Содержание

Перечень сокращений ... 4

Терминология ... 4

1. Введение ... 5

2. Предпроектное исследование ... 7

2.1. Основные положения языка РДО ... 7

2.2. Поиск решения на графе пространства состояний ... 7

2.3. Программный комплекс RAO-XT ... 9

2.4. Постановка задачи ... 9

3. Формирование ТЗ ... 11

3.1. Общие сведения ... 11

3.2. Назначение разработки ... 11

3.3. Требование к программе или программному изделию ... 11

3.3.1. Требования к функциональным характеристикам ... 11

3.3.2. Требования к надежности ... 11

3.3.3. Условия эксплуатации ... 12

3.3.4. Требование к составу и параметрам технических средств ... 12

3.3.5. Требование к информационной и программной совместимости ... 12

3.3.6. Требование к маркировке и упаковке ... 12

3.3.7. Требование к транспортированию и хранению ... 12

3.4. Требования к программной документации ... 12

3.5. Стадии и этапы разработки ... 12

3.6. Порядок контроля и приемки ... 13

4. Концептуальный этап проектирования системы ... 14

4.1. Графический интерфейс ... 14

4.2. Отображение статистики ... 14

4.2.1. Вывод статистики по результатам поиска ... 15

4.2.2. Вывод статистики по вершине ... 15

4.3. Вызов окна интерфейса подсистемы ... 16

4.4. Модуль визуализации в системе RAO-XT ... 16

5. Технический этап проектирования ... 18

5.1. Проектирование библиотечной части подсистемы визуализации ... 18

5.1.1. Формирование древовидной структуры ... 18

5.1.2. Чтение записи начала поиска ... 18

5.1.3. Чтение записи раскрытия вершины 19
5.1.4. Чтение записи порождения новой вершины 19
5.1.5. Чтение записи решения 20
5.1.6. Чтение записи завершения поиска 20
5.1.7. Хранение структур деревьев в системе 20
5.2. Проектирование графической части подсистемы визуализации 21
5.2.1. Выбор графической библиотеки 21
5.2.2. Реализация интерфейса пользователя 23
5.2.3. Вызов интерфейса подсистемы из интерфейса трассировщика 24
6. Рабочий этап проектирования 25
6.1. Запись данных бинарной сериализации в древовидную структуру 25
6.2. Отображение интерфейса на экране пользователя 27
6.2.1. Отображение графа в окне интерфейса 27
6.2.2. Отображение статистики 29
6.2.3. Окраска решения 30
6.3. Вызов интерфейса модуля визуализации 31
6.3.1. Определение записи начала поиска 31
6.3.2. Вывод окна интерфейса на экран 32
7. Апробирование разработанной подсистемы в модельных условиях 33
7.1. Методика тестирования 33
7.2. Результаты тестирования 33
8. Заключение 34
Список использованных источников 35
Список использованного программного обеспечения 35
Приложение А. Исходный код модели, использованной для тестирования модуля 36

Перечень сокращений

ИМ – Имитационное Моделирование

СДС – Сложная Дискретная Система

IDE – Integrated Development Environment (Интегрированная Среда Разработки)

GUI – Graphic User Interface (Графический Интерфейс Пользователя)

Терминология

Плагин — независимо компилируемый программный модуль, динамически подключаемый к основной программе и предназначенный для расширения и/или использования её возможностей.

Трассировка — получение информационных сообщений о работе приложения во время выполнения.

Сериализация — процесс перевода какой-либо структуры данных в последовательность битов.

Слушатель — механизм, позволяющий экземпляру какого-либо класса получать оповещения от других объектов об изменении их состояния и задавать алгоритм реакции системы на это изменение.

1. Введение

Имитационное моделирование (ИМ)[1] на ЭВМ находит широкое применение при исследовании и управлении сложными дискретными системами (СДС) и процессами, в них протекающими. К таким системам можно отнести экономические и производственные объекты, морские порты, аэропорты, комплексы перекачки нефти и газа, ирригационные системы, программное обеспечение сложных систем управления, вычислительные сети и многие другие. Широкое использование ИМ объясняется тем, что размерность решаемых задач и неформализуемость сложных систем не позволяют использовать строгие методы оптимизации. Эти классы задач определяются тем, что при их решении необходимо одновременно учитывать факторы неопределенности, динамическую взаимную обусловленность текущих решений и последующих событий, комплексную взаимозависимость между управляемыми переменными исследуемой системы, а часто и строго дискретную и четко определенную последовательность интервалов времени. Указанные особенности свойственны всем сложным системам.

Проведение имитационного эксперимента позволяет:

1. Сделать выводы о поведении СДС и ее особенностях:
   - без ее построения, если это проектируемая система;
   - без вмешательства в ее функционирование, если это действующая система, проведение экспериментов над которой или слишком дорого, или небезопасно;
   - без ее разрушения, если цель эксперимента состоит в определении пределов воздействия на систему.
2. Синтезировать и исследовать стратегии управления.
3. Прогнозировать и планировать функционирование системы в будущем.
4. Обучать и тренировать управленческий персонал и т.д.

ИМ является эффективным, но и не лишенным недостатков, методом. Трудности использования ИМ, связаны с обеспечением адекватности описания системы, интерпретацией результатов, обеспечением стохастической сходимости процесса моделирования, решением проблемы размерности и т.п. К проблемам применения ИМ следует отнести также и большую трудоемкость данного метода.

Интеллектуальное ИМ, характеризующиеся возможностью использования методов искусственного интеллекта и прежде всего знаний, при принятии решений в процессе имитации, при управлении имитационным экспериментом, при реализации интерфейса пользователя, создании информационных банков ИМ, использовании нечетких данных, снимает часть проблем использования ИМ.

Разработка интеллектуальной среды имитационного моделирования РДО выполнена в Московском государственном техническом университете (МГТУ им. Н. Э. Баумана) на кафедре "Компьютерные системы автоматизации производства". Причинами ее проведения и создания РДО явились требования универсальности ИМ относительно классов моделируемых систем и процессов, легкости модификации моделей,

моделирования сложных систем управления совместно с управляемым объектом (включая использование ИМ в управлении в реальном масштабе времени) и ряд других, сформировавшихся у разработчиков при выполнении работ, связанных с системным анализом и организационным управлением сложными системами различной природы.

2. Предпроектное исследование

2.1. Основные положения языка РДО

Основные положения системы РДО могут быть сформулированы следующим образом[3]:

- Все элементы СДС представлены как ресурсы, описываемые некоторыми параметрами. Ресурсы могут быть разбиты на несколько типов; каждый ресурс определенного типа описывается одними и теми же параметрами.
- Состояние ресурса определяется вектором значений всех его параметров; состояние СДС - значением всех параметров всех ресурсов.
- Процесс, протекающий в СДС, описывается как последовательность целенаправленных действий и нерегулярных событий, изменяющих определенным образом состояние ресурсов; действия ограничены во времени двумя событиями: событиями начала и событиями конца.
- Нерегулярные события описывают изменения состояния СДС, непредсказуемые в рамках продукционной модели системы (влияние внешних по отношению к СДС факторов либо факторов, внутренних по отношению к ресурсам СДС). Моменты наступления нерегулярных событий случайны.
- Действия описываются операциями, которые представляют собой модифицированные продукционные правила, учитывающие временные связи. Операция описывает предусловия, которым должно удовлетворять состояние участвующих в операции ресурсов, и правила изменения состояния ресурсов в начале и в конце соответствующего действия.
- Множество ресурсов R и множество операций O образуют модель СДС.

