.15 – Пример scp-программы

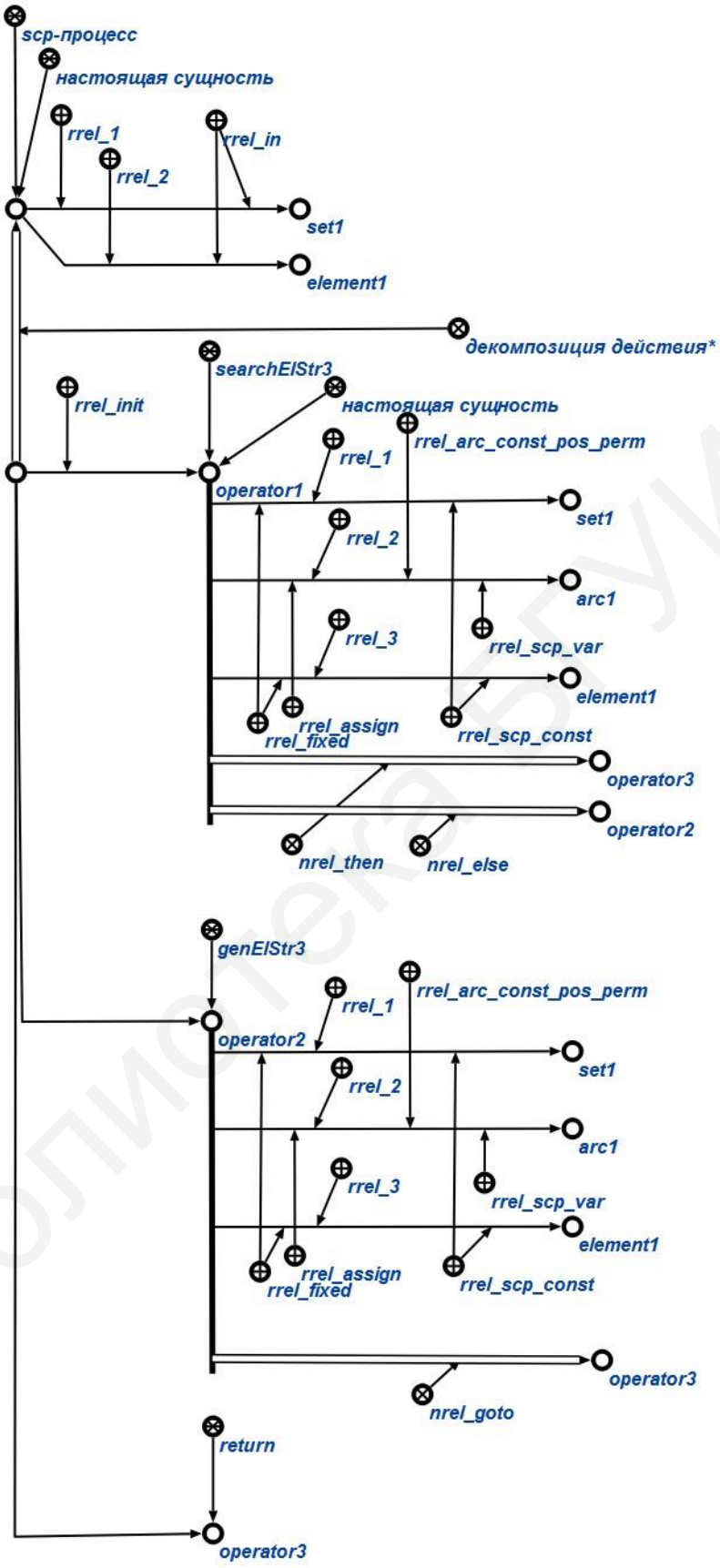

Рисунок 1.16 – Пример scp-процесса на начальной стадии выполнения

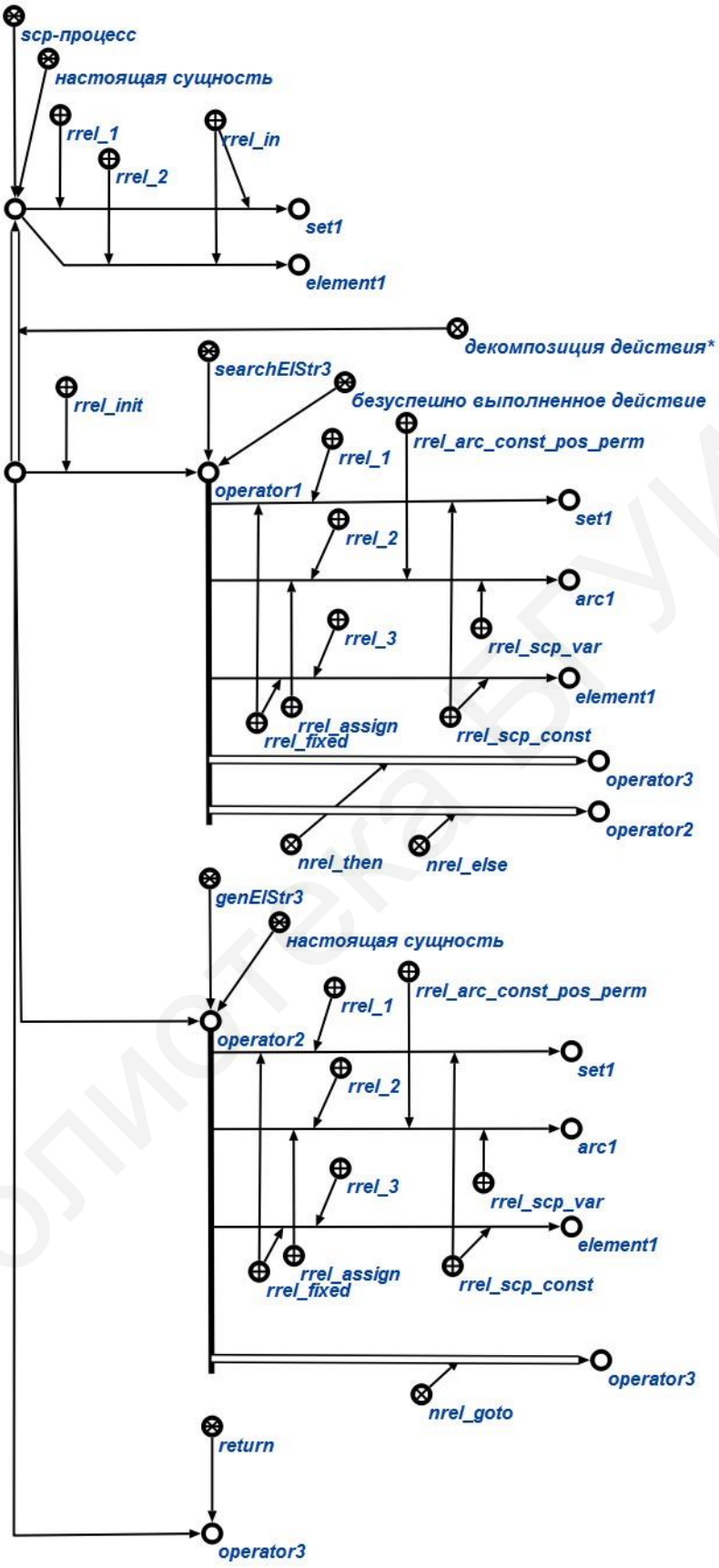

Рисунок 1.17 – Пример scp-процесса: безуспешно выполнен оператор поиска

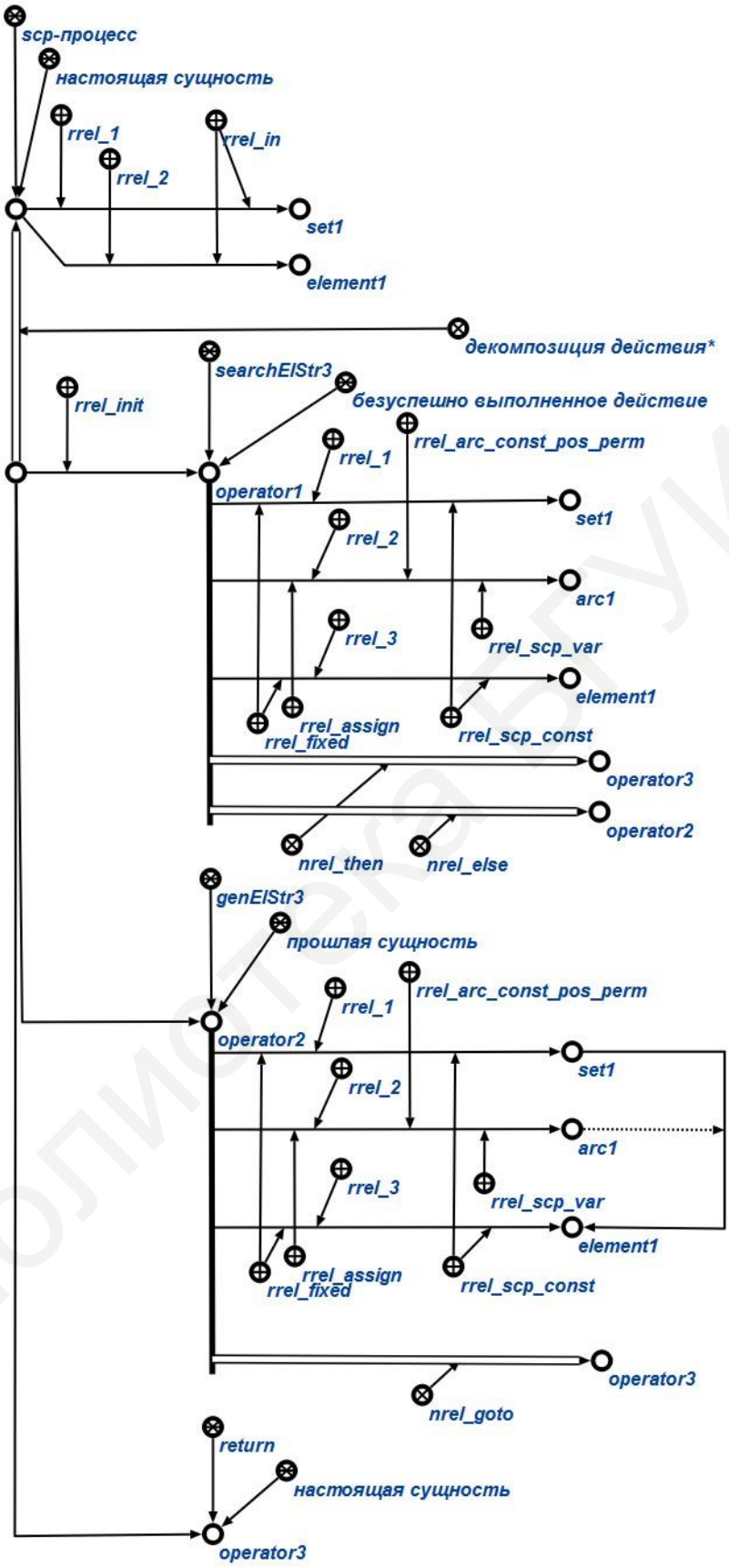

Рисунок 1.18 – Пример scp-процесса: выполнен оператор генерации, элемент добавлен во множество

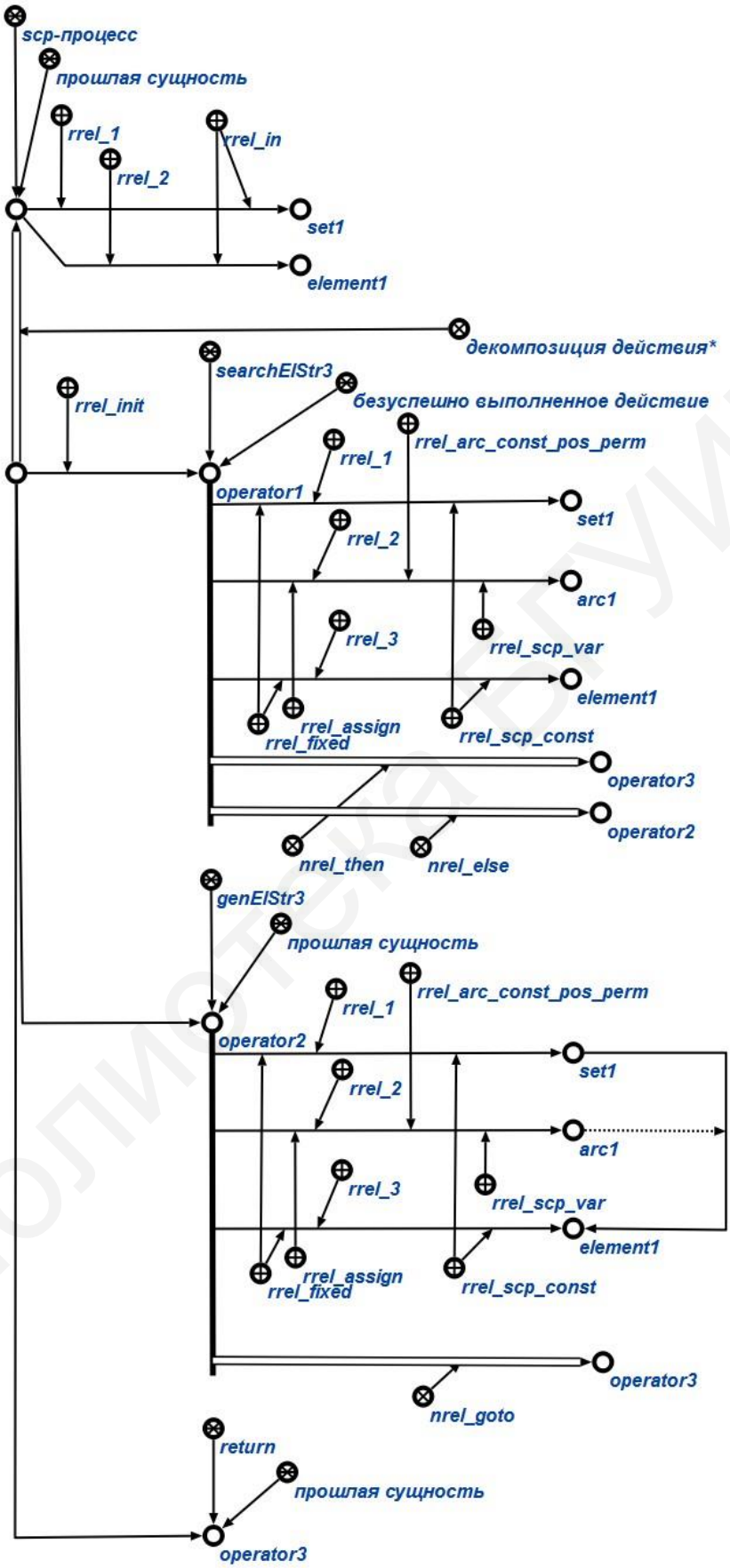

Рисунок 1.19 – Пример scp-процесса: выполнение завершено

Ролевое отношение *начальный оператор'* указывает в рамках декомпозиции соответствующего *scp-программе* действия те поддействия, которые должны быть выполнены в первую очередь, т. е. те, с которых собственно начинается выполнение *scp-процесса*.

Соответствующий *scp-программе scp-процесс* может иметь один и более параметров, каждый из которых имеет свой номер (*1', 2'* и т. д.). В настоящее время выделены два класса параметров (в SCn-коде):

параметр scp-программы'
*<= включение**:
аргумент действия':
*<= разбиение**:
{
- *in-параметр'*
- *out-параметр'*

}

Параметры типа *in-параметр'* хоть и соответствуют *переменным scp-программы'*, не могут менять значение в процессе ее интерпретации. Фиксированное значение переменной устанавливается при создании уникальной копии *scp-программы (scp-процесса)* для ее интерпретации, и, таким образом, соответствующая scp-переменная' на момент начала ее интерпретации становится *scp-константой'* в рамках каждого *scp-оператора*, в котором встречалась данная *scp-переменная'*. Использование *in-параметров* можно рассматривать по аналогии с использованием варианта механизма передачи по значению в традиционных языках программирования, с тем условием что значение локальной переменной в рамках дочерней программы не может быть изменено.

Параметры типа *out-параметр'* соответствуют *переменным scp-программы'* и обладают всеми теми же соответствующими свойствами. Чаще всего предполагается, что значение данного параметра необходимо родительской *scp-программе*, содержащей оператор вызова текущей *scp-программы*. При этом на момент начала интерпретации в качестве параметра дочернему процессу передается непосредственно узел, обозначающий переменную (а точнее, ее уникальную копию в рамках процесса) родительского процесса. Указанная переменная может при необходимости иметь значение либо не иметь. После завершения и во время интерпретации дочернего процесса родительский процесс по-прежнему может работать с переменной, переданной в качестве *out-параметра'*, при необходимости просматривая или изменяя ее значение. Использование

out-параметра можно рассматривать по аналогии с использованием механизма передачи по ссылке в традиционных языках программирования.

1.5.2 Спецификация действий базовой машины обработки

Каждый *scp-оператор* представляет собой некоторое элементарное *действие в sc-памяти*. Рассмотрим подробнее классификацию scp-операторов на языке SCn:

scp-оператор
*<= включение**:
действие в sc-памяти
*<= семейство подмножеств**:
атомарный тип scp-оператора
*<= разбиение**:
{
- *scp-оператор генерации конструкций*
  *<= разбиение**:
  {
  - *scp-оператор генерации конструкции по произвольному образцу*
  - *scp-оператор генерации пятиэлементной конструкции*
  - *scp-оператор генерации трехэлементной конструкции*
  - *scp-оператор генерации одноэлементной конструкции*

  }
- *scp-оператор ассоциативного поиска конструкций*
  *<= разбиение**:
  {
  - *scp-оператор поиска конструкции по произвольному образцу*
  - *scp-оператор поиска пятиэлементной конструкции с формированием множеств*
  - *scp-оператор поиска трехэлементной конструкции с формированием множеств*
  - *scp-оператор поиска пятиэлементной конструкции*
  - *scp-оператор поиска трехэлементной конструкции*

  }
- *scp-оператор удаления конструкций*
  *<= разбиение**:
  {
  - *scp-оператор удаления множества элементов трехэлементной конструкции*
  - *scp-оператор удаления одноэлементной конструкции*
  - *scp-оператор удаления пятиэлементной конструкции*
  - *scp-оператор удаления трехэлементной конструкции*

  }
- *scp-оператор проверки условий*
  *<= разбиение**:
  {

- *scp-оператор сравнения числовых содержимых файлов*
- *scp-оператор проверки равенства числовых содержимых файлов*
- *scp-оператор проверки совпадения значений операндов*
- *scp-оператор проверки наличия содержимого у файла*
- *scp-оператор проверки наличия значения у переменной*
- *scp-оператор проверки типа sc-элемента*

}

- *scp-оператор управления значениями операндов*
  *<= разбиение**:

  {
  - *scp-оператор удаления значения переменной*
  - *scp-оператор присваивания значения переменной*

  }
- *scp-оператор управления scp-процессами*
  *<= разбиение**:

  {
  - *scp-оператор удаления значения переменной*
  - *scp-оператор завершения выполнения программы*
  - *конъюнкция предшествующих scp-операторов*
  - *scp-оператор ожидания завершения выполнения множества scp-программ*
  - *scp-оператор ожидания завершения выполнения scp-программы*
  - *scp-оператор асинхронного вызова подпрограммы*

  }
- *scp-оператор управления событиями*
  *<= разбиение**:

  {
  - *scp-оператор ожидания события*

  }
- *scp-оператор обработки содержимых файлов*
  *<= разбиение**:

  {
  - *scp-оператор вычисления арксинуса числового содержимого файла*
  - *scp-оператор вычисления арккосинуса числового содержимого файла*
  - *scp-оператор деления числовых содержимых файлов*
  - *scp-оператор умножения числовых содержимых файлов*
  - *scp-оператор вычитания числовых содержимых файлов*
  - *scp-оператор сложения числовых содержимых файлов*
  - *scp-оператор вычисления тангенса числового содержимого файла*
  - *scp-оператор вычисления косинуса числового содержимого файла*
  - *scp-оператор вычисления синуса числового содержимого файла*
  - *scp-оператор вычисления логарифма числового содержимого файла*
  - *scp-оператор возведения числового содержимого файла в степень*
  - *scp-оператор удаления содержимого файла*
  - *scp-оператор копирования содержимого файла*

- *scp-оператор нахождения остатка от деления числовых содержимых файлов*
- *scp-оператор нахождения целой части от деления числовых содержимых файлов*
- *scp-оператор вычисления арктангенса числового содержимого файла*
- *scp-оператор перевода в верхний регистр строкового содержимого файла*
- *scp-оператор перевода в верхний регистр строкового содержимого файла*
- *scp-оператор замены определенной части строкового содержимого файла на содержимое указанного файла*
- *scp-оператор проверки совпадения конца строкового содержимого файла со строковым содержимым другого файла*
- *scp-оператор проверки совпадения начальной части строкового содержимого файла со строковым содержимым другого файла*
- *scp-оператор получения части строкового содержимого файла по индексам*
- *scp-оператор поиска строкового содержимого файла в строковом содержимом другого файла*
- *scp-оператор вычисления длины строкового содержимого файла*
- *scp-оператор разбиения строки на подстроки*
- *scp-оператор лексикографического сравнения строковых содержимых файлов*
- *scp-оператор проверки равенства строковых содержимых файлов*

}

- *scp-оператор управления блокировками*
  *<= разбиение**:
  {
  - *scp-оператор снятия всех блокировок данного scp-процесса*
  - *scp-оператор снятия блокировки с sc-элемента*
  - *scp-оператор установки полной блокировки на sc-элемент*
  - *scp-оператор установки блокировки на изменение sc-элемента*
  - *scp-оператор установки блокировки на удаление sc-элемента*
  - *scp-оператор снятия блокировки со структуры*
  - *scp-оператор установки полной блокировки на структуру*
  - *scp-оператор установки блокировки на изменение структуры*
  - *scp-оператор установки блокировки на удаление структуры*

  }

Выделяется также множество *атомарный тип scp-оператора*, каждый элемент которого представляет собой класс *scp-операторов*, не разбивающихся на более частные, и, соответственно, напрямую интерпретируется реализацией *Абстрактной scp-машины*.

Аргументы *scp-оператора* будем называть *scp-операндами'*.

Ролевое отношение *scp-операнд'* является неосновным понятием и указывает на принадлежность аргументов *scp-оператору*. Помимо указания какого-либо класса *scp-операндов'* порядок аргументов *scp-оператора* дополнительно уточняется *ролевыми отношениями 1'*, *2'* и т. д.

Классификация scp-операндов, представленная в SCn-коде:

scp-операнд'
*<= включение**:
аргумент действия'
∈ неосновное понятие:
∈ ролевое отношение:
*<= разбиение**:
{
- *scp-константа'*
- *scp-переменная'*

}
*<= разбиение**:
{
- *scp-операнд с заданным значением'*
- *scp-операнд со свободным значением'*

}
*<= разбиение**:
{
- *константный sc-элемент'*
- *переменный sc-элемент'*

}
*=> включение**:
- *формируемое множество'*
  *<= разбиение**:
  {
  - *формируемое множество 1'*
  - *формируемое множество 2'*
  - *формируемое множество 3'*
  - *формируемое множество 4'*
  - *формируемое множество 5'*

  }
- *удаляемый sc-элемент'*
- *тип sc-элемента'*
  *<= разбиение**:
  {
  - *sc-узел'*
    *<= разбиение**:
    {
    - *структура'*
    - *отношение'*
      *=> включение**:
      ролевое отношение'
    - *класс'*

    }
  - *sc-дуга'*
    *<= разбиение**:

{
- *sc-дуга общего вида’*
- *sc-дуга принадлежности’*

*=> включение**:
sc-дуга основного вида’
*<= разбиение**:
{
- *позитивная sc-дуга принадлежности’*
- *негативная sc-дуга принадлежности’*
- *нечеткая sc-дуга принадлежности’*

}
*<= разбиение**:
{
- *временная sc-дуга принадлежности’*
- *постоянная sc-дуга принадлежности’*

}
}
- *sc-ребро’*
- *файл’*

}

В рамках *scp-программы scp-константы’* явно участвуют в *scp-операторах* в качестве элементов (в теоретико-множественном смысле) и напрямую обрабатываются при интерпретации *scp-программы*. Константами в рамках *scp-программы* могут быть *sc-элементы* любого типа, как *sc-константы*, так и *sc-переменные*. Константа в рамках *scp-программы* остается неизменной в течение всего срока интерпретации. Константа *scp-программы* может быть рассмотрена как переменная, значение которой совпадает с самой переменной в каждый момент времени, и изменено быть не может. Таким образом, далее будем считать, что *scp-константа’* и ее значение это одно и то же. Каждый *in-параметр’* при интерпретации каждой конкретной копии *scp-программы* становится *scp-константой’* в рамках всех ее операторов, хотя в исходном теле данной программы в каждом из этих операторов он является *scp-переменной’*.

В рамках *scp-программы scp-переменные’* не обрабатываются явно при интерпретации, обрабатываются значения переменных. Каждая переменная *scp-программы* может иметь одно значение в каждый момент времени, т. е. представляет собой ситуативный *синглетон*, элементом которого является текущее значение *scp-переменной’*. Значение каждой *scp-переменной’* может меняться в ходе интерпретации *scp-программы*. При этом интерпретатор при обработке *scp-оператора* работает непосредственно со значениями

scp-переменных', а не самими *scp-переменными'* (которые также являются узлами той же семантической сети).

Значение операндов, помеченных ролевым отношением *scp-операнд с заданным значением'*, считается заданным в рамках текущего *scp-оператора*. Данное значение учитывается при выполнении *scp-оператора* и остается неизменным после окончания выполнения *scp-оператора*. Каждая *scp-константа'* по умолчанию рассматривается как *scp-операнд с заданным значением'*, в связи с чем явное использование данного ролевого отношения в таком случае является избыточным и в качестве значения рассматривается непосредственно сам операнд. В случае если отношением *scp-операнд с заданным значением'* помечена *scp-переменная'*, то осуществляется попытка поиска значения для данной *scp-переменной'* (ее элемента). Если попытка оказалась безуспешной, то возникает ошибка времени выполнения, которая должна быть обработана соответствующим образом.

Любой *scp-операнд с заданным значением'* независимо от конкретного типа *scp-оператора* может быть *scp-переменной'*.

Значение операндов, помеченных ролевым отношением *scp-операнд со свободным значением'*, считается свободным (не заданным заранее) в рамках текущего *scp-оператора*. В начале выполнения *scp-оператора* связь между *scp-переменной'*, помеченной данным ролевым отношением, и ее элементом (значением) всегда удаляется. В результате выполнения данного оператора может быть либо сгенерировано новое значение *scp-переменной'*, либо не сгенерировано, тогда *scp-переменная'* будет считаться не имеющей значения. Ни одна *scp-константа'* не может быть помечена как *scp-операнд со свободным значением'*, поскольку константа не может изменять свое значение в ходе интерпретации *scp-программы*.

Таблица 1.3 показывает возможные сочетания различных ролевых отношений, указывающих роль операнда в рамках scp-оператора.

Таблица 1.3 – Роли операндов в рамках scp-оператора

Константность	**Тип значения**	
	scp-операнд с заданным значением'	***scp-операнд со свободным значением'***
scp-константа'	Разрешено, может быть опущено	Запрещено
scp-переменная'	Разрешено, значение останется неизменным	Разрешено, значение переменной будет изменено либо потеряно

Ролевое отношение *тип sc-элемента'* используется для уточнения типа *sc-элемента*, выступающего в роли значения некоторого операнда. *Тип sc-элемента'* имеет смысл указывать только для операндов, помеченных как *scp-операнд со свободным значением'*, тогда данное уточнение типа *sc-элемента* будет использовано для сужения области поиска либо уточнения параметров генерации каких-либо конструкций. Значением *scp-операндов с заданным значением'* является конкретный, известный на момент начала выполнения *scp-оператора sc-элемент* с конкретным типом, не зависящим от указания *типа sc-элемента'*, в связи с чем использование ролевого отношения *тип sc-элемента'* в данном случае является некорректным.

Допускается использование комбинаций семантически непротиворечащих друг другу подмножеств указанного отношения. Например, допускается комбинация *константный sc-элемент'* и *sc-дуга общего вида'*, но не допускается комбинация *sc-узел'* и *sc-дуга'*.

Для удобства программирования выделяется также ролевое отношение *sc-дуга основного вида'*, являющееся объединением отношений *константный sc-элемент'*, *позитивная sc-дуга принадлежности'* и *постоянная sc-дуга принадлежности'*.

Ролевое отношение *формируемое множество'* используется для указания того факта, что в результате выполнения *scp-оператора* должно быть сформировано либо дополнено некоторое множество *sc-элементов*, являющееся значением одного из операндов данного *scp-оператора*. При этом если данный операнд помечен как *scp-операнд со свободным значением'*, то множество будет сформировано с нуля (сгенерирован новый *sc-элемент*, обозначающий данное множество), в противном случае уже существующее множество может быть дополнено. Использование данного ролевого отношения предполагает, что при его отсутствии множество бы не формировалось, а значением указанного операнда стал бы произвольный *sc-элемент* из данного множества.

Ролевое отношение *формируемое множество'* без уточнения порядкового номера используется только в *scp-операторах обработки произвольных конструкций*. Для явного указания номера операнда, которому соответствует *формируемое множество'*, используются подмножества данного ролевого отношения, аналогичные ролевым отношениям, задающим порядок элементов в ориентированном множестве (*1'*, *2'*, *3'* и т. д.), например, *формируемое множество 1'*, *формируемое множество 2'* и т. д. Указанные ролевые отношения используются только в *scp-операторах поиска конструкций с формированием множеств*.

Ролевое отношение *удаляемый sc-элемент'* используется для указания тех операндов, значение которых должно быть удалено в процессе выполнения *scp-операторов удаления*. Данным ролевым отношением может быть помечен как *scp-операнд с заданным значением'*, так и *scp-операнд со свободным значением'*. При этом удаляемым *sc-элементом* может быть как *scp-константа'*, так и *scp-переменная'* (в случае *scp-переменной'* удаляется не только связка принадлежности между этой *scp-переменной'* и ее значением, но и непосредственно сам *sc-элемент*, являющийся значением).

Порядок операндов указывается при помощи соответствующих ролевых отношений (*1'*, *2'*, *3'* и т. д.). Операнд, помеченный ролевым отношением *1'*, будем называть первым операндом, помеченный ролевым отношением *2'*, – вторым операндом и т. д. Тип и смысл каждого операнда также уточняется при помощи различных подклассов отношения *scp-операнд'*. В общем случае операндом может быть любой *sc-элемент*, в том числе знак какой-либо *scp-программы*, в том числе самой программы, содержащей данный оператор.

Каждый *scp-оператор* должен иметь один и более операндов. Кроме того, при помощи соответствующих отношений (*последовательность действий** и более частных) должен быть указан тот *scp-оператор* (или несколько scp-операторов), который должен быть выполнен следующим. Исключение из данного правила составляет *scp-оператор завершения выполнения программы*, который не содержит ни одного операнда и после выполнения которого никакие *scp-операторы* в рамках данной программы выполняться не могут.

1.5.3 Достоинства базовой модели обработки знаний

Большинство достоинств базовой модели обработки текстов *SC-кода* имеют место благодаря следующим основным ее особенностям:

– тексты scp-программ записываются при помощи тех же унифицированных семантических сетей, что и обрабатываемая информация;

– подход к описанию и интерпретации *scp-программ* основан на общих принципах описания деятельности различных субъектов, в частности, предполагается создание при каждом вызове *scp-программы* независимого *scp-процесса*.

Перечислим эти достоинства:

– одновременно в общей памяти могут выполняться несколько независимых процессов, при этом процессы, соответствующие одной и той же *scp-программе*, могут выполняться на разных серверах в случае распределенной реализации платформы интерпретации sc-моделей;

– *язык SCP* позволяет осуществлять параллельные асинхронные вызовы подпрограмм (создания подпроцессов в рамках scp-процессов), а также параллельно выполнять scp-операторы в рамках одного *scp-процесса*;

– поскольку *scp-программы* записываются при помощи *SC-кода*, то перенос реализованного на основе языка *SCP sc-агента* из одной системы в другую заключается в простом переносе фрагмента базы знаний без каких-либо дополнительных операций, зависящих от платформы интерпретации sc-моделей;

– тот факт, что спецификации sc-агентов и их программы могут быть записаны на том же языке, что и обрабатываемые знания, существенно сокращает перечень специализированных средств, предназначенных для построения и модификации решателей задач, и упрощает их разработку за счет использования более универсальных компонентов;

– тот факт, что для интерпретации *scp-программы* создается соответствующий ей уникальный *scp-процесс*, позволяет по возможности оптимизировать план его выполнения перед реализацией и даже непосредственно в процессе выполнения без потенциальной опасности испортить общий универсальный алгоритм всей программы; более того, такой подход к проектированию и интерпретации программ позволяет говорить о возможности создания самореконфигурируемых программ.

Нельзя утверждать, что идея создания отдельного независимого процесса на основе некоторой программы при каждом акте ее выполнения является принципиально новой и реализована только в рамках *языка SCP*. Аналогичный подход используется в большинстве современных систем, основанных на архитектуре фон Неймана и, соответственно, ориентированных на работу с традиционной линейной памятью. Однако применение такого подхода в семантической памяти имеет серьезное преимущество, которое заключается в ассоциативности такой памяти. Действительно, в традиционной памяти обращение к данным осуществляется исключительно по адресу. В случае построения реконфигурируемых программ адрес каждого фрагмента, в окрестности которого осуществляется изменение, или хотя бы структура изменяемого процесса должна быть известна процессу, осуществляющему такую реконфигурацию. При этом смысл отдельно взятого фрагмента информации в традиционной памяти практически никогда не может быть определен без заранее известного контекста. Данный факт приводит к большой трудоемкости разработки реконфигурируемых программ, сужению сферы их применения и высокому уровню зависимости программ, осуществляющих реконфигурацию, от изменений, вносимых разработчиками программ, в

которых эта реконфигурация осуществляется. Использование ассоциативной памяти и общей формальной семантической основы для описания программ любого вида позволяет снять эти ограничения. Доступ к элементам в рамках ассоциативной памяти выполняется не на основе адресов, а путем переходов по связям, при этом число ключевых узлов, к которым осуществляется привязка, относительно невелико. Спецификация каждого элемента такой памяти осуществляется путем формирования соответствующих связей, которые впоследствии могут быть проанализированы в рамках стороннего процесса, т. е. семантика каждого элемента может быть установлена любым процессом путем анализа связей этого элемента с другими. Число таких связей логически не ограничено, поэтому каждый элемент может быть специфицирован с необходимой степенью детализации. Кроме того, построение scp-программ и программ более высокого уровня на основе общих формальных средств описания деятельности любого рода субъектов позволяет сделать алгоритмы реконфигурации более универсальными, т. к. общий вид программы и семантика переходов в таком случае остаются неизменными независимо от уровня языка.

1.5.4 Семантическая модель интерпретатора scp-программ

Семантическая модель интерпретатора scp-программ трактуется как множество *sc-агентов*, на которые декомпозируется *Абстрактная scp-машина*, т. е. *атомарных sc-агентов*, реализуемых на платформенно-зависимом уровне. Для обеспечения работы таких агентов помимо базовых средств описания и синхронизации деятельности вводится набор дополнительных ключевых узлов.

Вся *Абстрактная scp-машина* рассматривается как целостный *абстрактный sc-агент*, который может иметь несколько вариантов декомпозиции. Один из вариантов принят за эталонный и описан в данном пункте. При этом одним из возможных вариантов, хоть и не самым эффективным, является вариант реализации всей *Абстрактной scp-машины* как одного *атомарного sc-агента*.

Декомпозиция ***Абстрактной scp-машины***, представленная в SCn-коде:

Абстрактная scp-машина
<= *декомпозиция абстрактного sc-агента**:
{
- *Абстрактный sc-агент создания scp-процессов*
- *Абстрактный sc-агент интерпретации scp-операторов*
- *Абстрактный sc-агент синхронизации процесса интерпретации scp-программ*
- *Абстрактный sc-агент уничтожения scp-процессов*
- *Абстрактный sc-агент синхронизации событий в sc-памяти и ее реализации*

}

Задачей *Абстрактного sc-агента создания scp-процессов* является создание *scp-процессов*, соответствующих заданной *scp-программе*. Данный *sc-агент* активируется при появлении в *sc-памяти инициированного действия*, принадлежащего классу *действие интерпретации scp-программы*.

После проверки *sc-агентом* условия инициирования выполняется создание *scp-процесса* с учетом конкретных параметров интерпретации *scp-программы*, после чего осуществляется поиск *начального оператора' scp-процесса* и добавление его во множество *настоящих сущностей*.

Задачей *Абстрактного sc-агента интерпретации scp-операторов* является собственно интерпретация операторов *scp-программы*, т. е. выполнение в *sc-памяти* действий, описываемых конкретным *scp-оператором*. Данный *sc-агент* активируется при появлении в *sc-памяти scp-оператора*, принадлежащего классу *настоящих сущностей*. После выполнения действия, описываемого *scp-оператором*, *scp-оператор* добавляется во множество *прошлых сущностей*. В случае когда семантика действия, описываемого *scp-оператором*, предполагает возможность ветвления *scp-программы* после выполнения данного *scp-оператора*, то используется одно из подмножеств класса *выполненных действий – безуспешно выполненное действие* или *успешно выполненное действие*.

В текущей реализации *Абстрактный sc-агент интерпретации scp-операторов декомпозируется* на более частные *абстрактные sc-агенты*, каждый из которых однозначно соответствует *атомарному классу scp-операторов*, в связи с чем не будем приводить перечень таких агентов.

Задачей *Абстрактного sc-агента синхронизации процесса интерпретации scp-программ* является обеспечение переходов между *scp-операторами* в рамках одного *scp-процесса*. Данный *sc-агент* активизируется при добавлении некоторого *scp-оператора* во множество *прошлых сущностей*. Далее осуществляется переход по *sc-дуге*, принадлежащей отношению *последовательность действий** (или более частным отношениям, в случае, если *scp-оператор* был добавлен во множество *успешно выполненных действий* или *безуспешно выполненных действий*). При этом очередной *scp-оператор* становится *настоящей сущностью* (активным *scp-оператором*) в том случае, если хотя бы один *scp-оператор*, связанный с ним входящими *sc-дугами*, принадлежащими отношению *последовательность действий** (или более частным отношениям), стал *прошлой сущностью* (или, соответственно, подмножеством прошлых сущностей). В случае когда необходимо дождаться завершения выполнения всех предыдущих операторов, для синхронизации используется оператор класса *конъюнкция предшествующих операторов*.

Задачей *Абстрактного sc-агента уничтожения scp-процессов* является уничтожение *scp-процесса*, т. е. удаление из *sc-памяти* всех *sc-элементов*, его составляющих. Данный *sc-агент* активируется при появлении в *sc-памяти scp-процесса*, принадлежащего множеству *прошлых сущностей*.

При этом уничтожаемый *scp-процесс* необязательно должен быть полностью сформирован. Необходимость уничтожения не до конца сформированного *scp-процесса* может возникнуть в случае, если при создании *scp-процесса* возникли проблемы, не позволяющие продолжить создание *scp-процесса* и его выполнение.

Задачей *Абстрактного sc-агента синхронизации событий в sc-памяти и ее реализации* является обеспечение работы *неатомарных sc-агентов*, реализованных на *языке SCP*.

В свою очередь *Абстрактный sc-агент синхронизации событий в sc-памяти и ее реализации* декомпозируется следующим образом:

Абстрактный sc-агент синхронизации событий в sc-памяти и ее реализации
*<= декомпозиция абстрактного sc-агента**:
{
- *Абстрактный sc-агент трансляции сформированной спецификации sc-события во внутреннее представление*
- *Абстрактный sc-агент обработки sc-события, инициирующего агентную scp-программу*

}

Задачей *Абстрактного sc-агента сформированной спецификации sc-события во внутреннее представление* является трансляция ориентированных пар, описывающих *первичное условие инициирования** некоторого *sc-агента* во внутреннее представление элементарных событий на уровне *sc-хранилища*, при условии, что этот *sc-агент* реализован на платформенно-независимом уровне (с использованием *языка SCP*). Условием инициирования данного *sc-агента* является появление в *sc-памяти* нового элемента множества *активных sc-агентов*, для которого будет найдена и протранслирована соответствующая ориентированная пара.

Задачей *Абстрактного sc-агента обработки sc-события, инициирующего агентную scp-программу*, является поиск *агентной scp-программы*, входящей во множество *программ sc-агента** для каждого *sc-агента*, первичное условие инициирования которого соответствует событию, произошедшему в *sc-памяти*, а также генерация и инициирование действия, направленного на интерпретацию этой программы. В результате работы данного *sc-агента* в

sc-памяти появляется *инициированное действие*, принадлежащее классу *действие интерпретации scp-программы.*

Рассмотрим пример конструкции в sc-памяти, описывающей инициированное действие интерпретации scp-программы с заданными аргументами в SCg-коде (рисунок 1.20).

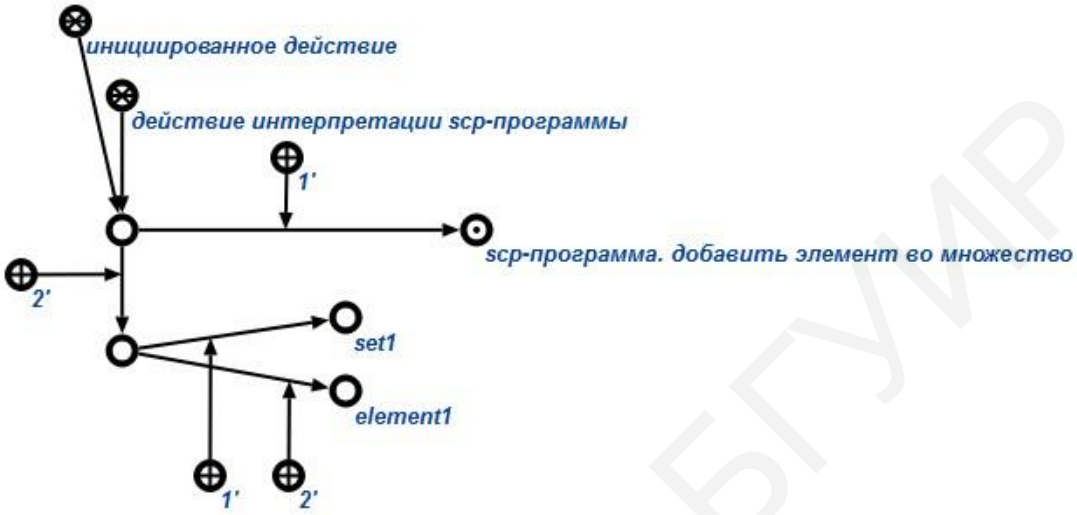

Рисунок 1.20 – Спецификация действия интерпретации scp-программы

Каждое *действие интерпретации scp-программы* имеет два аргумента. Первый *аргумент действия'* обозначает *scp-программу*, на основе которой будет создан интерпретируемый *scp-процесс*, второй *аргумент действия'* обозначает множество аргументов данного процесса. Порядок аргументов в этом множестве уточняется при помощи ролевых отношений *1'*, *2'* и т. д.

В свою очередь, на появление в sc-памяти инициированного *действия интерпретации scp-программы* реагирует *Абстрактный sc-агент интерпретации scp-операторов*, действия которого зависят от конкретного класса, которому принадлежит оператор.

1.6 ПРИНЦИПЫ КОММУНИКАЦИИ АГЕНТОВ В СЕМАНТИЧЕСКОЙ ПАМЯТИ

Основным принципом коммуникации агентов, предлагаемым в рамках технологии OSTIS, является коммуникация посредством спецификации выполняемых агентами процессов в семантической памяти.

Таким образом, чтобы говорить об общих принципах коммуникации агентов в семантической памяти, необходимо рассмотреть классификацию *процессов в sc-памяти*, описывающих конкретные преобразования в указанной

памяти, осуществляемые тем или иным субъектом. Речь в данном случае идет не о классификации, основанной на семантике осуществляемых преобразований (такая классификация представлена в пункте 1.3.4 для класса действий в sc-памяти), а о классификации, связанной с выделением классов процессов, использующих одинаковый механизм синхронизации. В рамках рассматриваемого подхода понятия *действие, выполняемое агентом в семантической памяти*, и *информационный процесс, выполняемый агентом в семантической памяти*, являются синонимичными. В данном учебном пособии будет использоваться термин «процесс», поскольку, когда речь идет о синхронизации выполнения каких-либо преобразований в памяти компьютерной системы, в литературе принято использовать именно термины «процесс», «взаимодействие процессов» [93, 94].

Классификация *процессов в sc-памяти* на языке SCn:

процесс в sc-памяти
*<= разбиение**:
{
- *процесс в sc-памяти, соответствующий платформенно-зависимому sc-агенту*
- *scp-процесс*

}

процесс в sc-памяти, соответствующий платформенно-зависимому sc-агенту
*<= разбиение**:
{
- *процесс в sc-памяти, соответствующий платформенно-зависимому sc-агенту и не являющийся действием абстрактной scp-машины*
- *действие абстрактной scp-машины*
  *=> включение**:
  действие интерпретации scp-программы:

}

scp-процесс
*<= разбиение**:
{
- *scp-процесс, не являющийся scp-метапроцессом*
- *scp-метапроцесс*

}

Под ***scp-метапроцессом*** будем понимать *процесс в sc-памяти*, описывающий деятельность *sc-метаагента*, реализованного на *языке SCP*.