2.2. Поиск решения на графе пространства состояний

В языке имитационного моделирования РДО помимо возможности использования для описания законов управления формализмов продукционных правил введены так называемые точки принятия решений, позволяющие осуществлять оптимальное управление[2].

Механизм точек принятия решений в языке имитационного моделирования РДО позволяет гибко сочетать имитацию с оптимизацией. Для этого используется поиск на графе состояний.

Примерами задач, которые решаются с использованием точек принятия решений, могут служить:

- Различные транспортные задачи (например, выбор последовательности объезда пунктов транспортным средством при минимуме пройденного пути, времени или стоимости).
- Задачи укладки грузов при минимизации занимаемого ими объема (в более общем случае – задачи размещения).
- Нахождение решения логических задач за минимальное число ходов (например, расстановка фишек в игре «Пятнашки»).

- Задачи теории расписаний (например, задачи определения последовательности обработки различных деталей на станках при минимизации времени обработки, либо обслуживания клиентов с минимумом отклонений от запланированного времени изготовления заказов и т. д.).

Граф — основной объект изучения математической теории графов, совокупность непустого множества вершин и наборов пар вершин (связей между вершинами).

Граф $G = (S, E)$ задается двумя множествами:

- $S = \{s_i\}$ – множество вершин графа. Каждой вершине s_i ставится в соответствие состояние системы (база данных);
- $E = \{e_{ij}(s_i, s_j : s_i \in S, s_j \in S)\}$ – множество дуг. Каждой дуге e_{ij}, принадлежащей E и соединяющей пару вершин, ставится в соответствие правило преобразования (продукционное правило). Если дуга направлена от вершины s_i к вершине s_j, то s_i в данном случае будет являться вершиной-родителем, а s_j – вершиной-потомком (преемником).

Маршрутом в графе называют конечную последовательность вершин, в которой каждая вершина (кроме последней) соединена со следующей в последовательности вершиной ребром.

Цепью называется маршрут без повторяющихся рёбер.

Циклом называют цепь, в которой первая и последняя вершины совпадают.

Граф называется связным, если для любых вершин u, v существует путь из u в v.

Граф называется деревом, если он связный и не содержит нетривиальных циклов.

В графах, представляющих интерес для поиска, у каждой вершины должно быть конечное число вершин-преемников. С дугой может быть связана некоторая величина c_{ij} – стоимость дуги, она отражает затраты (в смысле заданного критерия оптимизации) применения соответствующего правила.

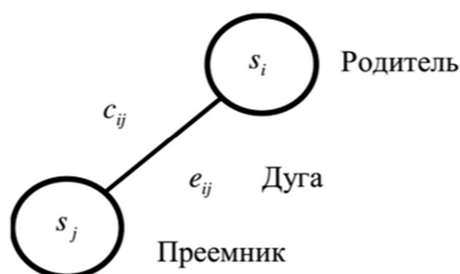

Рис 1. Простейший граф состояний

В графе имеются две особые вершины:

- s_0 – начальная вершина, или другими словами, вершина, представляющая собой исходную базу данных;
- s_t – целевая вершина, иначе – вершина, представляющая собой базу данных, удовлетворяющую терминальному условию поиска. Таких вершин может быть не одна, а множество, и такое множество будет называться целевым множеством.

Путь в графе – последовательность вершин, в которой каждая последующая является преемником:

$$m = \{s_n, s_{n+1}, s_{n+2}, \dots, s_{n+k}\},$$

где $k + 1$ – длина пути.

Каждому пути m ставится в соответствие его стоимость, которая равна сумме стоимостей применения правил по всему пути на графе G[2].

2.3. Программный комплекс RAO-XT

Программный комплекс RAO-XT предназначен для разработки и отладки имитационных моделей на языке РДО. Основные цели данного комплекса - обеспечение пользователя легким в обращении, но достаточно мощным средством разработки текстов моделей на языке РДО, обладающим большинством функций по работе с текстами программ, характерных для сред программирования, а также средствами проведения и обработки результатов имитационных экспериментов.

Система имитационного моделирования RAO-XT представляет собой плагин для интегрированной среды разработки Eclipse – свободной интегрированной среды разработки модульных кроссплатформенных приложений. Система написана на языке Java и состоит из трех основных компонентов:

- rdo – компонент, производящий преобразование кода на языке РДО в код на языке Java.
- rdo.lib – библиотека системы. Этот компонент реализует ядро системы имитационного моделирования.
- rdo.ui – компонент, реализующий графический интерфейс системы с помощью библиотеки SWT.

На момент начала выполнения курсового проекта, система не имела возможности выводить на экран пользователя граф пространства состояний для моделей, содержащих точки принятия решений. Вывод информации о поиске осуществлялся в текстовом формате в графическом окне модуля трассировки.

2.4. Постановка задачи

Проектирование любой системы начинается с выявления проблемы, для которой она создается. Под проблемой понимается несовпадение характеристик состояния систем, существующей и желаемой.

В результате предпроектного исследования было выявлено отсутствие в программном комплексе RAO-XT подсистемы графической визуализации поиска решения на графе пространства состояний.

Разработка и внедрение такой подсистемы позволит расширить функционал среды RAO-XT и повысить эффективность процесса моделирования.

3. Формирование ТЗ

3.1. Общие сведения

Основание для разработки: задание на курсовой проект.

Заказчик: Кафедра «Компьютерные системы автоматизации производства» МГТУ им. Н. Э. Баумана

Разработчик: студент кафедры «Компьютерные системы автоматизации производства» Стрижов К. А.

Наименование темы разработки: «Проектирование модуля визуализации поиска решения на графе состояний в РДО»

3.2. Назначение разработки

Разработать подсистему визуализации поиска решений на графе состояний и добиться ее интеграции в систему моделирования RAO-XT.

3.3. Требование к программе или программному изделию

3.3.1. Требования к функциональным характеристикам

Модуль визуализации должен удовлетворять следующим требованиям:

- Корректно отображать граф пространства состояний для соответствующей точки принятия решений типа search;
- Вызов графического окна модуля должен осуществляться из пользовательского интерфейса модуля трассировки системы RAO-XT;
- Графическое окно должно содержать всю необходимую информацию о точке принятия решений, как-то:
    - Название точки принятия решений;
    - Статистика по поиску на графе состояний;
    - Подробная информация по выделенной вершине;
    - Выделение цветом вершин графа, относящихся к решению;
- Иметь возможность отображать несколько графов в случае наличия в модели нескольких точек принятия решений.

3.3.2. Требования к надежности

Основное требование к надежности направлено на поддержание в исправном и работоспособном состоянии ЭВМ, на которой происходит использование программного комплекса RAO-XT.

Сохранение работоспособности системы при отказе по любым причинам подсистемы или ее части.

3.3.3. Условия эксплуатации

Эксплуатация должна производиться на оборудовании, отвечающем требованиями к составу и параметрам технических средств, и с применением программных средств, отвечающим требованиям к программной совместимости.

Аппаратные средства должны эксплуатироваться в помещениях с выделенной розеточной электросетью 220В ±10%, 50 Гц с защитным заземлением.

3.3.4. Требование к составу и параметрам технических средств

Программный продукт должен работать на компьютерах со следующими характеристиками:

- объем ОЗУ не менее 2 Гб;
- объем жесткого диска не менее 50 Гб;
- микропроцессор с тактовой частотой не менее 1ГГц;
- монитор с разрешением от 1280*1024 и выше.

3.3.5. Требование к информационной и программной совместимости

Система должна работать под управлением следующих ОС:

- Windows 7, 8;
- Ubuntu 14.10.

3.3.6. Требование к маркировке и упаковке

Требования к маркировке и упаковке не предъявляются.

3.3.7. Требование к транспортированию и хранению

Требования к транспортированию и хранению не предъявляются.

3.4. Требования к программной документации

Требования к программной документации не предъявляются.

3.5. Стадии и этапы разработки

Разработка должна быть проведена в три стадии:

- техническое задание
- технический и рабочий проекты
- внедрение

На стадии «Техническое задание» должен быть выполнен этап разработки и согласования настоящего технического задания.

На стадии «Технический и рабочий проект» должны быть выполнены перечисленные ниже этапы работ:

- разработка программы

- разработка методики тестирования
- испытания программы

На стадии «Внедрение» должен быть выполнен этап разработки «Подготовка и передача программы».

3.6. Порядок контроля и приемки

Контроль и приемка работоспособности системы осуществляются с помощью следующих методов:

- Опрос экспертов. Используется для оценки дизайна, качества и удобства использования новой системы графического вывода;
- Многократное ручное тестирование с помощью имитационных моделей, написанных на языке РДО.

4. Концептуальный этап проектирования системы

На концептуальном этапе проектирования требовалось:

- разработать графический интерфейс окна вывода графа на экран;
- определить перечень необходимой информации, которую должна содержать выводимая на экран статистика;
- определить, как должен быть организован вызов окна графа из интерфейса модуля трассировки;
- разработать общую структуру подсистемы и схему его взаимодействия с другими компонентами системы RAO-XT.

4.1. Графический интерфейс

Разрабатываемая подсистема должна предоставлять пользователю графический интерфейс, содержащий в себе построенный граф пространства состояний и собранную по результатам поиска статистику. На данном этапе проектирования были приняты следующие решения:

- большую часть пространства вызываемого окна должен занимать построенный граф пространства состояний, поскольку он является главным предметом реализации интерфейса;
- построенный граф должен содержать минимум необходимой информации, как-то:
    - номера построенных вершин – они отображают последовательность построения графа пространства состояний;
    - информацию о решении (согласно требованиям, указанным в п. 3.3.1);
- вывод статистики должен быть компактным и не отнимать пространство у отображенного графа.

Предполагается, что изначально интерфейс предоставляет пользователю информацию по графу состояний, и на экране должен быть отображен граф и собранная по нему статистика. Однако пользователю может потребоваться дополнительная информация по любой из отображенных вершин. Исходя из этого, было принято:

- при вызове интерфейса выводить на экран только граф и статистику по нему;
- при необходимости отображать информацию по интересующей вершине по клику мыши.

Для удобства использования было также решено именовать окна интерфейса согласно тем точкам принятия решений, для которых они были вызваны.

4.2. Отображение статистики

Интерфейс подсистемы реализует отображение на экране статистики двух типов:

- статистика по результатам поиска на графе пространства состояний;

- статистика по выделенной вершине графа, т. е. статистики для отдельного состояния.

Для составления перечня статистических данных, которые необходимо отобразить в интерфейсе разрабатываемой подсистемы, необходимо учесть, что в программном комплексе RAO-XT уже существует модуль трассировки, отображающий различную информацию о работе системы и результатах моделирования. Назначение разрабатываемой подсистемы – устранить недостаток текстового формата вывода трассировщика, т. е. визуализировать поиск решения на графе пространства состояний с отображением его древовидной структуры, и не потерять при этом в информативности.

4.2.1. Вывод статистики по результатам поиска

Всю статистику по работе модели, включая данные, относящиеся к алгоритму поиска, пользователь может получить из окна трассировщика, поэтому во избежание лишнего дублирования информации принято решение в разрабатываемом интерфейсе отображать только ту статистику, которая относится непосредственно к результатам поиска:

- стоимость решения – фактическая стоимость найденного пути от исходной вершины до целевой;
- количество раскрытых вершин – количество вершин, для которых были порождены потомки;
- количество вершин в графе – количество вершин, добавленных в граф в процессе поиска.

Перечисленный список является необходимым минимумом, который должен быть представлен пользователю разрабатываемой подсистемой.

4.2.2. Вывод статистики по вершине

Необходимая информация по результатам поиска представлена совокупностью информации, предоставляемой графом пространства состояний, и статистикой по нему. Если же пользователю требуется дополнительная информация по какой-либо вершине, он сможет получить ее, выделив ее кликом. Это сделано для того, чтобы исключить из действий пользователя операцию обратного переключения к интерфейсу модуля трассировщика, что позволит повысить эффективность работы и сэкономить время.

Модуль трассировки предоставляет следующую информацию при порождении новых вершин:

- номер самой вершины;
- номер ее родителя;
- значение фактической стоимости пути от исходной вершины до текущей;
- значение оценочной стоимости пути от текущей вершины до целевой;
- имя использованной активности с указанием в скобках имен релевантных ресурсов;

- значение стоимости применения правила.

Все перечисленные данные необходимо вывести на экран пользователя.

4.3. Вызов окна интерфейса подсистемы

На этапе формирования ТЗ в требованиях к функционалу подсистемы было указано, что вызов ее интерфейса должен осуществляться из интерфейса модуля трассировки. Это требование предъявлено к подсистеме по следующим причинам:

- модуль трассировки является основным средством вывода информации о состоянии и работе системы RAO-XT;
- в общем случае модель может содержать несколько точек принятия решений типа search.

Ситуация, когда при наличии в системе точек принятия решений типа search, статистика по ним не выведена в полотно трассировки, может возникнуть только в случае, когда пользователь вручную отключил вывод трассировки точек принятия решений, и, следовательно, не интересуется этой информацией. В любой другой ситуации пользователь будет иметь перед глазами записи, относящиеся к поиску на графе пространства состояний. К ним целесообразнее всего привязать вызов разрабатываемого модуля.

Ситуация с несколькими точкам принятия решений в модели так же является веским доводом против других вариантов вызова окна интерфейса (например, отдельной кнопки в меню), т.к. возникнет очевидная неоднозначность, граф какой точки хочет увидеть пользователь. Вызов окна интерфейса с привязкой к строке трассировщика исключит такую ситуацию, потому что каждая запись в полотне трассировки относится к соответствующей ей точке принятия решений.

4.4. Модуль визуализации в системе RAO-XT

На этапе концептуального проектирования ключевой задачей является разработка правильной схемы взаимодействия подсистемы визуализации с остальными частями системы RAO-XT. Ниже приведены основные требования к модулю визуализации как к компоненту системы:

- Сохранение информации о состоянии системы в каждый момент модельного времени производится компонентом *Database* в бинарном формате. Таким образом компонент *Database* осуществляет сериализацию данных, т.е. сохранение их в компактном, но недоступном для восприятия пользователем формате. Модуль визуализации должен уметь интерпретировать эти данные и формировать на выходе древовидную структуру, по которой можно будет восстановить и построить граф пространства состояний и затем вывести его на экран пользователя.
- Модуль визуализации должен предоставлять выходные данные исключительно для пользователя. Никакие другие компоненты системы не должны основывать свою работу на выходных данных модуля визуализации.