Далее на примере рассмотрим принцип коммуникации агентов в семантической памяти, предлагаемый в рамках технологии OSTIS (рисунки 1.21–1.25). Предположим, что в рамках некоторой системы существует

коллектив из двух sc-агентов, первый из которых умеет выполнять операцию сложения, а второй – операцию умножения. Изначально оба агента находятся в неактивном состоянии. В какой-то момент в системе возникает запрос на вычисление некоторого числа (рисунок 1.21). В данном случае указано, что данное число может быть получено путем сложения двух исходных чисел. Необходимо отметить, что в общем случае не важно, кто и каким образом породил данный запрос, это не влияет на алгоритм действий агентов.

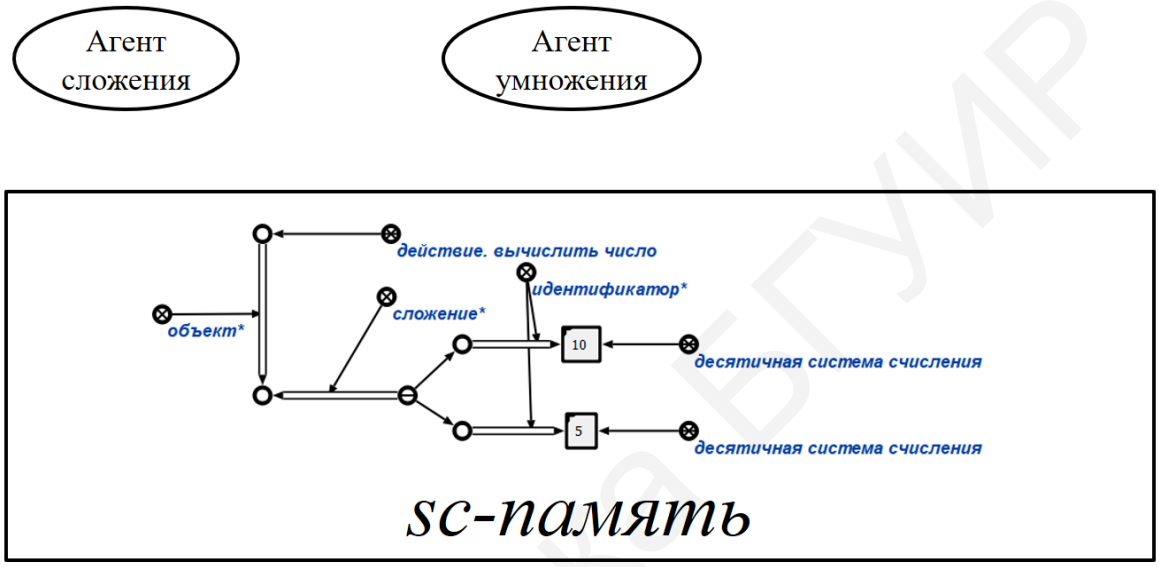

Рисунок 1.21 – Появление запроса

Оба агента реагируют на появление некоторого *инициированного действия* и приступают к анализу окрестности данного действия в базе знаний (рисунок 1.22).

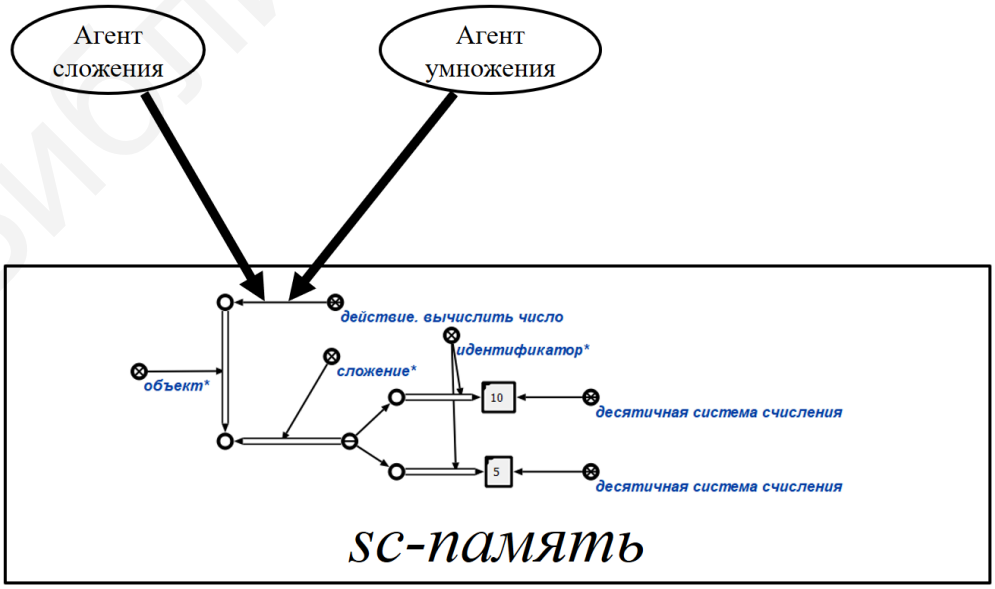

Рисунок 1.22 – Реакция sc-агентов

Проанализировав окрестность действия в базе знаний, агент умножения не находит нужного ключевого узла и снова переходит в неактивное состояние. В свою очередь, агент сложения находит нужное понятие и приступает к работе (рисунок 1.23).

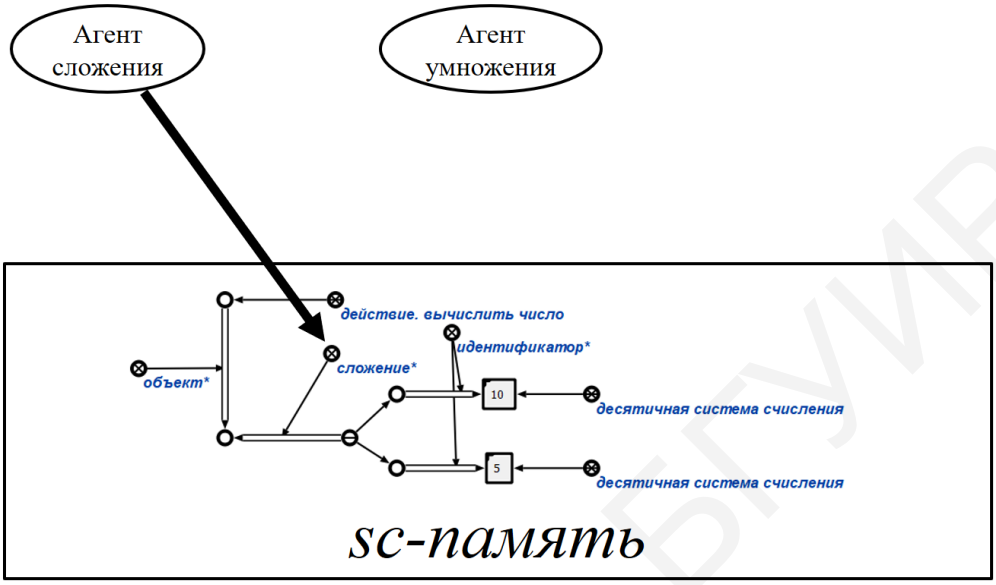

Рисунок 1.23 – Принятие sc-агентом решения о продолжении работы

Вычислив необходимое число, агент сложения должен специфицировать результат своей работы в базе знаний (рисунок 1.24). Выполнив работу, агент переходит в неактивное состояние. Агент или пользователь, создавший запрос на вычисление числа, реагирует на появление ответа и может продолжить свою работу, при этом ему также не важно, кто и как выполнил поставленную задачу.

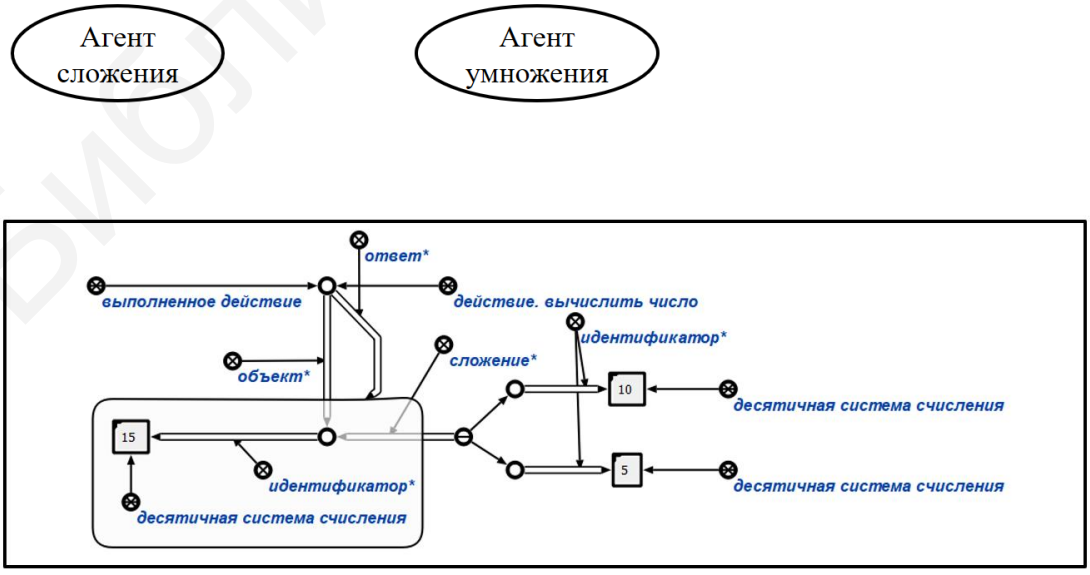

Рисунок 1.24 – Ответ на запрос

Как видно из приведенного примера, в общем случае агенты ничего не знают о других агентах, имеющихся в этой же системе. Такой подход позволяет легко добавить в систему новые агенты, при этом нет необходимости вносить изменения в уже существующие агенты (рисунок 1.25).

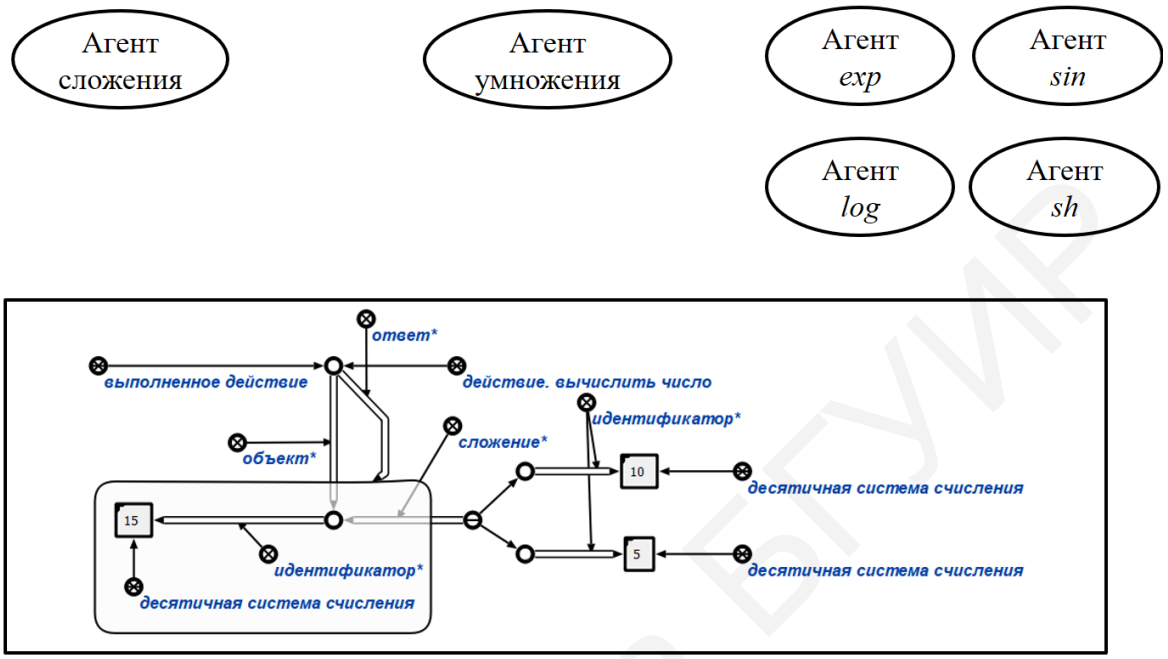

Рисунок 1.25 – Добавление новых sc-агентов

1.7 МЕХАНИЗМ СИНХРОНИЗАЦИИ ВЫПОЛНЕНИЯ ПАРАЛЛЕЛЬНЫХ ПРОЦЕССОВ

На рисунке 1.26 показан пример схемы, отражающей механизм синхронизации выполнения параллельных процессов в семантической памяти.

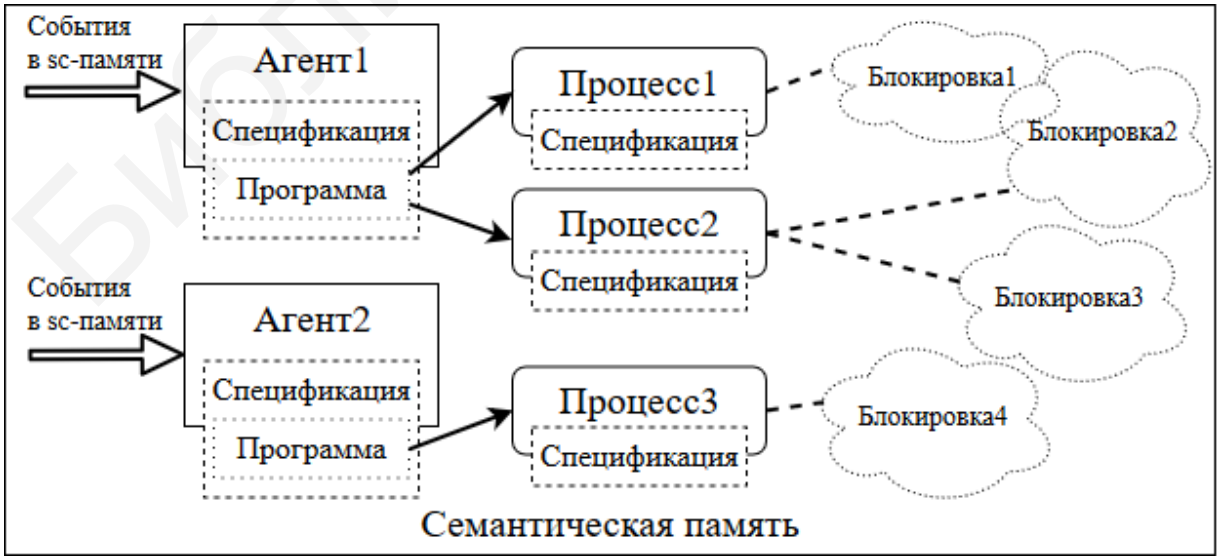

Рисунок 1.26 – Механизм синхронизации выполнения параллельных процессов в семантической памяти

Для синхронизации выполнения *процессов в sc-памяти* используется механизм блокировок. Отношение *блокировка** связывает знаки *действий в sc-памяти* со знаками ситуативных *структур*, которые содержат sc-элементы, заблокированные на время выполнения данного действия или на какую-то часть этого периода. Каждая такая *структура* принадлежит какому-либо из *типов блокировки*. С точки зрения средств синхронизации можно выделить три класса *sc-агентов*:

– *sc-агент интерпретации scp-программ*;

– *программный sc-агент*;

– *sc-метаагент*.

Механизм блокировок, описываемый в данном подразделе, используется для синхронизации деятельности *программных sc-агентов*. К ним относятся все агенты, отвечающие непосредственно за решение задач, поставленных перед конкретной *ostis-системой*, т. е. фактически *sc-агенты* данного класса обеспечивают функциональность *ostis-системы*.

Задачей *sc-агентов интерпретации scp-программ* является обеспечение соблюдения всех изложенных правил взаимодействия (на уровне платформы интерпретации sc-моделей). Принципы синхронизации агентов данного класса более тривиальны, чем в случае *программных sc-агентов*, и были изложены в подразделе 1.5.

Задачей *sc-метаагентов* является решение конфликтов и оптимизация деятельности *программных sc-агентов*.

Для того чтобы придерживаться общепринятой в литературе терминологии, будем говорить, что процесс выполняет некоторое преобразование sc-памяти (например, удаление или генерацию sc-элемента, установку или снятие блокировки), имея в виду, что соответствующее преобразование осуществляется некоторым sc-агентом и является частью некоторого *действия в sc-памяти* (т. е. *процессом в sc-памяти*, который описывает преобразования, выполняемые в sc-памяти неким активным субъектом), с которым указанный sc-агент связан отношением *исполнитель**. В текущей версии для синхронизации выполнения процессов в sc-памяти выделяются три *типа блокировок*:

– *полная блокировка;*

– *блокировка на любое изменение;*

– *блокировка на удаление.*

Каждая структура, принадлежащая множеству ***полная блокировка***, содержит *sc-элементы*, просмотр и изменение (удаление, добавление инцидентных *sc-коннекторов*, удаление самих *sc-элементов*, изменение

содержимого в случае файла) которых запрещены всем *sc-агентам*, кроме собственно *sc-агента*, выполняющего соответствующее данной структуре *действие в sc-памяти*, связанное с ней отношением *блокировка**. По сути sc-элементы, попадающие в полную блокировку, соответствующую некоторому процессу в sc-памяти, временно отсутствуют в текущем состоянии памяти для других процессов.

Для того чтобы исключить возможность реализации *sc-агентов*, которые могут внести изменения в конструкции, описывающие блокировки других *sc-агентов*, все элементы этих конструкций, в том числе сам знак *структуры*, содержащей заблокированные *sc-элементы* (принадлежащей как множеству *полная блокировка*, так и любому другому *типу блокировки*) и связки отношения *блокировка**, связывающие эту *структуру* и конкретное *действие в sc-памяти*, добавляются в полную блокировку, соответствующую данному действию в sc-памяти. Таким образом, каждой *полной блокировке* соответствует петля принадлежности, связывающая ее знак с самим собой.

Каждая *структура*, принадлежащая множеству ***блокировка на любое изменение***, содержит *sc-элементы*, изменение (физическое удаление, добавление инцидентных *sc-коннекторов*, физическое удаление самих *sc-элементов*, изменение содержимого в случае файла) которых запрещено всем *sc-агентам*, кроме собственно *sc-агента*, выполняющего соответствующее данной структуре *действие в sc-памяти*, связанное с ней отношением *блокировка**. Однако не запрещен просмотр (чтение) этих *sc-элементов* любым *sc-агентом*.

Каждая *структура*, принадлежащая множеству ***блокировка на удаление***, содержит *sc-элементы*, физическое удаление которых запрещено всем *sc-агентам*, кроме собственно *sc-агента*, выполняющего соответствующее данной структуре *действие в sc-памяти*, связанное с ней отношением *блокировка**. Однако не запрещен просмотр (чтение) этих *sc-элементов* любым *sc-агентом*, а также добавление или удаление инцидентных *sc-коннекторов*.

Конструкция SCg-кода в sc-памяти, описывающая блокировки, установленные некоторыми sc-агентами, представлена на рисунке 1.27.

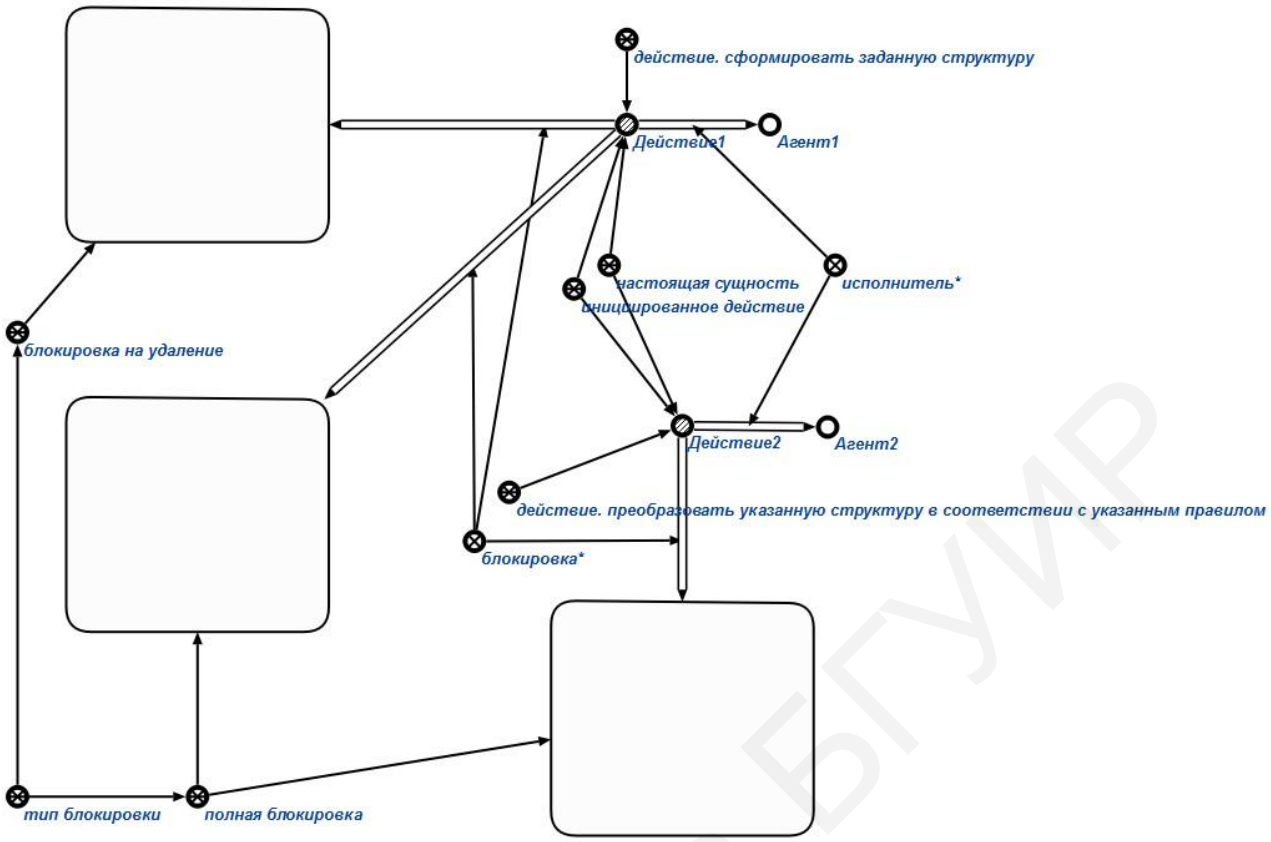

Рисунок 1.27 – Пример описания блокировок в семантической памяти

Перечислим базовые принципы работы с блокировками, не зависящие от типа блокировки:

– в каждый момент времени одному процессу в sc-памяти может соответствовать только одна блокировка каждого типа;

– в каждый момент времени одному процессу в sc-памяти может соответствовать только одна блокировка, установленная на некоторый конкретный sc-элемент;

– при завершении выполнения любого процесса в sc-памяти все установленные им блокировки автоматически снимаются;

– для повышения эффективности работы системы в целом каждый процесс должен в каждый момент времени блокировать минимально необходимое множество sc-элементов, снимая блокировку с каждого sc-элемента сразу же, как это становится возможным (безопасным).

В случае когда в рамках *процесса в sc-памяти* явно выделяются более частные подпроцессы (при помощи отношений *темпоральная часть*, поддействие*, декомпозиция действия** и т. д.), каждый такой подпроцесс с точки зрения синхронизации выполнения рассматривается как самостоятельный процесс, которому в соответствие могут быть поставлены все

необходимые блокировки. Однако для этого случая выделен ряд дополнительных правил работы с блокировками:

– все дочерние процессы в sc-памяти имеют доступ к блокировкам родительского процесса, так же как если бы это были блокировки, соответствующие каждому из таких дочерних процессов;

– в свою очередь, родительский процесс не имеет какого-либо привилегированного доступа к sc-элементам, заблокированным дочерними процессами, и работает с ними так же, как любой другой процесс в sc-памяти; исключение составляют sc-элементы, обозначающие сами дочерние процессы, поскольку родительский процесс должен иметь возможность управления дочерним, например, приостановки или прекращения их выполнения;

– все дочерние процессы по отношению друг к другу работают так же, как и по отношению к любым другим процессам;

– в случае когда родительский процесс приостанавливает выполнение (становится *отложенным действием*), все его дочерние процессы также приостанавливают выполнение; в свою очередь, приостановка одного из дочерних процессов в общем случае не инициирует явно остановку всего родительского процесса и соответственно других дочерних.

Принципы работы с *полными блокировками*, с одной стороны, наиболее просты, поскольку все процессы, кроме установившего такую блокировку, не имеют доступа к заблокированным sc-элементам и конфликты возникнуть не могут. С другой стороны, частое использование блокировок такого типа для, например, sc-элементов, описывающих предметную часть базы знаний [95], может привести к тому, что система не сможет использовать в полной мере имеющиеся у нее знания и будет давать неполные или даже некорректные ответы на поставленные вопросы.

Основные принципы работы с *полными блокировками*:

1. Если sc-элемент, инцидентный некоторому sc-коннектору, попадает в какую-либо полную блокировку, то сам этот sc-коннектор по умолчанию также считается заблокированным этой же блокировкой. Обратное в общем случае неверно, т. к. часть sc-коннекторов, инцидентных некоторому sc-элементу, может быть полностью заблокирована, при этом сам этот элемент заблокирован не будет. Такая ситуация типична, например, для sc-узлов, обозначающих классы понятий.

2. Каждый процесс в sc-памяти может свободно изменять или удалять любые sc-элементы, попадающие в полную блокировку, соответствующую этому процессу.

В случае блокировок на удаление или любое изменение возможны ситуации, когда несколько процессов пытаются получить доступ к одному и тому же sc-элементу, например, удалить его или инцидентные ему sc-коннекторы.

Перечислим несколько особенностей, касающихся принципов работы с блокировками на удаление и любое изменение:

– на один и тот же sc-элемент в один момент времени может быть установлена только одна блокировка одного типа, но разные процессы могут одновременно установить блокировки двух разных типов на один и тот же элемент. Это касается случая, когда первый процесс установил на некоторый sc-элемент блокировку на удаление, а второй процесс затем устанавливает блокировку на любое изменение. В других случаях возникает конфликт блокировок, который решается при помощи sc-метаагентов;

– установка блокировки любого типа также считается изменением, таким образом, если на некоторый sc-элемент была установлена блокировка на любое изменение, то другой процесс не сможет установить на этот же sc-элемент блокировку любого типа, пока первый процесс не снимет свою;

– если блокировка на удаление устанавливается на некоторый sc-коннектор, то по умолчанию та же блокировка устанавливается на инцидентные этому sc-коннектору sc-элементы, поскольку удаление этих элементов приведет к удалению этого коннектора.

В некоторых случаях для того чтобы обеспечить синхронизацию, необходимо объединять несколько элементарных действий над sc-памятью в одно неделимое действие (транзакцию), для которого гарантируется, что ни один сторонний процесс не сможет прочитать или изменить участвующие в этом действии sc-элементы, пока действие не завершится. При этом, в отличие от ситуации с полной блокировкой, процесс, пытающийся получить доступ к таким элементам, не продолжает выполнение так, как если бы этих элементов просто не было в sc-памяти, а ожидает завершения транзакции, после чего может выполнять с данными элементами любые действия согласно общим принципам синхронизации процессов. Проблема обеспечения транзакций не может быть решена на уровне SC-кода и требует реализации таких неделимых действий на уровне платформы интерпретации sc-моделей. Однако классы таких действий известны и число их сравнительно невелико.

Для того чтобы описать остальные принципы синхронизации процессов в sc-памяти, выделим базовые классы действий, которые может выполнять некоторый процесс в sc-памяти:

– поиск некоторого sc-элемента. Под поиском в данном случае понимается не только непосредственно поиск (так называемое чтение), но и сохранение найденного sc-элемента, например, в качестве операнда какого-либо оператора языка SCP (значения некоторой scp-переменной). В таком случае возможно возникновение конфликта, например, в ситуации, когда какой-либо процесс попытается удалить sc-элемент, найденный и сохраненный другим процессом;

– генерация некоторого sc-элемента (отдельного sc-узла или sc-коннектора, инцидентного двум заданным sc-элементам);

– удаление некоторого sc-элемента;

– установка блокировки некоторого типа на некоторый sc-элемент;

– снятие блокировки с некоторого sc-элемента.

В случае осуществления поиска все найденные и сохраненные в рамках какого-либо процесса sc-элементы попадают в соответствующую данному процессу *блокировку на любое изменение*. Таким образом, гарантируется целостность фрагмента базы знаний, с которым работает некоторый процесс в sc-памяти. При этом поиск и автоматическая установка такой блокировки должны быть реализованы как транзакция.

Такой подход также позволяет избежать ситуации, когда один процесс заблокировал некоторый sc-элемент на любое изменение, а второй процесс пытается сгенерировать или удалить sc-коннектор, инцидентный данному sc-элементу. В таком случае второй процесс должен будет предварительно найти и заблокировать указанный sc-элемент на любое изменение, что вызовет конфликт блокировок, который, в свою очередь, решается согласно алгоритму, изложенному ниже в данном разделе.

Таким образом, в случае генерации любого sc-элемента в рамках некоторого процесса он автоматически попадает в полную блокировку, соответствующую данному процессу. При этом генерация и автоматическая установка такой блокировки должны быть реализованы как транзакция. При необходимости сгенерированные элементы могут быть удалены (т. е. их временное существование вообще никак не отразится на деятельности других процессов) или разблокированы в случае, когда сгенерирована информация, которая может иметь некоторую ценность в дальнейшем.

Если какой-либо процесс пытается установить блокировку любого типа на какой-либо sc-элемент, уже заблокированный каким-либо другим процессом, то, с одной стороны, блокировка не может быть установлена, пока другой процесс не разблокирует указанный sc-элемент; с другой стороны, для того чтобы обеспечить возможность поиска и устранения взаимоблокировок,

необходимо явно указывать тот факт, что какой-либо процесс хочет получить доступ к какому-либо заблокированному другим процессом sc-элементу. Схожая ситуация и подход к ее решению для процессов в традиционной памяти описываются в работе [93]. В рамках рассматриваемого подхода для того чтобы иметь возможность указать, какие процессы пытаются заблокировать уже заблокированный *sc-элемент*, предлагается наряду с отношением *блокировка** использовать отношение *планируемая блокировка**, полностью аналогичное отношению *блокировка**. Процесс, которому в соответствие поставлена *планируемая блокировка**, приостанавливает выполнение до тех пор, пока уже установленные блокировки не будут сняты, после чего *планируемая блокировка** становится реальной *блокировкой** и процесс продолжает выполнение в соответствии с общими правилами. В случае когда на один и тот же sc-элемент планируют установить блокировку сразу несколько процессов, используется также отношение *приоритет блокировки**, связывающее между собой пары отношения *планируемая блокировка**. Как правило, приоритет блокировки определяется тем, какой из процессов раньше попытался установить блокировку на рассматриваемый sc-элемент, хотя в общем случае приоритет может устанавливаться или меняться в зависимости от дополнительных критериев.

Таким образом, описанный механизм регулирует также и процессы поиска, поскольку поиск и сохранение некоторого sc-элемента предполагает установку *блокировки на любое изменение*. Кроме того, следует учитывать, что на один sc-элемент *блокировка на любое изменение* может быть установлена после *блокировки на удаление*, соответствующей другому процессу. В этом случае использовать отношение *планируемые блокировки** нет необходимости.

Действие проверки наличия на некотором sc-элементе блокировки и в зависимости от результата проверки, установки блокировки или планируемой блокировки (с указанием приоритета при необходимости) должно быть реализовано как транзакция.

В случае попытки удаления некоторого sc-элемента некоторым процессом удаление может быть осуществлено только в случае, когда на данный sc-элемент не установлена (и не планируется) ни одна блокировка каким-либо другим процессом.

В других случаях необходимо обеспечить корректное завершение выполнения всех процессов, работающих с данным sc-элементом, и только потом удалить его физически.

Для реализации такой возможности каждому процессу может быть поставлено в соответствие множество *удаляемых данным процессом*

sc-элементов. Sc-элементы, попавшие в такое множество, доступны процессам, уже установившим (или планирующим установить) на эти sc-элементы блокировки ранее (до попытки их удаления), а для всех остальных процессов эти sc-элементы уже считаются удаленными. Процесс, пытающийся удалить sc-элемент, приостанавливает свое выполнение до того момента, пока все заблокировавшие и планирующие заблокировать данный sc-элемент процессы не разблокируют его. В общем случае один sc-элемент может входить во множества удаляемых элементов одновременно для нескольких процессов, в этом случае все такие процессы одновременно продолжат выполнение после снятия с этого sc-элемента всех блокировок. Если удаление пытается осуществить один из процессов, уже установивший на указанный sc-элемент блокировку, то алгоритм действий остается прежним: sc-элемент добавляется во множество удаляемых данным процессом sc-элементов и будет физически удален, как только все остальные процессы, установившие на данный sc-элемент блокировки, снимут их.

Действие проверки наличия блокировок или планируемых блокировок на удаляемый sc-элемент и собственно его удаление или добавление во множество удаляемых sc-элементов для соответствующего процесса должно быть реализовано как транзакция.

Рассмотрим пример использования описанного механизма блокировок в ситуации, когда с одним и тем же sc-элементом пытаются работать одновременно несколько процессов в sc-памяти (рисунки 1.28–1.31).

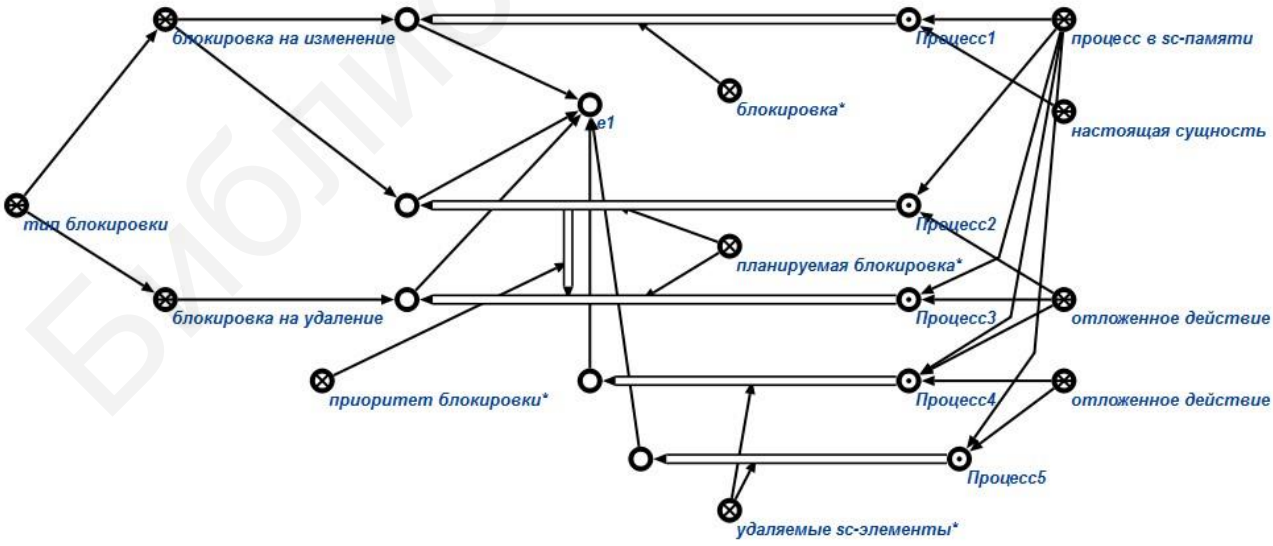

Рисунок 1.28 – Пример использования планируемых блокировок

В данном примере *Процесс1* непосредственно работает с sc-элементом *e1*, *Процесс2* и *Процесс3* планируют установить блокировку на любое изменение и блокировку на удаление соответственно, причем *Процесс2* попытался установить свою блокировку раньше, чем *Процесс3*, поэтому согласно направлению связки отношения *приоритет блокировки**, его блокировка будет установлена раньше. *Процесс4* и *Процесс5* ожидают снятия всех блокировок и планируемых блокировок, после чего *e1* будет удален, *Процесс1* и *Процесс2* продолжат свое выполнение. Никакие другие планируемые блокировки установлены быть уже не могут, поскольку *e1* попал во множество удаляемых sc-элементов как минимум одного процесса и, в соответствии с изложенными выше правилами все остальные процессы, кроме *Процесс1–Процесс5*, уже не смогут получить доступ к этому sc-элементу.

Выполняемый процесс принадлежит множеству *настоящая сущность*, приостановленные – множеству *отложенное действие*.

После того как *Процесс1* разблокировал sc-элемент *e1*, этот элемент будет заблокирован *Процессом2*, и *Процесс2* продолжит выполнение (см. рисунок 1.29).

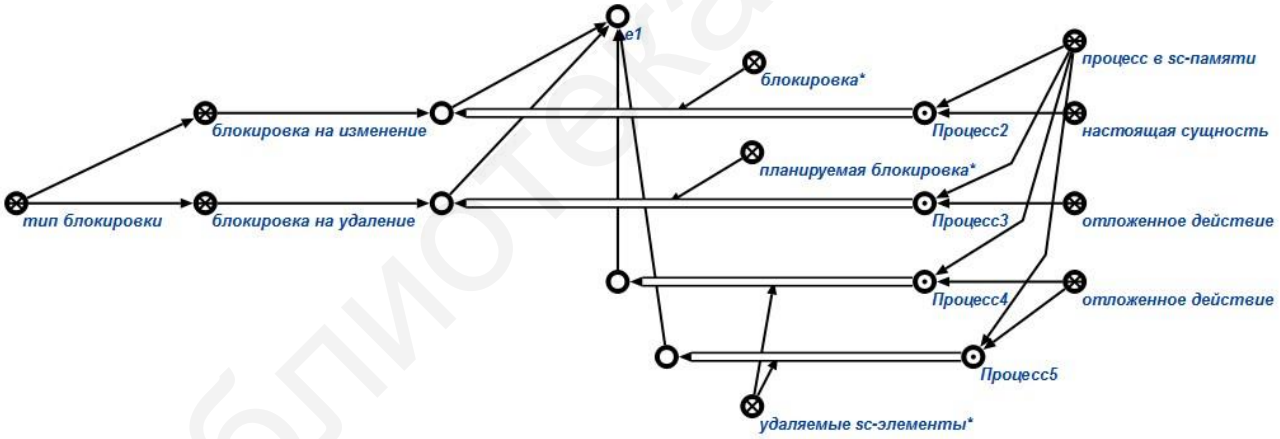

Рисунок 1.29 – Пример использования планируемых блокировок (продолжение)

Как видно из рисунка, *планируемая блокировка**, установленная *Процессом2*, становится обычной *блокировкой**.

Далее, после того как *Процесс2* разблокировал sc-элемент *e1*, этот элемент будет заблокирован *Процессом3*, и *Процесс3* продолжит выполнение (см. рисунок 1.30).

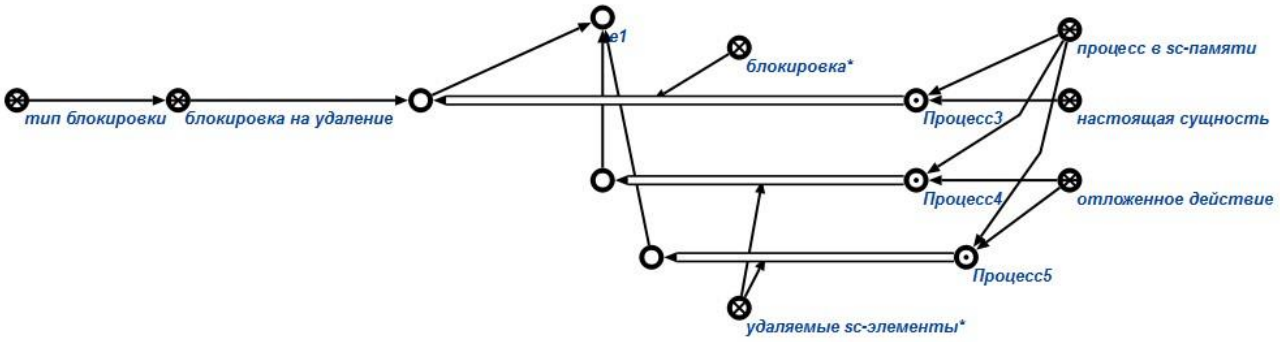

Рисунок 1.30 – Пример использования блокировки на удаление

Наконец, когда все процессы снимут блокировки с sc-элемента *e1*, он может быть физически удален, и *Процесс4* и *Процесс5* продолжат выполнение (см. рисунок 1.31).

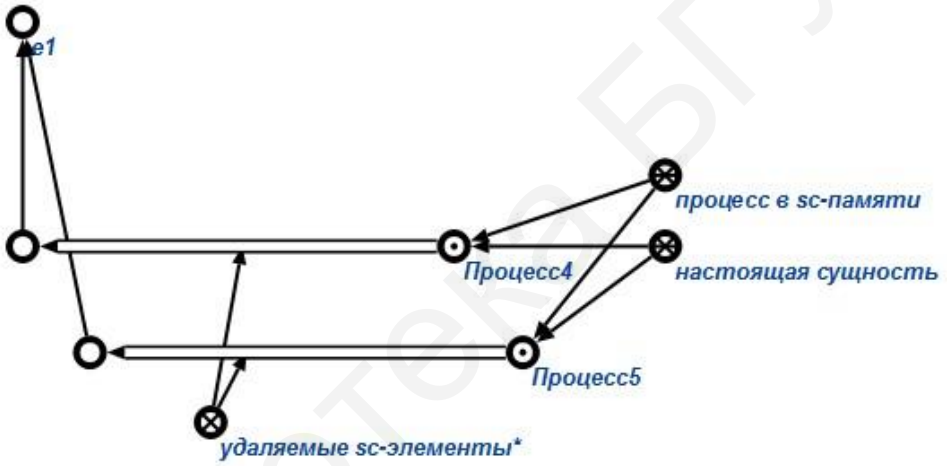

Рисунок 1.31 – Удаляемые sc-элементы

Алгоритм снятия блокировки любого типа с некоторого sc-элемента состоит из следующих шагов:

– если на данный sc-элемент установлена одна или несколько *планируемых блокировок**, то первая из них по приоритету (или единственная) становится *блокировкой**, а соответствующий ей процесс продолжает выполнение (становится настоящей сущностью); связка отношения *приоритет выполнения*, соответствовавшая удаленной связке отношения *планируемая блокировка** также удаляется, т. е. приоритет смещается на одну позицию;

– если *планируемых блокировок**, установленных на данный sc-элемент, нет, но он попадает в множество удаляемых sc-элементов для одного или нескольких процессов, то рассматриваемый sc-элемент физически удаляется, а приостановленные до его удаления процессы продолжают свое выполнение (становятся настоящими сущностями);

– если на данный sc-элемент не установлены планируемые блокировки, и он не входит в множество удаляемых для какого-либо процесса, то блокировка просто снимается без каких-либо дополнительных изменений.

Перечислим явно транзакции, реализация которых на уровне платформы интерпретации sc-моделей необходима для реализации рассматриваемого механизма синхронизации параллельных процессов:

– поиск некоторой конструкции в sc-памяти и автоматическая установка *блокировки на любое изменение* на найденные sc-элементы;

– генерация некоторого sc-элемента и автоматическая установка на него *полной блокировки*;

– проверка наличия на некотором sc-элементе блокировки и в зависимости от результата проверки установка *блокировки* или *планируемой блокировки* (с указанием приоритета при необходимости);

– проверка наличия *блокировок* или *планируемых блокировок* на удаляемый sc-элемент и собственно его удаление или добавление в множество удаляемых sc-элементов для соответствующего процесса;

– снятие блокировки с заданного sc-элемента и при необходимости установка первой по приоритету *планируемой блокировки** или удаление данного sc-элемента, если он входит в множество удаляемых sc-элементов для некоторого процесса;

– в случае добавления процесса в множество *отложенных действий* – поиск его подпроцессов и также добавление их в данное множество;

– в случае удаления процесса из множества *отложенных действий* – поиск его подпроцессов и также удаление их из данного множества.

При реализации *sc-агентов, не являющихся sc-агентами управления scp-процессами* на *языке SCP*, соблюдение всех принципов синхронизации соответствующих этим sc-агентам процессов обеспечивается на уровне *sc-агентов интерпретации scp-программ*, т. е. средствами платформы интерпретации sc-моделей. При реализации *sc-агентов, не являющихся sc-агентами управления scp-процессами* на уровне платформы, соблюдение всех принципов синхронизации возлагается, во-первых, непосредственно на разработчика агентов, во-вторых, – на разработчика платформы. Так, например, платформа может предоставлять доступ к хранимым в sc-памяти элементам через некоторый программный интерфейс, уже учитывающий принципы работы с блокировками, что избавит разработчика агентов от необходимости учитывать все эти принципы вручную.