- Модуль визуализации должен состоять из двух частей:
    - библиотечная часть;
    - графическая часть;
- Библиотечная часть модуля визуализации должна заниматься исключительно преобразованием бинарных данных в древовидную структуру. Далее с полученной структурой работает графическая часть модуля, расположенная в пакете *UI*. Эта часть отвечает за отрисовку графа на экране и за графический интерфейс пользователя. При этом данные, которые отображает графическая часть модуля, должны полностью соответствовать тем данным, которые на данный момент преобразовала библиотечная часть.
- Разрабатываемый модуль должен быть активен только после прогона модели. Модуль выводит результат на экран, основываясь на данных, сформированных только после окончания прогона модели.

5. Технический этап проектирования

5.1. Проектирование библиотечной части подсистемы визуализации

5.1.1. Формирование древовидной структуры

Библиотечная часть подсистемы визуализации, основным компонентом которой является класс *TreeBuilder*, должна взаимодействовать непосредственно с содержимым базы данных и формировать на его основе древовидную структуру графа. База содержит в себе записи с сериализованными данными о результатах работы системы, результатах моделирования и поиска. Типов записей в базе данных пять:

- Системные. Это записи, содержащие информацию о произошедших системных событиях;
- Записи ресурсов. Это записи, содержащие информацию об изменении состояния ресурсов модели;
- Записи образцов. Это записи, содержащие информацию о выполнившихся активностях и событиях;
- Записи поиска по графу. Это записи, содержащие информацию о порядке выполнения поиска по графу и найденном решении;
- Записи результатов. Это записи, содержащие информацию об изменении текущего значения результата.

Для разрабатываемой подсистемы интерес представляет только один тип записи, относящийся к поиску решения на графе. Сама запись поиска также бывает нескольких типов:

- начала поиска;
- раскрытия вершины;
- порождения новой вершины;
- решения;
- завершения поиска.

Зная правила сериализации данных для записи каждого типа, модуль должен считать нужное количество байт из записи, преобразовать их в переменную корректного типа и записать эту переменную в структуру. Для описания вершины графа описан вложенный по отношению к основному классу *TreeBuilder* класс *Node*. Экземпляр такого класса должен полностью охарактеризовывать конкретную вершину графа.

После чтения всех записей поиска, находящихся в базе данных, классом *TreeBuilder* будет сформирована полная структура дерева графа.

5.1.2. Чтение записи начала поиска

Встреча записи начала поиска в базе данных означает, что система осуществила поиск для определенной точки принятия решений типа search, описанной в тексте модели. В соответствии с этим библиотечная часть модуля визуализации должна отреагировать действием начала формирования дерева нового графа. Для этого необходимо:

- считать из записи номер соответствующей точки принятия решений;
- создать экземпляр класса *Node* с параметрами, соответствующими корневой вершине дерева, и добавить его в структуру дерева графа.

Корневая вершина обладает нулевым индексом (нумерация вершин в системе RAO-XT начинается с нуля, в отличие от системы RAO-Studio, где она начиналась с единицы) и не имеет предка.

5.1.3. Чтение записи раскрытия вершины

Запись раскрытия вершины содержит информацию о номере вершины, для которой будут порождены потомки, на определенном шаге поиска, и эта информация никак не интерпретируется разрабатываемым модулем. Записи данного типа им рассматриваться не должны.

5.1.4. Чтение записи порождения новой вершины

Запись порождения новой вершины бывает трех типов:

- порождение новой вершины. Вершина с таким состоянием системы не содержится в уже построенной части графа;
- порождение лучшей вершины. Вершина с таким состоянием уже есть в графе, и она перезаписывается, поскольку вновь найденный путь имеет меньшую стоимость;
- порождение худшей вершины. Вершина с таким состоянием уже есть и она не включается в граф (вновь найденный путь имеет большую стоимость).

Вершины, описываемые первыми двумя типами записей, добавляются на граф состояний. Обновление стоимостей при нахождении лучей вершины происходит внутри класса *DecisionPointSearch* библиотеки RAO-XT, поэтому внутри записей уже содержится вся необходимая информация, и операции, которые должен выполнить модуль при встрече записей этих типов, не отличаются. Модуль визуализации должен выполнить для них следующие действия:

- считать номер вершины и ее предка;
- считать стоимости g и h;
- определить имя активности;
- считать стоимость применения правила;
- создать экземпляр класса *Node* и добавить его в структуру графа.

Следует учитывать, что вершина имеет отношение к графу определенной точки принятия решения, номер которой модуль определил, встретив ранее запись начала поиска.

Вершина, описываемая последним типом записи порождения вершины, не добавляется на граф, а значит запись данного типа не должна рассматриваться разрабатываемым модулем.

5.1.5. Чтение записи решения

Записи данного типа присутствуют в полотне трассировке в случае существования решения и содержат информацию о том, какие правила были применены к исходной вершине графа (исходному состоянию системы) для перехода в целевое состояние. Так же они содержит номера вершин, входящих в решение, что и будет использоваться модулем визуализации.

Прочитав записи данного типа, класс *TreeBuilder* заполнит список, содержащий вершины, принадлежащие решению для текущей точки принятия решения.

5.1.6. Чтение записи завершения поиска

Среди записей данного типа модулем визуализации будут использоваться записи, обладающие следующими признаками:

- успешное завершение поиска. Наличие в базе данных записи такого типа означает, что решение найдено;
- неудачное завершение поиска. Наличие такой записи означает, что решение отсутствует.

Записи, обладающие признаками, отличными от этих, не рассматриваются разрабатываемым модулем, так как построение графа пространства состояний возможно только для перечисленных типов записей.

Запись завершения поиска содержит в себе информацию о результатах поиска на графе для текущей точки принятия решений, следовательно необходимо описание структуры, в которую она будет считана, и которая будет использоваться в дальнейшем для вывода статистики. Для описания такой структуры был описан вложенный по отношению к библиотечному классу *TreeBuilder* класс *GraphInfo*. Класс должен содержать поля для записи в них информации, описанной в п. 4.2.1.

В случае чтения записи об успешном завершении поиска, из нее необходимо считать данные по всем полям класса *GraphInfo*.

В случае чтения записи о неудачном завершении поиска, данные считываются только по двум полям:

- количество открытых вершин;
- количество вершин в графе.

Стоимость решения в таком случае необходимо задать равной нулю.

Данные из записи считываются модулем для текущей точки принятия решений.

5.1.7. Хранение структур деревьев в системе

При наличии нескольких точек принятия решений типа search в модели необходимо сохранять древовидную структуру пространства поиска для каждой из них. Наиболее

удобной реализацией их хранения является контейнер типа «карта» (Map), которая будет хранить пары <ключ, значение> и возвращать структуру нужного дерева по ключу.

В качестве значений ключей будут записаны номера точек принятия решений. Подобный подход реализован ранее в компоненте системы *Database*, где хранится карта номеров точек принятия решений, ключами являются строки типа *String* с их именами.

5.2. Проектирование графической части подсистемы визуализации

5.2.1. Выбор графической библиотеки

Реализация графического интерфейса разрабатываемой подсистемы предполагает использование некой библиотеки, которая предоставит разработчику широкий спектр возможностей как для настройки интерфейса, так и для описания требуемого функционала, описанного в ТЗ (п. 3.3.1.).