В общем случае весь механизм блокировок может как описываться на уровне SC-кода (для повышения уровня платформенной независимости), так и при необходимости быть реализован на уровне платформы интерпретации sc-моделей, например для повышения производительности. Для этого каждому выполняемому в sc-памяти процессу на нижнем уровне может быть поставлена в соответствие некая уникальная таблица, в каждый момент времени содержащая перечень заблокированных элементов с указанием типа блокировки.

В процессе обработки информации sc-агентами возможна ситуация, в которой может возникнуть конфликт двух *процессов в sc-памяти*, неразрешимый без участия *sc-метаагентов и* связанный с проблемой взаимоблокировки (deadlock).

Рассмотрим более подробно пример возникновения взаимоблокировки. Пусть в некоторый момент времени в *sc-памяти* выполняются (являются *настоящими сущностями*) два процесса – *Процесс1* и *Процесс2*. Существуют также *sc-элементы e1* и *e2*, при этом *Процесс1* имеет возможность найти *sc-элемент e1*, а *Процесс2* имеет возможность найти *sc-элемент e2*. Под фразой «имеет возможность найти» будем иметь в виду, что искомый sc-элемент не попадает в полную блокировку какого-либо другого процесса и ищущему процессу известны какие-либо sc-элементы, опираясь на которые и действуя по известной программе, процесс сможет путем перехода по связям в sc-памяти за конечное число шагов прийти к искомому sc-элементу. В SCg-коде такая ситуация может выглядеть следующим образом (рисунок 1.32).

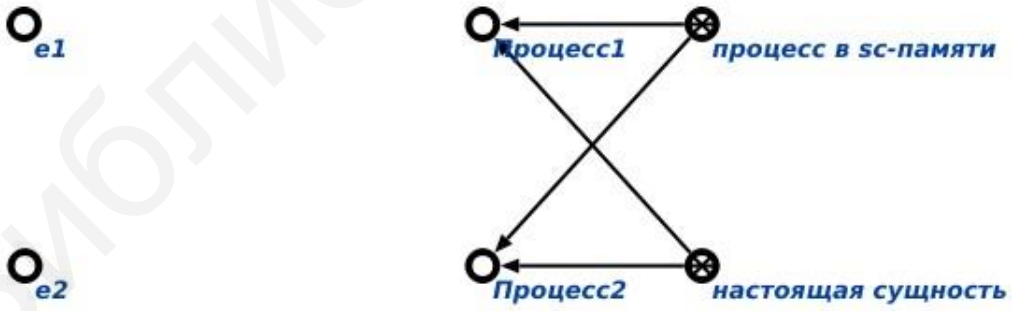

Рисунок 1.32 – Стартовая ситуация для взаимоблокировки

Далее предположим, что *Процесс1* заблокировал *sc-элемент e1* блокировкой *b1*, а *Процесс2* заблокировал *sc-элемент e1* блокировкой *b2*, после чего *Процесс1* пытается получить доступ к *sc-элементу e2*, а *Процесс2* пытается получить доступ к *sc-элементу e1* (сформированы соответствующие *планируемые блокировки**). В зависимости от конкретных *типов блокировок b1*

и *b2* и того, какие конкретно действия с *sc-элементами* предполагается выполнить далее в рамках *Процесса1* и *Процесса2*, возможны ситуации взаимоблокировки, когда каждый из указанных процессов будет ожидать снятия блокировки вторым процессом с нужного *sc-элемента*, не снимая при этом установленной им самим блокировки с *sc-элемента*, доступ к которому необходим второму процессу.

В случае когда хотя бы одна из блокировок *b1* и *b2* является *полной блокировкой*, ситуация взаимоблокировки возникнуть не может, поскольку *sc-элементы*, попавшие в *полную блокировку* некоторого *scp-процесса*, недоступны другим *scp-процессам* даже для чтения и, таким образом, остальные *scp-процессы* будут работать так, как будто заблокированные *sc-элементы* просто отсутствуют в текущем состоянии *sc-памяти*.

В случаях когда ни *b1* ни *b2* не являются *полной блокировкой*, возможно появление взаимоблокировок. Для того чтобы показать, в каких случаях это возможно, воспользуемся таблицей 1.4, в которой указаны типы блокировок *b1* и *b2* и дополнительные условия, при которых возникает взаимоблокировка.

Таблица 1.4 – Условия возникновения взаимоблокировок

Тип *b1*	Тип *b2*	Условие возникновения взаимоблокировки
Полная блокировка	*Полная блокировка*	Не возникает
Полная блокировка	*Блокировка на изменение*	Не возникает
Полная блокировка	*Блокировка на удаление*	Не возникает
Блокировка на изменение	*Блокировка на изменение*	*Процесс1* пытается изменить *sc-элемент e2*, *Процесс2* пытается изменить *sc-элемент e1*
Блокировка на изменение	*Блокировка на удаление*	*Процесс1* пытается удалить *sc-элемент e2*, *Процесс2* пытается изменить *sc-элемент e1*
Блокировка на удаление	*Блокировка на удаление*	*Процесс1* пытается удалить *sc-элемент e2*, *Процесс2* пытается удалить *sc-элемент e1*

Рассмотрим пример конкретной взаимоблокировки, возникшей в *sc-памяти*. Для этого положим, что *b1* является *блокировкой на изменение*, а *b2* является *блокировкой на удаление*. При этом *Процесс1* пытается установить на *sc-элемент e2*, заблокированный *Процессом2*, *блокировку на любое изменение*, а *Процесс2*, в свою очередь, пытается установить такую же блокировку на *sc-элемент e1*, заблокированный *Процессом1*.

В SCg-коде такая ситуация будет выглядеть, как показано на рисунке 1.33.

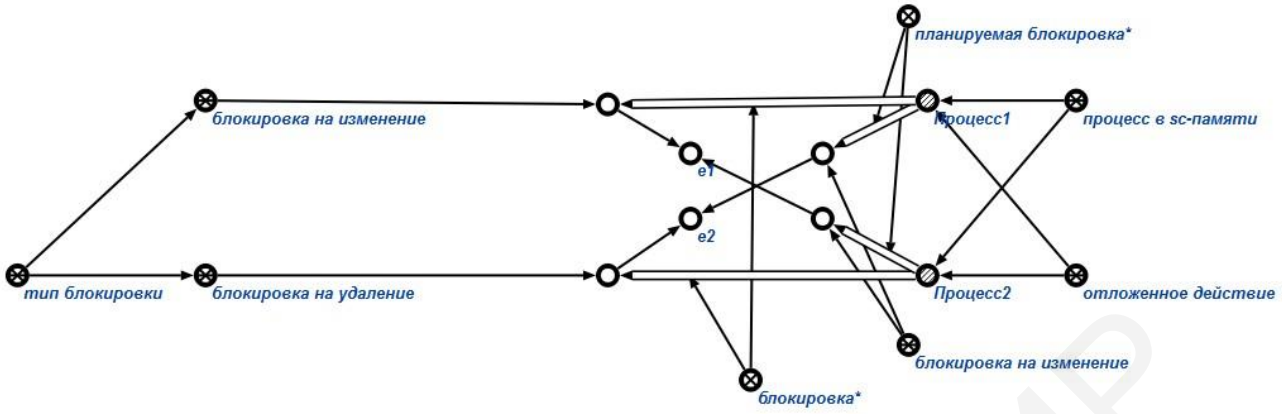

Рисунок 1.33 – Взаимоблокировка

Устранение такой взаимоблокировки невозможно без вмешательства специализированного *sc-метаагента*, который имеет право игнорировать блокировки, установленные другими процессами.

В общем случае проблема конкретной взаимоблокировки может быть решена путем выполнения специализированным *sc-метаагентом* следующих шагов:

– откат нескольких операций, выполненных одним из участвующих во взаимоблокировке процессов на столько шагов назад, на сколько это необходимо для того, чтобы второй процесс получил доступ к необходимым *sc-элементам* и смог продолжить выполнение;

– ожидание выполнения второго процесса вплоть до завершения или снятия им всех блокировок с *sc-элементов*, доступ к которым необходимо получить первому процессу;

– повторное выполнение в рамках первого процесса отмененных операций и продолжение его выполнения, но уже с учетом изменений в памяти, внесенных вторым процессом.

Изложенные правила синхронизации справедливы для *программных sc-агентов*. Для *sc-метаагентов* все sc-элементы, в том числе описывающие блокировки, планируемые блокировки и т. д., полностью эквивалентны между собой с точки зрения доступа к ним, т. е. любой *sc-метаагент* имеет доступ к любым sc-элементам, даже попавшим в полную блокировку для какого-либо другого процесса. Это необходимо для того, чтобы *sc-метаагенты* смогли выявлять и устранять различные проблемы, например, описанную проблему взаимоблокировки.

Таким образом, проблема синхронизации деятельности *sc-метаагентов* требует введения дополнительных правил.

Указанную проблему разделим на две частные:

– обеспечение синхронизации деятельности *sc-метаагентов* между собой;

– обеспечение синхронизации деятельности *sc-метаагентов* и *программных sc-агентов*.

Первую проблему предлагается решить за счет запрета параллельного выполнения *sc-метаагентов*. Таким образом, в каждый момент времени в рамках одной *ostis-системы* может существовать только один процесс, соответствующий *sc-метаагенту* и являющийся *настоящей сущностью*.

Вторую проблему предлагается решить за счет введения дополнительных привилегий для *sc-метаагентов* при обращении к какому-либо sc-элементу. Для этого достаточно одного правила: если некоторый sc-элемент стал использоваться в рамках процесса, соответствующего *sc-метаагенту* (например, стал элементом хотя бы одного scp-оператора, входящего в данный процесс), то все процессы, при которых указанный sc-элемент попадает в блокировку, становятся отложенными действиями (приостанавливают выполнение). Как только указанный sc-элемент перестает использоваться в рамках процесса, соответствующего *sc-метаагенту*, все приостановленные по этой причине процессы продолжают выполнение.

Рассмотренные ограничения не ухудшают производительность *ostis-системы* существенно, поскольку *sc-метаагенты* предназначены для решения достаточно узкого класса задач, которые, как показал опыт практической разработки прототипов различных *ostis-систем*, возникают достаточно редко.

Отдельно рассмотрим применение описанного механизма блокировок в том случае, когда некоторые преобразования sc-памяти посредством sc-агентов пользовательского интерфейса выполняет пользователь ostis-системы. Ранее говорилось, что каждый пользователь с точки зрения обработки знаний также рассматривается как sc-агент, а значит, имеет возможность блокировать sc-элементы блокировками различных типов. Для обеспечения такой возможности каждому пользователю ставится в соответствие действие, описывающее всю деятельность данного пользователя. Соответственно, все конкретные действия, выполняемые данным пользователем, будут для такого общего действия более частными, что формально выражается введением связок отношения *поддействие**. Все блокировки данного пользователя связываются с указанным общим действием, представленным на рисунке 1.34, в SCg-коде.

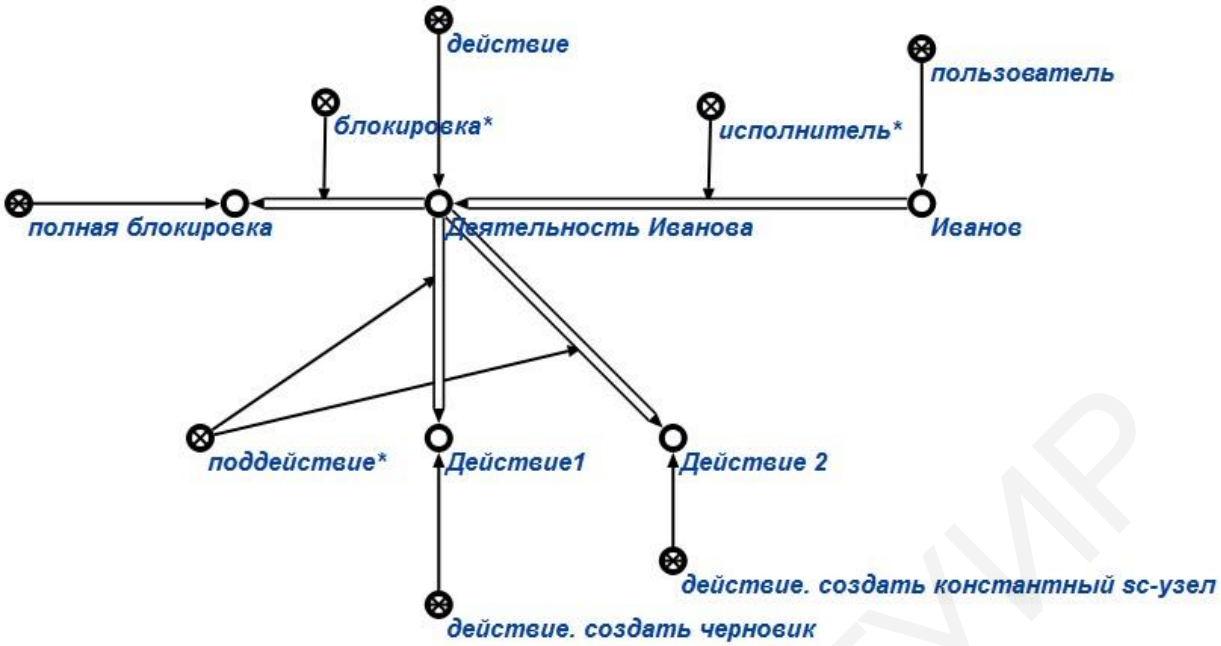

Рисунок 1.34 – Блокировки пользователя

Для sc-агентов, программы которых реализованы с использованием *языка SCP*, справедливы общие принципы организации взаимодействия *sc-агентов* и пользователей *ostis-системы* через общую *sc-память*.

Помимо общих принципов для *sc-агентов*, реализованных с использованием *языка SCP*, вводятся следующие дополнительные уточнения:

– в результате появления в sc-памяти некоторой конструкции, удовлетворяющей условию инициирования какого-либо *абстрактного sc-агента*, реализованного при помощи *языка SCP*, в *sc-памяти* генерируется и инициируется *scp-процесс*. В качестве шаблона для генерации используется *агентная scp-программа*, соответствующая данному *абстрактному sc-агенту*;

– каждый такой *scp-процесс*, соответствующий некоторой *агентной scp-программе*, может быть связан с набором структур, описывающих блокировки различных типов. Таким образом, синхронизация взаимодействия параллельно выполняемых *scp-процессов* осуществляется так же, как и в случае любых других *действий в sc-памяти*;

– несмотря на то что каждый *scp-оператор* представляет собой атомарное действие в sc-памяти, являющееся поддействием в рамках всего *scp-процесса*, блокировки, соответствующие одному оператору, не вводятся, чтобы избежать громоздкости и избытка дополнительных системных конструкций, создаваемых при выполнении некоторого *scp-процесса*. Вместо этого используются блокировки, общие для всего *scp-процесса*. Таким образом, *агенты интерпретации scp-программ* работают только с учетом блокировок, общих для всего интерпретируемого *scp-процесса*;

– процессы, описывающие деятельность агентов интерпретации scp-программ, как правило, не создаются, следовательно, и не вводятся соответствующие им блокировки. Поскольку такие агенты работают с уникальным scp-процессом и их число ограничено и известно, то использование блокировок для их синхронизации не требуется;

– в случае приостановки *scp-процесса* (добавления его во множество *отложенных действий*) в соответствии с общими правилами синхронизации все его дочерние процессы также должны быть приостановлены. В связи с этим все *scp-операторы*, которые в этот момент являются *настоящими сущностями*, становятся *отложенными действиями*;

– во избежание нежелательных изменений в самом теле *scp-процесса* вся конструкция, сгенерированная на основе некоторой *scp-программы* (весь sc-текст, описывающий декомпозицию *scp-процесса* на *scp-операторы*), должна быть добавлена в *полную блокировку*, соответствующую данному *scp-процессу*;

– при необходимости разблокировать или заблокировать некоторую конструкцию каким-либо типом блокировки используются соответствующие scp-операторы класса *scp-оператор управления блокировками*;

– после завершения выполнения некоторого scp-процесса его текст, как правило, удаляется из *sc-памяти*, а все заблокированные конструкции освобождаются (разрушаются знаки структур, обозначавших блокировки);

– как правило, частный *класс действий*, соответствующий конкретной *scp-программе,* явно не вводится, а используется более общий класс *scp-процесса*, за исключением тех случаев, когда введение специального *класса действий* необходимо по каким-либо другим соображениям.

В заключение стоит также отметить, что возможна ситуация, при которой выполнение некоторого процесса в sc-памяти прервано по причине возникновения какой-либо ошибки. В таком случае существует вероятность того, что блокировка, установленная данным процессом, не будет снята до тех пор, пока этого не сделает sc-метаагент, обнаруживший подобную ситуацию. Однако указанная проблема на уровне sc-модели может быть решена лишь частично, для случаев, когда ошибка возникает при интерпретации scp-программы, отслеживается scp-интепретатором и в памяти формируется соответствующая конструкция, сообщающая о проблеме sc-метаагенту. Случаи, когда возникла ошибка на уровне scp-интерпретатора или sc-хранилища, должны рассматриваться на уровне платформы интерпретации sc-моделей. Более строгое решение проблемы возникновения ошибок на настоящий момент находится в стадии исследования.

2 МЕТОДИКА И СРЕДСТВА РАЗРАБОТКИ РЕШАТЕЛЕЙ ЗАДАЧ

2.1 МЕТОДИКА ПОСТРОЕНИЯ РЕШАТЕЛЕЙ ЗАДАЧ

2.1.1 Общие принципы разработки решателей задач

Все платформенно-независимые компоненты гибридного решателя задач ostis-системы могут быть представлены при помощи SC-кода. При этом речь идет как о спецификациях sc-агентов, так и о полных текстах scp-программ, описывающих алгоритмы работы указанных агентов.

Таким образом, построение решателя задач некоторой ostis-системы сводится к разработке особого рода фрагмента базы знаний такой системы. В связи с этим при построении и модификации решателей могут использоваться все существующие средства автоматизации процесса построения и модификации баз знаний по технологии OSTIS, рассмотренные, в частности, в работах [53, 95].

В данном разделе более детально рассматриваются некоторые аспекты разработки, специфичные конкретно для гибридных решателей задач, в частности, методика построения и модификации таких решателей и средства автоматизации и информационной поддержки процесса построения и модификации решателей, построенных в соответствии с моделью гибридного решателя задач.

Указанные средства включают систему автоматизации процесса построения и модификации решателей и подсистему информационного обслуживания разработчиков решателей в рамках метасистемы IMS.

Важнейшим фрагментом метасистемы IMS является библиотека многократно используемых компонентов ostis-систем, в рамках которой выделяется библиотека многократно используемых компонентов решателей задач, построенная с учетом семантической классификации sc-агентов и решателей. Указанная библиотека включает как сами компоненты, так и набор агентов автоматизации поиска нужных компонентов на основе некоторой спецификации.

Методика построения и модификации гибридных решателей задач включает несколько этапов. На рисунке 2.1 представлен перечень таких этапов с указанием последовательности их выполнения. Темным фоном отмечены те этапы, которые частично автоматизированы в рамках средств, рассмотренных в данном подразделе.

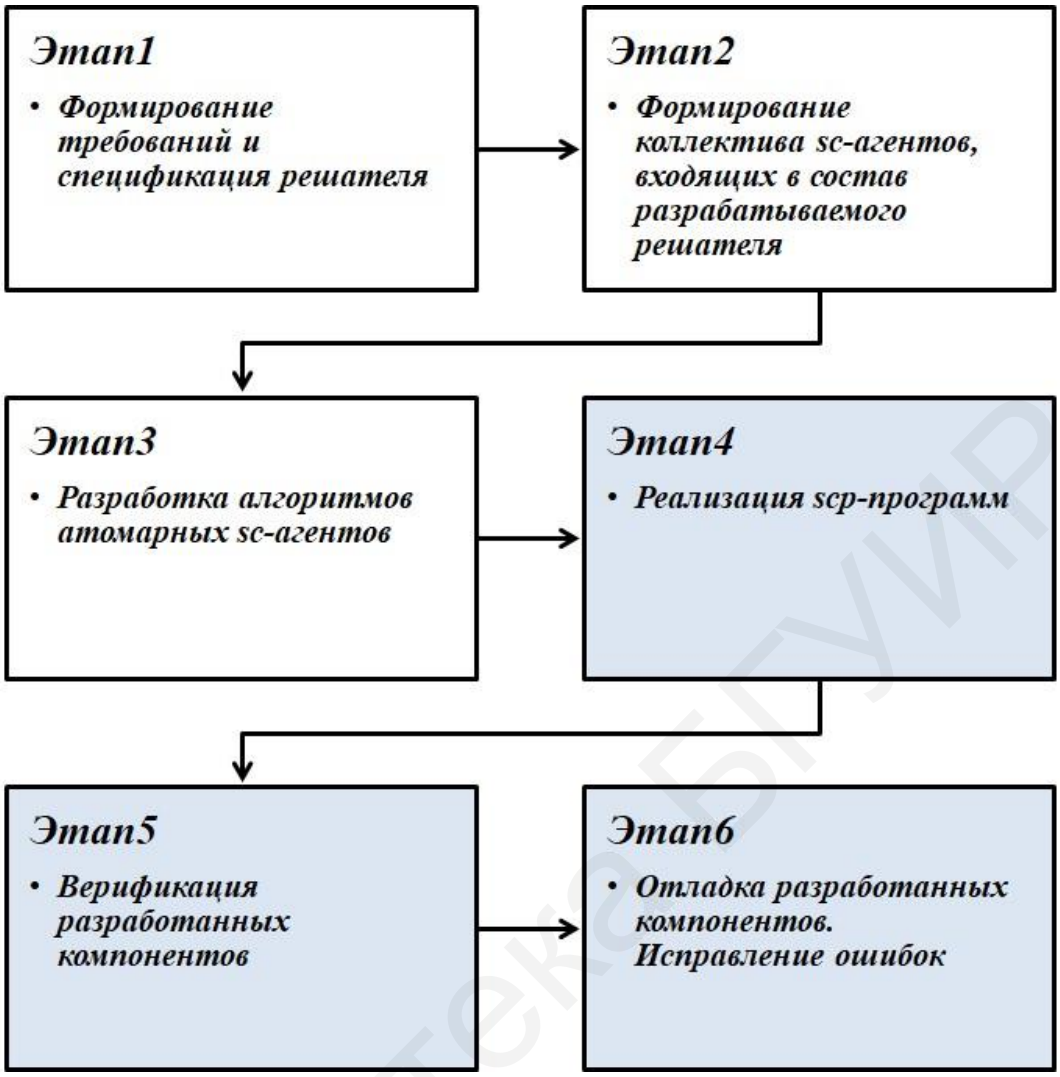

Рисунок 2.1 – Этапы процесса построения и модификации гибридных решателей задач

Рассматриваемая методика может быть применена как при разработке объединенных решателей, так и при разработке решателей частного вида, поскольку с формальной точки зрения все они трактуются как неатомарный абстрактный sc-агент.

Рассмотрим подробнее каждый из этапов на примере разработки решателя задач интеллектуальной справочной системы (ИСС) по геометрии Евклида.

Этап 1. Формирование требований и спецификация решателя задач.

На данном этапе необходимо четко выделить задачи, решение которых должен обеспечивать решатель задач, продумать предполагаемые способы их решения и на основе данного анализа определить место будущего решателя в общей иерархии решателей. Важность данного этапа заключается в том, что при правильной классификации существует вероятность того, что в составе библиотеки компонентов уже есть реализованный вариант требуемого

решателя. В противном случае у разработчика появляется возможность включить разработанный решатель в библиотеку компонентов для последующего использования. Данные факты обусловлены тем, что структура библиотеки компонентов решателей задач основана на семантической классификации таких решателей и, соответственно, их компонентов.

При недостаточно четкой спецификации и классификации разрабатываемого решателя повышается вероятность того, что подходящий решатель не будет найден в библиотеке компонентов даже в случае, если он там есть, а вновь разработанный решатель не сможет быть включен в библиотеку. Таким образом, идея многократного использования уже разработанных компонентов будет нарушена, что существенно повысит затраты на разработку такого решателя.

Для первой версии разрабатываемого решателя по геометрии можно выделить следующие классы задач:

– базовый поиск по базе знаний (поиск элементов заданного множества; поиск классов, которым принадлежит заданная сущность; поиск идентификаторов для заданной сущности; поиск декомпозиции заданной сущности; поиск семантической окрестности заданной сущности; поиск всех сущностей, являющихся частными/общими для заданной сущности);

– поиск по базе знаний, актуальный для интеллектуальных справочных систем (поиск иллюстраций для заданной сущности; поиск аксиом/теорем заданной формальной теории; поиск понятий, на основе которых определяется заданное понятие; поиск понятий, которые определяются на основе заданного понятия; поиск доказательства заданного логического утверждения, заранее представленного в базе знаний; поиск решения заданной задачи, заранее представленной в базе знаний и др.);

– поиск значения заданной величины с использованием простых механизмов логического вывода и стратегии поиска в глубину.

Этап 2. Формирование коллектива sc-агентов, входящих в состав разрабатываемого решателя.

В случае когда найти в библиотеке готовый решатель, удовлетворяющий всем предъявляемым требованиям, не представляется возможным, необходимо выделить и специфицировать все компоненты такого решателя.

Результатом данного этапа является перечень полностью специфицированных *sc-агентов*, которые войдут в состав разрабатываемого решателя, с их иерархией вплоть до *атомарных sc-агентов*. В рамках данного этапа очень важно проектировать коллектив агентов таким образом, чтобы максимально задействовать уже имеющиеся в библиотеке многократно

используемые компоненты, а при отсутствии нужного компонента – иметь возможность включить его в библиотеку после реализации. В качестве таких компонентов в зависимости от сложности разрабатываемого решателя могут выступать как *атомарные sc-агенты*, так и целые коллективы sc-агентов (*неатомарные sc-агенты*).

При разработке перечня агентов (в том числе их спецификаций) необходимо соблюдать ряд принципов:

1. Каждый разрабатываемый sc-агент должен быть по возможности предметно независим, т. е. в множество ключевых узлов данного sc-агента не должны входить понятия, имеющие отношение непосредственно к рассматриваемой предметной области. Исключение составляют понятия из общих предметных областей, которые носят междисциплинарный характер (например, отношение *включение** или понятие *действие*). Данное правило также может быть нарушено в случае, если sc-агент является вспомогательным и ориентирован на обработку какого-либо конкретного класса объектов (например, sc-агенты, выполняющие арифметические вычисления, могут напрямую работать с конкретными отношениями *сложение**, *умножение** и т. п.). Всю необходимую для решения задачи информацию sc-агент должен извлекать из семантической окрестности соответствующего инициированного действия. Очевидно, что sc-агент, разработанный с учетом указанных требований, может быть использован при проектировании большего числа ostis-систем, чем если бы он был реализован с ориентацией на конкретную частную предметную область. После завершения разработки и отладки такой sc-агент должен быть включен в *Библиотеку многократно используемых абстрактных sc-агентов.*

2. Не стоит путать понятия sc-агента и агентной программы (в том числе агентной scp-программы). Взаимодействие sc-агентов осуществляется исключительно посредством спецификации информационных процессов в общей памяти, каждый sc-агент реагирует на некоторый класс событий в sc-памяти. Таким образом, каждому sc-агенту соответствует некоторое условие инициирования и одна агентная программа, которая запускается автоматически при возникновении в sc-памяти соответствующего условия инициирования. При этом в рамках данной программы могут сколько угодно раз вызываться различные подпрограммы. Однако не стоит путать инициирование sc-агента, которое осуществляется при появлении в sc-памяти соответствующей конструкции, и вызов подпрограммы другой программой, предполагающий явное указание вызываемой подпрограммы и перечня ее параметров.

3. Каждый sc-агент должен самостоятельно проверять полноту соответствия своего условия инициирования текущему состоянию sc-памяти. В

процессе решения какой-либо задачи может возникнуть ситуация, когда на появление одной и той же структуры среагировали несколько sc-агентов. В таком случае выполнение продолжают только те из них, условие инициирования которых полностью соответствует сложившейся ситуации. Остальные sc-агенты в данном случае прекращают выполнение и возвращаются в режим ожидания. Выполнение данного принципа достигается за счет тщательного уточнения спецификаций разрабатываемых sc-агентов. В общем случае условия инициирования у нескольких sc-агентов могут совпадать, например, когда одна и та же задача может быть решена разными способами и заранее неизвестно, какой из них приведет к желаемому результату.

4. Необходимо помнить, что неатомарный sc-агент с точки зрения других sc-агентов, не входящих в его состав, должен функционировать как целостный sc-агент (выполнять логически атомарные действия), что накладывает определенные требования на спецификации атомарных sc-агентов, входящих в его состав: как минимум необходимо, чтобы в составе неатомарного sc-агента присутствовал хотя бы один атомарный sc-агент, условие инициирования которого полностью совпадает с условием инициирования данного неатомарного sc-агента.

5. При необходимости реализации нового sc-агента следует руководствоваться следующими принципами выделения атомарных абстрактных sc-агентов:

1) проектируемый sc-агент должен быть максимально независим от предметной области, что позволит в дальнейшем использовать его при разработке решателей максимально возможного числа ostis-систем. При этом универсальность предполагает не только минимизацию числа ключевых узлов sc-агента, но и выделение класса действий, выполняемых данным sc-агентом таким образом, чтобы имело смысл включить данный sc-агент в *Библиотеку многократно используемых абстрактных sc-агентов* и использовать его при разработке решателей других ostis-систем. Не следует искусственно увязывать ряд действий в один sc-агент и, наоборот, расчленять одно самодостаточное действие на поддействия: это вызовет сложности восприятия принципов работы sc-агента разработчиками и не позволит использовать sc-агент в ряде систем (например, в обучающих системах, которые должны объяснять ход решения пользователю);

2) акт деятельности каждого sc-агента (выполняемое данным sc-агентом действие) должен быть логически целостным и завершенным. Следует помнить, что все sc-агенты взаимодействуют исключительно через общую память и избегать ситуаций, в которых инициирование одного sc-агента

осуществляется путем явной генерации известного условия инициирования другим sc-агентом (т. е., по сути, явным непосредственным вызовом одного sc-агента другим);

3) имеет смысл выделять в отдельные sc-агенты те относительно крупные фрагменты реализации некоторого общего алгоритма, которые могут выполняться независимо друг от друга.

6. При объединении sc-агентов в коллективы рекомендуется проектировать их таким образом, чтобы они могли быть использованы не только как часть рассматриваемого неатомарного абстрактного sc-агента. В случае если это не представляется возможным и некоторые sc-агенты, будучи отделенными от коллектива, теряют смысл, необходимо указать данный факт при документировании рассматриваемых sc-агентов.

7. Фактическим инициатором запуска sc-агента посредством общей памяти (автором соответствующей конструкции) может быть как непосредственно пользователь системы, так и другой sc-агент, что никак не должно отражаться в работе самого sc-агента.

Рассмотрим подробнее процесс выделения агентов в составе разрабатываемого решателя по геометрии. Для реализации каждого из поисковых запросов представленного выше списка задач, решаемых таким решателем, выделяется соответствующий поисковый агент.

Для выделения коллектива агентов, реализующих поиск неизвестного значения величины, рассмотрим общий алгоритм решения задачи, который предполагается реализовывать в рамках данного решателя:

1. Осуществляется проверка, вычислено ли искомое значение величины, если это так, то процесс решения завершается, в противном случае – переход к шагу 2.

2. Анализируются связи искомой величины с другими сущностями. Для каждой сущности, связанной с величиной каким-либо отношением, осуществляется поиск классов, которым она принадлежит. Для каждого найденного класса осуществляется поиск логических утверждений, справедливых для данного класса. Для каждого найденного утверждения выполняются шаги:

– осуществляется попытка применения найденного логического утверждения с учетом информации, известной на данный момент: если утверждение может быть применено (информации достаточно, посылки истинны), то генерируются новые знания, и алгоритм возвращается к шагу 1 и повторяется уже с учетом новых знаний; при необходимости осуществляется расчет математических выражений; если для применения утверждения текущей

информации недостаточно, то осуществляется переход к следующему утверждению для текущего рассматриваемого класса;

– если все утверждения для данного класса просмотрены, то осуществляется переход к следующему классу для текущей сущности, и утверждения просматриваются аналогичным образом.

3. Если все классы для данной сущности рассмотрены, то осуществляется поиск непросмотренных сущностей, связанных с текущей по принципу «поиск в глубину».

На основании данного алгоритма целесообразно выделить следующие классы логически атомарных действий:

– поиск значения заданной величины;

– поиск утверждения, которое может быть применено к текущей рассматриваемой сущности или переход к другим сущностям (реализация стратегии поиска пути решения задачи в глубину);

– применение заданного логического утверждения в рамках заданного контекста (в текущей реализации рассматриваются простые логические утверждения об импликации (вида «если…, то…» и эквиваленции);

– расчет математического выражения: в рамках прототипа предполагается использование таких операций, как сложение/вычитание, умножение/деление, возведение в степень/извлечение корня/логарифмирование, сравнение; каждой из перечисленных операций также соответствует класс логически атомарных действий.

Можно заметить, что выделение таких классов действий позволяет, с одной стороны, сделать агенты достаточно универсальными для их последующего использования в других системах, с другой стороны – обеспечить возможность расширения функциональности решателя таким образом, чтобы вносимые изменения были локализованы. Так, агенты применения логических утверждений могут быть использованы самостоятельно без агента реализации стратегии, а перечень таких агентов может быть легко расширен, что позволит использовать для решения задачи утверждения более сложного вида, при этом нет необходимости вносить какие-либо изменения в агент реализации стратегии. В свою очередь, неатомарный агент расчета математических выражений может быть использован в других системах, при этом состав атомарных агентов в его составе может легко меняться, не затрагивая при этом общий принцип расчета.

Этап 3. Разработка алгоритмов атомарных sc-агентов.

В рамках данного этапа необходимо продумать алгоритм работы каждого разрабатываемого *атомарного sc-агента*. Разработка алгоритма подразумевает выделение в нем логически целостных фрагментов, которые могут быть реализованы как отдельные *scp-программы*, в том числе выполняемые параллельно. Таким образом, появляется необходимость говорить не только о *Библиотеке многократно используемых абстрактных sc-агентов*, но и о *Библиотеке многократно используемых программ обработки sc-текстов* на различных языках программирования, в том числе *Библиотеке многократно используемых scp-программ*. Благодаря этому часть scp-программ, реализующих алгоритм работы некоторого sc-агента, может быть заимствована из соответствующей библиотеки.

Важно помнить, что если в процессе работы *sc-агент* генерирует в памяти какие-либо временные структуры, то при завершении работы он обязан удалять всю информацию, использование которой в системе более нецелесообразно (убрать за собой информационный мусор). Исключение составляют ситуации, когда подобная информация необходима нескольким *sc-агентам* для решения одной задачи, однако после решения задачи информация становится бесполезной или избыточной и требует удаления. В данном случае может возникнуть ситуация, когда ни один из *sc-агентов* не в состоянии удалить информационный мусор. В таком случае возникает необходимость говорить о включении в состав решателя специализированных *sc-агентов*, задачей которых является выявление и уничтожение информационного мусора.

Пример решения задачи прототипом решателя задач по геометрии Евклида представлен в пункте 2.4.1.

Этап 4. Реализация scp-программ.

Конечным этапом непосредственно разработки является реализация специфицированных ранее *scp-программ* или при необходимости программ, реализуемых на уровне платформы.

Этап 5. Верификация разработанных компонентов.

Верификация разработанных компонентов может осуществляться как вручную, так и с использованием специфицированных средств, входящих в состав системы автоматизации процесса построения и модификации гибридных решателей задач по технологии OSTIS.

Этап 6. Отладка разработанных компонентов. Исправление ошибок.

Этап отладки разработанных компонентов, в свою очередь, можно также условно разделить на частные этапы:

– отладка отдельных scp-программ или программ, реализуемых на уровне платформы;

– отладка отдельных атомарных sc-агентов;

– отладка неатомарных sc-агентов, входящих в состав решателя задач;

– отладка всего решателя задач.

Заметим, что *этап 5* и *этап 6* могут выполняться параллельно и повторяются до тех пор, пока разработанные компоненты не будут удовлетворять необходимым требованиям.

2.1.2 Формальная онтология деятельности разработчиков решателей задач

В рамках технологии OSTIS методика построения и модификации гибридных решателей задач основана на формальной онтологии деятельности разработчиков таких решателей.

Важно отметить, что согласно модели гибридного решателя задач, решатель задач представляет собой *абстрактный sc-агент*, в связи с чем разработка решателя сводится к разработке такого агента.

Фрагмент формальной онтологии деятельности, направленной на построение и модификацию решателей задач, в SCn-коде выглядит следующим образом (для удобства чтения отношения, задающие порядок выполнения действий, опущены):

действие. разработать решатель задач ostis-системы
= действие. разработать абстрактный sc-агент
*<= разбиение**:
{
- *действие. разработать атомарный абстрактный sc-агент*
  *=> включение**:
  действие. разработать платформенно-независимый атомарный абстрактный sc-агент
- *действие. разработать неатомарный абстрактный sc-агент*

}
*=> абстрактное поддействие**:
- *действие. специфицировать абстрактный sc-агент*
- *действие. найти в библиотеке абстрактный sc-агент, удовлетворяющий заданной спецификации*
- *действие. верифицировать sc-агент*
- *действие. отладить sc-агент*

действие. разработать платформенно-независимый атомарный абстрактный sc-агент
*=> абстрактное поддействие**:

- *действие. декомпозировать платформенно-независимый атомарный абстрактный sc-агент на scp-программы*
- *действие. разработать scp-программу*

действие. разработать неатомарный абстрактный sc-агент
*=> абстрактное поддействие**:

- *действие. декомпозировать неатомарный абстрактный sc-агент на более частные*
- *действие. разработать абстрактный sc-агент*

действие. разработать scp-программу
*=> абстрактное поддействие**:

- *действие. специфицировать scp-программу*
- *действие. найти в библиотеке scp-программу, удовлетворяющую заданной спецификации*
- *действие. реализовать специфицированную scp-программу*
- *действие. верифицировать scp-программу*
- *действие. отладить scp-программу*

действие. верифицировать sc-агент
*<= разбиение**:

{

- *действие. верифицировать атомарный sc-агент*
- *действие. верифицировать неатомарный sc-агент*

}

действие. отладить sc-агент
*<= разбиение**:

{

- *действие. отладить атомарный sc-агент*
- *действие. отладить неатомарный sc-агент*

}

Наличие такой формальной онтологии позволяет, во-первых, частично автоматизировать процесс построения и модификации решателей, во-вторых, – повысить эффективность информационной поддержки разработчиков, поскольку данная онтология может быть включена в базу знаний интеллектуальной метасистемы IMS.

2.1.3 Пример разработки абстрактного sc-агента и соответствующей команды пользовательского интерфейса

Приведем пример последовательности шагов, которую необходимо выполнить при создании поискового sc-агента.

Рассмотрим sc-агент, выполняющий поиск иллюстраций для заданного объекта. Пример работы данного sc-агента будет продемонстрирован в системе по геометрии Евклида. Суть работы агента заключается в поиске имеющихся в системе иллюстраций для заданного объекта.

Создание команды меню

Для начала необходимо создать команду (подпункт) меню в системе, которая будет соответствовать sc-агенту. Для этого необходимо выполнить последовательность действий:

1. Перейти в каталог ~/ims.ostis.kb/knowledge_base_IMS/doc_technology_ostis/section_library_OSTIS/section_library_of_reusable_components_interfaces/lib_ui_menu.

2. Создать файл с именем ui_menu_file_for_finding_illustrations.scs (имя для файла может быть выбрано любое).

В этом файле указывается, где именно будет располагаться команда в web-интерфейсе. Также указывается имя класса действий, который будет инициировать данный sc-агент. Он будет уникален для рассматриваемого sc-агента.

```
ui_menu_file_for_finding_illustrations<-ui_user_command_class_atom;
ui_user_command_class_view_kb;;
// Указываем русский идентификатор команды меню
ui_menu_file_for_finding_illustrations => nrel_main_idtf:
    [Запрос иллюстраций для заданного объекта]
    (* <- lang_ru;; *);;
// Указываем английский идентификатор команды меню
ui_menu_file_for_finding_illustrations => nrel_main_idtf:
    [Request of illustrations for given object]
    (* <- lang_en;; *);;
// Указываем шаблон для команды меню
ui_menu_file_for_finding_illustrations => ui_nrel_command_template:
    [*
        question_of_finding_illustrations _-> ._question_of_finding_illustrations_instance
            (*
            _-> ui_arg_1;;
            *);;
        ._question_of_finding_illustrations_instance _<- question;;
    *];;
// Указываем текстовый шаблон команды на русском языке
ui_menu_file_for_finding_illustrations => ui_nrel_command_lang_template:
    [Запрос иллюстраций для заданного понятия: $ui_arg_1]
```

```
        (* <- lang_ru;; *);;
// Указываем текстовый шаблон команды на английском языке
ui_menu_file_for_finding_illustrations => ui_nrel_command_lang_template:
    [Request of illustrations for given object: $ui_arg_1]
        (* <- lang_en;; *);;
```

3. В этой же папке необходимо открыть файл ui_na_various_objects.scs. Внутри него необходимо добавить имя подпункта меню (добавленный текст выделен курсивом).

```
ui_na_various_objects <- ui_user_command_class_noatom;
=> nrel_main_idtf: [Команды для произвольных объектов] (* <- lang_ru;; *);
=> nrel_main_idtf: [Commands for various objects] (* <- lang_en;; *);
<= nrel_ui_commands_decomposition:
{
    ui_menu_file_for_finding_illustrations;
    ui_menu_file_for_finding_definitions;
    ui_menu_view_links_of_relation_connected_with_element
};;
```

Важно отметить, что точка с запятой после последнего имени не нужна.

4. Необходимо пересобрать базу знаний и перезапустить сервер (restart_sctp, run_scweb). В результате создания подпункта меню отобразится следующее (рисунок 2.2).

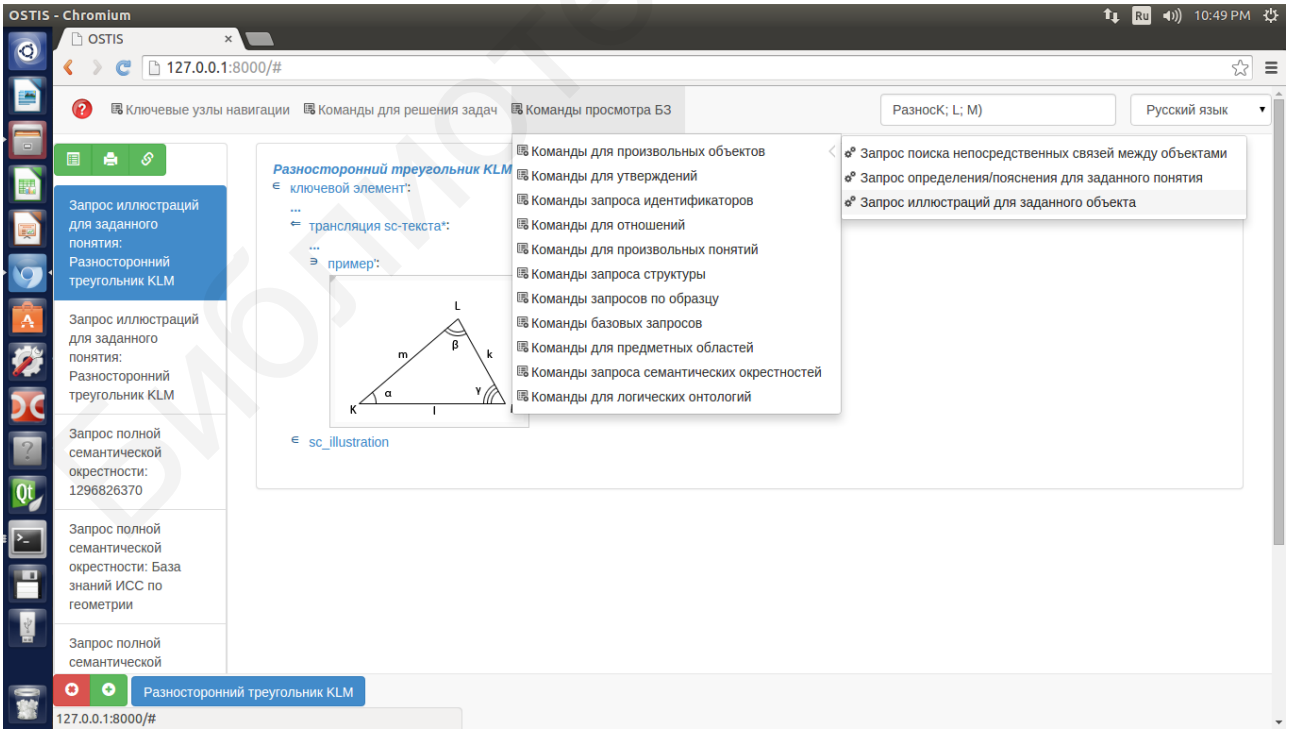

Рисунок 2.2 – Созданный подпункт меню в системе

Создание scp-программы для поиска

Для создания sc-агента нужна агентная программа, поисковая программа (процедура) и спецификация sc-агента.

Для создания процедуры, которая будет выполнять поиск иллюстраций, необходимо выполнить следующую последовательность действий:

1. Создать папку, в которой будут храниться процедуры. Например, lib_component_proc_of_finding_illustrations. Тогда путь будет следующим: ims.ostis.kb/knowledge_base_IMS/doc_technology_ostis/section_library_OSTIS/section_library_of_reusable_components_processing_machinery_knowledge/section_library_of_reusable_programs_for_sc_text_processing/lib_scp_program/lib_component_proc_of_finding_illustrations.

2. В этой папке необходимо создать файл proc_of_finding_illustrations.scs. В качестве его содержимого необходимо установить следующий код.

```
// Объявляем процедуру как scp-программу
scp_program -> proc_of_finding_illustrations
	(*
	-> rrel_params: .proc_of_finding_illustrations_params
		(*
		-> rrel_1: rrel_in: _elem;;
		-> rrel_2: rrel_in: _answer;;
		*);;
// Устанавливаем связь между процедурой и множеством ее операторов
	-> rrel_operators: .proc_of_finding_illustrations_operator_set
		(*
// Данный оператор принадлежит множеству операторов
// с атрибутом rrel_init, это означает, что он будет выполняться первым
		-> rrel_init: .proc_of_finding_illustrations_operator1A
			(*
			<- searchSetStr5;;
			-> rrel_1: rrel_assign: rrel_scp_var: _elem2;;
			-> rrel_2: rrel_assign: rrel_const: rrel_common: rrel_scp_var: _arc1;;
			-> rrel_3: rrel_fixed: rrel_scp_var: _elem;;
		-> rrel_4: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc2;;
			-> rrel_5: rrel_fixed: rrel_scp_const: nrel_sc_text_translation;;
		-> rrel_set_1: rrel_assign: rrel_scp_var: _mainset;;
			=> nrel_then: .proc_of_finding_illustrations_operator1B;;
			=> nrel_else: .proc_of_finding_illustrations_operator_return;;
		*);;
		-> .proc_of_finding_illustrations_operator1B
			(*
			<- searchElStr3;;
			-> rrel_1: rrel_fixed: rrel_scp_var: _mainset;;
			-> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc2;;
			-> rrel_3: rrel_assign: rrel_scp_var: _translit;;
			=> nrel_then: .proc_of_finding_illustrations_operator1C;;
```

```
        => nrel_else: .proc_of_finding_illustrations_operator3A;;
            *);;
        -> .proc_of_finding_illustrations_operator1C
            (*
            <- eraseEl;;
            -> rrel_1: rrel_fixed: rrel_erase: rrel_scp_var: _arc2;;
        => nrel_goto: .proc_of_finding_illustrations_operator1D;;
            *);;
// Добавим параметр операции во множество, содержащее конструкцию ответа
        -> .proc_of_finding_illustrations_operator1D
            (*
            <- genElStr3;;
            -> rrel_1: rrel_fixed: rrel_scp_var: _answer;;
            -> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc1;;
            -> rrel_3: rrel_fixed: rrel_scp_var: _translit;;
        => nrel_goto: .proc_of_finding_illustrations_operator2A;;
            *);;
        -> .proc_of_finding_illustrations_operator2A
            (*
            <- searchSetStr5;;
            -> rrel_1: rrel_fixed: rrel_scp_var: _translit;;
            -> rrel_2: rrel_assign: rrel_const: rrel_common: rrel_scp_var: _arc1;;
            -> rrel_3: rrel_fixed: rrel_scp_var: _elem;;
        -> rrel_4: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc2;;
            -> rrel_5: rrel_fixed: rrel_scp_const: nrel_sc_text_translation;;
        -> rrel_set_2: rrel_fixed: rrel_scp_var: _answer;;
            -> rrel_set_4: rrel_fixed: rrel_scp_var: _answer;;
        => nrel_then: .proc_of_finding_illustrations_operator2B;;
            => nrel_else: .proc_of_finding_illustrations_operator_return;;
        *);;
        -> .proc_of_finding_illustrations_operator2B
            (*
            <- searchSetStr5;;
            -> rrel_1: rrel_fixed: rrel_scp_var: _translit;;
            -> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc1;;
            -> rrel_3: rrel_assign: rrel_scp_var: _opr;;
        -> rrel_4: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc2;;
            -> rrel_5: rrel_fixed: rrel_scp_const: rrel_example;;
        -> rrel_set_2: rrel_fixed: rrel_scp_var: _answer;;
        -> rrel_set_3: rrel_fixed: rrel_scp_var: _answer;;
            -> rrel_set_4: rrel_fixed: rrel_scp_var: _answer;;
        => nrel_then: .proc_of_finding_illustrations_operator1B;;
            => nrel_else: .proc_of_finding_illustrations_operator_return;;
        *);;

// Проверяем, есть ли в узле с конструкцией ответа найденный элемент
            -> .proc_of_finding_illustrations_operator3A
            (*
            <- genElStr3;;
            -> rrel_1: rrel_fixed: rrel_scp_var: _answer;;
            -> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc1;;
```

```
                -> rrel_3: rrel_fixed: rrel_scp_const: rrel_example;;

        => nrel_goto: .proc_of_finding_illustrations_operator3B;;
                *);;

        -> .proc_of_finding_illustrations_operator3B
                (*
                <- genElStr3;;
                -> rrel_1: rrel_fixed: rrel_scp_var: _answer;;
                -> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc1;;
                -> rrel_3: rrel_fixed: rrel_scp_const: nrel_sc_text_translation;;
// Переходим к оператору отчистки памяти
        => nrel_goto: .proc_of_finding_illustrations_operator3C;;
                *);;
//Очистка памяти, не забываем про атрибут rrel_erase при удаляемом элементе
         -> .proc_of_finding_illustrations_operator3C
                (*
                <- eraseEl;;
                -> rrel_1: rrel_fixed: rrel_erase: rrel_scp_var: _mainset;;
// Переходим к оператору завершения операции
                => nrel_goto: .proc_of_finding_illustrations_operator_return;;
                *);;
// Оператор завершения операции
        -> .proc_of_finding_illustrations_operator_return
                (*
                <- return;;
                *);;
        *);;
*);;
```

Создание агентной scp-программы

Далее необходимо создать агентную scp-программу, которая будет вызывать созданную ранее процедуру.

Агентные scp-программы представляют собой реализации программ агентов обработки знаний и имеют жестко фиксированную структуру множества параметров. Множество параметров в данном случае состоит из двух in-параметров.

Для создания агентной scp-программы необходимо выполнить следующие шаги:

1. Перейти в следующий каталог: ims.ostis.kb/knowledge_base_IMS/doc_technology_ostis/section_library_OSTIS/section_library_of_reusable_components_processing_machinery_knowledge/lib_scp_agents/lib_scp_agents_search/lib_component_agent_of_finding_illustrations.

2. Создать файл agent_of_finding_illustrations.scs. В качестве его содержимого необходимо установить следующий код:

```
agent_of_finding_illustrations
// Множество идентификаторов
=> nrel_main_idtf:
	[агентная scp-программа поиска иллюстраций для заданного объекта] (* <- lang_ru;; *);
	[agent scp-program of finding illustrations for given object] (* <- lang_en;; *);
	<- agent_scp_program;;
// Указываем, что операция поиска агента является scp-программой
scp_program -> agent_of_finding_illustrations
	(*
// Множество параметров агентной операции
	-> rrel_params: .agent_of_finding_illustrations_params
		(*
		-> rrel_1: rrel_in: _event;;
		-> rrel_2: rrel_in: _input_arc;;
		*);;
// Множество операторов агентной операции
	-> rrel_operators: .agent_of_finding_illustrations_operator_set
		(*
// Первый исполняемый оператор операции
		-> rrel_init: .agent_of_finding_illustrations_operator1A
			(*
			<- searchElStr3;;
			-> rrel_1: rrel_assign: rrel_scp_var: _temp;;
			-> rrel_2: rrel_fixed: rrel_scp_var: _input_arc;;
			-> rrel_3: rrel_assign: rrel_scp_var: _quest;;

			=> nrel_goto: .agent_of_finding_illustrations_operator1B;;
			*);;
// В rrel_1 устанавливаем имя придуманного нами вопроса
// Агент будет просыпаться, когда обнаружит наш вопрос
		-> .agent_of_finding_illustrations_operator1B
			(*
			<- searchElStr3;;
			-> rrel_1: rrel_fixed: rrel_scp_const: question_of_finding_illustrations;;
			-> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc;;
			-> rrel_3: rrel_fixed: rrel_scp_var: _quest;;

			=> nrel_then: .agent_of_finding_illustrations_operator1C;;
			=> nrel_else: .agent_of_finding_illustrations_operator_return;;
			*);;
// Найдем параметры операции – их подал на вход сам пользователь
		-> .agent_of_finding_illustrations_operator1C
			(*
			<- searchElStr3;;
			-> rrel_1: rrel_fixed: rrel_scp_var: _quest;;
			-> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc;;
```

```
                    -> rrel_3: rrel_assign: rrel_scp_var: _param;;

                    => nrel_then: .agent_of_finding_illustrations_operator1D;;
                    => nrel_else: .agent_of_finding_illustrations_operator_return;;
                    *);;
// Сгенерировать узел, в который мы поместим конструкцию ответа
          -> .agent_of_finding_illustrations_operator1D
                    (*
                    <- genEl;;
                    -> rrel_1: rrel_assign: rrel_const: rrel_node: rrel_scp_var: _answer;;

                    => nrel_goto: .agent_of_finding_illustrations_operator2A;;
                    *);;
           -> .agent_of_finding_illustrations_operator2A
                    (*
                    <- searchSetStr5;;
                    -> rrel_1: rrel_assign: rrel_scp_var: _elem;;
                    -> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc1;;
                    -> rrel_3: rrel_fixed: rrel_scp_var: _param;;
              -> rrel_4: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc2;;
                    -> rrel_5: rrel_fixed: rrel_scp_const: rrel_key_sc_element;;
              -> rrel_set_1: rrel_assign: rrel_scp_var: _set1;;
              => nrel_then: .agent_of_finding_illustrations_operator2B;;
                    => nrel_else: .agent_of_finding_illustrations_operator_return;;
              *);;

              -> .agent_of_finding_illustrations_operator2B
                    (*
                    <- searchElStr3;;
                    -> rrel_1: rrel_fixed: rrel_scp_var: _set1;;
                    -> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc;;
                    -> rrel_3: rrel_assign: rrel_scp_var: _inst;;

                    => nrel_then: .agent_of_finding_illustrations_operator2C;;
                    => nrel_else: .agent_of_finding_illustrations_operator4A;;
                    *);;

              -> .agent_of_finding_illustrations_operator2C
                    (*
                    <- eraseEl;;
              -> rrel_1: rrel_fixed: rrel_erase: rrel_scp_var: _arc;;

              => nrel_goto:  .agent_of_finding_illustrations_operator2D;;
                    *);;

              -> .agent_of_finding_illustrations_operator2D
                    (*
                    <- searchElStr3;;
                    -> rrel_1: rrel_fixed: rrel_scp_const: sc_illustration;;
                    -> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc1;;
                    -> rrel_3: rrel_fixed: rrel_scp_var: _inst;;
```

```
                => nrel_then: .agent_of_finding_illustrations_operator2E;;
                => nrel_else: .agent_of_finding_illustrations_operator2B;;
                *);;

        -> .agent_of_finding_illustrations_operator2E
                (*
                <- genElStr3;;
                -> rrel_1: rrel_fixed: rrel_scp_var: _answer;;
                -> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc1;;
                -> rrel_3: rrel_fixed: rrel_scp_var: _inst;;

        => nrel_goto: .agent_of_finding_illustrations_operator2F;;
        *);;

        -> .agent_of_finding_illustrations_operator2F
                (*
                <- searchSetStr5;;
                -> rrel_1: rrel_fixed: rrel_scp_var: _inst;;
                -> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc1;;
                -> rrel_3: rrel_fixed: rrel_scp_var: _param;;
        -> rrel_4: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc2;;
                -> rrel_5: rrel_fixed: rrel_scp_const: rrel_key_sc_element;;

        -> rrel_set_2: rrel_fixed: rrel_scp_var: _answer;;
        -> rrel_set_4: rrel_fixed: rrel_scp_var: _answer;;

        => nrel_then: .agent_of_finding_illustrations_operator2G;;
        => nrel_else: .agent_of_finding_illustrations_operator_return;;
        *);;

        -> .agent_of_finding_illustrations_operator2G
                (*
                <- searchSetStr3;;
                -> rrel_1: rrel_fixed: rrel_scp_const: sc_illustration;;
                -> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc1;;
                -> rrel_3: rrel_fixed: rrel_scp_var: _inst;;

        -> rrel_set_2: rrel_fixed: rrel_scp_var: _answer;;

        => nrel_then: .agent_of_finding_illustrations_operator3A;;
        => nrel_else: .agent_of_finding_illustrations_operator_return;;
        *);;

        -> .agent_of_finding_illustrations_operator3A
                (*
                <- call;;
                -> rrel_1: rrel_fixed: rrel_scp_const: proc_of_finding_illustrations;;
                -> rrel_2: rrel_fixed: rrel_scp_const:
.agent_of_finding_illustrations_operator3A_params
                        (*
```

```
                        -> rrel_1: rrel_fixed: rrel_scp_var: _inst;;
                        -> rrel_2: rrel_fixed: rrel_scp_var: _answer;;
                        *);;
                -> rrel_3: rrel_assign: rrel_scp_var: _descr1;;

                => nrel_goto: .agent_of_finding_illustrations_operator3B;;
                *);;
// scp-оператор ожидания завершения выполнения программы
        -> .agent_of_finding_illustrations_operator3B
                (*
                <- waitReturn;;
                -> rrel_1: rrel_fixed: rrel_scp_var: _descr1;;

                => nrel_goto: .agent_of_finding_illustrations_operator2B;;
                *);;

        -> .agent_of_finding_illustrations_operator4A
                (*
                <- genElStr3;;
                -> rrel_1: rrel_fixed: rrel_scp_var: _answer;;
                -> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc1;;
                -> rrel_3: rrel_fixed: rrel_scp_const: rrel_key_sc_element;;

        => nrel_goto: .agent_of_finding_illustrations_operator4B;;
        *);;

        -> .agent_of_finding_illustrations_operator4B
                (*
                <- genElStr3;;
                -> rrel_1: rrel_fixed: rrel_scp_var: _answer;;
                -> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc1;;
                -> rrel_3: rrel_fixed: rrel_scp_var: _param;;

        => nrel_goto: .agent_of_finding_illustrations_operator4C;;
        *);;

        -> .agent_of_finding_illustrations_operator4C
                (*
                <- genElStr3;;
                -> rrel_1: rrel_fixed: rrel_scp_var: _answer;;
                -> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc1;;
                -> rrel_3: rrel_fixed: rrel_scp_const: sc_illustration;;

        => nrel_goto: .agent_of_finding_illustrations_operator4D;;
        *);;

      -> .agent_of_finding_illustrations_operator4D
                (*
                <- eraseEl;;
        -> rrel_1: rrel_fixed: rrel_erase: rrel_scp_var: _set1;;
```

```
            => nrel_goto:  .agent_of_finding_illustrations_operator_gen_answer;;
                *);;
// Добавляем узел с конструкцией ответа
            -> .agent_of_finding_illustrations_operator_gen_answer
                (*
                <- genElStr5;;
                -> rrel_1: rrel_fixed: rrel_scp_var: _quest;;
                -> rrel_2: rrel_assign: rrel_const: rrel_common: rrel_scp_var: _arc;;
                -> rrel_3: rrel_fixed: rrel_scp_var: _answer;;
                -> rrel_4: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc2;;
                -> rrel_5: rrel_fixed: rrel_scp_const: nrel_answer;;

                => nrel_goto: .agent_of_finding_illustrations_operator_return;;
                *);;
// Оператор завершения агентной операции
            -> .agent_of_finding_illustrations_operator_return
                (*
                <- return;;
                *);;
            *);;
    *);;
```

Спецификация sc-агента

Каждый абстрактный sc-агент имеет соответствующую ему спецификацию, рассмотренную в пункте 1.4.3.

Для спецификации агента необходимо выполнить следующую последовательность действий:

1. Перейти в ~/ims.ostis.kb/knowledge_base_IMS/doc_technology_ostis/section_library_OSTIS/section_library_of_reusable_components_processing_machinery_knowledge/lib_scp_agents/lib_scp_agents_search/lib_component_agent_of_finding_illustrations.

2. Открыть sc_agent_of_finding_illustrations.scs.

3. Дописать в самый конец файла:

```
// Объявляем sc-агент
sc_agent_of_finding_illustrations
// Указываем множество идентификаторов агента
=> nrel_main_idtf:
// lang_ru –идентификатор на русском языке
    [sc-агент поиска иллюстраций для заданного объекта] (* <- lang_ru;; *);
// lang_en –идентификатор на английском языке
    [sc-agent of finding illustrations for given object] (* <- lang_en;; *);
// Указываем, что наш агент принадлежит классу абстрактных sc-агентов
<- abstract_sc_agent;
// Указываем первичное условие инициирования sc-агента
=> nrel_primary_initiation_condition:
    (sc_event_add_output_arc => question_initiated);
```

```
// Указываем начальное условие и результат для sc-агента
=> nrel_initiation_condition_and_result:
        (..sc_agent_of_finding_illustrations_condition                    =>
..sc_agent_of_finding_illustrations_result);
// Описание ключевых sc-элементов sc-агента
<= nrel_sc_agent_key_sc_elements:
        {
        question_initiated;
        question;
        question_of_finding_illustrations
        };
// Описание реализации sc-агента, т. е. связи sc-агента и его программ
=> nrel_inclusion:
        .platform_independent_realization_of_sc_agent_of_finding_illustrations
        (*
        <- platform_independent_abstract_sc_agent;;
        <= nrel_sc_agent_program:
                {
                agent_of_finding_illustrations;
                proc_of_finding_illustrations
                };;
        -> sc_agent_of_finding_illustrations_scp
                (* <- active_sc_agent;; *);;
        *);;
// Условие инициирования агента
..sc_agent_of_finding_illustrations_condition
= [*
        question_of_finding_illustrations _-> .._question;;
        question_initiated _-> .._question;;
        question _-> .._question;;
        .._question _-> .._parameter;;
*];;
// Описание результатов выполнения sc-агента
..sc_agent_of_finding_illustrations_result
= [*
        question_of_finding_illustrations _-> .._question;;
        question_finished _-> .._question;;
        question _-> .._question;;
        .._question _=> nrel_answer:: .._answer;;
        .._question _-> .._parameter;;
*];;
```

4. Пересобрать базу и перезапустить sctp-сервер. Вот что будет выдавать sctp-server (выделен разработанный sc-агент и процедура, рисунок 2.3).

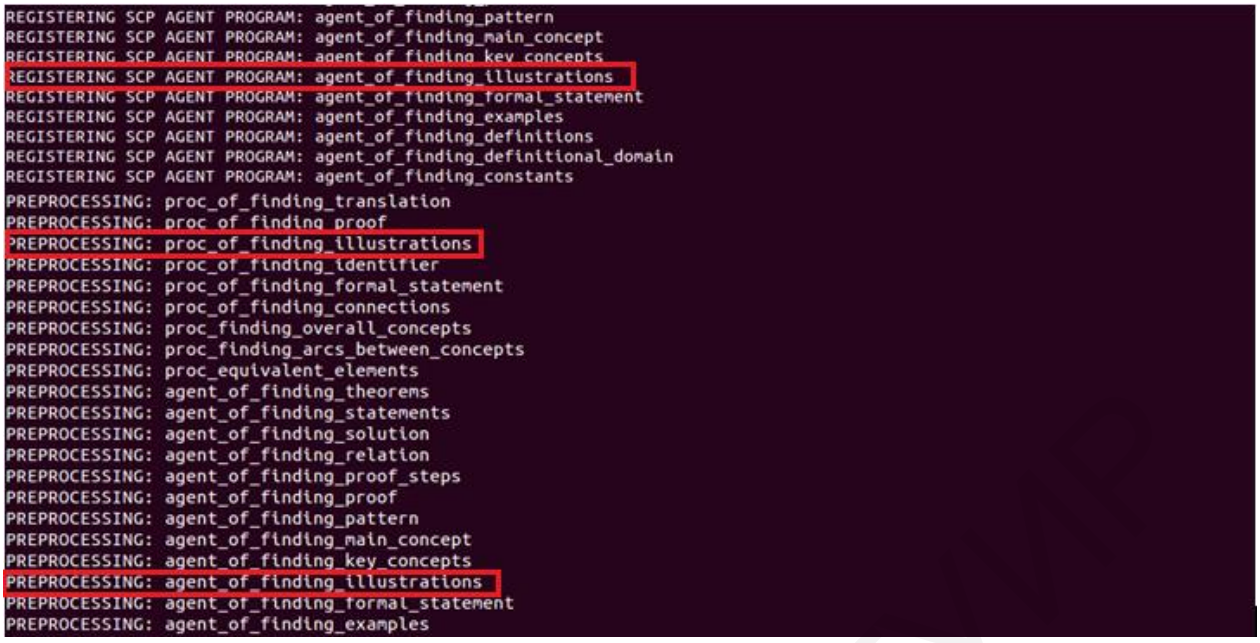

Рисунок 2.3 – Результат запуска sctp-сервера

Демонстрация работы sc-агента

Продемонстрируем простейшие примеры работы созданного sc-агента на примере системы по геометрии Евклида (рисунки 2.4–2.10):

1. Запрос поиска иллюстраций для узла «Разносторонний треугольник».

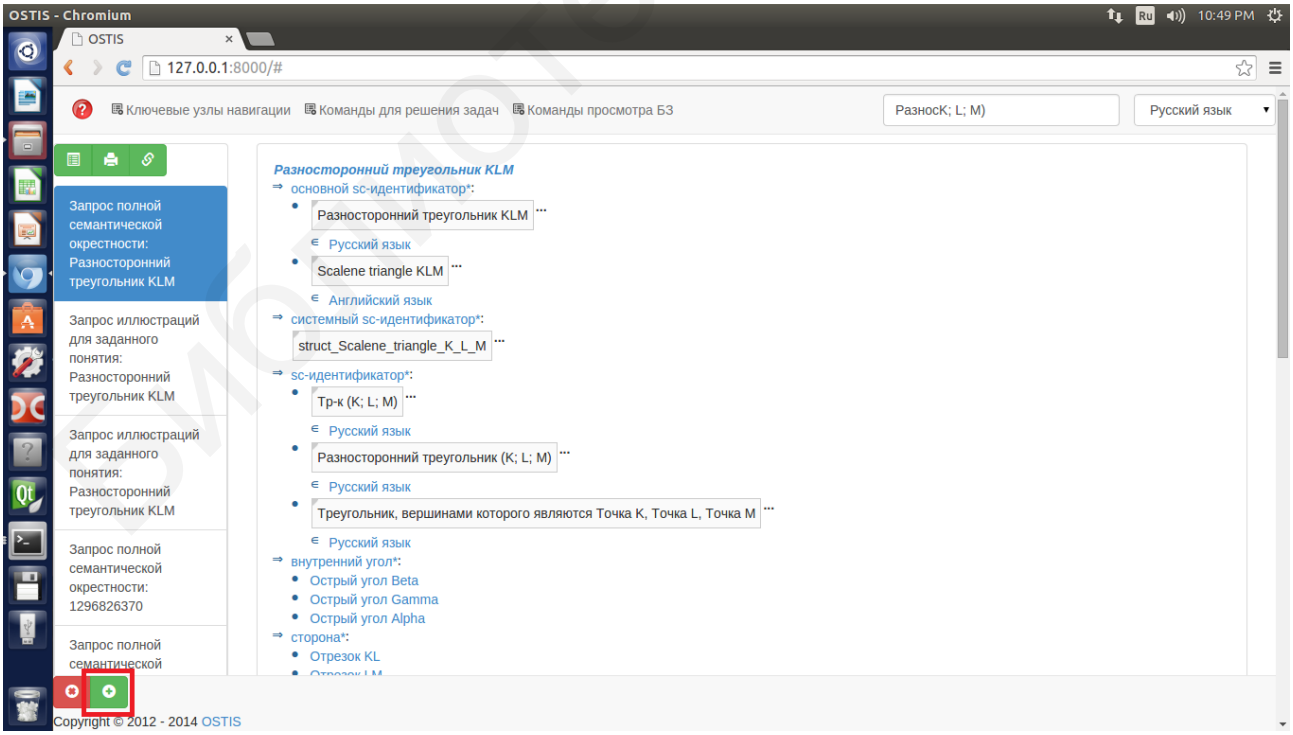

Рисунок 2.4 – Выбор режима задания аргумента

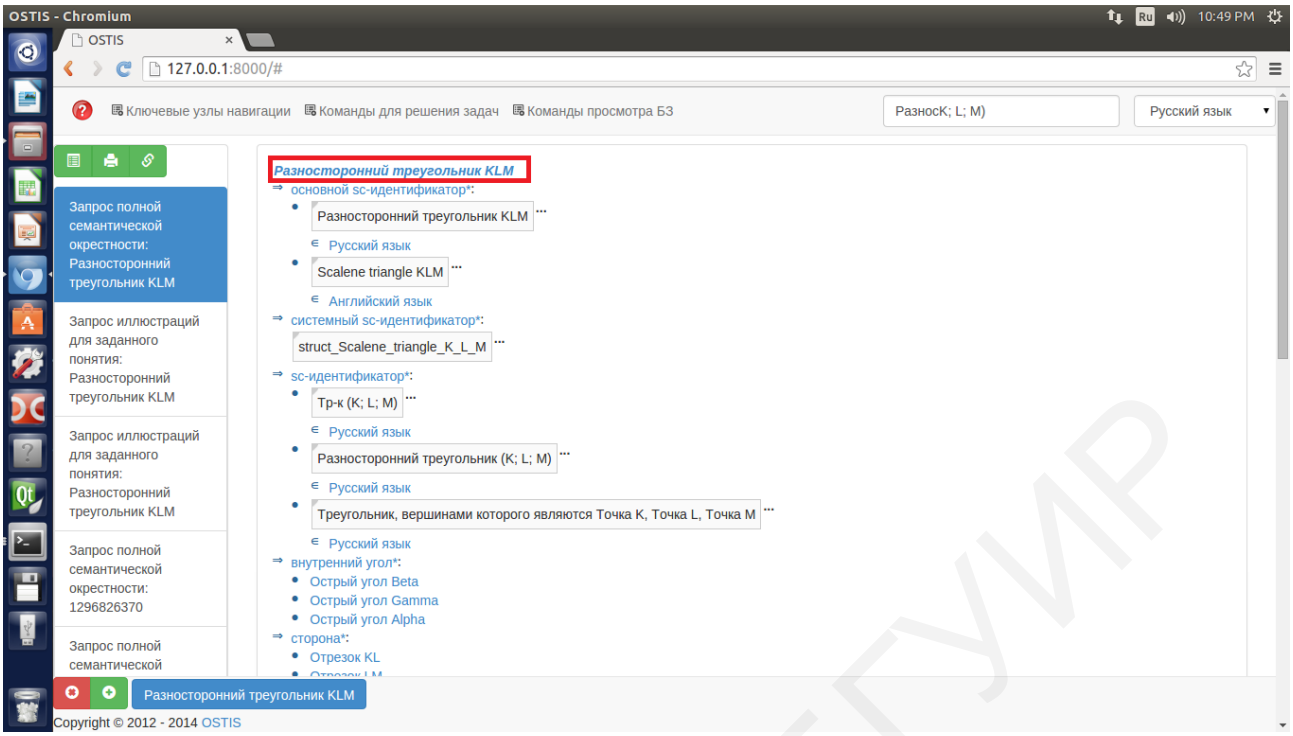

Рисунок 2.5 – Выбор треугольника KLM в качестве аргумента

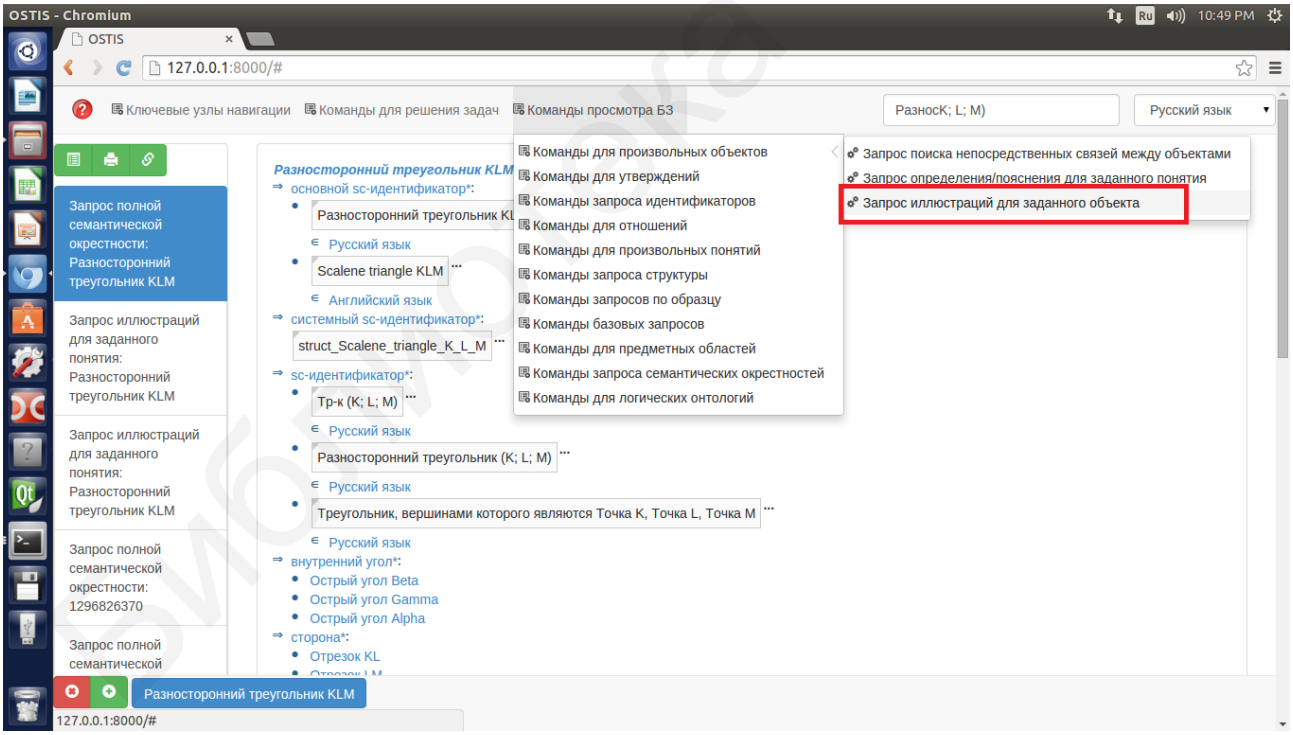

Рисунок 2.6 – Выбор команды запроса иллюстраций

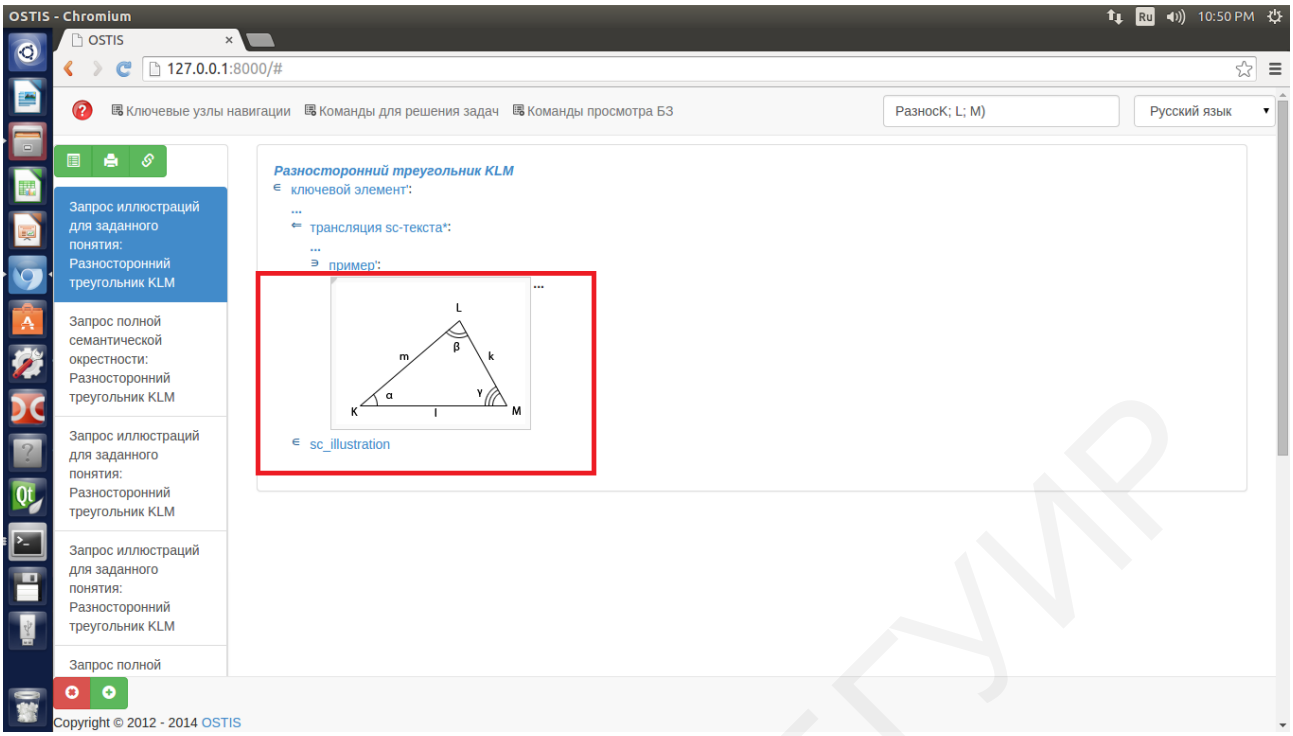

Рисунок 2.7 – Результат работы sc-агента поиска иллюстраций

2. Запрос поиска иллюстраций для узла «Прямоугольная трапеция» (рисунки 2.8–2.10).

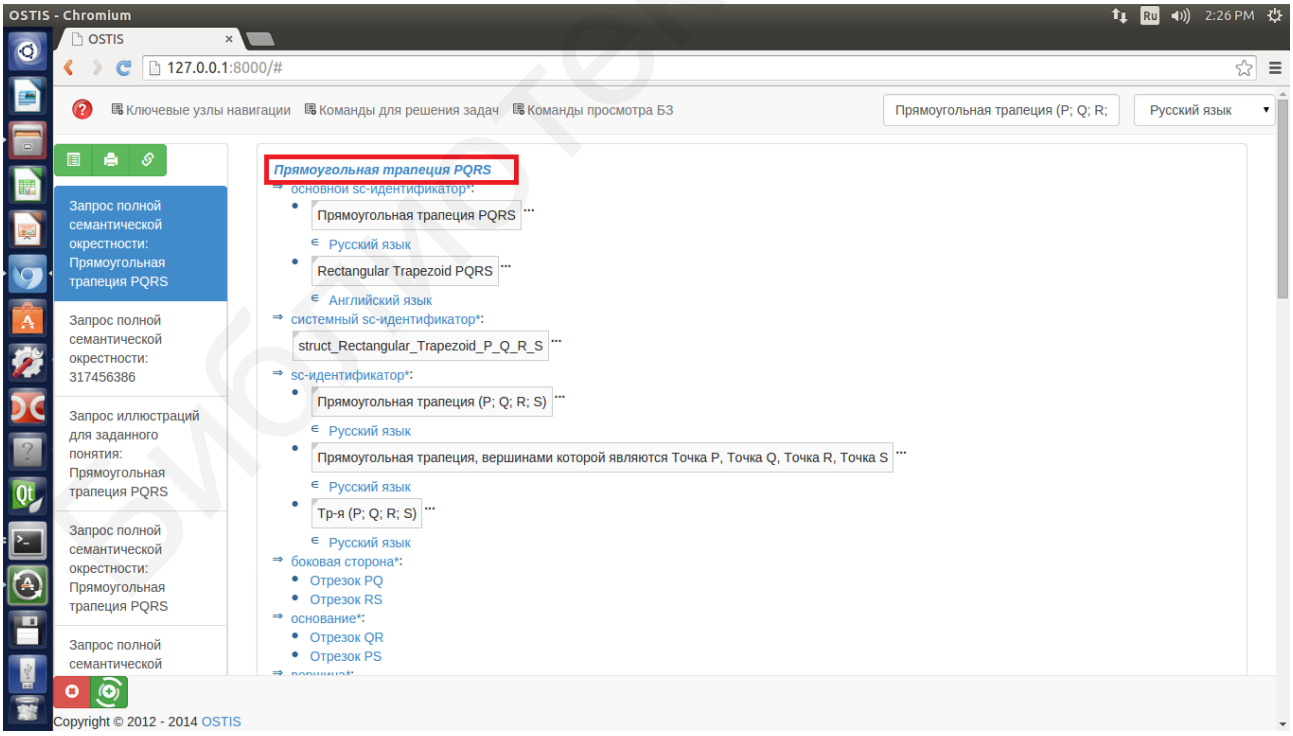

Рисунок 2.8 – Выбор трапеции PQRS в качестве аргумента

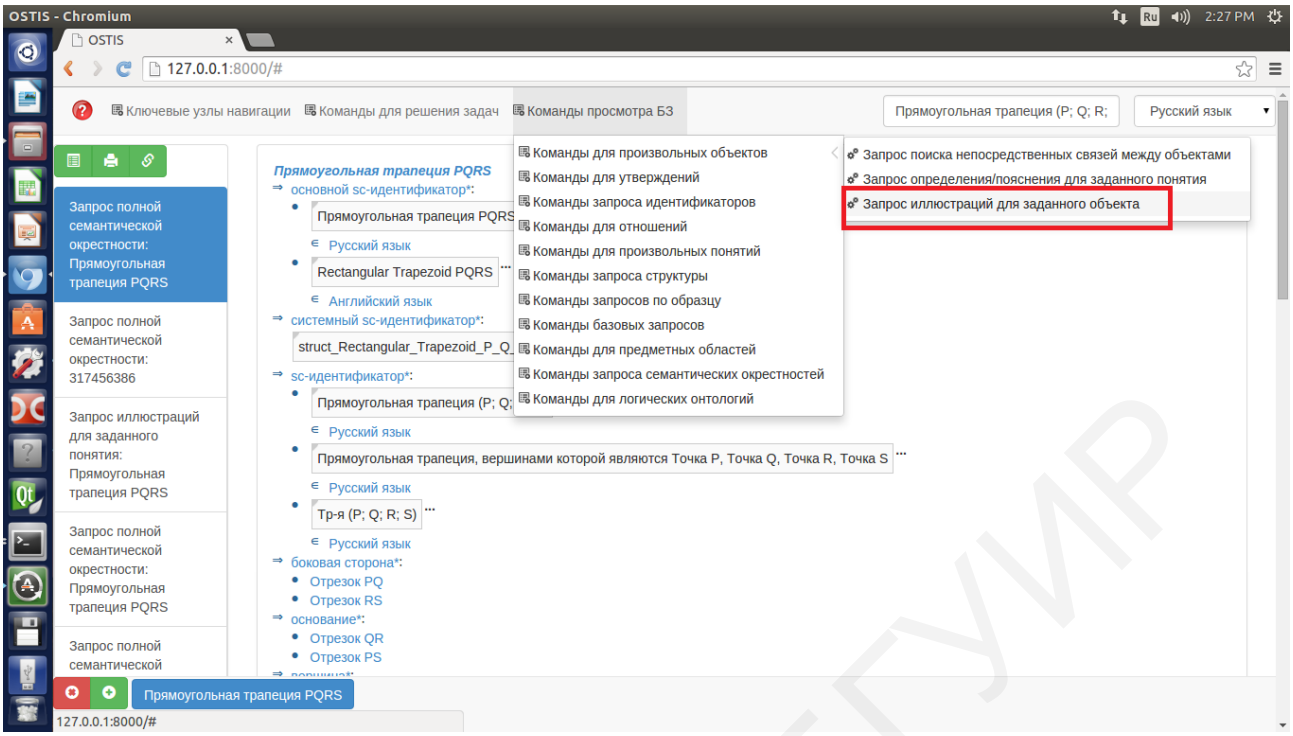

Рисунок 2.9 – Выбор команды поиска иллюстраций для трапеции

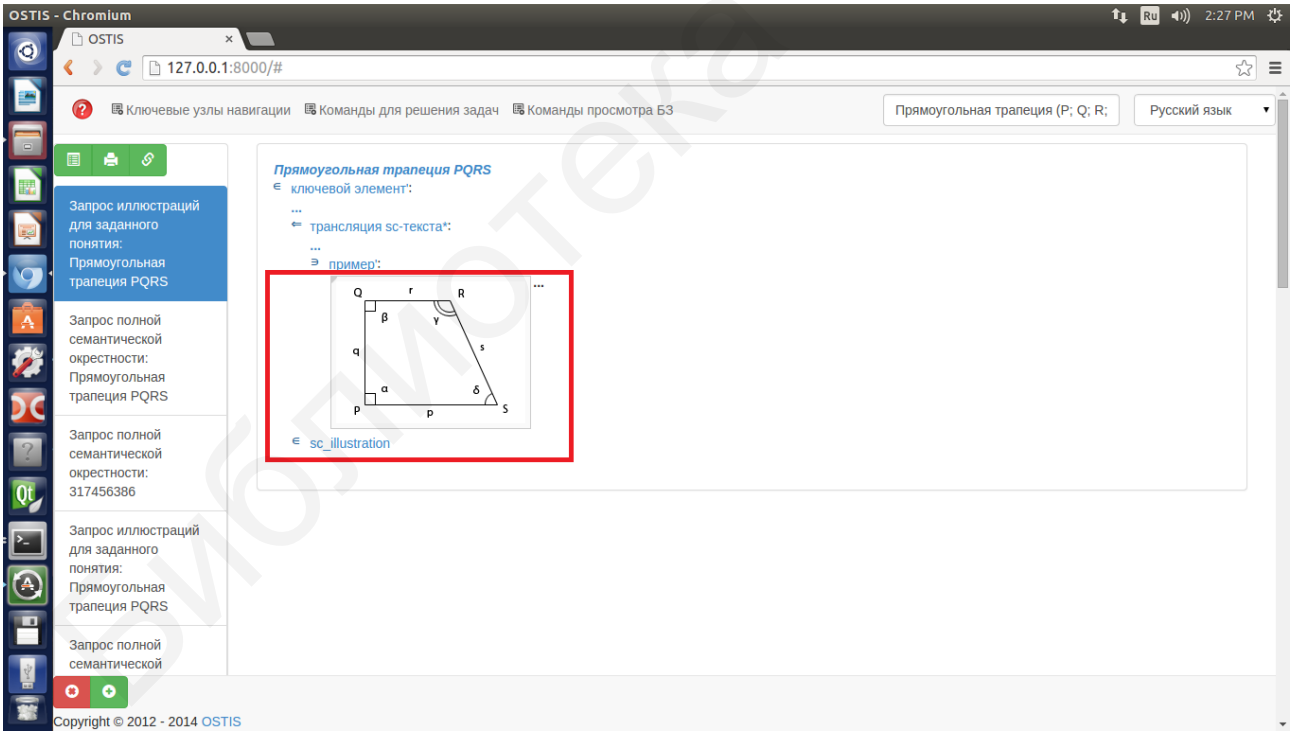

Рисунок 2.10 – Результат работы sc-агента поиска иллюстраций для трапеции

2.2 БИБЛИОТЕКА КОМПОНЕНТОВ РЕШАТЕЛЕЙ ЗАДАЧ

Библиотека многократно используемых компонентов решателей задач является важнейшим фрагментом метасистемы IMS, обеспечивающим надежность проектируемых решателей и повышение скорости их разработки.

Библиотека включает:

– собственно множество компонентов решателей задач;

– средства спецификации компонентов решателей задач;

– средства поиска компонентов решателей задач на основе их спецификации.

Под *многократно используемым компонентом OSTIS* вообще понимается компонент некоторой ostis-системы, который может быть использован в другой ostis-системе. Для этого необходимо выполнение как минимум двух условий:

1) есть техническая возможность встроить компонент в дочернюю ostis-систему путем либо физического копирования, переноса и встраивания его в проектируемую систему, либо использования компонента, размещенного в исходной системе наподобие сервиса, т. е. без явного копирования и переноса компонента. Трудоемкость встраивания зависит в том числе от реализации компонента;

2) использование компонента в каких-либо ostis-системах, кроме метасистемы IMS, является целесообразным, т. е. компонентом не может быть частное решение, ориентированное на узкий круг задач. Стоит, однако, отметить, что в общем случае практически каждое решение может быть использовано в каких-либо других системах, круг которых определяется степенью общности и предметной зависимостью такого решения.