На данный момент существует три основных библиотеки для разработки графического интерфейса пользователя (GUI) на языке Java:

- AWT (Abstract Window Toolkit);
- Swing;
- SWT (Standard Widget Toolkit).

Библиотека AWT является первой попыткой разработчиков языка Java компании Sun Microsystems создать встроенную библиотеку для реализации GUI. Методы библиотеки AWT используют нативные элементы операционной системы для создания интерфейса пользователя, в результате чего разрабатываемые GUI похожи на интерфейсы других программ, разработанных под данную платформу, но функционал, обеспечиваемый библиотекой, достаточно прост.

Достоинства библиотеки AWT:

- часть JDK (Java Developer Kit) – нет необходимости подключать дополнительные библиотеки;
- относительное быстродействие;
- графические компоненты похожи на стандартные.

Недостатки библиотеки AWT:

- использование нативных компонентов налагает ограничения на использование их свойств (некоторые компоненты могут вообще не работать на других платформах);
- отсутствие большого числа возможностей, реализованных в других библиотеках;
- стандартных компонентов AWT немного – во многих случаях реализация требуемого функционала требует от разработчика дополнительных усилий.

В настоящее время библиотека AWT используется разработчиками крайне редко.

Библиотека Swing была разработана вслед за AWT компанией Sun Microsystems, и в отличие от AWT, которая использует методы библиотек, написанных на языке C, полностью реализована на языке Java. Набор функциональных компонентов значительно превосходит возможности библиотеки AWT как по многообразию, так и по функционалу.

Достоинства библиотеки Swing:

- часть JDK;
- богатая документация;
- встроенный редактор форм почти для всех сред разработки;
- в Сети доступно большое количество расширений на базе данной библиотеки;
- настраиваемые стили.

Недостатки библиотеки Swing:

- разработка сложного интерфейса может повлечь за собой трудности при работе с менеджером компоновки;
- падение быстродействия при увеличении сложности интерфейса.

Несмотря на существование других перспективных разработок (например, библиотека JavaFX) на сегодняшний день Swing остается самой популярной и широко используемой библиотекой для разработки GUI на языке Java.

Библиотека SWT – разработка компании IBM, задумывалась как альтернатива библиотеке Swing, первые версии которых отличались низкой производительностью. SWT использует нативные компоненты системы аналогично библиотеке AWT, но в отличие от нее для каждой платформы созданы свои интерфейсы взаимодействия. Это повлекло за собой расширение функционала и позволило более полно использовать возможности существующих компонентов платформы, на которой ведется разработка, однако отрицательно сказалось на кросс-платформенности данной библиотеки. Кроме того, при использовании SWT задача освобождения ресурсов возлагается на разработчика.

Достоинства библиотеки SWT:

- высокая производительность за счет использования функционала компонентов системы;
- Eclipse IDE предоставляет визуальный редактор форм данной библиотеки;
- возможно использование AWT- и Swing- компонентов.

Недостатки библиотеки SWT:

- для каждой платформы необходимо подключать отдельную библиотеку;
- важно тщательно следить за использованием ресурсов и их своевременным освобождением;
- сложная архитектура, повышающая порог вхождения для разработчика.

При сравнении трех основных библиотек разработки GUI было выделено два критерия, наиболее важных в рамках поставленных задач:

- кросс-платформенность;
- широкий функционал – для минимизации усилий разработчика.

Как итог, в качестве проектного решения, на основе анализа достоинств и недостатков каждой из библиотек, был сделан выбор в пользу библиотеки Swing и ее расширения JGraphx – графической библиотеки для визуализации графов и диаграмм, предоставляющей как возможности для создания интерфейса пользователя, так и готовый функционал для визуализации структур, сформированных разрабатываемым в рамках проекта модулем. Выбранная библиотека наиболее полно подходит для решения поставленных задач.

5.2.2. Реализация интерфейса пользователя

Для описания объекта, описывающего вызываемое окно интерфейса пользователя, необходимо задать класс, который будет содержаться в пакете rdo.ui.graph системы и описывать графическую часть модуля:

```
public class GraphFrame extends JFrame {
    ...
}
```

Данный класс наследуется от стандартного класса *JFrame* библиотеки Swing, который реализует интерфейс графического окна, выводимого на экран пользователя.

Класс *GraphFrame* должен реализовывать следующие методы:

- построения графа по древовидной структуре, принятой в качестве параметра от библиотечной части разрабатываемой подсистемы;
- окрашивания решения, вызываемого в модуле в случае его наличия и принимающего в качестве параметра список объектов-вершин класса *Node* от библиотечной части;
- отображения статистики по поиску на графе, принимающего в качестве параметра объект класса *GraphInfo* от библиотечной части;
- отображения статистики по вершине, принимающего в качестве параметра вершину, представленную объектом класса библиотеки JGraphx, содержащего в себе объект данных класса *Node.*

Конструктор класса *GraphFrame* должен вызываться из интерфейса трассировщика и выводить на экран графическое окно интерфейса подсистемы визуализации.

5.2.3. Вызов интерфейса подсистемы из интерфейса трассировщика

Графический интерфейс трассировщика описан в классе *RDOTraceView* и представлен объектом класса *TableViewer*, предоставляющим пользователю таблицу строк полотна трассировки. Наиболее простой и удобный способ вызова интерфейса разрабатываемого модуля – его вызов по клику на строку таблицы. Существует несколько типов строк, относящихся к поиску по графу (п. 5.1.1.), и только одна из них содержит информацию о точке принятия решений, для которой необходимо вывести граф (запись начала поиска). Наиболее простым способом вызова окна с графом представляется его вызов по строке начала поиска. Для остальных строк в качестве решения предложено выбирать предшествующие строки из полотна трассировки до тех пор, пока не встретится запись начала поиска. Учитывая этот факт, требуется реализовать следующие методы в классе *RDOTraceView:*

- метод вызова графического окна интерфейса разрабатываемого модуля, принимающий в качестве параметра запись полотна трассировки;
- метод нахождения соответствующей строки начала поиска для произвольной строки другого типа, принимаемой в качестве параметра.

Также для объекта класса TableViewer необходимо задать слушателя событий, реагирующего на двойной щелчок мышью, при помощи которого пользователь будет вызывать интерфейс модуля визуализации.

```
viewer.addDoubleClickListener(new IDoubleClickListener() {

            ...

            });
```

В теле слушателя будут вызываться два приведенных выше метода.

6. Рабочий этап проектирования

На рабочем этапе проектирования были реализованы разработанные на предыдущих этапах концепции и решения.

6.1. Запись данных бинарной сериализации в древовидную структуру

Для перевода данных из записей бинарной сериализации, поставляемой классу *TreeBuilder* библиотекой *Database*, в древовидную структуру графа был описан вложенный по отношению к *TreeBuilder* класс Node:

```
public class Node {

	public Node parent;

	public ArrayList<Node> children = new ArrayList<Node>();

	public int index;

	public double g;

	public double h;

	public int ruleNumber;

	public String ruleName;

	public double ruleCost;

	public String label;

	@Override
	public String toString() {
		return label;
	}
}
```

Данный класс содержит всю необходимую информацию о вершине графа и предоставляет возможность перемещения по нему в обе стороны, имея поля для связи как с вершиной родителем, так и с вершинами-потомками.