С формальной точки зрения каждый многократно используемый компонент OSTIS представляет собой структуру, которая содержит все те (и только те) sc-элементы, которые необходимы для функционирования компонента в дочерней ostis-системе и, соответственно, должны быть в нее скопированы при включении компонента в одну из таких систем. Конкретный состав данной структуры зависит от типа компонента и уточняется для каждого типа отдельно. По сути, данная структура представляет собой эталон (или образец), который копируется при включении соответствующего компонента в дочернюю систему.

Каждый *многократно используемый компонент OSTIS* может быть атомарным либо неатомарным, т. е. состоять из более простых самодостаточных компонентов.

В зависимости от типа компонента в его состав, т. е. в состав соответствующей *структуры*, могут дополнительно вводиться роли некоторых *sc-элементов*, если это необходимо. Например, в случае *многократно используемого sc-агента* сам *sc-узел*, обозначающий *sc-агент*, будет являться *ключевым sc-элементом'* в рамках компонента.

В каждый момент времени в текущем состоянии sc-памяти каждый многократно используемый компонент может быть представлен полностью, т. е. в памяти явно присутствуют все *sc-дуги принадлежности*, соединяющие соответствующую компоненту *структуру* и все ее элементы, или представлен неявно, например, при помощи указания *ключевых sc-элементов'* данного компонента или путем задания декомпозиции данного компонента на более частные.

Рассмотрим общую классификацию многократно используемых компонентов OSTIS в SCn-коде.

Библиотека многократно используемых компонентов OSTIS
= Библиотека OSTIS
= многократно используемый компонент OSTIS
= многократно используемый компонент интеллектуальных систем, построенных по технологии OSTIS
*<= разбиение**:
{
- *Семейство платформ интерпретации sc-моделей компьютерных систем*
- *Библиотека многократно используемых компонентов sc-моделей баз знаний*
- *Библиотека шаблонов типовых компонентов sc-моделей компьютерных систем*
- *Библиотека многократно используемых компонентов решателей задач*
- *Библиотека многократно используемых компонентов sc-моделей интерфейсов компьютерных систем*
- *Библиотека типовых подсистем компьютерных систем, разрабатываемых по технологии OSTIS*

}

В данном учебном пособии основное внимание уделено *многократно используемым компонентам решателей задач*. Формально *Библиотека многократно используемых компонентов решателей задач* является фрагментом *предметной области абстрактных агентов, работающих над унифицированной семантической памятью*.

Если *многократно используемый компонент решателей задач* является *платформенно-зависимым многократно используемым компонентом OSTIS*, то его интеграция производится в соответствии с инструкцией, предоставляемой разработчиком в зависимости от платформы, как и для любого компонента

такого рода. В противном случае процесс интеграции можно конкретизировать в зависимости от подклассов данного типа компонентов.

Рассмотрим классификацию многократно используемых компонентов решателей задач в SCn-коде.

Библиотека многократно используемых компонентов решателей задач
= многократно используемый компонент решателей задач
*<= разбиение**:
{
- *Библиотека многократно используемых решателей задач*
- *Библиотека многократно используемых атомарных абстрактных sc-агентов*
- *Библиотека многократно используемых программ обработки sc-текстов*

}

Под *многократно используемым абстрактным sc-агентом* подразумевается компонент, соответствующий некоторому *абстрактному sc-агенту*, который может быть использован в других системах, возможно, в составе более сложных *неатомарных абстрактных sc-агентов*. Указанный абстрактный sc-агент входит в соответствующую компоненту *структуру* под атрибутом *ключевой sc-элемент'*. Каждый *многократно используемый абстрактный sc-агент* должен содержать всю информацию, необходимую для функционирования соответствующего *sc-агента* в дочерней системе.

Рассмотрим классификацию многократно используемых sc-агентов в SCn-коде.

Библиотека многократно используемых абстрактных sc-агентов
= многократно используемый абстрактный sc-агент
*<= разбиение**:
{
- *Библиотека sc-агентов информационного поиска*
- *Библиотека sc-агентов погружения интегрируемого знания в базу знаний*
- *Библиотека sc-агентов выравнивания онтологии интегрируемого знания с основной онтологией текущего состояния базы знаний*
- *Библиотека sc-агентов планирования решения явно сформулированных задач*
- *Библиотека sc-агентов логического вывода*
- *Библиотека sc-моделей языков программирования высокого уровня и соответствующих им интерпретаторов*
- *Библиотека sc-агентов верификации базы знаний*
- *Библиотека sc-агентов редактирования базы знаний*
- *Библиотека sc-агентов автоматизации деятельности разработчиков базы знаний*

}

Под *многократно используемой программой обработки sc-текстов* подразумевается компонент, соответствующий программе, записанной на произвольном языке программирования и ориентированной на обработку *структур*, хранящихся в памяти *ostis-системы*. Приоритетным в данном случае является использование *scp-программ* по причине их платформенной независимости, за исключением случаев проектирования некоторых компонентов интерфейса, когда полная платформенная независимость невозможна (например, при проектировании *эффекторных sc-агентов* и *рецепторных sc-агентов*).

В свою очередь, под *многократно используемой scp-программой* понимается компонент, соответствующий некоторой достаточно универсальной *scp-программе*, которая может быть использована в составе сразу нескольких *sc-агентов*.

В *многократно используемую scp-программу* включается полный текст *scp-программы*, т. е. все *sc-элементы*, принадлежащие *структуре*, являющейся *scp-программой*, а также все пары принадлежности между этой *структурой* и ее элементами и знак самой *структуры*. При этом sc-узел, обозначающий scp-программу, входит в соответствующий компонент под атрибутом *ключевой sc-элемент'*.

После того как *многократно используемая scp-программа* была скопирована в дочернюю систему, необходимо добавить ее в множество корректных *scp-программ* (корректность верифицируется при попадании в библиотеку компонентов в рамках IMS).

Для удобства работы с библиотекой многократно используемых компонентов были также разработаны средства автоматизации поиска компонентов на основе заданной спецификации, представляющие собой неатомарный sc-агент, который декомпозируется на более частные.

Далее представлена структура такого агента на языке SCn.

Средства автоматизации библиотеки многократно используемых компонентов решателей задач

*<= декомпозиция sc-агента**:

{

- *Абстрактный sc-агент формирования неатомарного компонента из атомарных*
- *Абстрактный sc-агент поиска всех неатомарных компонентов, частью которых является заданный атомарный компонент*
- *Абстрактный sc-агент поиска всех сопутствующих компонентов*
- *Абстрактный sc-агент поиска sc-агента по условию инициирования*
- *Абстрактный sc-агент поиска sc-агента по результату работы*
- *Абстрактный sc-агент поиска scp-программы по входным/выходным параметрам*

- *Абстрактный sc-агент поиска sc-агентов, для которых элементы заданного множества являются ключевыми sc-элементами*

}

Под *неатомарным компонентом решателей задач* понимается такой компонент, в составе которого можно выделить другие компоненты, используемые также и самостоятельно, отдельно от исходного компонента. Чаще всего в роли таких неатомарных компонентов выступают неатомарные sc-агенты, в составе которых могут быть выделены самодостаточные sc-агенты, используемые также отдельно от исходного неатомарного, или scp-программы, которые являются общими для нескольких агентов и могут быть использованы не только в составе неатомарного sc-агента. Таким образом, задачей *Абстрактного sc-агента формирования неатомарного компонента из атомарных* является формирование структуры, содержащей в себе полный sc-текст неатомарного компонента, включая спецификации всех sc-агентов в его составе, а также тексты всех необходимых scp-программ. Формирование такой структуры необходимо для того, чтобы упростить процесс копирования указанного компонента в другие ostis-системы.

Под сопутствующим компонентом понимается компонент, который часто используется в ostis-системе одновременно с некоторым другим компонентом. Такая связь между компонентами задается явно при помощи отношения *сопутствующий компонент**. Примерами таких компонентов являются некоторый sc-агент и команда пользовательского интерфейса, позволяющая пользователю инициировать выполнение указанного агента с заданными аргументами. При этом sc-агент будет функционировать и без наличия в системе такой команды, однако для его инициирования придется сформировать соответствующую конструкцию в sc-памяти вручную.

Абстрактный sc-агент поиска sc-агентов, для которых элементы заданного множества являются *ключевыми sc-элементами,* играет важную роль при внесении изменений в базу знаний, в частности, при переопределении каких-либо понятий. Указанный sc-агент позволяет выявить те sc-агенты, для которых могут потребоваться изменения в алгоритме работы в связи с изменением семантической трактовки каких-либо понятий.

Рассмотрим пример описания компонента решателей задач на языке SCs с использованием системных идентификаторов. Компонент включает собственно поисковый sc-агент, его программу на языке SCP, а также команду пользовательского интерфейса, которая позволяет пользователю запустить этот sc-агент. Для пояснения фрагментов sc.s-текста в рамках описания приводятся естественно-языковые комментарии. В качестве примера рассмотрим sc-агент поиска конструкций по заданному образцу (шаблону).

```
lib_component_agent_of_finding_pattern
=> nrel_main_idtf:
      [Компонент библиотеки. sc-агент поиска конструкций для заданного шаблона]
      (* <- lang_ru;; *);
      [Library component. sc-agent of finding constructions for given pattern]
      (* <- lang_en;; *);

      //Классы, которым принадлежит компонент
      <- library_of_platform_independent_reusable_components;
      <- library_of_atomic_reusable_components;

      //Связь с зависимыми компонентами
      => nrel_attendant_component:
            lib_component_ui_menu_file_for_finding_pattern;

      //Связь с конкретным sc-агентом
      -> rrel_key_sc_element:
            .platform_independent_realization_of_sc_agent_of_finding_pattern;

      //Естественно-языковое описание деятельности sc-агента
      <- rrel_key_sc_element:
            ...
            (*
            <- explanation;;
            <= nrel_sc_text_translation:
                  ...
                  (*
                  -> rrel_example:
                        "file://htmls/Agent_Pattern.html"
                              (* <- lang_ru;; *);;
                  *);;
            *);;
```

Важно отметить, что в приведенном примере указана связь между зависимыми компонентами. В данном случае связью отношения «зависимый компонент» связаны компонент-команда меню пользовательского интерфейса и компонент-sc-агент. Это означает, что команда пользовательского интерфейса не имеет смысла как компонент при отсутствии соответствующего sc-агента, поскольку пользователь не сможет ей воспользоваться.

2.3 СРЕДСТВА ПОСТРОЕНИЯ И МОДИФИКАЦИИ РЕШАТЕЛЕЙ ЗАДАЧ

Средства построения и модификации решателей задач включают систему автоматизации процесса построения и модификации решателей и подсистему информационного обслуживания разработчиков решателей в рамках метасистемы IMS.

Важнейшим фрагментом метасистемы IMS является библиотека многократно используемых компонентов ostis-систем, в рамках которой выделяется библиотека многократно используемых компонентов решателей задач, построенная с учетом классификации sc-агентов (см. пункт 1.4.2).

К числу задач системы автоматизации процесса построения и модификации решателей относится техническая поддержка разработчиков решателей, в том числе обеспечение корректного и эффективного выполнения этапов, предусмотренных методикой, рассмотренной в подразделе 2.1.

При разработке любых компонентов ostis-систем используются схожие принципы. Одним из основных является принцип использования готовых компонентов различного рода, уже имеющихся в библиотеке компонентов OSTIS, входящей в состав метасистемы IMS.

Система поддержки построения и модификации решателей задач сама по себе также является ostis-системой и имеет соответствующую структуру. Таким образом, модель данной системы включает sc-модель базы знаний, sc-модель объединенного решателя задач и sc-модель пользовательского интерфейса.

С другой стороны, в рамках рассматриваемой системы условно выделяются две подсистемы – автоматизации процесса построения и модификации агентов обработки знаний, автоматизации процесса построения и модификации scp-программ.

Графически архитектуру средств построения и модификации решателей задач, включая структуру системы поддержки построения и модификации решателей задач и ее подсистем, можно изобразить схемой, представленной на рисунке 2.11.

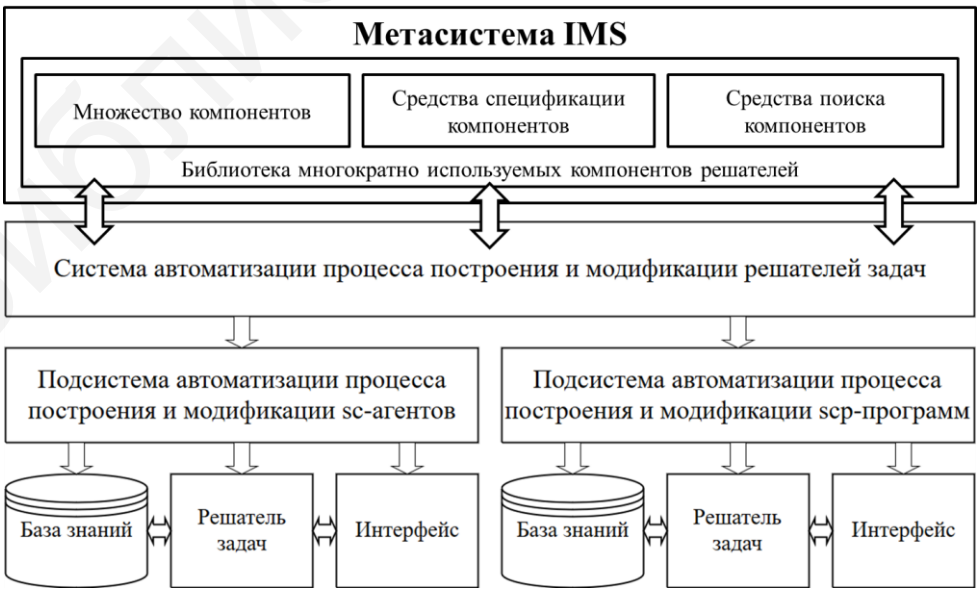

Рисунок 2.11 – Структура системы автоматизации процесса построения и модификации решателей задач

Рассматриваемая система может фактически использоваться тремя способами:

– как подсистема в рамках метасистемы поддержки проектирования компьютерных систем, управляемых знаниями (IMS): данный вариант использования предполагает отладку необходимых компонентов в рамках метасистемы с последующим переносом их в дочернюю систему;

– как самостоятельная ostis-система, предназначенная исключительно для разработки и отладки компонентов решателей задач: в этом случае проектируемые компоненты отлаживаются в рамках такой системы, а затем должны быть перенесены в дочернюю ostis-систему;

– как подсистема в рамках дочерней ostis-системы: в таком варианте отладка компонентов осуществляется непосредственно в той же системе, в которой предполагается их использование, и дополнительного переноса не требуется.

Независимо от выбранного способа использования системы разработанные компоненты впоследствии могут быть включены в состав библиотеки компонентов OSTIS.

Необходимо отметить, что выделяются два принципиально разных уровня отладки решателя задач:

– отладка на уровне sc-агентов;

– отладка на уровне scp-программ.

В случае отладки на уровне sc-агентов акт выполнения каждого агента считается неделимым и не может быть прерван. При этом может выполняться отладка как атомарных sc-агентов, так и неатомарных. Инициирование того или иного агента, в том числе входящего в состав неатомарного, осуществляется путем создания соответствующих конструкций в sc-памяти. Таким образом, отладка может осуществляться на разных уровнях детализации агентов, вплоть до атомарных.

С учетом того что предлагаемая в данной работе модель взаимодействия агентов использует универсальный вариант взаимодействия агентов через общую память, рассматриваемая система поддержки проектирования агентов может служить основой для моделирования систем агентов, применяющих другие принципы коммуникации, например, непосредственный обмен сообщениями между агентами.

Отладка на уровне scp-программ осуществляется аналогично существующим современным подходам к отладке процедурных программ и предполагает возможность установки точек останова, пошагового выполнения программы и т. д.

Система автоматизации процесса построения и модификации решателей задач и, соответственно, ее sc-модель, разделяется на две частные.

Система автоматизации процесса построения и модификации решателей задач по технологии OSTIS
*<= базовая декомпозиция**:
{
- *Система автоматизации процесса построения и модификации агентов обработки знаний*
- *Система автоматизации процесса построения и модификации scp-программ*

}

В свою очередь, указанные подсистемы декомпозируются в соответствии с общими принципами построения ostis-систем.

Система автоматизации процесса построения и модификации агентов обработки знаний
*<= базовая декомпозиция раздела**:
{
- *База знаний системы автоматизации процесса построения и модификации агентов обработки знаний*
- *Решатель задач системы автоматизации процесса построения и модификации агентов обработки знаний*
- *Пользовательский интерфейс системы автоматизации процесса построения и модификации агентов обработки знаний*

}

Система автоматизации процесса построения и модификации scp-программ
*<= базовая декомпозиция**:
{
- *База знаний системы автоматизации процесса построения и модификации scp-программ*
- *Решатель задач системы автоматизации процесса построения и модификации scp-программ*
- *Пользовательский интерфейс системы автоматизации процесса построения и модификации scp-программ*

}

Далее рассмотрим подробнее перечисленные компоненты.

2.3.1 Семантическая модель базы знаний системы автоматизации процесса построения и модификации решателей задач

База знаний системы автоматизации процесса построения и модификации решателей задач включает в себя кроме ядра и расширений ядра sc-моделей баз знаний, предоставляемых на уровне технологии OSTIS, и моделей предметных областей scp-программ и scp-интерпретатора также описание ключевых понятий, связанных с верификацией и отладкой scp-программ.

Рассмотрим основные понятия, специфичные для базы знаний системы автоматизации процесса построения и модификации scp-программ, специфицированные в SCn-коде.

точка останова*
∈ *квазибинарное отношение*

Связки отношения *точки останова** связывают *scp-программу* с некоторым множеством sc-переменных, соответствующих *scp-операторам* в рамках этой программы. При генерации каждого *scp-процесса*, соответствующего этой *scp-программе*, все *scp-операторы*, соответствующие таким переменным, будут добавлены в множество *точка останова*, т. е. выполнение данного scp-процесса будет прерываться при достижении каждого из этих *scp-операторов*. Использование данного отношения приводит к указанию точек останова для всех *scp-процессов*, формируемых на основе заданной *scp-программы*. Для указания точки останова в рамках отдельно взятого *scp-процесса* нужный scp-оператор явно включается во множество *точка останова*.

точка останова
<= *включение**:
scp-оператор

В множество *точка останова* входят все *scp-операторы*, являющиеся точками останова в рамках какого-либо *scp-процесса*. Это означает, что в момент, когда в соответствии с переходами между *scp-операторами* по связкам отношения *последовательность действий** указанный *scp-оператор* должен стать *активным действием*, он становится *отложенным действием*, и, соответственно, выполнение всего *scp-процесса* по данной ветке приостанавливается. Чтобы продолжить выполнение, необходимо удалить указанный *scp-оператор* из множества *отложенных действий* и добавить его в множество *активных действий*.

некорректность в scp-программе
<= *включение**:
некорректная структура
=> *включение**:
ошибка в scp-программе
=> *включение**:
- *недостижимый scp-оператор*
- *потенциально бесконечный цикл*

Под *некорректностью в scp-программе* понимается *некорректная структура*, описывающая некорректность (необязательно делающую невозможным выполнение соответствующих данной *scp-программе scp-процессов*), выявленную в рамках какой-либо конкретной *scp-программы*.

ошибка в scp-программе
*<= разбиение**:
{
- *синтаксическая ошибка в scp-программе*
- *семантическая ошибка в scp-программе*

}
*<= разбиение**:
{
- *ошибка в scp-программе на уровне программы*
- *ошибка в scp-программе на уровне множества параметров*
- *ошибка в scp-программе на уровне множества операторов*
- *ошибка в scp-программе на уровне оператора*
- *ошибка в scp-программе на уровне операнда*

}

Под *ошибкой в scp-программе* понимается такая *некорректность в scp-программе*, которая делает невозможным успешное выполнение любого *scp-процесса*, соответствующего данной *scp-программе*, или даже создание такого *scp-процесса*.

Под *синтаксической ошибкой в scp-программе* понимается *ошибка в scp-программе*, в состав которой входит некоторая конструкция, не соответствующая синтаксису *scp-программы* или какой-либо ее части, например, конкретного *scp-оператора*.

Под *семантической ошибкой в scp-программе* понимается *ошибка в scp-программе*, в состав которой входит некоторая конструкция, корректная с точки зрения синтаксиса, но некорректная с семантической точки зрения, например, нарушающая логическую целостность *scp-программы*.

Каждая *ошибка в scp-программе на уровне программы* описывает некорректный фрагмент, выявление которого требует анализа всей *scp-программы* как единого целого, и не может быть выполнено путем анализа ее отдельных частей, например, конкретных *scp-операторов*.

ошибка в scp-программе на уровне программы
*=> включение**:
- *отсутствует scp-процесс, соответствующий данной scp-программе*
  ∈ синтаксическая ошибка в scp-программе
- *не указана декомпозиция scp-процесса, соответствующего данной scp-программе*

∈ *синтаксическая ошибка в scp-программе*

Каждая *ошибка в scp-программе на уровне множества параметров* описывает некорректный фрагмент, для выявления которого достаточно анализа параметров некоторой *scp-программы*, т. е. явным образом выделенных аргументов *действия (scp-процесса)*, соответствующего данной scp-программе. К такого рода ошибкам относятся, например, неверное указание ролей этих аргументов в рамках данного действия.

ошибка в scp-программе на уровне множества параметров
*=> включение**:

- *не указан тип параметра scp-программы*
  ∈ *синтаксическая ошибка в scp-программе*
- *не указан порядковый номер параметра scp-программы*
  ∈ *синтаксическая ошибка в scp-программе*

Каждая *ошибка в scp-программе на уровне множества операторов* описывает некорректный фрагмент, для выявления которого достаточно анализа множества операторов некоторой *scp-программы*, т. е. элементов декомпозиции *действия (scp-процесса)*, соответствующего данной *scp-программе*. К таким ошибкам относится, например, факт отсутствия *начального оператора' scp-программы* или факт отсутствия в программе *scp-оператора завершения выполнения программы*.

ошибка в scp-программе на уровне множества операторов
*=> включение**:

- *декомпозиция scp-процесса не содержит ни одного элемента*
  ∈ *синтаксическая ошибка в scp-программе*
- *отсутствует scp-оператор завершения выполнения программы*
  ∈ *синтаксическая ошибка в scp-программе*
- *scp-оператор, к которому осуществляется переход, не является частью текущего scp-процесса*
  ∈ *синтаксическая ошибка в scp-программе*
- *не указана последовательность действий после выполнения текущего scp-оператора*
  ∈ *синтаксическая ошибка в scp-программе*
- *отсутствует начальный оператор scp-программы*
  ∈ *синтаксическая ошибка в scp-программе*

Каждая *ошибка в scp-программе на уровне оператора* описывает некорректный фрагмент, для выявления которого достаточно анализа одного конкретного *scp-оператора*, при этом неважно, в состав какой *scp-программы* он входит. К такого рода ошибкам относится, например, факт указания

количества операндов *scp-оператора*, не соответствующего спецификации соответствующего класса *scp-операторов*.

ошибка в scp-программе на уровне оператора
*=> включение**:

- *scp-оператор не принадлежит ни одному из атомарных классов scp-операторов*
  ∈ *синтаксическая ошибка в scp-программе*
- *ни один операнд scp-оператора удаления не помечен как удаляемый sc-элемент*
  ∈ *синтаксическая ошибка в scp-программе*
- *в scp-операторе поиска пятиэлементной конструкции совпадает второй и четвертый операнд*
  ∈ *синтаксическая ошибка в scp-программе*
- *scp-оператор поиска не содержит ни одного операнда с заданным значением*
  ∈ *синтаксическая ошибка в scp-программе*
- *scp-оператор поиска с формированием множеств не содержит ни одного операнда с атрибутом формируемое множество*
  ∈ *синтаксическая ошибка в scp-программе*
- *атрибутом формируемое множество отмечен операнд, которому соответствует операнд с заданным значением*
  ∈ *синтаксическая ошибка в scp-программе*
- *количество операндов scp-оператора не совпадает со спецификацией*
  ∈ *синтаксическая ошибка в scp-программе*

Каждая *ошибка в scp-программе на уровне оператора* описывает некорректный фрагмент, для выявления которого достаточно анализа одного конкретного операнда в рамках scp-программы, точнее sc-дуги принадлежности, связывающей указанный операнд и соответствующий *scp-оператор*, при этом неважно, какой именно *scp-оператор*. К такого рода ошибкам относится, например, факт отсутствия ролевого отношения, указывающего на номер операнда в рамках *scp-оператора*.

ошибка в scp-программе на уровне операнда
*=> включение**:

- *не указан номер операнда в рамках scp-оператора*
  ∈ *синтаксическая ошибка в scp-программе*

некорректность в scp-программе*
∈ *бинарное отношение*
*=> первый домен**:
некорректность в scp-программе
*=> второй домен**:
scp-программа

Отношение *scp-программа поиска некорректности в scp-программе** связывает *класс некорректностей в scp-программе* и *scp-программу*, которая может использоваться для выявления соответствующей некорректности в какой-либо другой *scp-программе*.

Указанная *scp-программа* должна иметь единственный параметр, который является *in-параметром'* и в зависимости от соответствующего класса некорректностей в *scp-программе* обозначает:

– саму *scp-программу* в случае выявления *некорректности в scp-программе* вообще или *ошибки в scp-программе на уровне программы*;

– *scp-процесс*, являющийся *ключевым sc-элементом* данной *scp-программы* в случае выявления ошибки в *scp-программе на уровне множества параметров*;

– *множество операторов* данной *scp-программы* в случае выявления *ошибки в scp-программе на уровне множества операторов*;

– *знак конкретного scp-оператора* в случае выявления ошибки в *scp-программе на уровне оператора*;

– *sc-дугу принадлежности* в случае выявления *ошибки в scp-программе на уровне операнда*.

Если в результате верификации *scp-программы* выявлена некорректность, то формируется соответствующая *структура* и генерируется связка отношения *некорректность в scp-программе**.

2.3.2 Семантическая модель решателя задач системы автоматизации процесса построения и модификации решателей задач

Рассмотрим подробнее состав решателя задач системы автоматизации процесса построения и модификации агентов обработки знаний в SCn-коде.

Решатель задач системы автоматизации процесса построения и модификации агентов обработки знаний
*<= декомпозиция sc-агента**:
{
- *Абстрактный sc-агент верификации sc-агентов*
- *Абстрактный sc-агент отладки коллективов sc-агентов*

}

Абстрактный sc-агент верификации sc-агентов
*<= декомпозиция sc-агента**:
{
- *Абстрактный sc-агент верификации спецификации sc-агента*

- *Абстрактный sc-агент проверки неатомарного sc-агента на непротиворечивость его спецификации спецификациям частных sc-агентов в его составе*

}

Абстрактный sc-агент верификации sc-агентов
*<= декомпозиция sc-агента**:

{

- *Абстрактный sc-агент поиска всех выполняющихся процессов, соответствующих заданному sc-агенту*
- *Абстрактный sc-агент инициирования заданного sc-агента на заданных аргументах*
- *Абстрактный sc-агент активации заданного sc-агента*
- *Абстрактный sc-агент деактивации заданного sc-агента*
- *Абстрактный sc-агент установки блокировки заданного типа для заданного процесса на заданный sc-элемент*
- *Абстрактный sc-агент снятия всех блокировок заданного процесса*
- *Абстрактный sc-агент снятия всех блокировок с заданного sc-элемента*

}

Единственным аргументом класса действий, соответствующего *Абстрактному sc-агенту поиска всех выполняющихся процессов, соответствующих заданному sc-агенту, Абстрактному sc-агенту активации заданного sc-агента, Абстрактному sc-агенту деактивации заданного sc-агента*, является знак этого *sc-агента*.

Класс действий, соответствующий *Абстрактному sc-агенту инициирования заданного sc-агента на заданных аргументах*, имеет два аргумента. Первый аргумент является знаком инициируемого sc-агента, второй – знаком связки, в которую под соответствующими атрибутами входят sc-элементы, которые станут аргументами соответствующего *действия в sc-памяти*.

Класс действий, соответствующий *Абстрактному sc-агенту установки блокировки заданного типа на заданный sc-элемент*, имеет три аргумента. Первый аргумент является знаком класса блокировок, второй – знаком процесса в sc-памяти, третий – sc-элементом, на который должна быть установлена блокировка.

Единственным аргументом класса действий, соответствующего *Абстрактному sc-агенту снятия всех блокировок заданного процесса*, является знак этого *процесса в sc-памяти*.

Единственным аргументом класса действий, соответствующего *Абстрактному sc-агенту снятия всех блокировок с заданного sc-элемента*, является знак этого *sc-элемента*.

Рассмотрим подробнее состав решателя задач системы автоматизации процесса построения и модификации scp-программ в SCn-коде:

Решатель задач системы автоматизации процесса построения и модификации scp-программ
*<= декомпозиция sc-агента**:
{
- *Абстрактный sc-агент верификации scp-программ*
- *Абстрактный sc-агент отладки scp-программ*

}

Алгоритм работы *Абстрактного sc-агента верификации scp-программ* сводится к поиску некорректностей в рамках *scp-программы* на основе определений, соответствующих различным классам таких некорректностей, а также посредством запуска соответствующих данным классам некорректностей *scp-программ поиска некорректности в scp-программе**.

Результатом работы *Абстрактного sc-агента верификации scp-программ* является формирование в *sc-памяти структур*, описывающих некорректности в исследуемой *scp-программе*, если таковые имеются.

Единственным аргументом класса действий, соответствующего *Абстрактному sc-агенту верификации scp-программ*, является знак верифицируемой *scp-программы*.

Абстрактный sc-агент отладки scp-программ
*<= декомпозиция sc-агента**:
{
- *Абстрактный sc-агент запуска заданной scp-программы для заданного множества входных данных*
- *Абстрактный sc-агент запуска заданной scp-программы для заданного множества входных данных в режиме пошагового выполнения*
- *Абстрактный sc-агент поиска всех scp-операторов в рамках scp-программы*
- *Абстрактный sc-агент поиска всех точек останова в рамках scp-процесса*
- *Абстрактный sc-агент добавления точки останова в scp-программу*
- *Абстрактный sc-агент удаления точки останова из scp-программы*
- *Абстрактный sc-агент добавления точки останова в scp-процесс*
- *Абстрактный sc-агент удаления точки останова из scp-процесса*
- *Абстрактный sc-агент продолжения выполнения scp-процесса на один шаг*
- *Абстрактный sc-агент продолжения выполнения scp-процесса до точки останова или завершения*
- *Абстрактный sc-агент просмотра информации об scp-процессе*
- *Абстрактный sc-агент просмотра информации об scp-операторе*

}

Классы действий, соответствующие *Абстрактному sc-агенту запуска заданной scp-программы для заданного множества входных данных* и *Абстрактному sc-агенту запуска заданной scp-программы для заданного множества входных данных в режиме пошагового выполнения*, имеют два аргумента. Первый аргумент является знаком запускаемой scp-программы, второй – знаком связки, в которую под соответствующими атрибутами входят sc-элементы, которые станут аргументами соответствующего scp-процесса.

В режиме пошагового выполнения предполагается, что на каждом шаге инициируется выполнение всех scp-операторов в рамках заданного scp-процесса, для которых предыдущий scp-оператор стал прошлой сущностью (выполнился). В свою очередь, шаг заканчивается, когда все инициированные таким образом операторы закончат выполнение. Таким образом, в случае если в рамках scp-программы есть параллельные ветви, то на одном шаге могут одновременно инициироваться два и более scp-оператора.

Классы действий, соответствующие *Абстрактному sc-агенту добавления точки останова в scp-программу, Абстрактному sc-агенту удаления точки останова из scp-программы, Абстрактному sc-агенту добавления точки останова в scp-процесс* и *Абстрактному sc-агенту удаления точки останова из scp-процесса*, имеют два аргумента. Первый аргумент является знаком scp-программы или scp-процесса соответственно, второй – знаком scp-оператора, входящего в состав этой scp-программы или scp-процесса.

Единственным аргументом классов действий, соответствующих *Абстрактному sc-агенту поиска всех точек останова в рамках scp-процесса, Абстрактному sc-агенту продолжения выполнения scp-процесса на один шаг, Абстрактному sc-агенту продолжения выполнения scp-процесса до точки останова или завершения* и *Абстрактному sc-агенту просмотра информации об scp-процессе*, является знак scp-процесса, с которым будет выполнено соответствующее действие.

Единственным аргументом класса действий, соответствующего *Абстрактному sc-агенту поиска всех scp-операторов в рамках scp-программы*, является знак этой scp-программы.

Единственным аргументом класса действий, соответствующего *Абстрактному sc-агенту просмотра информации об scp-операторе*, является знак scp-оператора, входящего в состав некоторого scp-процесса. Результатом работы данного агента становится структура, описывающая значения операндов данного scp-оператора, его атомарный тип и другую служебную информацию, полезную для разработчика.

2.3.3 Семантическая модель пользовательского интерфейса системы автоматизации процесса построения и модификации решателей задач

Поскольку объектами проектирования описываемой системы автоматизации являются компоненты решателей задач, в частности, агенты и программы обработки знаний, представленные в SC-коде, то в такой системе могут использоваться базовые средства внешнего представления текстов SC-кода, например, на языках SCn или SCg.

Для того чтобы визуально упростить процесс верификации и отладки компонентов решателя, используется подход, предполагающий, что пользователю системы в каждый момент времени отображается только минимально необходимый набор sc-элементов. Например, при отладке scp-процесса достаточно отображать scp-операторы и переходы между ними. При необходимости пользователь может вручную запросить и просмотреть спецификацию нужного scp-оператора в момент останова. Указанный подход заложен в алгоритмы работы всех агентов описываемой системы.

Таким образом, в настоящее время пользовательский интерфейс системы автоматизации процесса построения и модификации решателей задач представлен набором интерфейсных команд, позволяющих пользователю инициировать деятельность нужного агента, входящего в состав этой системы.

2.3.4 Примеры использования системы автоматизации процесса построения и модификации решателей задач

Агенты системы реализованы с использованием языка SCP. В настоящий момент *Решатель задач системы автоматизации процесса построения и модификации решателей задач* имеет в своем составе 22 агента.

Пользовательский интерфейс системы автоматизации процесса построения и модификации решателей задач построен на основе существующей реализации платформы интерпретации sc-моделей интерфейса, доступной в репозитории [96]. Указанная реализация выполнена в web-ориентированном варианте, таким образом, доступ к системе осуществляется посредством web-браузера. Такой вариант реализации позволяет работать с системой удаленно, в том числе осуществлять коллективную работу, при этом не требуется никаких дополнительных средств, кроме обычного браузера.

Указанная реализация ориентирована на работу с программной реализацией семантической памяти [97] и в комплексе с ней в настоящее время используется в качестве основного варианта реализации *платформы интерпретации sc-моделей*, используемого в рамках технологии OSTIS на текущем этапе развития.

Рассмотрим примеры работы системы автоматизации процесса построения и модификации scp-программ.

Добавление точки останова в тестовую программу (рисунок 2.12).

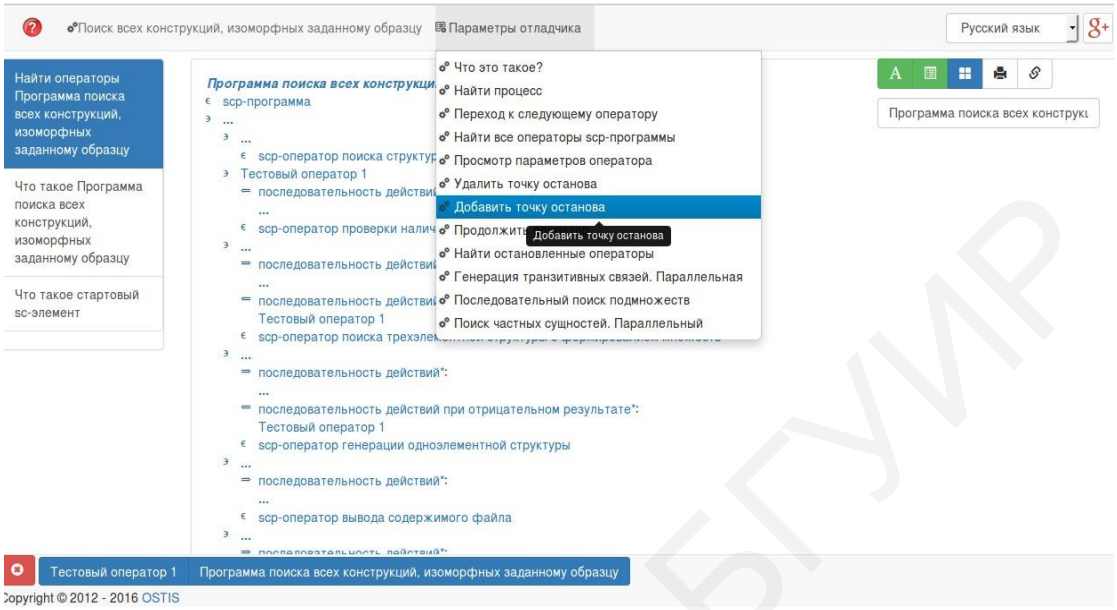

Рисунок 2.12 – Добавление точки останова

После запуска программы на тестовых данных выполнение соответствующего процесса остановлено при достижении указанной ранее точки останова. Пользователь выполняет поиск оператора, на котором остановлено выполнение (рисунок 2.13).

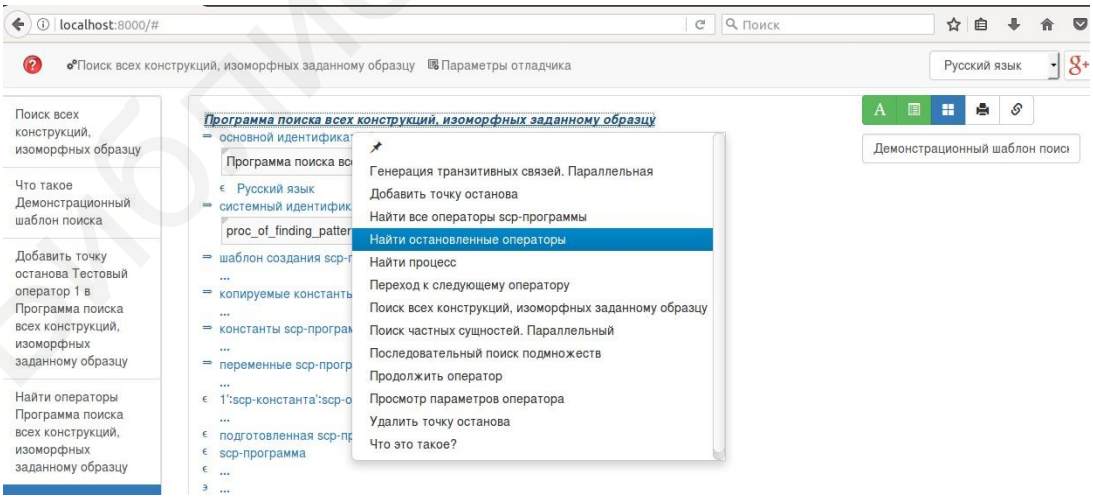

Рисунок 2.13 – Поиск остановленных операторов

Оператор найден (рисунок 2.14).

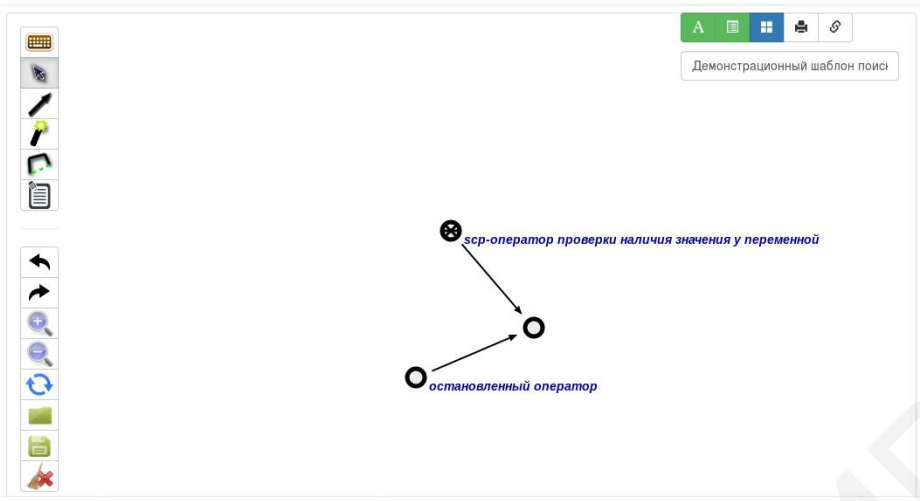

Рисунок 2.14 – Оператор, на котором остановилось выполнение отлаживаемого процесса

Далее, используя стандартные средства навигации, пользователь может просмотреть состояние памяти в окрестности остановленного оператора, например, просмотреть значения его операндов (рисунок 2.15).

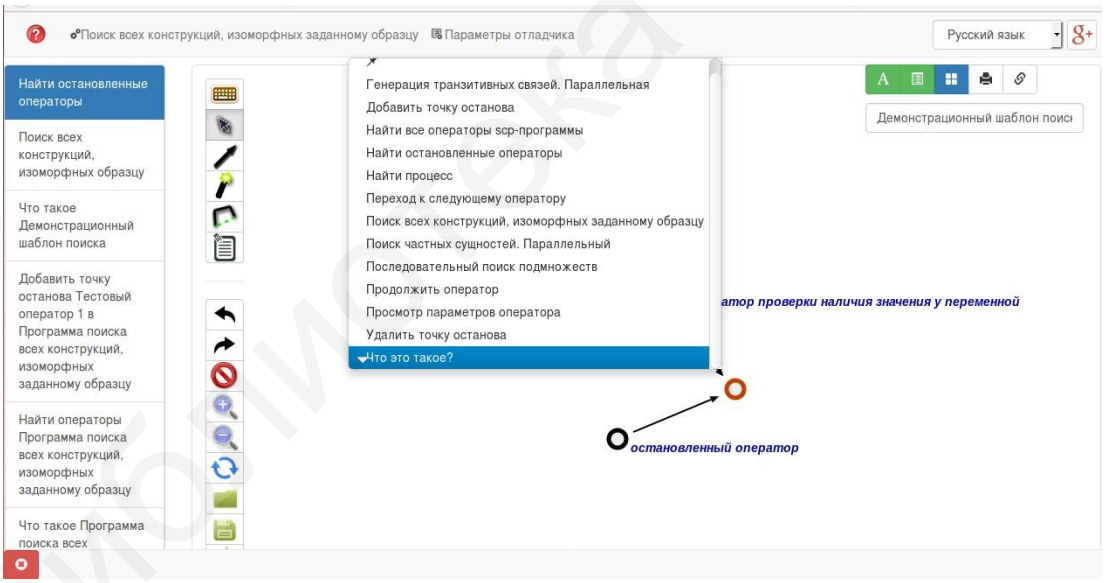

Рисунок 2.15 – Запрос семантической окрестности оператора

Семантическая окрестность остановленного оператора (рисунок 2.16).

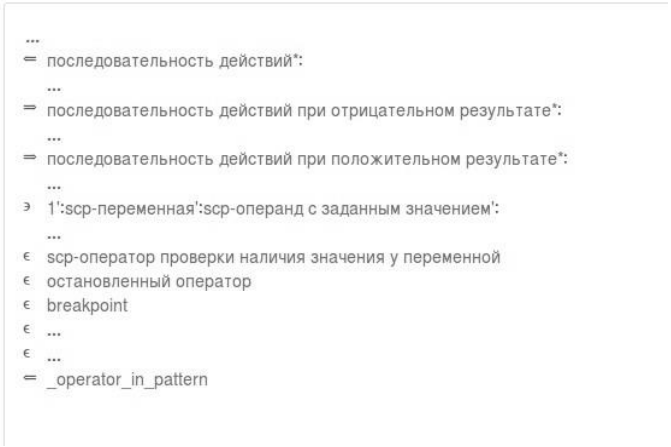

Рисунок 2.16 – Семантическая окрестность оператора

Далее выполнение процесса может быть продолжено в режиме пошагового выполнения или до конца процесса.

Как видно из приведенных примеров, спроектированная и реализованная система автоматизации процесса построения и модификации scp-программ обладает всей базовой функциональностью современных сред проектирования программ, таким как возможность использования точек останова и пошагового выполнения. При этом средства навигации в составе описанной системы позволяют просматривать любую информацию в окрестности остановленного процесса, в том числе – значения переменных, а также позволяют отслеживать изменения, происходящие в памяти. Кроме того, по аналогии с современными средами рассматриваемая система содержит средства автоматической верификации проектируемых программ.

2.4 ПРИМЕРЫ ПОСТРОЕНИЯ ГИБРИДНЫХ РЕШАТЕЛЕЙ ЗАДАЧ

С использованием описанных в данном учебном пособии моделей и средств осуществлена разработка решателя задач самой метасистемы поддержки проектирования интеллектуальных систем IMS [4, 56].

Кроме того, указанные модели и средства использованы для построения объединенных решателей задач интеллектуальных справочных систем (ИСС) по различным предметным областям, исходные тексты которых доступны по [98].

Наибольший интерес представляют прототипы решателей задач ИСС по геометрии Евклида и теории графов. Это обусловлено, во-первых, относительной развитостью и сложностью указанных прототипов, а во-вторых, тем, что указанные системы используют принципиально разные подходы к решению задач.

В рамках ИСС по геометрии Евклида реализованы стратегия поиска решения задачи в глубину и механизм прямого логического вывода на основе продукций, позволяющие решать задачи в несколько действий, используя логические утверждения (теоремы, аксиомы) вида «если…, то…», хранящиеся в базе знаний, путем применения правила modus ponens.

В свою очередь, в рамках ИСС по теории графов реализуется концепция пакета программ, в основе которой лежит механизм сведения задачи к подзадачам, каждая из которых в конечном итоге решается путем выполнения хранимой в памяти системы программой на некоторых входных данных. Указанный механизм также позволяет решать задачи в несколько действий, т. е. такие задачи, для которых нет заранее заготовленной программы, с помощью которой можно было бы решить данную задачу.

Кроме того, объединенный решатель задач каждого из рассматриваемых прототипов имеет в своем составе набор поисковых агентов, многие из которых были заимствованы из библиотеки многократно используемых sc-агентов, а некоторые реализованы специально для соответствующей системы, поскольку являются предметно-зависимыми.

Кроме того, описанные модели и средства использованы при разработке различных подсистем комплексной системы автоматизации предприятия рецептурного производства (в сотрудничестве с ОАО «Савушкин продукт»).

2.4.1 Решатель задач метасистемы IMS

В настоящий момент объединенный решатель задач IMS, без учета подсистем, включает в себя набор агентов информационного поиска, реализующих базовые механизмы навигации по базе знаний.

Рассмотрим структуру объединенного решателя задач в SCn-коде.

Решатель задач IMS
*<= декомпозиция абстрактного sc-агента**:
{
- *Абстрактный sc-агент поиска всех входящих константных позитивных стационарных sc-дуг принадлежности*
- *Абстрактный sc-агент поиска всех идентификаторов заданного sc-элемента*
- *Абстрактный sc-агент поиска полной семантической окрестности заданного элемента*
- *Абстрактный sc-агент поиска связок декомпозиции для заданного sc-элемента*
- *Абстрактный sc-агент поиска всех известных сущностей, являющихся общими по отношению к заданной*
- *Абстрактный sc-агент поиска определения или пояснения для заданного объекта*
- *Абстрактный sc-агент поиска всех известных сущностей, являющихся частными по отношению к заданной*

- *Абстрактный sc-агент поиска всех выходящих константных позитивных стационарных sc-дуг принадлежности с их ролевыми отношениями*
- *Абстрактный sc-агент поиска всех выходящих константных позитивных стационарных sc-дуг принадлежности*
- *Абстрактный sc-агент поиска всех входящих константных позитивных стационарных sc-дуг принадлежности с их ролевыми отношениями*

}

Подробная спецификация разработанного решателя приведена в соответствующем разделе базы знаний метасистемы [56].

Рассмотрим некоторые примеры работы решателя задач метасистемы IMS.

Пример работы *Абстрактного sc-агента поиска всех выходящих константных позитивных стационарных sc-дуг принадлежности* представлен на рисунке 2.17.

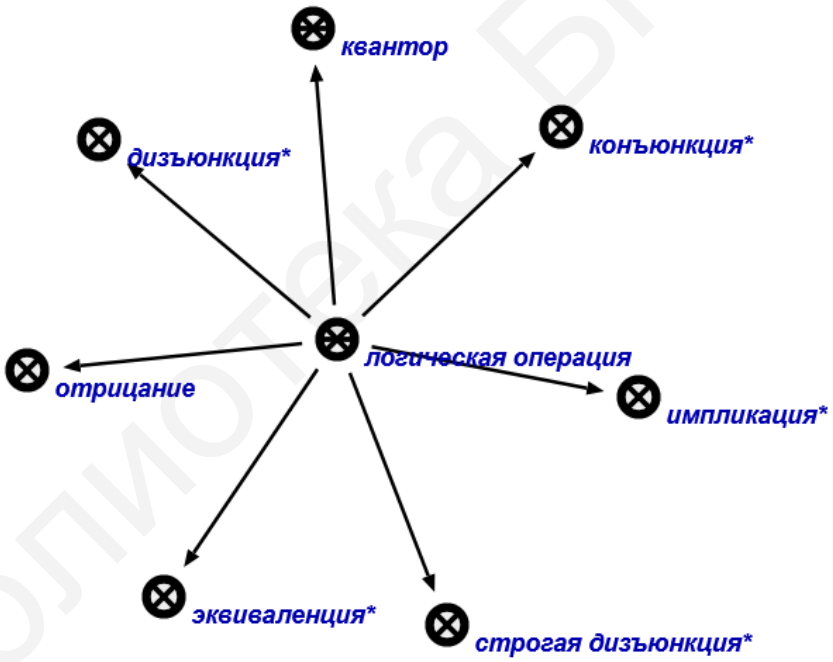

Рисунок 2.17 – Результат поиска выходящих sc-дуг принадлежности

Аналогичным образом выглядит результат работы *Абстрактного sc-агента поиска всех входящих константных позитивных стационарных sc-дуг принадлежности, Абстрактного sc-агента поиска всех выходящих константных позитивных стационарных sc-дуг принадлежности с их ролевыми отношениями, Абстрактного sc-агента поиска всех входящих константных позитивных стационарных sc-дуг принадлежности с их ролевыми отношениями*.

Пример работы *Абстрактного sc-агента поиска всех известных сущностей, являющихся частными по отношению к заданной*, для понятия «комплексное число» представлен на рисунке 2.18.

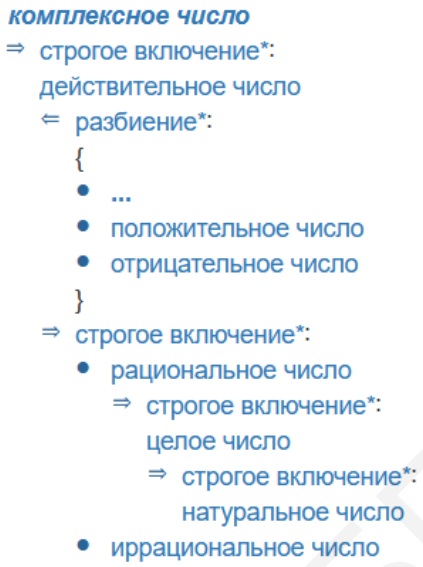

Рисунок 2.18 – Результат поиска сущностей, частных по отношению к заданной

В этом примере предполагается, что натуральное число и другие классы чисел являются подмножествами множества комплексных чисел. Аналогичным образом выглядит результат работы *Абстрактного sc-агента поиска всех известных сущностей, являющихся общими по отношению к заданной*.

Пример работы *Абстрактного sc-агента поиска всех идентификаторов заданного sc-элемента* для понятия «множество» представлен на рисунке 2.19.

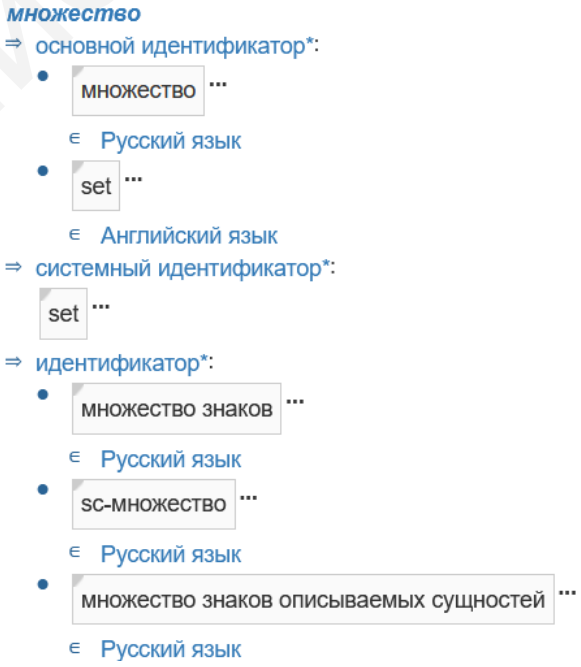

Рисунок 2.19 – Результат поиска идентификаторов заданной сущности

Пример работы *Абстрактного sc-агента поиска связок декомпозиции для заданного sc-элемента* для сущности «База знаний IMS» представлен на рисунке 2.20.

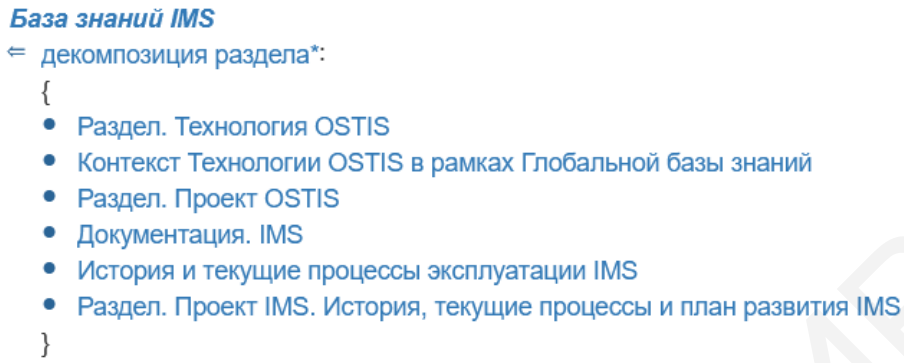

Рисунок 2.20 – Результат поиска идентификаторов заданной сущности

2.4.2 Решатель задач ИСС по геометрии Евклида

Кроме стандартного набора базовых агентов информационного поиска, входящих в состав решателя задач метасистемы IMS, для ИСС по геометрии Евклида были реализованы дополнительные поисковые агенты, впоследствии включенные в состав *Библиотеки sc-агентов информационного поиска* IMS, а также реализована одна из моделей решения задач.

Перечень агентов информационного поиска, реализованных в рамках ИСС по геометрии Евклида:

– *Абстрактный sc-агент поиска аннотации для заданного раздела;*

– *Абстрактный sc-агент поиска аксиом заданной онтологии;*

– *Абстрактный sc-агент поиска теорем заданной онтологии;*

– *Абстрактный sc-агент поиска непосредственных связей между двумя объектами;*

– *Абстрактный sc-агент поиска понятий, через которые определяется заданное понятие;*

– *Абстрактный sc-агент поиска области определения отношения;*

– *Абстрактный sc-агент поиска определения или пояснения для заданного объекта;*

– *Абстрактный sc-агент поиска примеров для заданного понятия;*

– *Абстрактный sc-агент поиска формальной записи утверждения для заданного знака утверждения;*

– *Абстрактный sc-агент поиска иллюстраций для заданного объекта;*

– *Абстрактный sc-агент нахождения ключевых sc-элементов для заданной предметной области;*

– Абстрактный sc-агент поиска понятий, которые определяются на основе заданного;

– Абстрактный sc-агент поиска всех конструкций, изоморфных заданному образцу;

– Абстрактный sc-агент поиска sc-текста доказательства для заданного утверждения;

– Абстрактный sc-агент поиска отношений, заданных на понятии;

– Абстрактный sc-агент поиска sc-текста условия и решения задачи;

– Абстрактный sc-агент поиска утверждений об объекте.

В рамках ИСС по геометрии Евклида также реализован прототип интеллектуального решателя задач, который, как и некоторые его компоненты, может быть использован и в других системах. Рассмотрим структуру решателя.

Неатомарный абстрактный sc-агент решения задач
<= декомпозиция абстрактного sc-агента:*
{
- *Абстрактный sc-агент поиска значения неизвестной величины*
- *Абстрактный sc-агент проверки истинности утверждения*
- *Абстрактный sc-агент применения стратегий решения задач*
- *Абстрактный sc-агент выполнения логического вывода*
- *Неатомарный абстрактный sc-агент расчета математических выражений*

<= декомпозиция абстрактного sc-агента:*
{
- *Абстрактный sc-агент координации вычисления математических выражений*
- *Абстрактный sc-агент возведения в степень, извлечения корня и нахождения натурального логарифма*
- *Абстрактный sc-агент сложения и вычитания величин и чисел*
- *Абстрактный sc-агент произведения и деления величин и чисел*
- *Абстрактный sc-агент сравнения величин и чисел*
- *Абстрактный sc-агент вычисления тригонометрических выражений*

}
}

В сокращенной форме проиллюстрировать структуру разработанного с использованием предложенной модели решателя задач ИСС по геометрии Евклида можно так, как показано на рисунке 2.21, из которого видно, что такая структура решателя позволяет при необходимости легко расширять функциональные возможности решателя на разных уровнях, не затрагивая при этом другие компоненты решателя. Так, например, можно реализовать в рамках того же решателя другие стратегии решения задач; расширить число типов логических правил, которые могут интерпретироваться соответствующим

неатомарным агентом; расширить число интерпретируемых решателем арифметических операций.

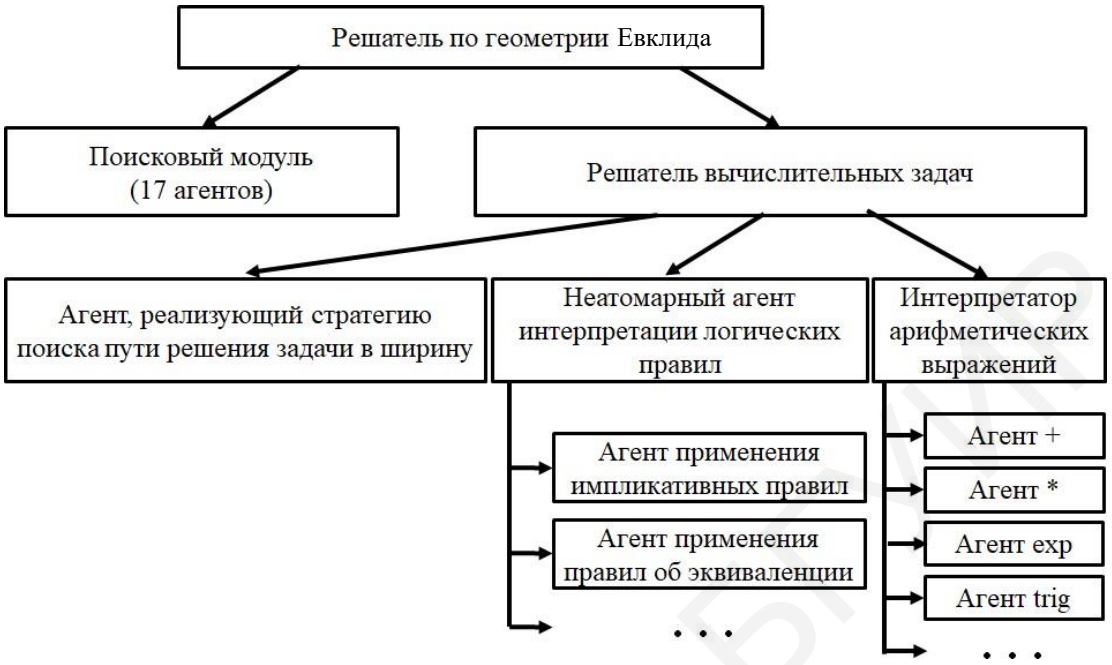

Рисунок 2.21 – Структура решателя задач по геометрии Евклида

Рассмотрим примеры работы некоторых из перечисленных агентов, которые на данный момент реализованы в рамках ИСС по геометрии Евклида (без учета агентов, заимствованных из библиотеки):

1. Агент поиска непосредственных связей между двумя объектами.

Результат работы агента для поиска непосредственных связей между объектами *Тупоугольный треугольник LDK* и *Равносторонний треугольник XYZ* представлен на рисунке 2.22.

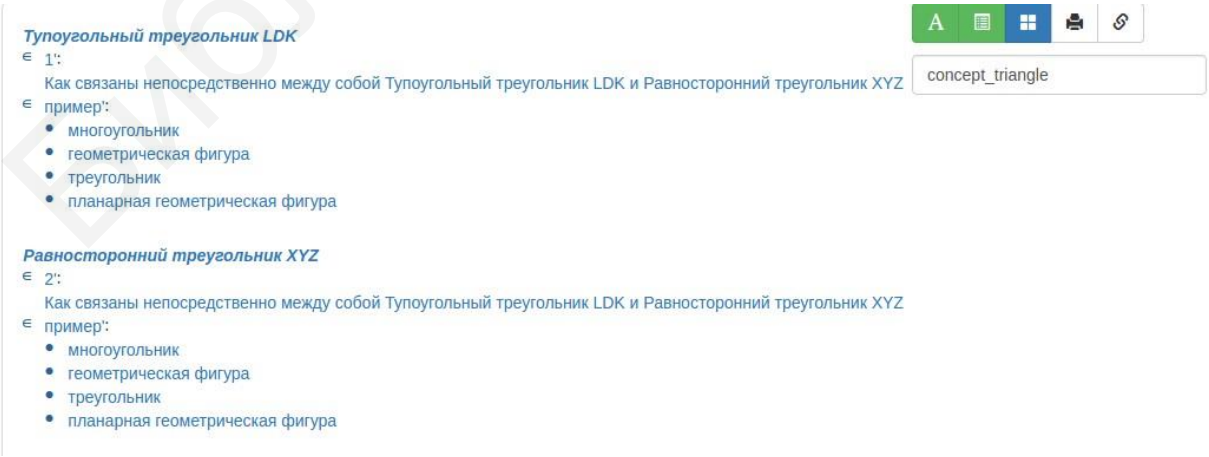

Рисунок 2.22 – Результат работы агента поиска непосредственных связей между объектами

2. Агент поиска понятий, через которые определяется заданное понятие.

Результат работы агента для поиска понятий, через которые определяется понятие *квадрат*, приведен на рисунке 2.23.

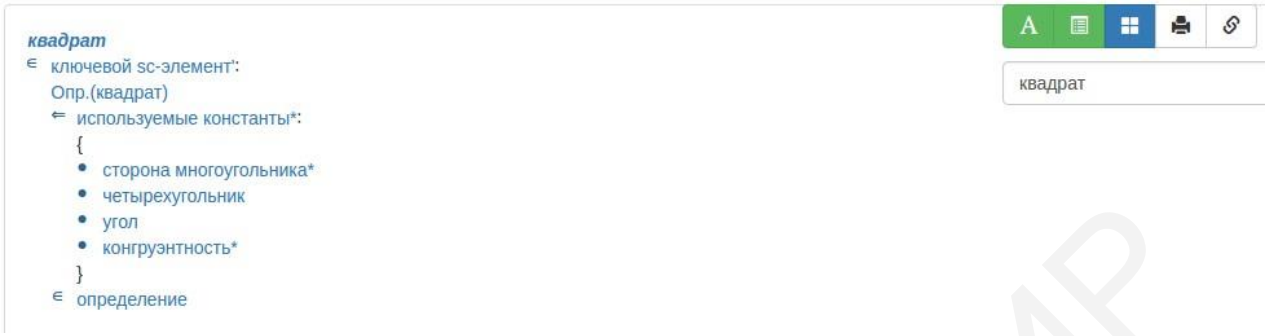

Рисунок 2.23 – Результат работы агента поиска понятий, через которые определяется заданное понятие

3. Агент поиска понятий, которые определяются на основе заданного.

Результат работы агента поиска понятий, которые определяются на основе понятия *отрезок*, приведен на рисунке 2.24.

Рисунок 2.24 – Результат работы агента поиска понятий, которые определяются на основе заданного

4. Агент поиска всех конструкций, изоморфных заданному образцу.

Задание образца поиска представлено на рисунке 2.25.

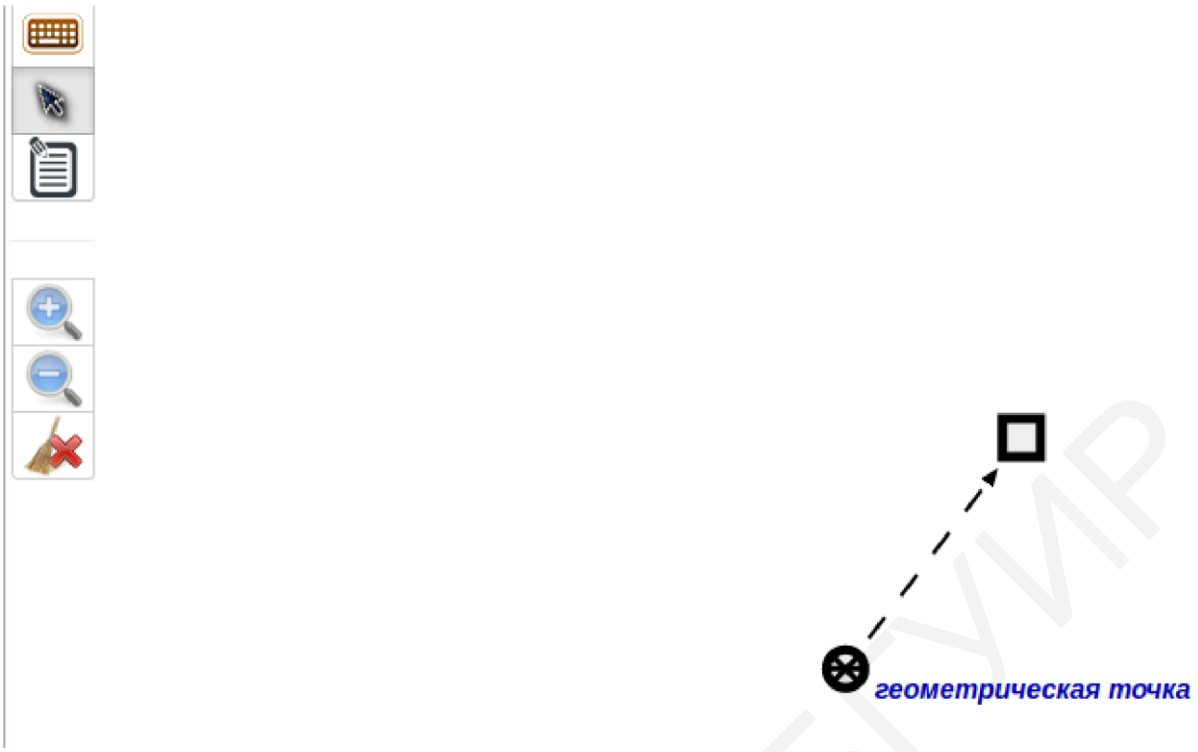

Рисунок 2.25 – Образец поиска

Результат работы агента для заданного образца приведен на рисунке 2.26.

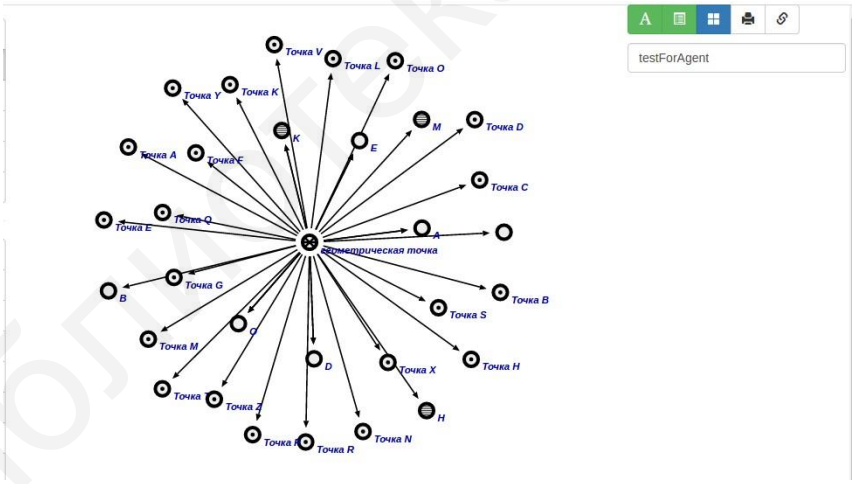

Рисунок 2.26 – Результат работы агента поиска всех конструкций, изоморфных заданному образцу

5. Агент поиска sc-текста доказательства для заданного утверждения.

Указанный агент принимает в качестве аргумента соответствующего действия sc-элемент, обозначающий логическое утверждение, полный текст доказательства которого требуется найти. Пример результата работы агента для поиска sc-текста доказательства признака подобия треугольников по двум

углам (если два угла одного треугольника равны двум углам другого треугольника, то такие треугольники подобны) приведен на рисунке 2.27.

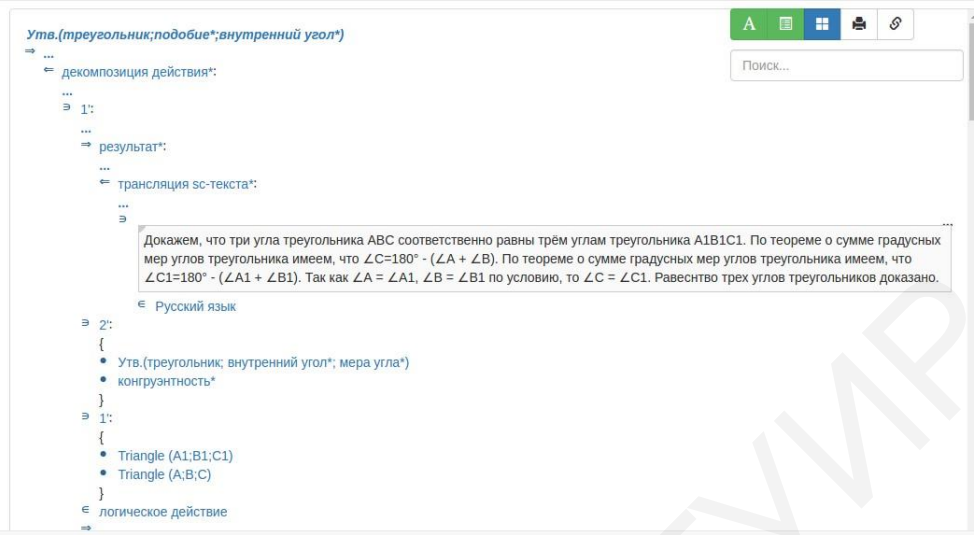

Рисунок 2.27 – Результат работы агента поиска sc-текста доказательства для заданного утверждения

6. Агент поиска отношений, заданных на понятии.

Результат работы команды пользовательского интерфейса поиска отношений, заданных на понятии *геометрическая точка*, представлен на рисунке 2.28.

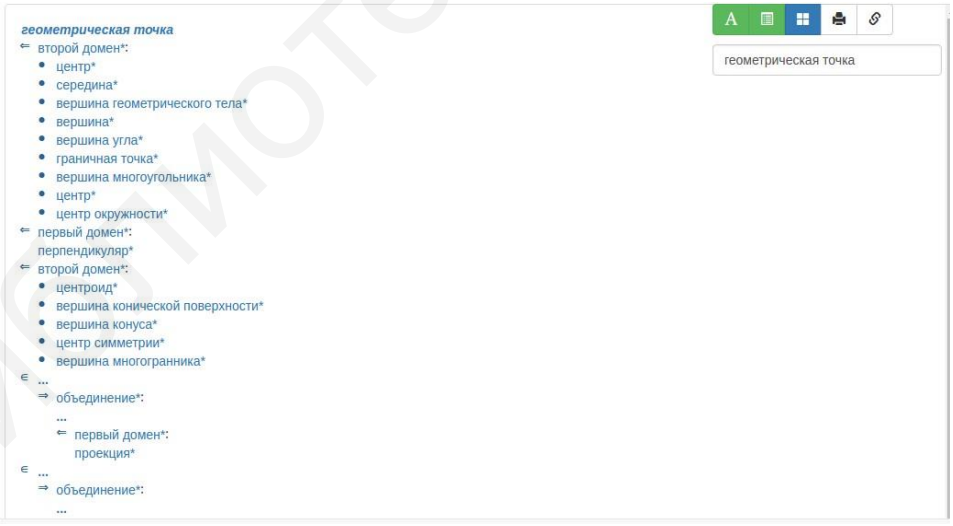

Рисунок 2.28 – Результат работы агента поиска отношений, заданных на понятии

7. Агент поиска утверждений об объекте.

Результат работы агента для поиска утверждений о понятии *прямоугольный треугольник* представлен на рисунке 2.29.

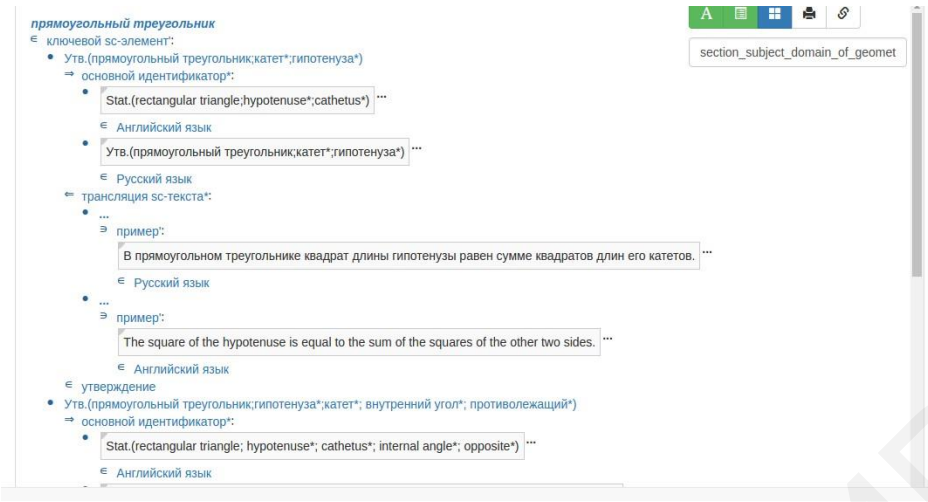

Рисунок 2.29 – Результат работы агента поиска утверждений об объекте

Далее рассмотрим пример работы гибридного решателя задач, реализованного для интеллектуальной справочной системы по геометрии Евклида.

На примере данного решателя будут продемонстрированы следующие принципы:

– информационного поиска в базе знаний как первого этапа решения задачи;

– реализации некоторой стратегии решения задачи;

– реализации некоторого способа логического вывода;

– реализации интерпретации (вычисления) математических выражений, полученных в результате логического вывода.

В настоящий момент интеллектуальный решатель реализует смешанную стратегию решения, представляющую собой комбинацию традиционной продукционной модели и метода поиска в глубину.

Уровень логического вывода представлен возможностью вывода на основе продукций вида «если…, то…» по правилу modus ponens. При этом возможен вариант преобразования найденной продукции в более частую форму, если это возможно с точки зрения исходных данных и не противоречит логической структуре утверждения. Также интеллектуальный решатель способен оперировать утверждениями, представленными в виде эквиваленции, такими как определения и утверждения о необходимости и достаточности. В указанном случае утверждение может быть применено как продукция, где в качестве посылки и заключения используются атомарные высказывания, входящие в указанное утверждение об эквиваленции. Роли данных

высказываний определяются в зависимости от контекста в каждом конкретном случае.

Процесс решения задачи в текущей реализации можно разделить на следующие этапы:

1. Этап работы поисковых sc-агентов. Вне зависимости от типа задачи всегда имеется вероятность того, что данная задача уже была решена системой ранее или системе уже откуда-либо известен ответ на поставленный вопрос. На данном этапе работу осуществляет коллектив поисковых sc-агентов, каждый из которых, как правило, соответствует некоторому классу решаемых задач. Если ответ найден, решатель прекращает свою работу. В противном случае происходит переход на следующий этап решения.

2. Этап применения стратегий решения задач. На данном этапе осуществляется выбор между различными стратегиями решения задач и при необходимости параллельный запуск различных стратегий. На данный момент, как уже было сказано, интеллектуальный решатель реализует комбинированную стратегию. Вначале рассматривается объект, для которого осуществляется поиск всех классов объектов, которым он принадлежит. Далее для каждого класса осуществляется поиск утверждений, справедливых для данного класса объектов (с целью оптимизации данный факт должен быть явно указан проектировщиком базы знаний). При рассмотрении каждого утверждения осуществляется попытка применить его в рамках некоторой семантической окрестности рассматриваемого объекта, для чего осуществляется переход на следующий этап решения.

3. Этап применения правил логического вывода. Здесь происходит попытка применения утверждения, полученного на предыдущем этапе, с целью генерации в системе новых знаний. Если такое применение справедливо (например, посылка истинна) и имеет смысл (в результате применения будут сгенерированы новые знания), то осуществляется генерация новых знаний на основе одного из правил логического вывода. При этом применение происходит в контексте объекта, рассматриваемого на предыдущем этапе (в общем случае – ряда объектов). На данный момент реализован логический вывод на основе правила modus ponens. В будущем предполагается расширить набор подобных правил с целью увеличения количества различных типов утверждений, интерпретируемых решателем. Если в данном контексте вывод на основе данного утверждения невозможен или нецелесообразен, решение возвращается на предыдущий этап. В случае успешного применения утверждения происходит переход к следующему этапу решения.

4. Этап оптимизации сгенерированных знаний и сборки мусора. Здесь происходит интерпретация арифметических отношений, сгенерированных в процессе решения на предыдущем этапе, т. е. попытка вычисления недостающих значений компонентов связок арифметических отношений (например, сложение и произведение величин) на основе имеющихся значений. Если вычислить все недостающие значения не представляется возможным, то все знания, сгенерированные на предыдущем этапе, уничтожаются и решение переходит на этап применения стратегий. В таком случае применение логического вывода для рассматриваемого на предыдущем этапе утверждения считается нецелесообразным. Также на данном этапе происходит устранение синонимии, если таковая появилась на предыдущем этапе решения, например, сгенерирована связка отношения *совпадение** между некоторыми объектами. В конечном итоге происходит удаление конструкций, ставших ненужными и по каким-либо причинам не удаленных на предыдущих этапах решения.

Если все этапы решения выполнены успешно, то оно возвращается к первому этапу, в случае если ответ не получен, процесс повторяется еще раз. Стоит отметить, что в процессе решения один и тот же объект или одно и то же высказывание может быть использовано многократно, если это целесообразно. Однако очевидно, что применение одного и того же утверждения для одного объекта несколько раз не имеет смысла, при условии, что нужные знания из памяти не удаляются в процессе решения какими-либо сторонними sc-агентами.

Одним из возможных вариантов стратегии решения задач является также использование интеллектуального пакета программ. В настоящее время данный подход в качестве эксперимента реализован в прототипе интеллектуальной справочной системы по теории графов. В указанном подходе после возникновения в памяти вопросной ситуации осуществляется просмотр спецификаций имеющихся в системе программ, ориентированных на решение какой-либо задачи. Программы могут быть реализованы как на внешних языках программирования, так и на языке SCP. В случае если условие запуска программы соответствует вопросной конструкции, программа запускается на выполнение с соответствующими параметрами. При этом допускается возможность существования программ, необходимых для ответа на один и тот же вопрос, однако с различным количеством параметров. В результате работы программа генерирует в памяти некоторую ответную конструкцию. Применение интеллектуального пакета программ позволяет ускорить процесс решения задачи, однако увеличивает зависимость всего решателя от конкретной платформы или предметной области. Данный подход легко

интегрируется с другими стратегиями решения задач и встраивается в общий процесс решения.

Как можно заключить из описанного процесса решения задачи, временная сложность решения напрямую не зависит от непосредственно количества объектов и утверждений, имеющихся в базе знаний, так как сам процесс решения осуществляется в некотором контексте вопроса, а не во всей базе знаний. Объем данного контекста определяется, во-первых, конкретной задачей, во-вторых, качеством проектирования базы знаний.

Существует несколько путей уменьшения времени решения задачи:

– оптимизация исходных ходов реализованных sc-агентов и программ;

– оптимизация процесса взаимодействия sc-агентов через общую память, модификация языка вопросов;

– применение различных эвристик для оптимизации перебора и применения утверждений в базе знаний, а также выбора между различными стратегиями решения и моделями логического вывода;

– оптимизация текущей реализации модели графодинамической ассоциативной памяти, а в конечном счете переход на аппаратную реализацию.

Рассмотрим процесс применения решателя в рамках интеллектуальной справочной системы по геометрии Евклида.

Пример условия задачи.

Исходные данные:

– в треугольнике *Треугк(ТчкА;ТчкВ;ТчкС)* заданы три биссектрисы *Отр(ТчкА;ТчкА1)*, *Отр(ТчкВ;ТчкВ1)* и *Отр(ТчкС;ТчкС1)*;

– в треугольник *Треугк(ТчкА;ТчкВ;ТчкС)* вписана окружность *Окр(ТчкО;ТчкА2)*;

– окружность *Окр(ТчкО;ТчкА2)* и треугольник *Треугк(ТчкА;ТчкВ;ТчкС)* имеют общие точки *ТчкА2*, *ТчкВ2*, *ТчкС2*;

– длина отрезка *Отр(ТчкА;ТчкВ)* равна 12 см;

– длина отрезка *Отр(ТчкА;ТчкС)* равна 10 см;

– длина отрезка *Отр(ТчкА1;ТчкС)* равна 5 см.

Задача: определить длину радиуса окружности *Окр(ТчкО;ТчкА2)* (рисунок 2.30).

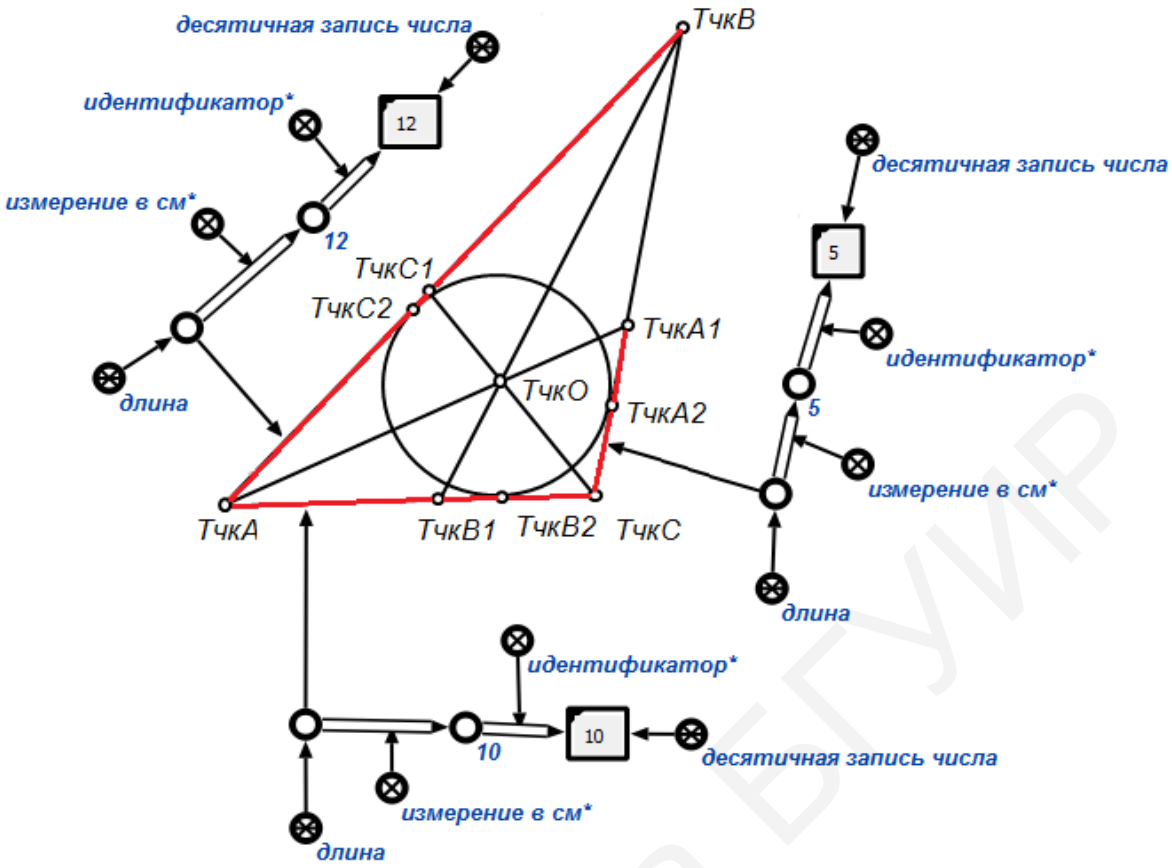

Рисунок 2.30 – Иллюстрация к задаче

Содержимое базы знаний системы (контекст решения задачи):

– теорема о биссектрисе (биссектриса внутреннего угла треугольника делит противоположную сторону в отношении, равном отношению двух прилежащих сторон);

– формула вычисления длины отрезка как суммы длин двух отрезков, его составляющих;

– формула вычисления периметра треугольника как суммы длин трех его сторон;

– формула Герона для вычисления площади треугольника по длинам трех его сторон;

– формула вычисления радиуса вписанной в треугольник окружности как отношения площади треугольника к его полупериметру.

На рисунках 2.31–2.34 представлено формальное описание условия задачи, а также формальное описание некоторых фрагментов базы знаний.