Структуру деревьев для всех точек принятия решений, описанных в модели, удобно хэшировать в специальные контейнеры типа Map, откуда их будет удобно считывать:

```
public HashMap<Integer, HashMap<Integer, Node>> mapList = new
HashMap<Integer,

	HashMap<Integer, Node>>();
```

После вызова конструктора класса *TreeBuilder* этот контейнер будет заполнен информацией, полученной от базы данных системы.

Информация от базы данных представляет собой список всех записей бинарной сериализации, сформированных в ходе работы системы. Этот список необходимо получить для дальнейшей работы:

```
final ArrayList<Database.Entry> entries = Simulator.getDatabase().getAllEntries();
```

Далее в цикле отбираются записи, необходимые для построения структуры графа. Из каждой записи бинарные данные считываются таким образом, чтобы не испортить данные в записи в случае множественного обращения к ней:

```
final ByteBuffer header = Tracer.prepareBufferForReading(entry.getHeader());
final ByteBuffer data = Tracer.prepareBufferForReading(entry.getData());
```

Далее приведен пример того, как бинарные данные интерпретируются разрабатываемым модулем и записываются в объект вершины:

```
{

...

Node treeNode = new Node();

...

final int nodeNumber = data.getInt();
final int parentNumber = data.getInt();
final double g = data.getDouble();
final double h = data.getDouble();
final int ruleNum = data.getInt();

...

treeNode.g = g;
treeNode.h = h;
treeNode.ruleNumber = ruleNum;


...

}
```

Списки вершин, входящих в решение, захэшированы в контейнер типа Map, значениями ключей являются номера точек принятия решений модели:

```
public HashMap<Integer, ArrayList<Node>> solutionMap = new HashMap<Integer, ArrayList<Node>>();
```

Для хранения статистической информации по поиску на графе описан вложенный по отношению к TreeBuilder класс GraphInfo:

```
public class GraphInfo {
	public double solutionCost;
	public int numOpened;
	public int numNodes;
}
```

Статистика считывается в объекты этого класса из записей завершения поиска (п. 5.1.6).

Объекты класса *GraphInfo* хранятся в подсистеме аналогично спискам вершин, входящих в решение:

```
public HashMap<Integer, GraphInfo> infoMap = new HashMap<Integer,
GraphInfo>();
```

Данные сформированных контейнеров передаются в графическую часть модуля для последующего отображения результата на экране пользователя.

6.2. Отображение интерфейса на экране пользователя

6.2.1. Отображение графа в окне интерфейса

Сущность графа как набора входящих в него вершин и связей между ними представлена классом *mxGraph* библиотеки JGraphx. Можно сказать, что это нижнего уровня, на который будут добавляться графические объекты, представляющие вершины и ребра графа.

Вершины графа с точки зрения графической части разрабатываемого модуля представлены экземплярами класса *mxCell* библиотеки JGraphx. Они добавляются на граф по рекурсивному алгоритму, реализованному в следующем методе:

```
private void drawGraph(mxGraph graph, HashMap<Integer, Node> nodeMap,
Node parentNode) {
	mxCell vertex = (mxCell)
graph.insertVertex(graph.getDefaultParent(), null, parentNode, 385,
100, 30, 30, fontColor + strokeColor);
	vertexMap.put(parentNode, vertex);
	if (parentNode.parent != null)
		graph.insertEdge(graph.getDefaultParent(), null, null,
vertexMap.get(parentNode.parent), vertexMap.get(parentNode),
strokeColor);
	if (parentNode.children.size() != 0)
		for (int i = 0; i < parentNode.children.size(); i++)
			drawGraph(graph, nodeMap,
nodeMap.get(parentNode.children.get(i).index));
}
```

Все добавленные на граф вершины хэшируются в контейнер:

```
Map<Node, mxCell> vertexMap = new HashMap<Node, mxCell>();
```

Для отображения графической информации на этом слое библиотека JGraphx предполагает использование следующей конструкции:

```
final mxGraph graph = new mxGraph();

graph.getModel().beginUpdate();
try {

...

} finally {
	graph.getModel().endUpdate();
}
```

Все изменения, вносимые в изображение, могут быть внесены только с использованием такой конструкции. В разработанном классе *GraphFrame*, экземпляр которого будет реализовывать графический интерфейс подсистемы, использует эту конструкцию для вызова трех методов из четырех, указанных в п 5.2.2.:

```
graph.getModel().beginUpdate();
try {
	drawGraph(graph, treeMap, treeMap.get(0));
	colorNodes(vertexMap, treeMap, solution);
	graphInfoCell = insertInfo(graph, info);
} finally {
	graph.getModel().endUpdate();
}
```

Оставшийся четвертый метод, отвечающий за отображение статистики по вершине, вызывается только в случае выделения пользователем определенной вершины графа. В таком случае необходимо реализовать слушателя событий, ожидающего выделения вершины:

```
graph.getSelectionModel().addListener(mxEvent.CHANGE, new
mxIEventListener() {

	private mxCell cellInfo;

	@Override
	public void invoke(Object sender, mxEventObject evt) {
		// TODO Auto-generated method stub
		mxGraphSelectionModel selectionModel =
(mxGraphSelectionModel) sender;
		mxCell cell = (mxCell) selectionModel.getCell();
		if (cell != null && cell.isVertex() && cell != cellInfo &&
cell != graphInfoCell){
			graph.getModel().beginUpdate();
			try {
```

```
                    if (cellInfo != null)
                        cellInfo.removeFromParent();
                    cellInfo = showCellInfo(graph, cell, height);
                }
                finally {
                    graph.getModel().endUpdate();
                }
            }
        }
});
```

В теле слушателя происходит проверка выделенного объекта на предмет того, является ли он объектом класса *mxCell*, и является ли он вершиной, относящейся к построенному графу.

Расположение вершин внутри графического окна осуществляется средствами графической библиотеки JGraphx. Эти средства позволяют осуществлять настройку слоев, при помощи которых графические объекты отображаются на экране пользователя:

```
mxCompactTreeLayout layout = new mxCompactTreeLayout(graph, false);

layout.setLevelDistance(30);
layout.setNodeDistance(5);
layout.setEdgeRouting(false);

layout.execute(graph.getDefaultParent());
```

Параметры расстояния между вершинами графа и между смежными уровнями вершин в глубину графа можно варьировать, задавая их через свойства слоя класса *mxCompactTreeLayout* библиотеки JGraphx.

6.2.2. Отображение статистики

Вывод статистики осуществляется в методе *insertInfo()*, который описан ниже:

```
public mxCell insertInfo(mxGraph graph, GraphInfo info) {

    final String solutionCost = "Стоимость решения: " +
Double.toString(info.solutionCost) + "\n";
    final String numOpened = "Количество раскрытых вершин: " +
Integer.toString(info.numOpened) + "\n";
    final String numNodes = "Количество вершин в графе: " +
Integer.toString(info.numNodes);
    final String text = solutionCost + numOpened + numNodes;

    int fontSize = mxConstants.DEFAULT_FONTSIZE;

    int style = Font.PLAIN;
    Font font = new Font("Arial", style, fontSize);
    double scale = 1.0;
    mxRectangle bounds = mxUtils.getSizeForString(text, font, scale);
```

```
        final double delta = 20;

        double width = bounds.getWidth() + delta;
        double height = bounds.getHeight() + delta;

        return (mxCell) graph.insertVertex(graph.getDefaultParent(),
null, text, delta / 2, delta / 2, width, height);
}
```

Статистика добавляется на граф в виде объекта вершины, представленного экземпляром класса *mxCell*, содержащего в себе текст с информацией. Этот объект имеет стиль, отличный от вершин графа, не имеет с ним связей, и следовательно представляет собой отдельный графический элемент, не имеющий отношения к графу пространства состояний.