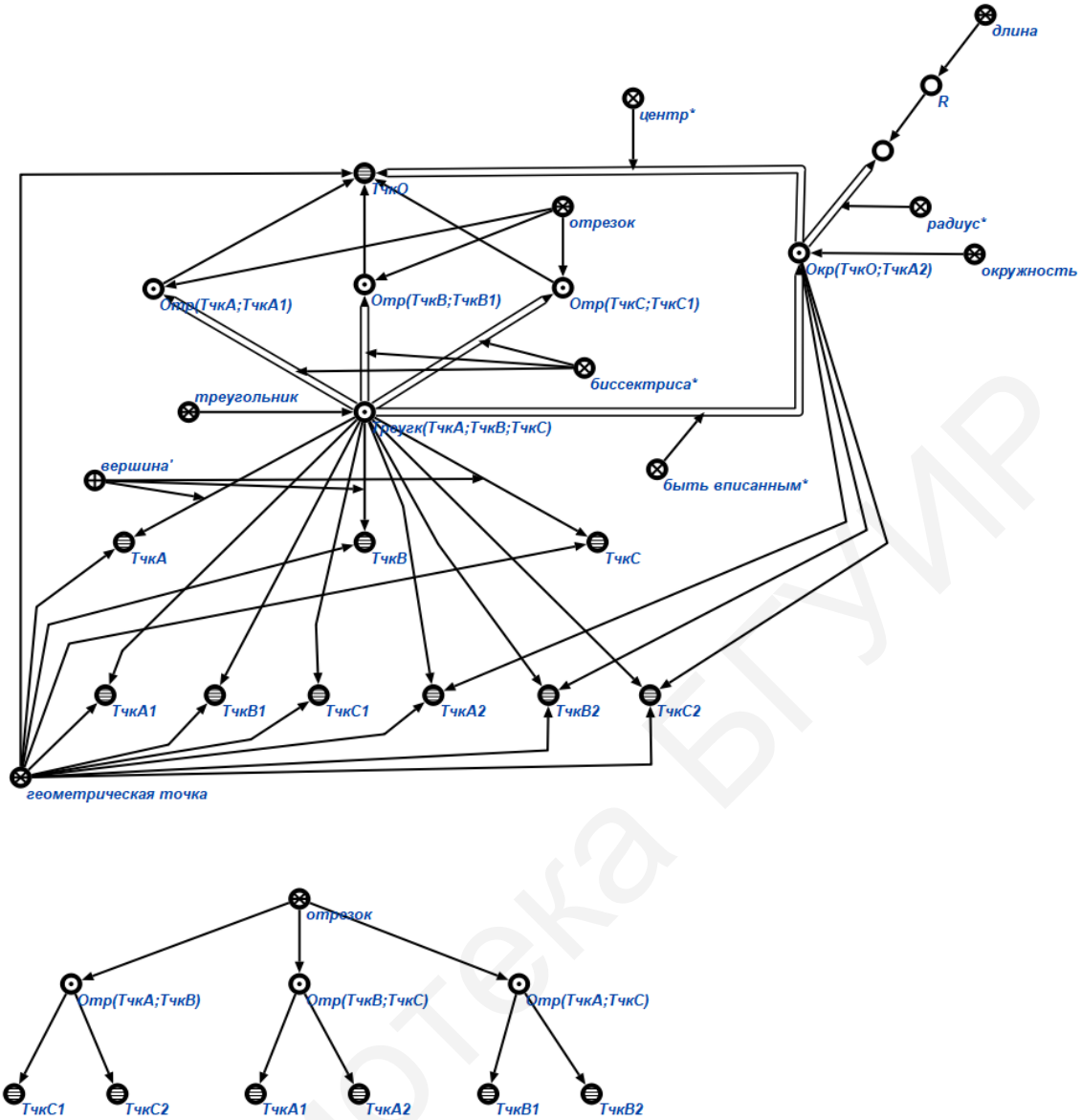

Рисунок 2.31 – Формальное описание условия задачи (фрагмент 1)

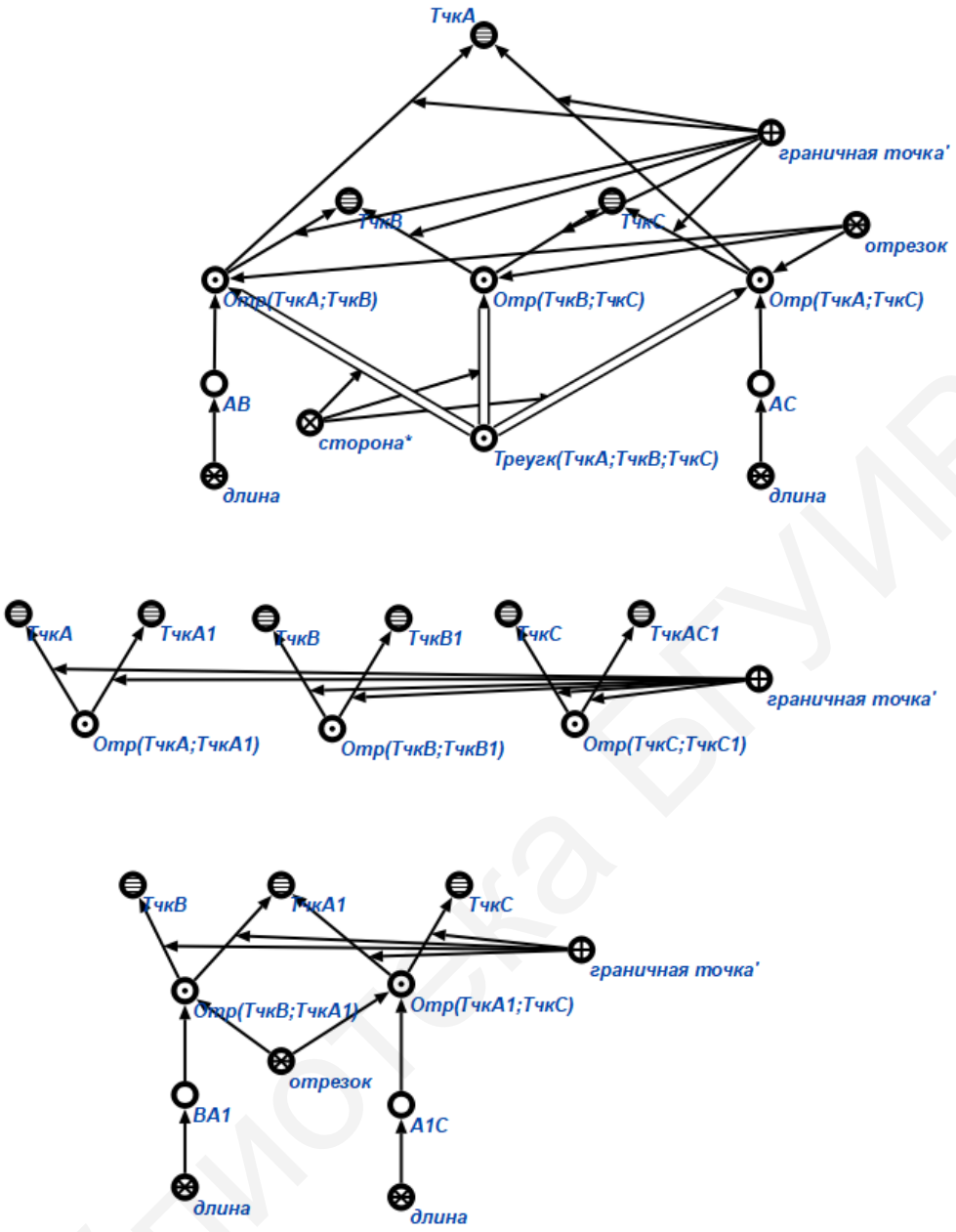

Рисунок 2.32 – Формальное описание условия задачи (фрагмент 2)

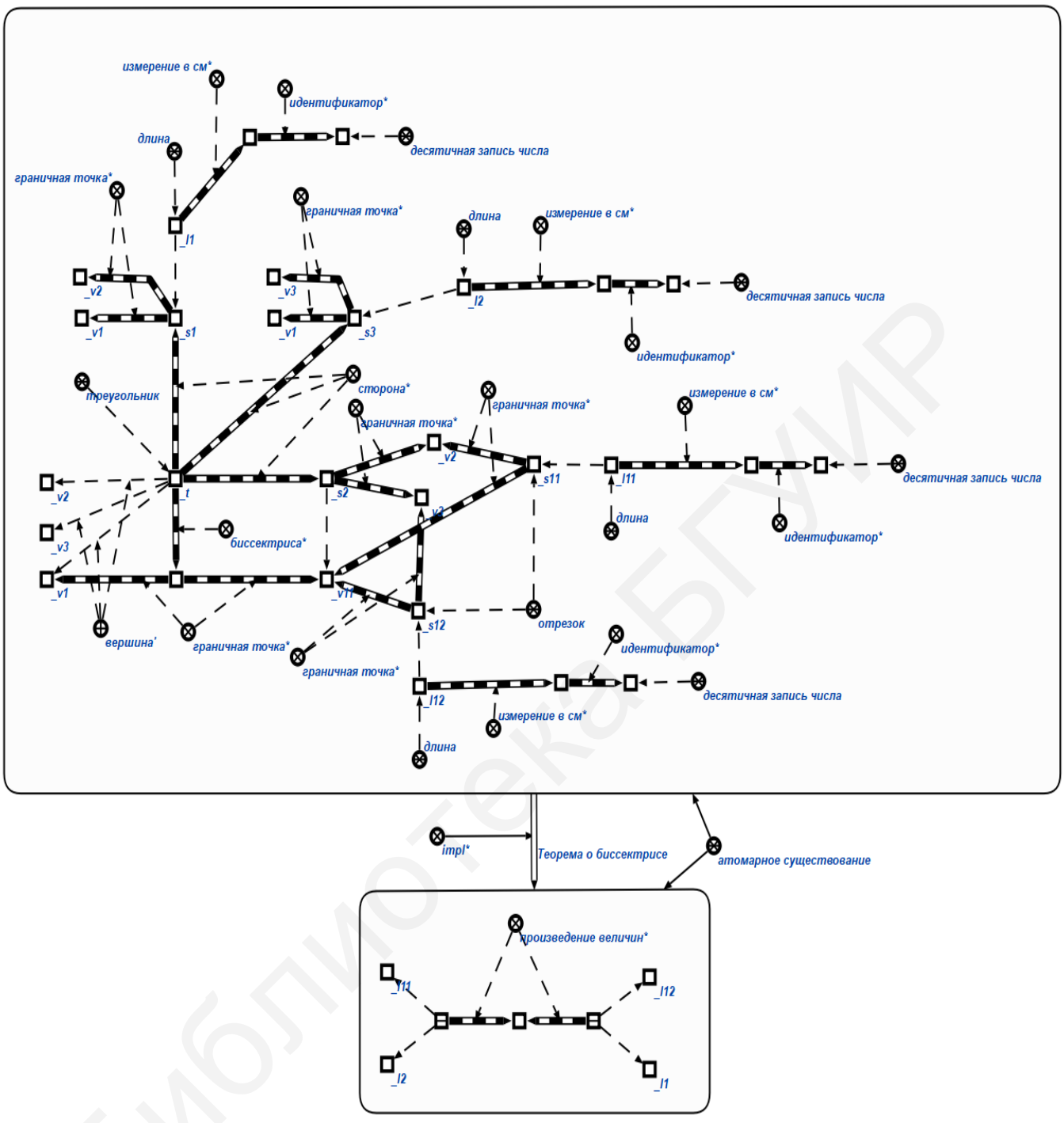

Рисунок 2.33 – Теорема о биссектрисе

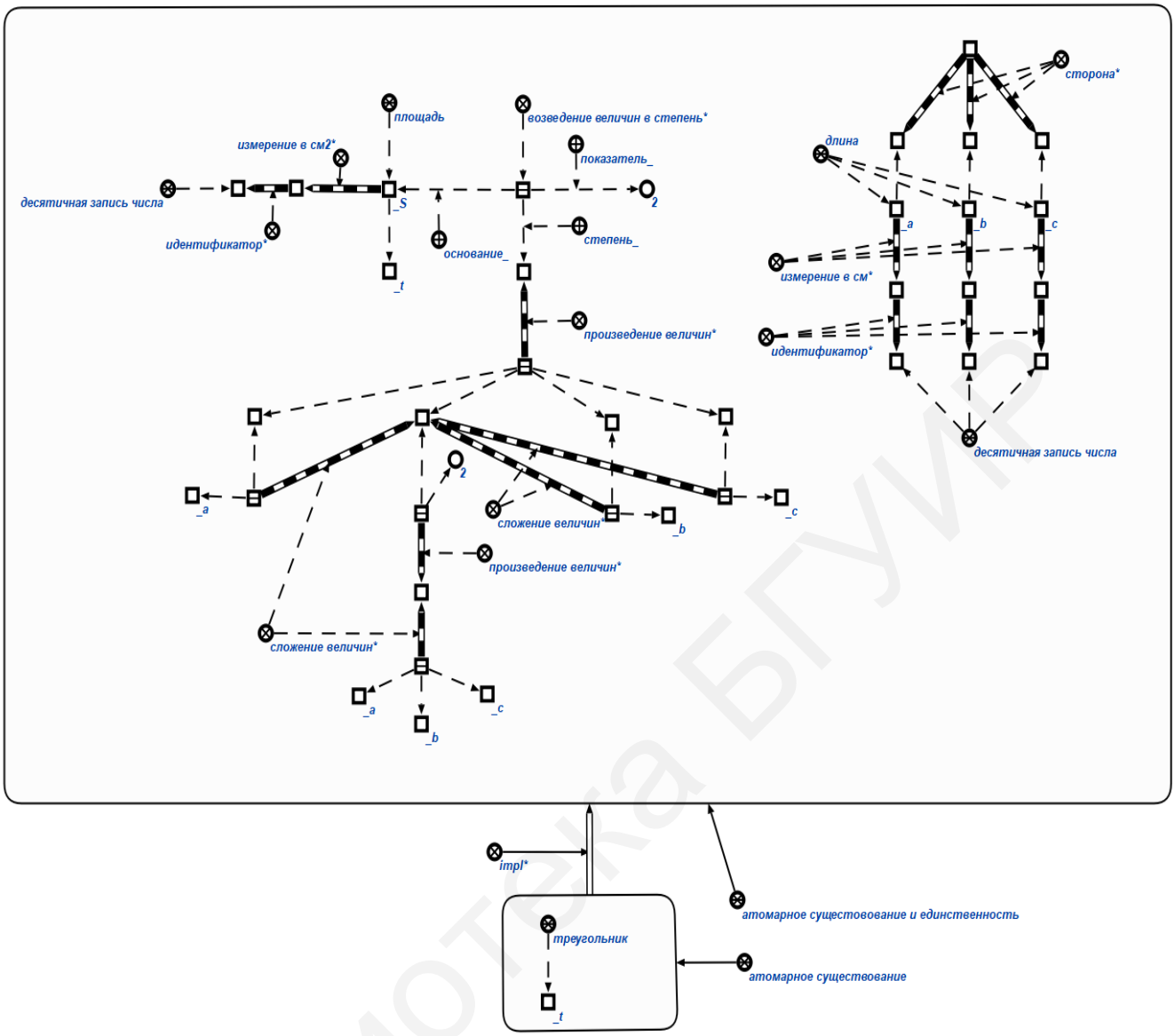

Рисунок 2.34 – Формула Герона

Для инициирования набора sc-агентов необходимо создать в памяти вопросную ситуацию (рисунок 2.35).

Рисунок 2.35 – Формат вопроса

Опишем краткий протокол решения задачи по шагам:

Шаг 1.

Используемый sc-агент.

Sc-агент поиска значения величины.

Пояснение:

Sc-агент пытается найти уже имеющееся значение требуемой величины. Требуемое значение отсутствует.

Условие инициирования на шаге 1 представлено на рисунке 2.36.

Рисунок 2.36 – Условие инициирования sc-агента на шаге 1

Результат работы на шаге 1 изображен на рисунке 2.37.

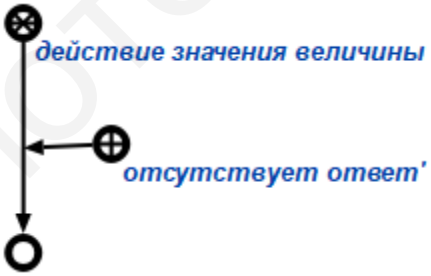

Рисунок 2.37 – Результат работы sc-агента на шаге 1

Шаг 2.

Используемый sc-агент.

Sc-агент, осуществляющий поиск в глубину.

Пояснение:

Sc-агент организует запуск рекурсивной операции поиска в глубину.

Условие инициирования на шаге 2 представлено на рисунке 2.38.

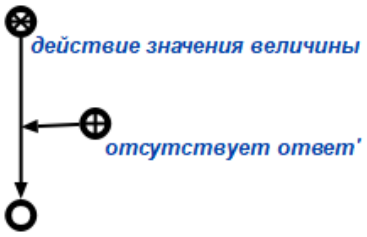

Рисунок 2.38 – Условие инициирования sc-агента на шаге 2

Результат работы на шаге 2 изображен на рисунке 2.39.

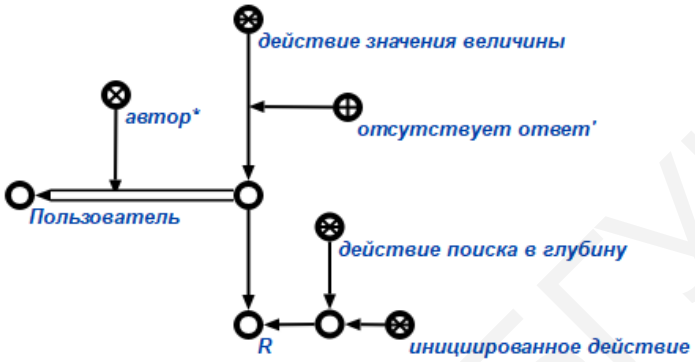

Рисунок 2.39 – Результат работы sc-агента на шаге 2

Шаг 3.

Используемый sc-агент.

Sc-агент поиска в глубину.

Пояснение:

Операция просматривает все объекты, связанные с исходным объектом и пытается сгенерировать новые знания. Если знания сгенерировать не удалось, запрос поиска в глубину устанавливается на объекты, связанные с данным. Просмотренные узлы добавляются в множество просмотренных узлов. В данном случае новые знания генерируются для объекта *Треугк(ТчкА;ТчкВ;ТчкС)*. При этом используется утверждение «*Теорема о биссектрисе*».

Условие инициирования на шаге 3 представлено на рисунке 2.40.

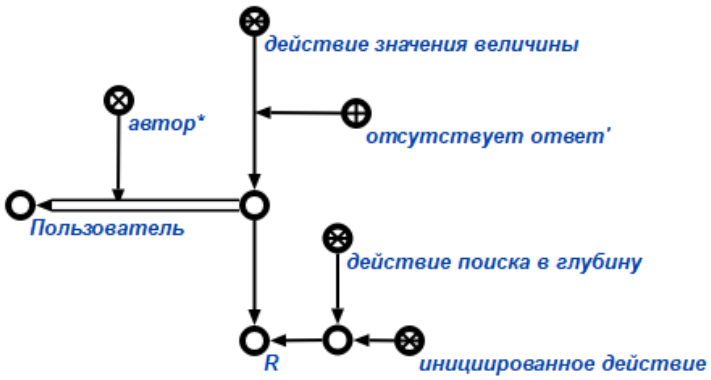

Рисунок 2.40 – Условие инициирования sc-агента на шаге 3

Результат работы на шаге 3 изображен на рисунке 2.41.

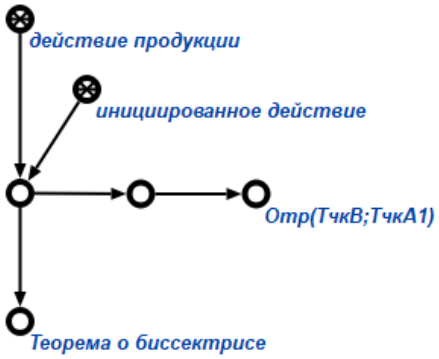

Рисунок 2.41 – Результат работы sc-агента на шаге 3

Шаг 4.

Используемый sc-агент.

Sc-агент генерации значения продукции.

Sc-агент интерпретации арифметических выражений.

Пояснение:

На основании теоремы о биссектрисе вычисляется длина отрезка *Отр(ТчкВ;ТчкА1)* – 6 см.

Условие инициирования на шаге 4 представлено на рисунке 2.42.

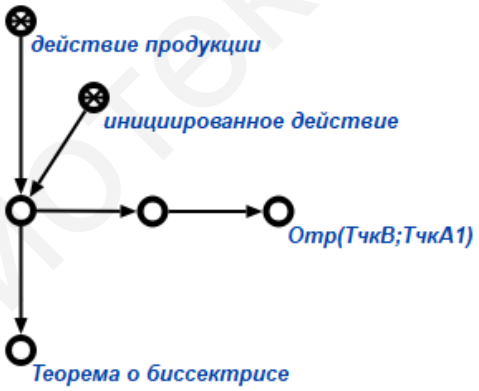

Рисунок 2.42 – Условие инициирования sc-агента на шаге 4

Результат работы на шаге 4 изображен на рисунке 2.43.

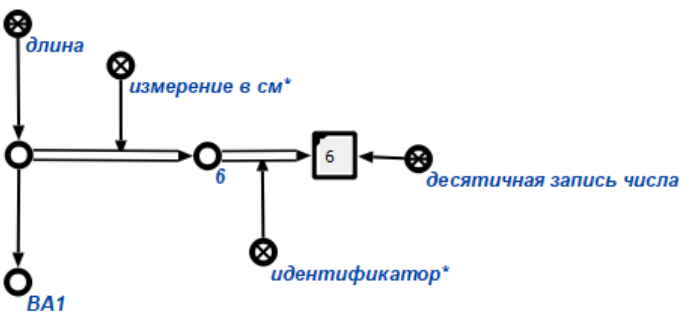

Рисунок 2.43 – Результат работы sc-агента на шаге 4

Шаг 5 аналогичен шагу 1.

Используемый sc-агент.

Sc-агент поиска значения величины.

Шаг 6 аналогичен шагу 2.

Используемый sc-агент.

Sc-агент, осуществляющий поиск в глубину.

Шаг 7 аналогичен шагу 3.

Используемый sc-агент.

Sc-агент поиска в глубину.

Пояснение:

В данном случае новые знания генерируются для объекта *Отр(ТчкВ;ТчкС)*. При этом используется утверждение «*Формула вычисления длины отрезка*».

Шаг 8.

Используемый sc-агент.

Sc-агент генерации значения продукции.

Sc-агент интерпретации арифметических выражений.

Пояснение:

Вычисляется длина отрезка *Отр(ТчкВ;ТчкС)* как сумма длин отрезков *Отр(ТчкВ;ТчкА1) и Отр(ТчА1;ТчкС)* – 11 см.

Результат работы на шаге 8 изображен на рисунке 2.44.

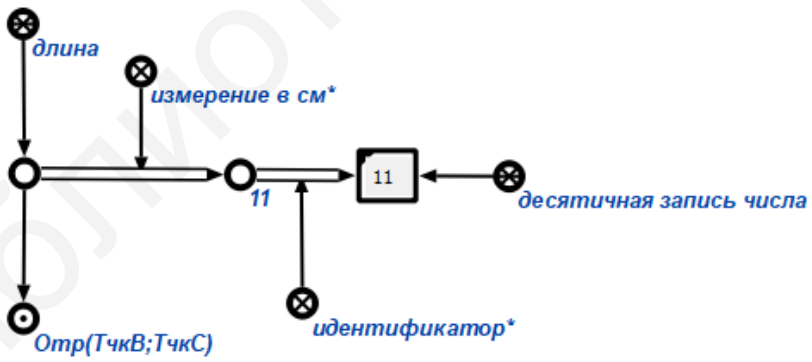

Рисунок 2.44 – Результат работы sc-агента на шаге 8

Шаг 9 аналогичен шагу 1.

Используемый sc-агент.

Sc-агент поиска значения величины.

Шаг 10 аналогичен шагу 2.

Используемый sc-агент.

Sc-агент, осуществляющий поиск в глубину.

Шаг 11 аналогичен шагу 3.

Используемый sc-агент.

Sc-агент поиска в глубину.

Пояснение:

В данном случае новые знания генерируются для объекта *Треугк(ТчкА;ТчкВ;ТчкС)*. При этом используется утверждение «*Формула вычисления периметра треугольника*».

Шаг 12.

Используемый sc-агент.

Sc-агент генерации значения продукции.

Sc-агент интерпретации арифметических выражений.

Пояснение:

Вычисляется периметр треугольника *Треугк(ТчкА;ТчкВ;ТчкС)* – 33 см.

Результат работы на шаге 12 изображен на рисунке 2.45.

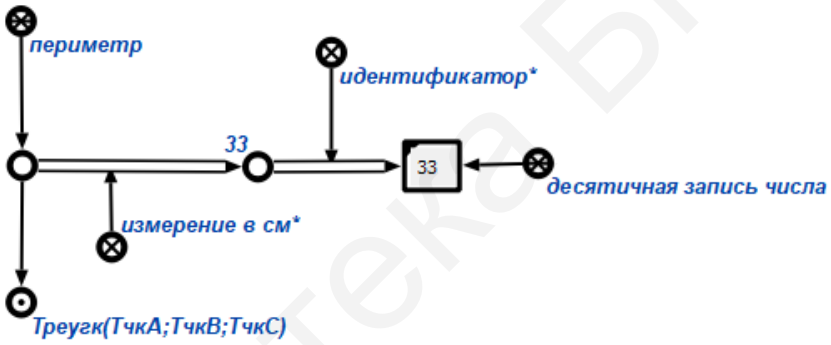

Рисунок 2.45 – Результат работы sc-агента на шаге 12

Шаг 13 аналогичен шагу 1.

Используемый sc-агент.

Sc-агент поиска значения величины.

Шаг 14 аналогичен шагу 2.

Используемый sc-агент.

Sc-агент, осуществляющий поиск в глубину (postorder_tree_search_manager).

Шаг 15 аналогичен шагу 3.

Используемый sc-агент.

Sc-агент поиска в глубину.

Пояснение:

В данном случае новые знания генерируются для объекта *Треугк(ТчкА;ТчкВ;ТчкС)*. При этом используется утверждение «*Формула Герона*».

Шаг 16.

Используемый sc-агент.

Sc-агент генерации значения продукции.

Sc-агент интерпретации арифметических выражений.

Пояснение:

Вычисляется площадь треугольника *Треугк(ТчкА;ТчкВ;ТчкС)* – 51,52 см^2.

Результат работы на шаге 16 изображен на рисунке 2.46.

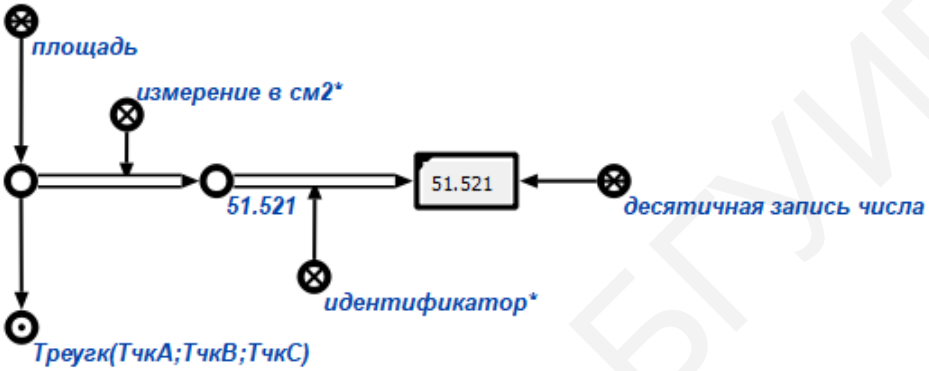

Рисунок 2.46 – Результат работы sc-агента на шаге 16

Шаг 17 аналогичен шагу 1.

Используемый sc-агент.

Sc-агент поиска значения величины.

Шаг 18 аналогичен шагу 2.

Используемый sc-агент.

Sc-агент, осуществляющий поиск в глубину.

Шаг 19 аналогичен шагу 3.

Используемый sc-агент.

Sc-агент поиска в глубину.

Пояснение:

В данном случае новые знания генерируются для объекта *Окр(ТчкО;ТчкА2)*. При этом используется утверждение «*Соотношение площади треугольника и радиуса вписанной окружности*».

Шаг 20.

Используемый sc-агент.

Sc-агент генерации значения продукции.

Sc-агент интерпретации арифметических выражений.

Пояснение:

Вычисляется радиус окружности *Окр(ТчкО;ТчкА2)* – 3,1225 см.
Результат работы на шаге 20 изображен на рисунке 2.47.

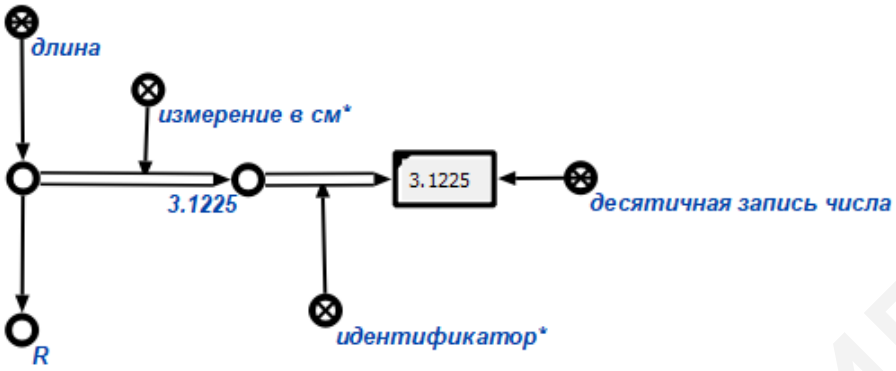

Рисунок 2.47 – Результат работы sc-агента на шаге 20

Шаг 21.

Используемый sc-агент.

Sc-агент поиска значения величины.

Пояснение:

Sc-агент пытается найти уже имеющееся значение требуемой величины.

Требуемое значение присутствует.

Результат работы на шаге 21 изображен на рисунке 2.48.

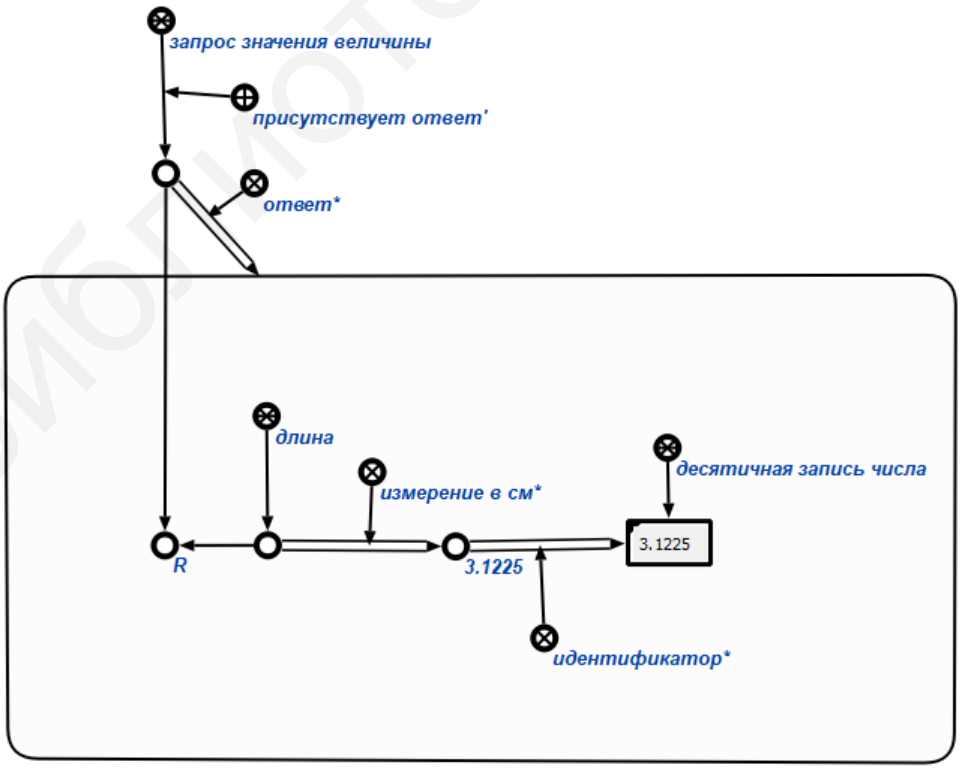

Рисунок 2.48 – Результат работы sc-агента на шаге 21

Выбор sc-агентов, необходимых для решения задач в каждой конкретной прикладной интеллектуальной системе, определяется разработчиком интеллектуального решателя. В связи с этим некоторые sc-агенты, необходимые в одной предметной области, будут избыточными в другой.

Например, sc-агенты нечеткого и правдоподобного вывода будут очень полезны в системах, где имеется много критериев для принятия решения, анализируется множество характеристик, которые просто невозможно описать с точки зрения однозначной истинности или ложности.

Эти же sc-агенты в геометрии Евклида, напротив, будут избыточными, так как решение задач осуществляется только по правилам классического вывода (дедуктивного, обратного и т. д.).

2.4.3 Решатель задач ИСС по теории графов

Для ИСС по теории графов реализованы следующие агенты (без учета агентов, заимствованных из библиотеки):

1) агенты, отвечающие на общие вопросы к графу:

– *Абстрактный sc-агент спецификации графа;*

– *Абстрактный sc-агент поиска свойств графа;*

– *Абстрактный sc-агент поиска числовых характеристик графа;*

– *Абстрактный sc-агент поиска множеств, характеризующих граф.*

2) агенты формирования различных множеств, характеризующих граф:

– *Абстрактный sc-агент поиска минимального остовного дерева графа;*

– *Абстрактный sc-агент поиска множества точек сочленения графа;*

– *Абстрактный sc-агент поиска множества мостов графа;*

– *Абстрактный sc-агент поиска множества тупиков графа;*

– *Абстрактный sc-агент поиска множества антитупиков графа.*

3) агенты по определению вида графа:

– *Абстрактный sc-агент проверки графа на планарность;*

– *Абстрактный sc-агент проверки графа на связность;*

– *Абстрактный sc-агент проверки графа на ориентированность;*

– *Абстрактный sc-агент проверки графа на цикличность;*

– *Абстрактный sc-агент проверки графа на симметричность;*

– *Абстрактный sc-агент проверки графа на транзитивность;*

– *Абстрактный sc-агент проверки графа на рефлексивность*.

4) агенты расчета числовых характеристик графа:

– *Абстрактный sc-агент поиска количества компонентов связности;*

– *Абстрактный sc-агент расчета диаметра графа;*

– *Абстрактный sc-агент расчета радиуса графа.*

Большинству из представленных выше агентов соответствует scp-программа, реализующая основной алгоритм данного агента и имеющая спецификацию, которая позволяет оценить возможность и целесообразность применения этой программы в процессе решения некоторой задачи.

Для того чтобы обеспечить возможность использования нескольких программ или логических утверждений в процессе решения одной задачи, в рамках ИИС по теории графов был заимствован и доработан рассмотренный выше *Неатомарный абстрактный sc-агент решения задач*. После доработки данный агент получил возможность анализировать не только логические утверждения, но и спецификации программ и при необходимости инициировать выполнение этих программ на необходимых исходных данных. В отличие от исходного модифицированный агент реализует стратегию поиска решения задачи от цели (обратный вывод) и пытается выстроить такую последовательность программ и логических утверждений, применение которых на имеющихся входных данных позволит получить требуемый результат.

Действия, выполняемые модифицированным агентом, имеют два аргумента. Первым аргументом является знак *сущности*, характеристику которой необходимо найти или вычислить (например, знак конкретного графа), вторым – знак *класса*, соответствующего описываемой характеристике, при этом вторым аргументом может быть как абсолютное понятие, так и относительное. Например, если необходимо проверить, является ли заданный граф ациклическим, вторым аргументом будет знак понятия *ациклический граф*; если необходимо определить диаметр заданного графа, то вторым аргументом будет знак отношения *диаметр**.

Рассмотрим структуру решателя задач по теории графов на языке SCn.

Неатомарный абстрактный sc-агент решения задач
*<= декомпозиция абстрактного sc-агента**:
{
- *Абстрактный sc-агент генерации условия задачи по шаблону*
- *Абстрактный sc-агент приведения графа к заданному виду*
- *Абстрактный sc-агент решения составной задачи*
- *Неатомарный абстрактный sc-агент решения простой задачи*
  *<= декомпозиция абстрактного sc-агента**:
  {
  - *Абстрактный sc-агент поиска связи заданного sc-элемента с заданным понятием*

- *Абстрактный sc-агент применения бинарной операции*

}

}

Рассмотрим примеры работы некоторых из реализованных агентов.

Агент определения свойств графа

Задача агента состоит в том, чтобы для заданного графа определить классы графовых структур, которым принадлежит или не принадлежит указанный граф. Для определения каждого свойства графа используется соответствующая программа.

Исходный граф представлен на рисунке 2.49.

Рисунок 2.49 – Исходный граф упражнения 2

Результат работы агента представлен на рисунке 2.50.

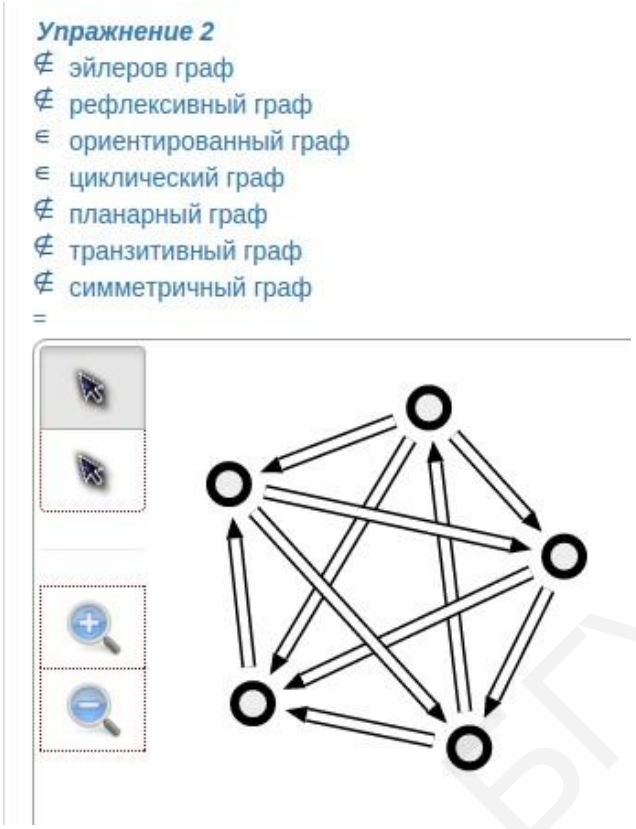

Рисунок 2.50 – Результат работы агента для упражнения 2

Агент нахождения множеств, характеризующих граф

Исходный граф представлен на рисунке 2.51.

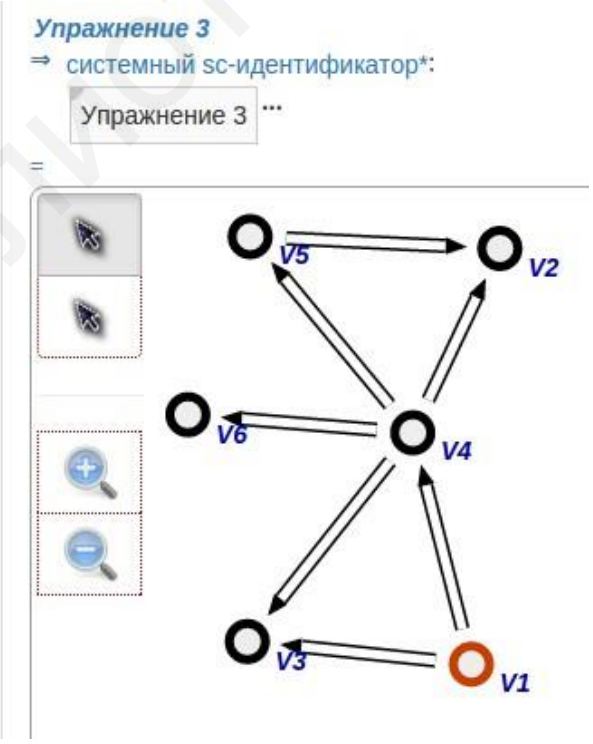

Рисунок 2.51 – Исходный граф для упражнения 3

Результат работы агента приведен на рисунке 2.52.

Рисунок 2.52 – Результат работы агента для упражнения 3

Минимальный остов графа представлен на рисунке 2.53.

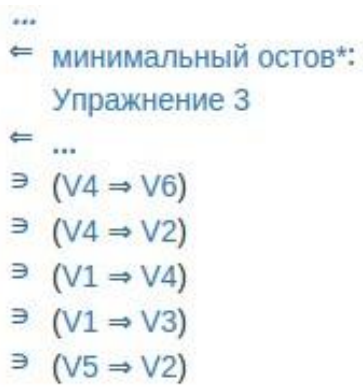

Рисунок 2.53 – Минимальный остов исходного графа

На рисунке 2.54 представлено множество мостов исходного графа.

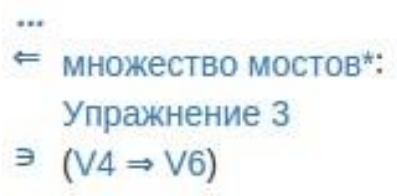

Рисунок 2.54 – Множество мостов исходного графа

Агент нахождения числовых характеристик графа

Исходный граф приведен на рисунке 2.55.

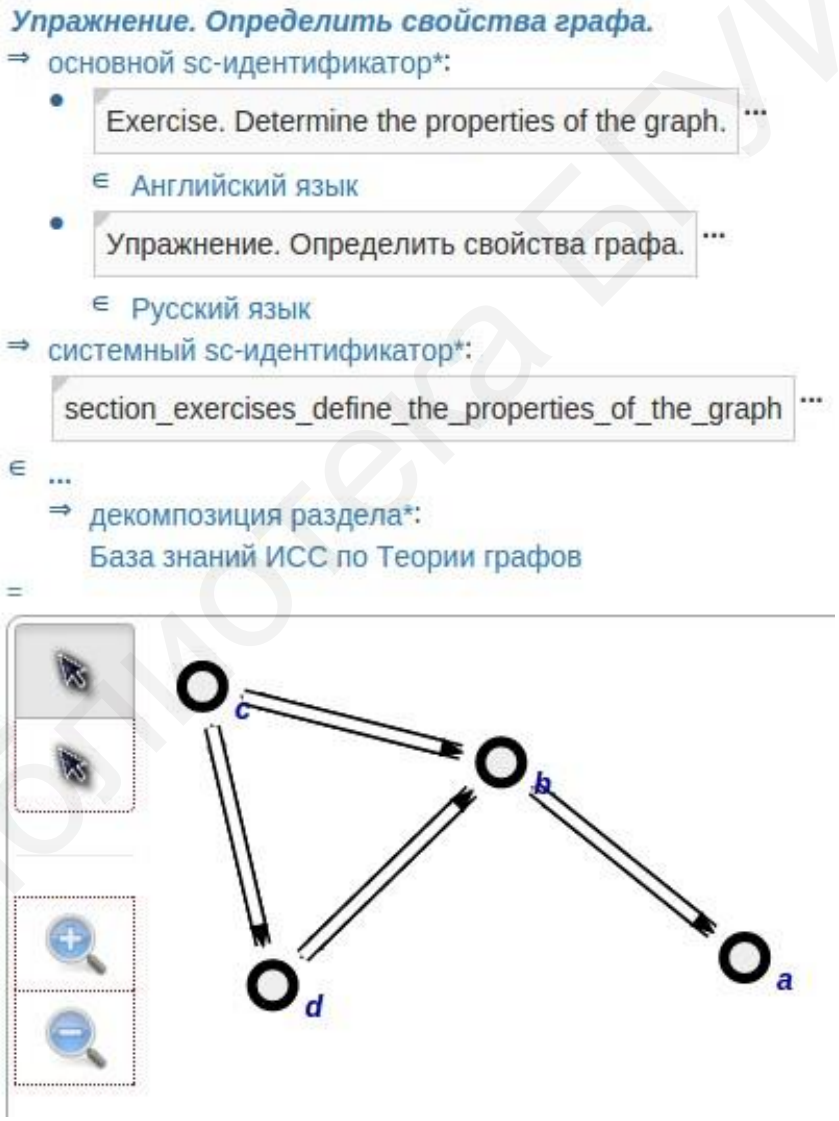

Рисунок 2.55 – Исходный граф для определения его свойств

Результат работы агента представлен на рисунке 2.56.

Рисунок 2.56 – Результат работы агента определения свойств графа

Агент спецификации графов

Задача агента состоит в том, чтобы для заданного графа определить все возможные его характеристики, используя все программы, имеющиеся в системе.

Исходный граф приведен на рисунке 2.57.

Упражнение. Нахождение двух множеств вершин, характеризующих двудольный граф.

⇒ основной sc-идентификатор*:

- Exercise. Finding two sets of peaks that characterize the bipartite graph. ...

  ∈ Английский язык

- Упражнение. Нахождение двух множеств вершин, характеризующих двудольный граф. ...

  ∈ Русский язык

⇒ системный sc-идентификатор*:

section_exercises_bipartite_graph_find_set ...

=

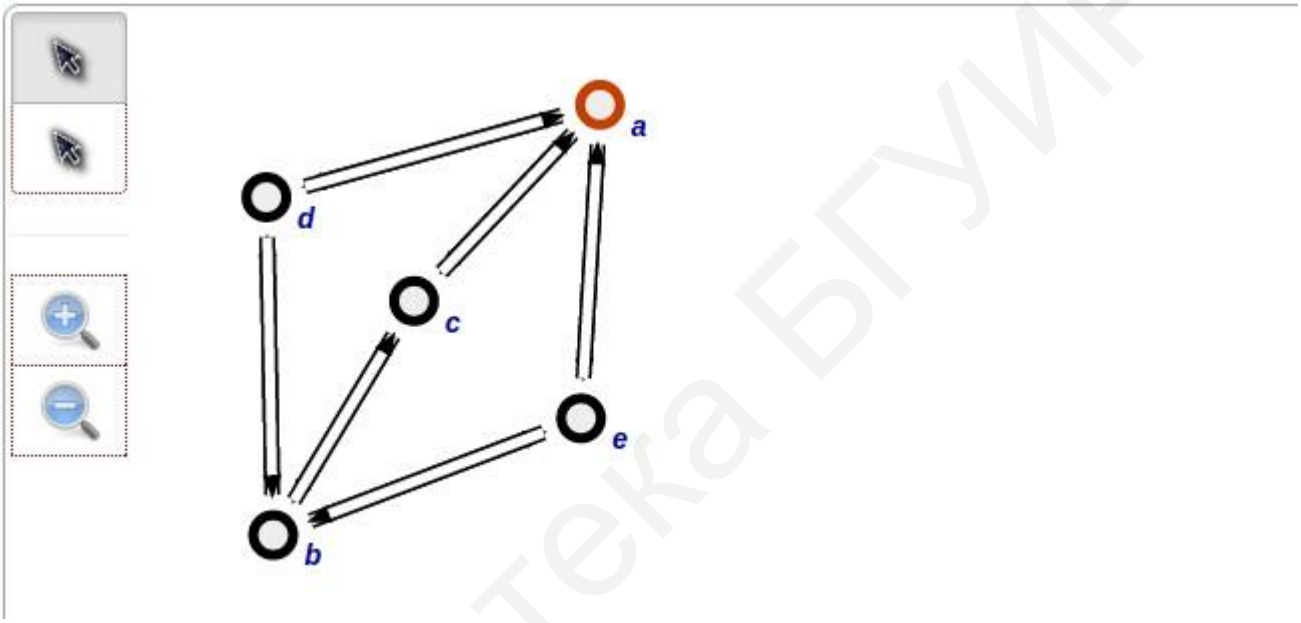

Рисунок 2.57 – Исходный двудольный граф

Результат работы агента приведен на рисунке 2.58.