Размер объекта *mxCell* посчитан, исходя из размеров шрифта текста, который должен быть вписан в его рамки. При помощи встроенных средств библиотеки JGraphx определяется прямоугольник – объект класса *mxRectangle* библиотеки – и высчитываются ширина и высота этого прямоугольника. Текст вписывается внутрь с небольшой дельтой, для того чтобы текст не сливался с границей содержащего его объекта.

Решение о подобной реализации было принято, исходя из требования располагать граф так, чтобы он занимал большую часть окна интерфейса. Вследствие этого от альтернативного варианта размещения статистики с использованием других компонентов библиотеки и различных слоев было решено отказаться.

Аналогичным образом реализован метод отображения статистики по вершине.

6.2.3. Окраска решения

Смена цвета для вершин, входящих в решение, реализована в методе colorNodes(). В нем происходит замена стилей для каждой вершины, входящей в решение, путем извлечения из карты вершин vertexMap объектов mxCell, по ключу, задаваемому элементом списка вершин, входящих в решение, на каждой итерации цикла:

```
public void colorNodes(Map<Node, mxCell> vertexMap, HashMap<Integer,
Node> nodeMap, ArrayList<Node> solution) {
        if (!solution.isEmpty()) {
                Node rootNode = nodeMap.get(0);
                vertexMap.get(rootNode).setStyle(solutionColor + fontColor +
strokeColor);
                for (int i = 0; i < solution.size(); i++) {
                        vertexMap.get(solution.get(i)).setStyle(solutionColor
+ fontColor + strokeColor);
                }
        }
}
```

В результате моделирования может оказаться, что решения для заданного исходного состояния модели не существует. В теле метода присутствует проверка на наличие элементов в списке вершин, принадлежащих решению. Алгоритм метода реализуется только в случае, если этот список не пуст.

6.3. Вызов интерфейса модуля визуализации

На предыдущем этапе проектирования (в п. 5.2.3.) были описаны два метода, которые необходимо реализовать в классе, описывающем интерфейс трассировщика – *RDOTraceView*.

Метод вызова окна интерфейса в качестве параметра должен получить запись, соответствующую строке, выделенной пользователем в полотне трассировки. Возможно два случая:

- строка представляет запись начала поиска;
- строка представляет запись иного типа.

Во втором случае необходимо определить ближайшую сверху запись начала поиска, т.к. только она содержит информацию о точке принятия решений, для которой необходимо вывести граф.

6.3.1. Определение записи начала поиска

Для определения записи начала поиска на графе необходимо получить список всех записей трассировки:

```
final ArrayList<TraceOutput> traceList =
Simulator.getTracer().getTraceList();
```

В нем будет осуществляться поиск записи, которую выделит пользователь в интерфейсе трассировщика двойным щелчком мыши:

```
TraceOutput traceOutput = (TraceOutput)
viewer.getTable().getSelection()[0].getData();
```

Список записей трассировки и строка, выделенная пользователем, определяются в теле слушателя, который задан для объекта класса *TableViewer*. Далее в нем в случае необходимости вызывается метод, которому передается полученный список и строка в качестве параметров:

```
private TraceOutput upToSearchBegin(ArrayList<TraceOutput> traceList,
TraceOutput traceOutput) {
	for (int index = traceList.indexOf(traceOutput) - 1;
(traceOutput.type() != TraceType.SEARCH_BEGIN) && (index != 0); index-
-) {
		traceOutput = traceList.get(index);
	}
	return traceOutput;
}
```

Метод возвращает объект класса *TraceOutput*, представляющего собой запись начала поиска на графе.

6.3.2. Вывод окна интерфейса на экран

Метод, реализующий непосредственно вывод на экран окна интерфейса разрабатываемой подсистемы, работает с записью начала поиска, принимаемой в качестве параметра:

```
private void createWindow(TraceOutput traceOutput) {
	String content = traceOutput.content();
	String frameName = "";
	int dptNum = -1;
	for (String dptNameKey : Database.searchIndex.keySet()) {
		int dotIndex = dptNameKey.indexOf('.');
		String dptName = dptNameKey.substring(dotIndex + 1);
		if (content.contains(dptName)) {
			frameName = dptName;
			dptNum = Database.searchIndex.get(dptNameKey).number;
		}
	}
	if (dptNum != -1) {
		TreeBuilder treeBuilder = new TreeBuilder();
		GraphFrame graphFrame = new
GraphFrame(treeBuilder.mapList.get(dptNum),
treeBuilder.infoMap.get(dptNum), treeBuilder.solutionMap.get(dptNum));

	graphFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
		graphFrame.setSize(800, 600);
		graphFrame.setTitle(frameName);
		graphFrame.setLocationRelativeTo(null);
		graphFrame.setVisible(true);
	}
}
```

Метод выделяет из записи имя точки принятия решений, после чего запрашивает у базы данных контейнер, содержащий номера точек принятия решений. Значениями ключей являются имена точек типа search, описанных в тексте модели. При нахождении совпадения имени из записи со значением одного из ключей, из контейнера выбирается соответствующий номер точки принятия решений.

Вызывается конструктор класса *GraphFrame* с параметрами, соответствующими номеру точки принятия решений, в результате чего на экран пользователя выводится окно интерфейса.

7. Апробирование разработанной подсистемы в модельных условиях

7.1. Методика тестирования

Для проведения тестов и отладки кода была разработана методика тестирования подсистемы:

1. Создать в среде RAO-XT новый проект;
2. Создать тестовую модель, используя исходный код модели, приведенный в приложении А;
3. В интерфейсе модуля конфигурации трассировки настроить вывод трассировки точек принятия решений;
4. Запустить модель;
5. Проверить работу интерфейса модуля визуализации, вызвав его из интерфейса окна вывода трассировки;
6. Сверить отображенную статистику с трассировкой.

По приведенной методике следует провести многократное ручное тестирование. После этого, в случае положительного результата, необходимо усложнить модель, поочередно изменяя ее следующими способами:

- ввести в модель вторую точку принятия решений;
- задать для ресурсов такие значения параметров, при которых решения не существует;
- изменить количество ресурсов.

Дополнительно следует провести тесты, используя описанные в модели эвристики, отличные от поиска в ширину.

7.2. Результаты тестирования

По результатам тестирования текущей версии подсистемы визуализации было установлено:

- подсистема исправна;
- весь функционал, требуемый от ее интерфейса, может быть использован;
- результаты моделирования, выводимые и предоставляемые системой пользователю, соответствуют результатам, выводимым независимо другой подсистемой (подсистемой трассировки).

8. Заключение

В рамках данного курсового проекта были получены следующие результаты:

- Проведено предпроектное исследование системы имитационного моделирования RAO-XT и ее компонентов на предмет необходимости внедрения новой подсистемы;
- Определены основные функции и требования к подсистеме, на основе которых было составлено техническое задание;
- На этапах концептуального и технического проектирования была разработана основная структура подсистемы, схема ее взаимодействия с другими компонентами системы RAO-XT, обоснован выбор графической библиотеки и рассмотрены аспекты технической реализации;
- На этапе рабочего проектирования был разработан и реализован модуль визуализации поиска на графе пространства состояний для системы имитационного моделирования RAO-XT. Модуль реализован таким образом, что другим компонентам системы не требуется использовать его вывод для своего функционирования;
- Была разработана и испытана методика ручного тестирования модуля визуализации. Результаты тестов показали, что интеграция модуля в систему RAO-XT проведена успешно.

Список использованных источников

1. **Емельянов, В. В.** Введение в интеллектуальное имитационное моделирование сложных дискретных систем и процессов. Язык РДО. / В. В. Емельянов, С. И. Ясиновский - М.: "Анвик", 1998. - 427 с., ил. 136.

2. **Емельянов, В. В.** Принятие оптимальных решений в интеллектуальных имитационных системах: Учебное пособие по курсам «Методы системного анализа и синтеза» и «Моделирование технологических и производственных процессов» / В.В. Емельянов, В.И. Майорова, Ю.В. Разумцова и др. – М.: Изд-во МГТУ им. Н.Э. Баумана. – 2002. – 60 с.: ил.

3. Документация по языку РДО [http://www.rdostudio.com/help/index.html]

4. Java™ Platform, Standard Edition 7. API Specification. [http://docs.oracle.com/javase/7/docs/api/]

5. JGraphX (JGraph 6) User Manual [https://jgraph.github.io/mxgraph/docs/manual_javavis.html]

6. JGraphx API Specification [https://jgraph.github.io/mxgraph/java/docs/index.html]

Список использованного программного обеспечения

7. Autodesk AutoCAD2012

8. Eclipse IDE for Java Developers Luna Service Release 1 (4.4.1)

9. Inkscape 0.48.4

10. Microsoft Office Word 2010

11. openjdk version "1.8.0_40-internal"

12. Visual Paradigm for UML 8.0

Приложение А. Исходный код модели, использованной для тестирования модуля

```
/* Game 5 */

$Resource_type Фишка : permanent
$Parameters
  Номер: integer
  Местоположение: integer
$End

$Resource_type Дырка_t : permanent
$Parameters
  Место: integer
$End

$Resources
  Фишка1 = Фишка(1, 2);
  Фишка2 = Фишка(2, 3);
  Фишка3 = Фишка(3, 6);
  Фишка4 = Фишка(4, 4);
  Фишка5 = Фишка(5, 5);
  Фишка6 = Фишка(6, 9);
  Фишка7 = Фишка(7, 7);
  Фишка8 = Фишка(8, 8);
  Дырка = Дырка_t(1);
$End


$Pattern Перемещение_фишки : rule
$Parameters
  Куда_перемещать: such_as Место_дырки
  На_сколько_перемещать: integer
$Relevant_resources
  _Фишка: Фишка   Keep
  _Дырка: Дырка_t Keep
$Body
  _Фишка:
    Choice from Где_дырка(_Фишка.Местоположение) ==
Куда_перемещать
    first
      Convert_rule Местоположение = _Фишка.Местоположение +
На_сколько_перемещать;
  _Дырка:
    Choice NoCheck
    first
      Convert_rule Место = _Дырка.Место - На_сколько_перемещать;
$End

$Decision_point Расстановка_фишек : search
$Condition Exist(Фишка: Фишка.Номер <> Фишка.Местоположение)
$Term_condition
  For_All(Фишка: Фишка.Номер == Фишка.Местоположение)
$Evaluate_by IDS()
```

```
$Compare_tops = YES
$Activities
  Перемещение_вправо: Перемещение_фишки(справа, 1) value after
1;
  Перемещение_влево : Перемещение_фишки(слева, -1) value after
1;
  Перемещение_вверх : Перемещение_фишки(сверху, -3) value after
1;
  Перемещение_вниз  : Перемещение_фишки(снизу, 3) value after 1;
$End

$Constant
    Место_дырки: (справа, слева, сверху, снизу, дырки_рядом_нет)
= дырки_рядом_нет
    Длина_поля : integer = 3
$End

$Function IDS: integer
$Type = algorithmic
$Parameters
$Body
    return 0;
$End

$Function Ряд: integer
$Type = algorithmic
$Parameters
    Местоположение: integer
$Body
    return (Местоположение - 1)/Длина_поля + 1;
$End

$Function Остаток_от_деления : integer
$Type = algorithmic
$Parameters
    Делимое   : integer
    Делитель  : integer
$Body
    integer Целая_часть   = Делимое/Делитель;
    integer Макс_делимое = Делитель * Целая_часть;
    return Делимое -  Макс_делимое;
$End

$Function Столбец: integer
$Type = algorithmic
$Parameters
    Местоположение: integer
$Body
    return Остаток_от_деления(Местоположение - 1,Длина_поля) +
1;
$End

$Function Где_дырка : such_as Место_дырки
```

```
$Type = algorithmic
$Parameters
    _Место: such_as Фишка.Местоположение
$Body
    if (Столбец(_Место) == Столбец(Дырка.Место) and Ряд(_Место)
== Ряд(Дырка.Место)+ 1) return сверху;
    if (Столбец(_Место) == Столбец(Дырка.Место) and Ряд(_Место)
== Ряд(Дырка.Место)- 1) return снизу;
    if (Ряд(_Место) == Ряд(Дырка.Место) and Столбец(_Место) ==
Столбец(Дырка.Место)- 1) return справа;
    if (Ряд(_Место) == Ряд(Дырка.Место) and Столбец(_Место) ==
Столбец(Дырка.Место)+ 1) return слева;
    return дырки_рядом_нет;
$End

$Function Фишка_на_месте : integer
$Type = algorithmic
$Parameters
    _Номер: such_as Фишка.Номер
    _Место: such_as Фишка.Местоположение
$Body
    if (_Номер == _Место) return 1;
    else                  return 0;
$End

$Function Кол_во_фишек_не_на_месте : integer
$Type = algorithmic
$Parameters
$Body
    return 5 - (Фишка_на_месте(Фишка1.Номер,
Фишка1.Местоположение)+
                Фишка_на_месте(Фишка2.Номер,
Фишка2.Местоположение)+
                Фишка_на_месте(Фишка3.Номер,
Фишка3.Местоположение)+
                Фишка_на_месте(Фишка4.Номер,
Фишка4.Местоположение)+
                Фишка_на_месте(Фишка5.Номер,
Фишка5.Местоположение));
$End

$Function Расстояние_фишки_до_места : integer
$Type = algorithmic
$Parameters
    Откуда: integer
    Куда  : integer
$Body
    return Math.abs(Ряд(Откуда)-Ряд(Куда)) +
Math.abs(Столбец(Откуда)-Столбец(Куда));
$End

$Function Расстояния_фишек_до_мест : integer
$Type = algorithmic
```

```
$Parameters
$Body
    return Расстояние_фишки_до_места(Фишка1.Номер,
Фишка1.Местоположение)+
           Расстояние_фишки_до_места(Фишка2.Номер,
Фишка2.Местоположение)+
           Расстояние_фишки_до_места(Фишка3.Номер,
Фишка3.Местоположение)+
           Расстояние_фишки_до_места(Фишка4.Номер,
Фишка4.Местоположение)+
           Расстояние_фишки_до_места(Фишка5.Номер,
Фишка5.Местоположение);
$End

$Results
    f1 : get_value Фишка1.Местоположение
    f2 : get_value Фишка2.Местоположение
    f6 : get_value Дырка.Место

    ft1 : get_value 1
    ft2 : get_value 5
$End
```