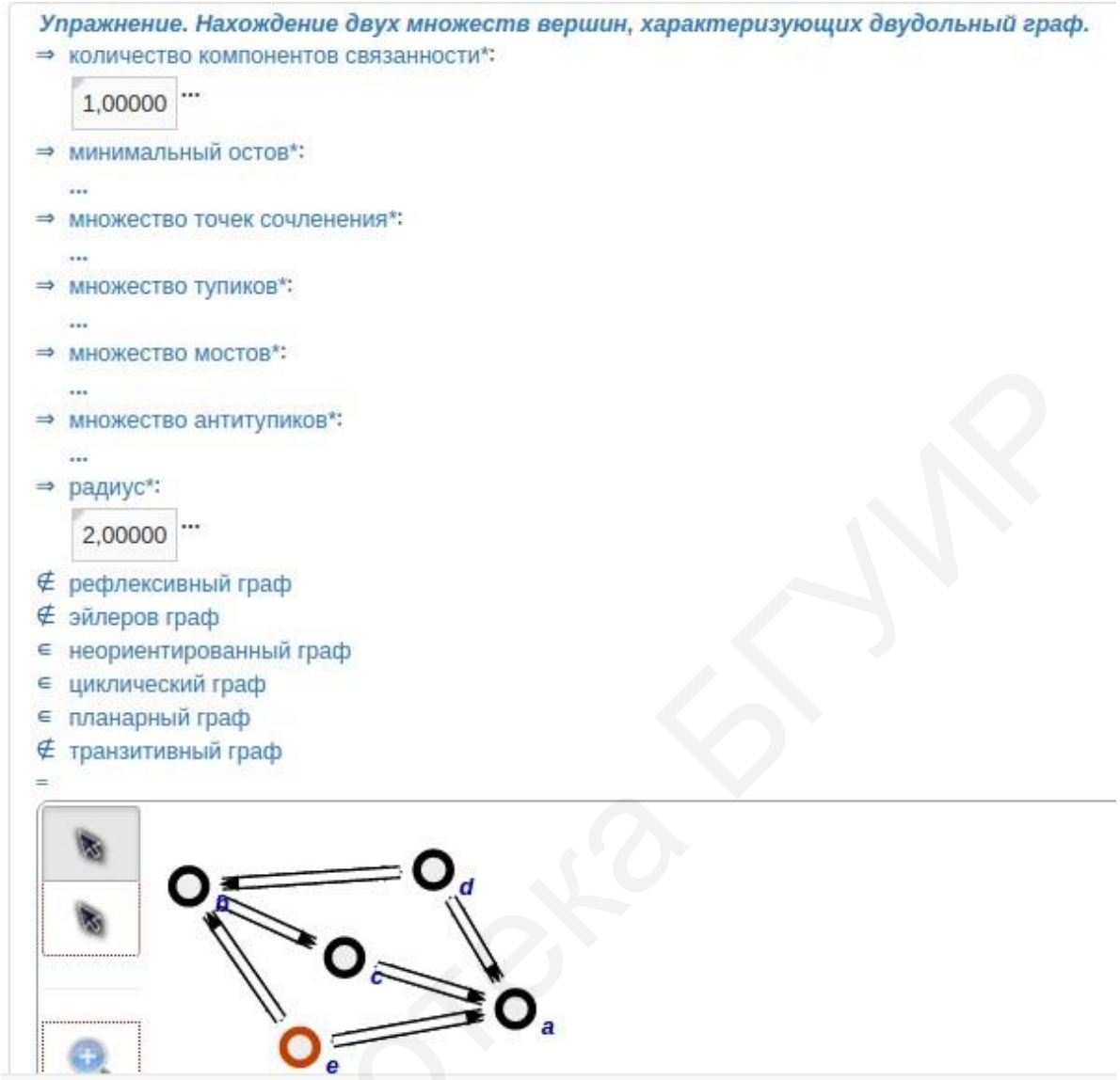

Рисунок 2.58 – Результат работы агента для двудольного графа

Разработанные решатели не уступают аналогам по большинству показателей, а также имеют ряд уникальных свойств, не присущих другим системам.

2.4.4 Решатель задач прототипа системы автоматизации рецептурного производства

В рамках сотрудничества между кафедрой интеллектуальных информационных технологий БГУИР и ОАО «Савушкин продукт» с использованием предлагаемых в данном учебном пособии моделей, методики и средств разработан прототип системы автоматизации производства.

При разработке системы использовались готовые компоненты из библиотеки многократно используемых компонентов решателей задач, а также были разработаны некоторые дополнительные sc-агенты:

1. *Агент определения текущего состояния указанных объектов*. Данный sc-агент в качестве аргумента соответствующего действия принимает некоторый sc-элемент. Результатом работы агента является конструкция, описывающая все ситуации, связанные с указанным sc-элементом, представленные в данный момент в базе знаний и являющиеся настоящими сущностями.

2. *Агент определения значений выходных характеристик указываемого процесса*. Данный sc-агент в качестве аргумента соответствующего действия принимает некоторый процесс. Результатом работы агента является конструкция, описывающая значения величин измеряемых параметров, полученных в результате выполнения процесса (например, объем или масса продукта, температура смеси в результате ее нагревания и т. д.).

3. *Агент поиска подклассов указанного класса, используемых в указанном процессе*. Данный sc-агент позволяет системе дать ответ на вопросы вида «Какое оборудование задействовано в процессе производства творога согласно указанному рецепту?», «Какие вещества участвуют в процессе производства творога согласно указанному рецепту?» и др.

4. *Агент поиска точки начала и окончания указанной временной сущности*. Данный sc-агент позволяет системе дать ответ на вопросы вида «Когда начинается/заканчивается первая (вторая, третья) смена?», «На какое время запланировано завершение текущего процесса производства творога?» и др.

Как можно заметить, ориентация указанных агентов на обработку семантических моделей баз знаний [53] и использование SC-кода в качестве основы для представления знаний позволяет сделать агенты в большой степени независимыми от предметной области рецептурного производства и впоследствии включить их в библиотеку многократно используемых компонентов решателей задач.

Кроме того, в рамках разрабатываемой системы реализовано несколько sc-агентов, взаимодействующих в процессе решения задачи с аппаратными средствами предприятия. В качестве примера такой задачи выбрана задача наполнения жидкостью небольшой емкости (например, пластиковой бутылки). Процесс наполнения состоит из следующих этапов:

– оператор нажимает на кнопку подачи жидкости, факт нажатия фиксируется в памяти системы;

– *Агент открытия клапана* реагирует на факт нажатия кнопки, осуществляет в sc-памяти поиск связанного с кнопкой клапана, формирует команду на его открытие (помещает клапан во множество *открытых*);

– жидкость начинает подаваться в емкость, которая находится на весах. Весы с заданной периодичностью изменяют в sc-памяти содержимое *файла*, описывающего массу бутылки с жидкостью. *Агент закрытия клапана* реагирует на изменение данного файла и при достижении требуемого значения закрывает клапан (помещает клапан во множество *закрытых*). Требуемое значение массы, клапан, который нужно закрыть, и другая необходимая информация фиксируется в памяти, и указанный агент находит ее по мере необходимости;

– *Агент отжатия кнопки* реагирует на закрытие клапана, находит в памяти связанную с клапаном кнопку и помещает ее во множество *отжатых*, позволяя таким образом нажимать ее снова.

Схематично описанная установка изображена на рисунке 2.59. На рисунке использованы следующие условные обозначения:

1 – пульт управления (кнопка запуска);
2 – контроллер;
3 – клапан, регулирующий подачу жидкости;
4 – источник жидкости;
5 – наполняемая емкость;
6 – весы.

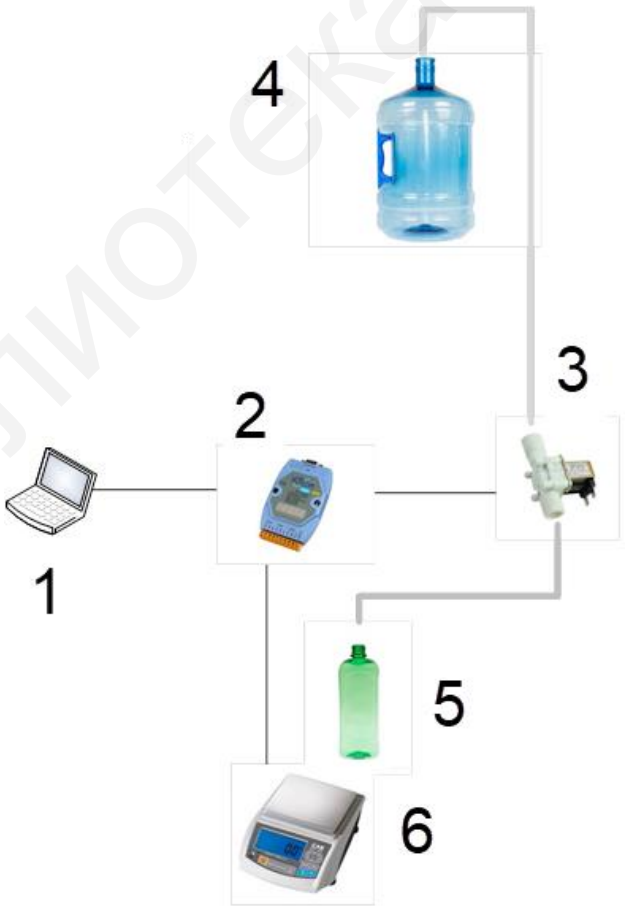

Рисунок 2.59 – Схема наполнения емкости

Использование предлагаемого многоагентного подхода для решения поставленной задачи позволяет при необходимости легко добавлять в процесс новые этапы, при этом внесение изменений в уже реализованные агенты не потребуется, поскольку агенты реагируют только на ситуации в sc-памяти, а не обмениваются сообщениями напрямую. Так, например, на факт достижения бутылкой необходимой массы может реагировать агент, запускающий звуковую или световую сигнализацию, при этом все остальные агенты, участвующие в процессе наполнения, изменений не потребуют.

Более подробно принципы автоматизации предприятия рецептурного производства с использованием моделей, методики и средств разработки гибридных решателей задач описаны в совместных с представителями ОАО «Савушкин продукт» работах [99, 100, 101].

ПРАКТИЧЕСКАЯ ЧАСТЬ

Лабораторная работа №1
Описание действий и задач в семантической памяти

Цель работы: описать в семантической памяти при помощи соответствующих языковых средств условие, пошаговое решение и результат для задач нескольких типов.

Задачи лабораторной работы:

1. Ознакомиться с понятиями действия и задачи, с классификацией действий и классификацией задач.
2. Изучить язык описания действий и задач в семантической памяти.

Методические указания. Вариант задания выбирается магистрантом самостоятельно и согласовывается с преподавателем. Средства разработки выбираются магистрантом самостоятельно.

Требования к отчету. В отчете необходимо описать условия, пошаговое решение и результат для задач нескольких типов с использованием формальных средств представления знаний. Отчет предоставить для проверки в электронном виде.

Варианты заданий:

1. Описать в семантической памяти условия пошагового выполнения для действия «Элементом каких множеств является указываемая сущность и какие роли она там выполняет?».
2. Описать в семантической памяти условия пошагового выполнения для действия «Какие элементы принадлежат указываемому множеству и какие роли они выполняют?».
3. Описать в семантической памяти условия пошагового выполнения для действия «Элементом каких множеств является указываемая сущность?».
4. Описать в семантической памяти условия пошагового выполнения для действия «Какие элементы принадлежат указываемому множеству?».
5. Описать в семантической памяти условия пошагового выполнения для действия «Какие варианты декомпозиции соответствуют указываемой сущности?».
6. Описать в семантической памяти условия пошагового выполнения для действия «Какие сущности являются частными по отношению к указываемой сущности?».

7. Описать в семантической памяти условия пошагового выполнения для действия «Какие сущности являются общими по отношению к указываемой сущности?».

8. Описать в семантической памяти условия пошагового выполнения для действия «Какие утверждения об указываемой сущности известны?».

9. Описать в семантической памяти условия пошагового выполнения для действия «В рамках каких предметных областей исследуется указанная сущность?».

10. Описать в семантической памяти условия пошагового выполнения для действия «С какими сущностями связана указываемая сущность?».

Теоретический материал: подраздел 1.3.

Лабораторная работа №2
Спецификация агентов обработки знаний

Цель работы: разработать спецификацию набора агентов обработки знаний для решения заданной задачи.

Задачи лабораторной работы:

1. Ознакомиться с понятием агента обработки знаний.
2. Изучить язык описания программ базовой машины обработки знаний.

Методические указания. Вариант задания выбирается магистрантом самостоятельно и согласовывается с преподавателем. Средства разработки выбираются магистрантом самостоятельно.

Требования к отчету. В отчете необходимо описать процесс разработки и спецификацию набора агентов обработки знаний для решения заданной задачи. Отчет предоставить для проверки в электронном виде.

Варианты заданий:

1. Разработать и специфицировать наборы агентов обработки знаний для действия «Элементом каких множеств является указываемая сущность и какие роли она там выполняет?».

2. Разработать и специфицировать наборы агентов обработки знаний для действия «Какие элементы принадлежат указываемому множеству и какие роли они выполняют?».

3. Разработать и специфицировать наборы агентов обработки знаний для действия «Элементом каких множеств является указываемая сущность?».

4. Разработать и специфицировать наборы агентов обработки знаний для действия «Какие элементы принадлежат указываемому множеству?».

5. Разработать и специфицировать наборы агентов обработки знаний для действия «Какие варианты декомпозиции соответствуют указываемой сущности?».

6. Разработать и специфицировать наборы агентов обработки знаний для действия «Какие сущности являются частными по отношению к указываемой сущности?».

7. Разработать и специфицировать наборы агентов обработки знаний для действия «Какие сущности являются общими по отношению к указываемой сущности?».

8. Разработать и специфицировать наборы агентов обработки знаний для действия «Какие утверждения об указываемой сущности известны?».

9. Разработать и специфицировать наборы агентов обработки знаний для действия «В рамках каких предметных областей исследуется указанная сущность?».

10. Разработать и специфицировать наборы агентов обработки знаний для действия «С какими сущностями связана указываемая сущность?».

Теоретический материал: подраздел 1.4.

Лабораторная работа №3
Разработка прототипа решателя задач

Цель работы: разработать прототип решателя задач для решения задач заданного класса.

Задачи лабораторной работы:

1. Ознакомиться с понятием решателя задач.
2. Изучить понятие процесса в семантической памяти.

Методические указания. Вариант задания выбирается магистрантом самостоятельно и согласовывается с преподавателем. Средства разработки выбираются магистрантом самостоятельно.

Требования к отчету. В отчете необходимо описать процесс разработки прототипа решателя задач для решения задач заданного класса, а также отразить иерархию агентов и спецификации агента. Можно использовать наработки из лабораторной работы №2. Отчет предоставить для проверки в электронном виде.

Варианты заданий:

1. Разработать прототип решателя задач для решения задач по теории графов.
2. Разработать прототип решателя задач для решения задач по геометрии.
3. Разработать прототип решателя задач для решения задач по алгебре.
4. Разработать прототип решателя задач для решения задач по правилам дорожного движения.
5. Разработать прототип решателя задач для решения задач по теории множеств.
6. Разработать прототип решателя задач для решения задач по теории нечетких множеств.
7. Разработать прототип решателя задач по химии.
8. Разработать прототип решателя задач по физике.
9. Разработать прототип решателя задач по логике.
10. Разработать прототип решателя задач по электротехнике.

Теоретический материал: подраздел 1.4.

Лабораторная работа №4
Компоненты решателей задач

Цель работы: разработать спецификацию многократно используемого компонента решателей задач.

Задачи лабораторной работы:

1. Изучить методику построения решателей задач.
2. Ознакомиться с библиотекой компонентов решателей задач.

Методические указания. Вариант задания выбирается магистрантом самостоятельно и согласовывается с преподавателем. Средства разработки выбираются магистрантом самостоятельно.

Требования к отчету. В отчете необходимо описать процесс разработки и спецификацию многократно используемого компонента решателя задач по заданной предметной области. Отчет предоставить для проверки в электронном виде.

Варианты заданий:

1. Разработать компонент для решателя задач по теории графов.
2. Разработать компонент для решателя задач по геометрии.
3. Разработать компонент для решателя задач по алгебре.
4. Разработать компонент для решателя задач по правилам дорожного движения.
5. Разработать компонент для решателя задач по теории множеств.
6. Разработать компонент для решателя задач по теории нечетких множеств.
7. Разработать компонент для решателя задач по химии.
8. Разработать компонент для решателя задач по физике.
9. Разработать компонент для решателя задач по логике.
10. Разработать компонент для решателя задач по электротехнике.

Теоретический материал: подраздел 2.2.

Лабораторная работа №5
Машины информационного поиска

Цель работы: разработать прототип машины информационного поиска в базе знаний.

Задачи лабораторной работы:

1. Изучить принципы построения машин информационного поиска в базе знаний.
2. Ознакомиться с классификацией поисковых агентов в семантической памяти.

Методические указания. Вариант задания выбирается магистрантом самостоятельно и согласовывается с преподавателем. Средства разработки выбираются магистрантом самостоятельно.

Требования к отчету. В отчете необходимо описать процесс разработки прототипа машины информационного поиска в базе знаний по заданной предметной области, выделить соответствующие агенты и описать алгоритм их работы. Отчет предоставить для проверки в электронном виде.

Варианты заданий:

1. Разработать машину информационного поиска в базе знаний по теории графов.
2. Разработать машину информационного поиска в базе знаний по геометрии.
3. Разработать машину информационного поиска в базе знаний по алгебре.
4. Разработать машину информационного поиска в базе знаний по правилам дорожного движения.
5. Разработать машину информационного поиска в базе знаний по теории множеств.
6. Разработать машину информационного поиска в базе знаний по теории нечетких множеств.
7. Разработать машину информационного поиска в базе знаний по химии.
8. Разработать машину информационного поиска в базе знаний по физике.
9. Разработать машину информационного поиска в базе знаний по логике.
10. Разработать машину информационного поиска в базе знаний по электротехнике.

Теоретический материал: подраздел 2.4.

Лабораторная работа №6
Решение вычислительных задач

Цель работы: разработать прототип решателя вычислительных задач в базе знаний.

Задачи лабораторной работы:

1. Изучить принципы организации решателей вычислительных задач в базе знаний.
2. Изучить стратегии поиска путей решения вычислительных задач.

3. Ознакомиться с принципами интерпретации математических выражений в базе знаний.

Методические указания. Вариант задания выбирается магистрантом самостоятельно и согласовывается с преподавателем. Средства разработки выбираются магистрантом самостоятельно.

Требования к отчету. В отчете необходимо описать алгоритм разработки прототипа решателя вычислительных задач в базе знаний по заданной предметной области, выделить соответствующие агенты и описать алгоритм их работы. Отчет предоставить для проверки в электронном виде.

Варианты заданий:

1. Разработать прототип решателя вычислительных задач в базе знаний по теории графов.
2. Разработать прототип решателя вычислительных задач в базе знаний по геометрии.
3. Разработать прототип решателя вычислительных задач в базе знаний по алгебре.
4. Разработать прототип решателя вычислительных задач в базе знаний по правилам дорожного движения.
5. Разработать прототип решателя вычислительных задач в базе знаний по теории множеств.
6. Разработать прототип решателя вычислительных задач в базе знаний по теории нечетких множеств.
7. Разработать прототип решателя вычислительных задач в базе знаний по химии.
8. Разработать прототип решателя вычислительных задач в базе знаний по физике.
9. Разработать прототип решателя вычислительных задач в базе знаний по логике.
10. Разработать прототип решателя вычислительных задач в базе знаний по электротехнике.

Теоретический материал: подраздел 2.4.

Лабораторная работа №7
Машины логического вывода

Цель работы: разработать прототип машины логического вывода в базе знаний.

Задачи лабораторной работы:

1. Изучить принципы организации логического вывода в базе знаний.
2. Изучить прямой и обратный вывод в базе знаний.
3. Ознакомиться с принципами автоматического преобразования логических формул в базе знаний.

Методические указания. Вариант задания выбирается магистрантом самостоятельно и согласовывается с преподавателем. Средства разработки выбираются магистрантом самостоятельно.

Требования к отчету. В отчете необходимо описать алгоритм разработки прототипа машины логического вывода в базе знаний по заданной предметной области, выделить соответствующие агенты и описать алгоритм их работы. Отчет предоставить для проверки в электронном виде.

Варианты заданий:

1. Разработать прототип машины логического вывода в базе знаний по теории графов.
2. Разработать прототип машины логического вывода в базе знаний по геометрии.
3. Разработать прототип машины логического вывода в базе знаний по алгебре.
4. Разработать прототип машины логического вывода в базе знаний по правилам дорожного движения.
5. Разработать прототип машины логического вывода в базе знаний по теории множеств.
6. Разработать прототип машины логического вывода в базе знаний по теории нечетких множеств.
7. Разработать прототип машины логического вывода в базе знаний по химии.
8. Разработать прототип машины логического вывода в базе знаний по физике.

9. Разработать прототип машины логического вывода в базе знаний по логике.

10. Разработать прототип машины логического вывода в базе знаний по электротехнике.

Теоретический материал: подраздел 2.4.

ЗАКЛЮЧЕНИЕ

В рамках данного учебного пособия подробно рассмотрены модели обработки знаний в интеллектуальных системах и модели решателей задач, реализующих указанные модели решения задач. Рассмотрена авторская агентно-ориентированная модель гибридного решателя задач, рассматривающая каждый такой решатель как иерархическую систему агентов, управляемых ситуациями и событиями в общей семантической памяти, и обеспечивающая модифицируемость таких решателей задач, а также возможность решения задач, требующих совместного использования различных методов решения задач. Кроме того, рассмотрена методика построения и модификации гибридных решателей задач, основанная на формальной онтологии деятельности разработчиков таких решателей и ориентированная на применение многократно используемых компонентов решателей, а также средства автоматизации и информационной поддержки процесса построения и модификации гибридных решателей задач, включающие средства автоматизации процесса построения агентов обработки знаний и библиотеку многократно используемых компонентов таких решателей.

Приведенный материал основан на результатах диссертационного исследования автора, поэтому является актуальным и соответствует современному уровню развития области разработки решателей задач.

Рассмотренные модели, средства и их программная реализация могут быть использованы при разработке решателей задач интеллектуальных систем различного назначения, как инструментальных, так и прикладных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Шункевич, Д. В. Агентно-ориентированные решатели задач интеллектуальных систем : автореф. дис. ... канд. техн. наук : 05.13.17 / Д. В. Шункевич ; Белорус. гос. ун-т информатики и радиоэлектроники – Минск : БГУИР, 2018. – 27 с.

2. Заливако, С. С. Семантическая технология компонентного проектирования интеллектуальных решателей задач / С. С. Заливако, Д. В. Шункевич // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2012) : материалы II Междунар. науч.-техн. конф., Минск, 16–18 февр. 2012 г. / Белорус. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков (отв. ред.) [и др.]. – Минск : БГУИР, 2012. – С. 297–314.

3. Нечеткие гибридные системы. Теория и практика / И. З. Батыршин [и др.] ; под ред. Н. Г. Ярушиной. – М. : Физматлит, 2007. – 208 с.

4. Голенков, В. В. Проект открытой семантической технологии компонентного проектирования интеллектуальных систем. В 2 ч. Ч. 2 : Унифицированные модели проектирования / В. В. Голенков, Н. А. Гулякина // Онтология проектирования . – 2014. – №4. – С. 34–53.

5. Пратт, Т. Языки программирования: разработка и реализация / Т. Пратт, М. Зелковиц ; под общ. ред. А. Матросова ; пер. с англ. – 4-е изд. – СПб. : Питер : Питер принт, 2002. – 688 с.

6. Гладков, Л. А. Генетические алгоритмы : учеб. пособие / Л. А. Гладков, В. В. Курейчик, В. М. Курейчик. – 2-е изд., испр. и доп. – М. : Физматлит, 2006. – 320 с.

7. Емельянов, В. В. Теория и практика эволюционного моделирования / В. В. Емельянов, В. В. Курейчик, В. М. Курейчик. – М. : Физматлит, 2003. – 432 с.

8. Беркинблит, М. Б. Нейронные сети : эксперим. учеб. пособие / М. Б. Беркинблит. – М. : МИРОС : Всерос. заоч. многопредмет. шк. Рос. акад. наук, 1993. – 96 с.

9. Головко, В. А. Нейрокомпьютеры и их применение. В 4 кн. Кн. 4 : Нейронные сети: обучение, организация и применение : учеб. пособие / В. А. Головко, А. И. Галушкин ; под общ. ред. А. И. Галушкина. – М. : Радиотехника, 2001. – 256 с.

10. Горбань, А. Н. Нейронные сети на персональном компьютере / А. Н. Горбань, Д. А. Россиев. – Новосибирск : Наука, 1996. – 276 с.

11. Достоверный и правдоподобный вывод в интеллектуальных системах / В. Н. Вагин [и др.] ; под ред. В. Н. Вагина, Д. А. Поспелова. – 2-е изд., испр. и доп. – М. : Физматлит, 2008. – 712 с.

12. Кулик, Б. А. Логика естественных рассуждений / Б. А. Кулик. – СПб. : Нев. диалект, 2001. – 128 с.

13. Пойа, Д. Математика и правдоподобные рассуждения / Д. Пойа ; пер. с англ. И. А. Ванштейна ; под ред. С. А. Яновской. – М. : Наука, 1975. – 463 с.

14. Батыршин, И. З. Основные операции нечеткой логики и их обобщения / И. З. Батыршин. – Казань : Отечество, 2001. – 100 с.

15. Деменков, Н. П. Нечеткое управление в технических системах : учеб. пособие / Н. П. Деменков. – М. : Изд-во Моск. гос. техн. ун-та, 2005. – 198 с.

16. Поспелов, Д. А. Моделирование рассуждений: опыт анализа мыслительных актов / Д. А. Поспелов. – М. : Радио и связь, 1989. – 184 с.

17. Reiter, R. A logic for default reasoning / R. Reiter // Artificial Intelligence. – 1980. – Vol. 13, №13. – P. 81–132.

18. Еремеев, А. П. Построение решающих функций на базе тернарной логики в системах принятия решений в условиях неопределенности / А. П. Еремеев // Изв. Рос. акад. наук. Теория и системы упр. – 1997. – №2. – С. 138–143.

19. Кахро, М. В. Инструментальная система программирования ЕС ЭВМ (ПРИЗ) / М. В. Кахро, А. П. Калья, Э. Х. Тыугу. – М. : Финансы и статистика, 1988. – 181 с.

20. Ефимов, Е. И. Решатели интеллектуальных задач / Е. И. Ефимов. – М. : Наука, 1982. – 320 с.

21. Раговский, А. П. Интеллектуальная многоагентная система дедуктивного вывода на основе сетевой организации / А. П. Раговский // Искусств. интеллект и принятие решений. – 2011. – №2. – С. 73–86.

22. Подколзин, А. С. Компьютерное моделирование логических процессов: архитектура решателя и язык решателя задач / А. С. Подколзин. – М. : Физматлит, 2008. – 1024 с.

23. Курбатов, С. С. Программное обеспечение для автоматического решения задач по планиметрии / С. С. Курбатов, А. П. Лобзин, Г. К. Хахалин // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2016) : материалы VI Междунар. науч.-техн. конф., Минск, 18–20 февр. 2016 г. / Белорус. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков (отв. ред.) [и др.]. – Минск : БГУИР, 2016. – С. 159–164.

24. Wolfram Alpha [Electronic resource]. – Mode of access: http://www.wolframalpha.com. – Date of access: 07.04.2021.

25. Таранчук, В. Б. Wolfram mathematica – средства и технологии разработки интеллектуальных обучающих систем / В. Б. Таранчук // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2015) : материалы V Междунар. науч.-техн. конф., Минск, 19–21 февр. 2015 г. / Белорус. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков (отв. ред.) [и др.]. – Минск : БГУИР, 2015. – С. 339–346.

26. Программный комплекс «УДАВ»: практическая реализация активного обучаемого логического ввода с линейной вычислительной сложностью на основе миварной сети правил / А. Н. Владимиров [и др.] // Труды НИИР : сб. науч. ст. / Науч.-исслед. ин-т радио. – М., 2010. – №1. – С. 108–116.

27. Хахалин, Г. К. Мультизадачное использование прикладной онтологии / Г. К. Хахалин, А. Л. Воскресенский // Одиннадцатая национальная конференция по искусственному интеллекту с международным участием, Дубна, 28 сентября – 3 октября 2008 г. : КИИ-2008 : тр. конф. : в 3 т. / Рос. ассоц. искусств. интеллекта. – М., 2008. – Т. 3. – С. 112–123.

28. Artificial General Intelligence [Electronic resource]. – Mode of access: http://agi-conf.org. – Date of access: 01.04.2021.

29. Kaner, C. Testing computer software / C. Kaner, J. Falk, H. Q. Nguyen. – 2nd ed. – Hoboken : Wiley, 1999. – 496 p.

30. Искусственный интеллект. В 3 кн. Кн. 1 : Системы общения и экспертные системы : Справочник / под ред. Э. В. Попова. – М. : Радио и связь, 1990. – 464 с.

31. Jackson, P. Introduction to expert systems / P. Jackson. – 3rd ed. – Boston : Addison-Wesley, 1998. – 542 p.

32. World Wide Web Consortium [Electronic resource]. – Mode of access: https://www.w3.org. – Date of access: 06.04.2021.

33. OWL 2 Web Ontology Language document overview [Electronic resource] // World Wide Web Consortium (W3C). – Mode of access: http://www.w3.org/TR/owl2-overview. – Date of access: 14.04.2021.

34. 1.1 concepts and abstract syntax [Electronic resource] // World Wide Web Consortium. – Mode of access: https://www.w3.org/TR/rdf11-concepts. – Date of access: 24.04.2021.

35. SPARQL 1.1 overview [Electronic resource] // World Wide Web Consortium (W3C). – Mode of access: https://www.w3.org/TR/sparql11-overview. – Date of access: 08.04.2021.

36. Chapter 3. Cypher [Electronic resource]. – Mode of access: http://neo4j.com/docs/stable/cypher-query-lang.html. – Date of access: 03.04.2021.

37. OWL Implementations [Electronic resource] // World Wide Web Consortium (W3C). – Mode of access: https://www.w3.org/2001/sw/wiki/OWL/Implementations. – Date of access: 11.04.2021.

38. Базовая технология разработки интеллектуальных сервисов на облачной платформе IACPaaS. В 2 ч. Ч. 1: Разработка базы знаний и решателя задач / В. В. Грибова [и др.] // Програм. инженерия. – 2015. – №12. – С. 3–11.

39. Проект IACPaaS. Комплекс для интеллектуальных систем на основе облачных вычислений / В. В. Грибова [и др.] // Искусств. интеллект и принятие решений. – 2011. – №1. – С. 27–35.

40. Загорулько, Ю. А. Технология конструирования средств обработки знаний на основе семантических сетей. Общая схема и базовые средства / Ю. А. Загорулько. – Новосибирск : ВЦ СО АН СССР, 1987. – 30 с. – (Препринт / Вычисл. центр Сиб. отд-ия Акад. наук ; №749).

41. Загорулько, Ю. А. Технология конструирования средств обработки знаний на основе семантических сетей. Средства спецификации и настройки / Ю. А. Загорулько. – Новосибирск : ВЦ СО АН СССР, 1988. – 24 с. – (Препринт / Вычисл. центр Сиб. отд-ия Акад. наук ; №793).

42. Загорулько, Ю. А. Технологии разработки интеллектуальных систем, основанные на интегрированной модели представления знаний / Ю. А. Загорулько // Открытые семантические технологии проектирования интеллектуальных систем (OSTIS-2013) : материалы III Междунар. науч.-техн. конф., Минск, 16–18 февр. 2013 г. / Белорус. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков (отв. ред.) [и др.]. – Минск, 2013. – С. 31–42.

43. Загорулько, Г. Б. Подход к организации комплексной поддержки процесса разработки интеллектуальных СППР в слабоформализованных предметных областях / Г. Б. Загорулько, Ю. А. Загорулько // Открытые семантические технологии проектирования интеллектуальных систем (OSTIS-2016) : материалы VI Междунар. науч.-техн. конф., Минск, 18–20 февр. 2016 г. / Белорус. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков (отв. ред.) [и др.]. – Минск, 2016. – С. 61–64.

44. Загорулько, Г. Б. Подход к интеграции разнородных методов поддержки принятия решений для сложных задач / Г. Б. Загорулько, Ю. А. Загорулько // Открытые семантические технологии проектирования интеллектуальных систем (OSTIS-2013) : материалы III Междунар. науч.-техн. конф., Минск, 16–18 февр. 2013 г. / Белорус. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков (отв. ред.) [и др.]. – Минск, 2013. – С. 265–268.

45. Единая онтологическая платформа интеллектуального анализа данных / А. А. Филиппов [и др.] // Открытые семантические технологии проектирования интеллектуальных систем (OSTIS-2016) : материалы VI Междунар. науч.-техн. конф., Минск, 18–20 февр. 2016 г. / Белорус. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков (отв. ред.) [и др.]. – Минск, 2016. – С. 77–82.

46. Борисов, А. Н. Построение интеллектуальных систем, основанных на знаниях, с повторным использованием компонентов / А. Н. Борисов // Открытые семантические технологии проектирования интеллектуальных систем (OSTIS-2014) : материалы IV Междунар. науч.-техн. конф., Минск, 20–22 февр. 2014 г. / Белорус. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков (отв. ред.) [и др.]. – Минск, 2014. – С. 97–102.

47. Dutta, S. Knowledge processing and applied artificial intelligence / S. Dutta. – Oxford : Butterworth-Heinemann, 1993. – 353 p.

48. Pau, L. F. Knowledge-based processing / L. F. Pau // Computer vision for electronics manufacturing. – Boston, 1990. – P. 127–131.

49. Голенков, В. В. Графодинамические модели и методы параллельной асинхронной переработки информации в интеллектуальных системах : дис. ... д-ра техн. наук : 05.13.11 ; 05.13.17 / В. В. Голенков. – Минск, 1996. – 396 л.

50. Golenkov, V. V. Ontology-based design of intelligent systems / V. V. Golenkov // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2017) : материалы VII Междунар. науч.-техн. конф., Минск, 16–18 февр. 2017 г. / Белорус. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков (отв. ред.) [и др.]. – Минск : БГУИР, 2017. – Вып. 1. – С. 37–56.

51. Голенков, В. В. Проект открытой семантической технологии компонентного проектирования интеллектуальных систем. В 2 ч. Ч. 1: Принципы создания / В. В. Голенков, Н. А. Гулякина // Онтология проектирования. – 2014. – №1. – С. 42–64.

52. Средства структуризации семантических моделей баз знаний / И. Т. Давыденко [и др.] // Открытые семантические технологии проектирования

интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2016) : материалы VI Междунар. науч.-техн. конф., Минск, 18–20 февр. 2016 г. / Белорус. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков (отв. ред.) [и др.]. – Минск : БГУИР, 2016. – С. 93–106.

53. Davydenko, I. T. Ontology-based knowledge base design / I. T. Davydenko // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2017) : материалы VII Междунар. науч.-техн. конф., Минск, 16–18 февр. 2017 г. / Белорус. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков (отв. ред.) [и др.]. – Минск : БГУИР, 2017. – Вып. 1. – С. 57–72.

54. Корончик, Д. Н. Реализация хранилища унифицированных семантических сетей / Д. Н. Корончик // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2013) : материалы III Междунар. науч.-техн. конф., Минск, 21–23 февр. 2013 г. / Белорус. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков (отв. ред.) [и др.]. – Минск : БГУИР, 2013. – С. 125–129.

55. Голенков, В. В. Графодинамические модели параллельной обработки знаний: принципы построения, реализации и проектирования / В. В. Голенков, Н. А. Гулякина // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2012) : материалы II Междунар. науч.-техн. конф., Минск, 16–18 февр. 2012 г. / Белорус. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков (отв. ред.) [и др.]. – Минск : БГУИР, 2012. – С. 23–52.

56. Метасистема IMS.ostis [Электронный ресурс]. – Режим доступа: http://ims.ostis.net. – Дата доступа: 02.04.2021.

57. Davydenko, I. Semantic models, method and tools of knowledge bases coordinated development based on reusable components / I. Davydenko // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2018) : материалы VIII Междунар. науч.-техн. конф., Минск, 15–17 февр. 2018 г. / Белорус. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков (гл. ред.) [и др.]. – Минск : БГУИР, 2018. – С. 99–118.

58. Воеводин, В. В. Параллельные вычисления / В. В. Воеводин. – СПб. : БХВ Петербург, 2002. – 609 с.

59. Барский, А. Б. Параллельные процессы в вычислительных системах: планирование и организация / А. Б. Барский. – М. : Радио и связь, 1990. – 256 с.

60. Головкин, Б. А. Параллельные вычислительные системы / Б. А. Головкин. – М. : Наука, 1980. — 520 с.

61. Городняя, Л. В. О парадигме параллельного программирования / Л. В. Городняя // Информ. и мат. технологии в науке и упр. – 2016. – №1. – С. 136–139.

62. Борщёв, В. Б. Вегетативная машина / В. Б. Борщёв // Программирование. – 1989. – №5. – С. 16–28.

63. Котов, В. Е. Асинхронные вычислительные процессы над общей памятью / В. Е. Котов, А. С. Нариньяни // Кибернетика. – 1966. – №3. – С. 64–71.

64. Мартынов, В. В. Семиологические основы информатики / В. В. Мартынов. – Минск : Наука и техника, 1974. – 192 с.

65. Мартынов, В. В. Универсальный семантический код: Грамматика. Словарь. Тексты / В. В. Мартынов. – Минск : Наука и техника, 1977. – 189 с.

66. Мартынов, В. В. Универсальный семантический код: УСК-3 / В. В. Мартынов ; под ред. А. Е. Михневича. – Минск : Наука и техника, 1984. – 132 с.

67. Boyko, I. Semantic classification of actions for knowledge inference / I. Boyko // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2016) : материалы VI Междунар. науч.-техн. конф., Минск, 18–20 февр. 2016 г. / Белорус. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков (отв. ред.) [и др.]. – Минск : БГУИР, 2016. – С. 115–124.

68. Шенк, Р. Обработка концептуальной информации / Р. Шенк ; под ред. В. М. Брябрина ; пер. с англ. – М. : Энергия, 1980. – 361 с.

69. Wooldridge, M. An introduction to multiagent systems / M. Wooldridge. – 2nd ed. – Chichester : J. Wiley, 2009. – 484 p.

70. Russel, S. Artificial intelligence: a modern approach / S. Russel, P. Norvig. – 3rd ed. – Upper Saddle River : Prentice Hall, 2010. – 1150 p.

71. Тарасов, В. Б. От многоагентных систем к интеллектуальным организациям: философия, психология, информатика / В. Б. Тарасов. – М. : Эдиториал УРСС, 2002. – 348 с.

72. Городецкий, В. И. Многоагентные системы (обзор) / В. И. Городецкий, М. С. Грушинский, А. В. Хабалов // Новости искусств. интеллекта. – 1998. – №2. – С. 64–116.

73. Razvan, V. F. Autonomous artificial intelligent agents : techn. rep. Coneural-03-01, 4 Febr. 2003 / V. F. Razvan. – Cluj-Napoca : Center for Cognitive a. Neural Studies, 2003. – 50 p.

74. Autonomous agents and multi-agent systems [Electronic resource]. – Mode of access: http://www.springer.com/computer/ai/journal/10458. – Date of access: 09.04.2021.

75. Давыдов, А. А. О компьютационной теории социальных агентов / А. А. Давыдов // Социс. – 2006. – №2. – С. 19–28.

76. Скрипкин, С. К. Концептуальное моделирование энергетических систем на основе интеграции агентно-базированных платформ / С. К. Скрипкин // Информ. и мат. технологии в науке и упр. – 2016. – №1. – С. 123–131.

77. Замятина, Е. Б. Опыт разработки системы имитации, основанной на агентах / Е. Б. Замятина, Д. Ф. Каримов, А. А. Митраков // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2015) : материалы V Междунар. науч.-техн. конф., Минск, 19–21 февр. 2015 г. / Белорус. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков (отв. ред.) [и др.]. – Минск : БГУИР, 2015. – С. 597–604.

78. AgentBuilder [Electronic resource]. – Mode of access: http://www.agentbuilder.com. – Date of access: 08.04.2021.

79. Eve – a web-based agent platform [Electronic resource]. – Mode of access: http://eve.almende.com/index.html. – Date of access: 16.04.2021.

80. GAMA Platform [Electronic resource]. – Mode of access: http://gama-platform.org. – Date of access: 18.04.2021.

81. Macal, C. M. Tutorial on agent-based modeling and simulation / C. M. Macal, M. J. North // WSC'05 : proc. of the 2005 Winter Simulation Conf., Orlando, 4–7 Dec. 2005 / Assoc. for Computing Machinery. – New York, 2005. – P. 2–15.

82. Requirements analysis of agent-based simulation platforms: state of the art and new prospects / M. Marietto [et al.] // Multi-agent-based simulation : third intern. workshop, Bologna, 15–16 July 2002 / ed.: J. S. Sichman, F. Bousquet, P. Davidsson. – Bologna, 2002. – P. 125–141.

83. Агентно-ориентированная технология проектирования / В. А. Лещёв [и др.] // Програм. продукты и системы. – 2006. – № 1. – С. 23–29.

84. Bordini, R. H. Programming multi-agent systems in AgentSpeak using Jason / R. H. Bordini, J. F. Hubner, M. J. Wooldridge. – Chichester : Wiley, 2007. – 294 p.

85. Castillo, O. Recent advances on hybrid intelligent systems / O. Castillo, P. Melin, J. Kacprzyk. – Berlin : Springer, 2014. – 572 p.

86. GOAL [Electronic resource] // Atlassian. – Mode of access: https://goalapl.atlassian.net/wiki/spaces/GOAL/overview. – Date of access: 23.04.2021.

87. Implementing Industrial Multi-agent Systems using JACK / R. Evertsz [et al.] // Programming multi-agent systems : first Intern. workshop, PROMAS 2003, Melbourne, 15 July, 2003 : sel. rev. a. invited papers / ed.: M. M. Dastani, J. Dix, A. E. Fallah-Seghrouchni. – Berlin, 2004. – P. 18–48.

88. JAVA Agent DEvelopment Framework [Electronic resource]. – Mode of access: http://jade.tilab.com. – Date of access: 17.04.2021.

89. Multi-agent oriented programming with JaCaMo / O. Boissier [et al.] // Science of Computer Programming. – 2013. – Vol. 78, № 6. – P. 747–761.

90. Omicini, A. Coordination for internet application development / A. Omicini, F. Zambonelli // Autonomous Agents a. Multi-Agent Systems. – 1999. – Vol. 2, №2. – P. 251–269.

91. Jagannathan, V. Blackboard architectures and applications / V. Jagannathan, K. Dodhiawala, L. Baum. – New York : Academic Press, 1989. – 556 p.

92. Шункевич, Д. В. Взаимодействие асинхронных параллельных процессов обработки знаний в общей семантической памяти / Д. В. Шункевич // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2016) : материалы VI Междунар. науч.-техн. конф., Минск, 18–20 февр. 2016 г. / редкол.: В. В. Голенков (отв. ред.) [и др.]. – Минск : БГУИР, 2016. – С. 137–144.

93. Дейкстра, Э. Языки программирования : учеб. пособие / Э. Дейкстра / ред. Ф. Женюи ; под ред. В. М. Курочкина ; пер. с англ. В. П. Кузнецова. – М. : Мир, 1972. – С. 9–86.

94. Хоар, Ч. Взаимодействующие последовательные процессы / Ч. Хоар ; под ред. А. П. Ершова ; пер. с англ. – М. : Мир, 1989. – 264 с.

95. Давыденко, И. Т. Семантическая модель коллективного проектирования баз знаний / И. Т. Давыденко // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2016) : материалы VI Междунар. науч.-техн. конф., Минск, 18–20 февр. 2016 г. / Белорус. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков (отв. ред.) [и др.]. – Минск : БГУИР, 2016. – С. 107–114.

96. The sc-web enchancement [Electronic resource] // GitHub. – Mode of access: https://github.com/ostis-apps/sc-web. – Date of access: 19.04.2021.

97. Software implementation of semantic network storage and processing [Electronic resource] // GitHub. – Mode of access: https://github.com/ostis-dev/sc-machine. – Date of access: 22.04.2021.

98. Store for Intelligent systems developed on OSTIS(ims.ostis.net) [Electronic resource] // GitHub. – Mode of access: https://github.com/ostis-apps. – Date of access: 24.04.2021.

99. Golenkov, V. V. Ontology-based Design of Batch Manufacturing Enterprises / Golenkov V. V. and other // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2017) : материалы междунар. науч.-техн. конф., Минск, 16–18 февраля 2017 года / Белорус. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков (отв. ред.) [и др.]. – Минск : БГУИР, 2017. – С. 265 – 280.

100. Design of batch manufacturing enterprises in the context of Industry 4.0 / V. Taberko and others // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2018) : материалы междунар. науч.-техн. конф., Минск, 15–17 февраля 2018 года / Белорус. гос. ун-т информатики и радиоэлектроники ; редкол. : В. В. Голенков (отв. ред.) [и др.]. – Минск : БГУИР, 2018. – С. 307 – 320.

101. Taberko, V. Design Principles of Integrated Information Services for Batch Manufacturing Enterprise Employees / V. Taberko [et al.] // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2019) : материалы междунар. науч.-техн. конф., Минск, 21–23 февраля 2019 года / Белорус. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков (гл. ред.) [и др.]. – Минск : БГУИР, 2019. – С. 215–224.

102. Гулякина, Н. А. Языки и технологии программирования, ориентированные на обработку семантических сетей / Н. А. Гулякина, О. В. Пивоварчик, Д. А. Лазуркин // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2012): материалы II Междунар. науч.-техн. конф., Минск, 16–18 февр. 2012 г. / Белорус. гос. ун-т информатики и радиоэлектроники ; редкол.: В. В. Голенков (отв. ред.) [и др.]. – Минск : БГУИР, 2012. – С. 221–228.

Св. план 2020, поз. 9

Учебное издание

Шункевич Даниил Вячеславович

СЕМАНТИЧЕСКИЕ ТЕХНОЛОГИИ ПРОЕКТИРОВАНИЯ РЕШАТЕЛЕЙ ЗАДАЧ

УЧЕБНОЕ ПОСОБИЕ

Редактор *Е. С. Юрец*
Корректор *Е. Н. Батурчик*
Компьютерная правка, оригинал-макет *В. М. Задоля*

Подписано в печать 08.06.2022. Формат 60×84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 12,9. Уч.-изд. л. 14,6. Тираж 30 экз. Заказ 246.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
Ул. П. Бровки, 6, 220013, г. Минск