Оглавление

Перечень сокращений **4**
Терминология **4**
1. Введение **5**
2. Предпроектное исследование **7**
2.1. Основные положения языка РДО[2] 7
2.2. Система имитационного моделирования RAO-studio[2] 7
2.3. Трассировка в системе имитационного моделирования RAO-studio[2] 8
2.4. Система имитационного моделирования RAO-XT 9
3. Формирование ТЗ **10**
3.1. Введение 10
3.2. Общие сведения 10
3.3. Назначение разработки 10
3.4. Требования к программе или программному изделию 10
3.4.1. Требования к функциональным характеристикам 10
3.4.2. Требования к надежности 10
3.4.3. Условия эксплуатации 11
3.4.4. Требования к составу и параметрам технических средств 11
3.4.5. Требования к информационной и программной совместимости 11
3.4.6. Требования к маркировке и упаковке 11
3.4.7. Требования к транспортированию и хранению 11
3.5. Требования к программной документации 11
3.6. Стадии и этапы разработки 11
3.7. Порядок контроля и приемки 11
4. Концептуальный этап проектирования системы **13**
4.1. Формат трассировки 13
4.1.1. Формат трассировки ресурсов 13
4.1.2. Формат трассировки образцов 14
4.1.3. Формат трассировки результатов 14
4.1.4. Формат трассировки точек принятия решений 15
4.1.5. Формат системных событий 16
4.1.6. Формат вывода параметров 16
4.2. Пользовательский интерфейс конфигурации трассировки 17
4.3. Модуль трассировки в системе 17
4.4. Использование трассировки для тестирования 18
5. Технический этап проектирования системы **20**
5.1. Перевод сериализованных бинарных данных в текстовый формат 20
5.2. Работа с большими объемами данных 20
5.3. Схема работы в реальном времени 21
5.4. Вывод трассировки в формате, идентичном формату RAO-studio 21
5.4.1. Преобразование системных записей 21
5.4.2. Преобразование записей ресурсов 21
5.4.3. Преобразование записей образцов 22
5.4.4. Преобразование записей поиска по графу 22
5.4.5. Преобразование записей результатов 23
5.5. Схема работы пользовательского интерфейса конфигурации трассировки 23
6. Рабочий этап проектирования системы **24**
6.1. Реализация читаемого вывода трассировки 24

6.2. Реализация вывода трассировки в реальном времени 25
6.3. Реализация вывода трассировки в формате RAO-studio 26
6.3.1. Приведение номеров ресурсов 26
6.3.2. Приведение номеров выполнения активностей 27
7. Апробирование разработанной системы в модельных условиях 29
7.1. Сравнение вывода трассировки RAO-XT в legacy-режиме и вывода трассировки RAO-studio 29
7.2. Сравнение производительности 29
7.2.1. Тест на скорость добавления данных в вывод трассировки. 29
7.2.2. Тест на скорость обновления данных в таблице 31
8. Заключение 33
Список используемых источников 34
Список использованного программного обеспечения 34
Приложение 1 – Исходный код модели для тестирования трассировки 35
Приложение 2 – Вывод трассировки модели из Приложения 1 в новом формате 35
Приложение 3 – Вывод трассировки модели из Приложения 1 в legacy-формате 35

Перечень сокращений

ИМ – Имитационное Моделирование

СДС – Сложная Дискретная Система

IDE - Integrated Development Environment (Интегрированная Среда Разработки)

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.

Legacy-code (legacy-код) — код, унаследованный от прошлой версии программы.

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[2]

Основные положения системы РДО могут быть сформулированы следующим образом[1]:

- 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 отоношению 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. Система имитационного моделирования RAO-studio[2]

Программный комплекс RAO-studio 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.

В соответствии с основной целью программный комплекс решает следующие задачи:

- синтаксический разбор текста модели и настраиваемая подсветка синтаксических конструкций языка РДО;

- открытие и сохранение моделей;

- расширенные возможности для редактирования текстов моделей;

- автоматическое завершение ключевых слов языка;

- поиск и замена фрагментов текста внутри одного модуля модели;

- поиск интересующего фрагмента текста по всей модели;

- навигация по тексту моделей с помощью закладок;
- наличие нескольких буферов обмена для хранения фрагментов текста;
- вставка синтаксических конструкций языка и заготовок (шаблонов) для написания элементов модели;
- настройка отображения текста моделей, в т.ч. скрытие фрагментов текста и масштабирование;
- запуск и остановка процесса моделирования;
- изменение режима моделирования;
- изменение скорости работающей модели;
- переключение между кадрами анимации в процессе моделирования;
- отображение хода работы модели в режиме реального времени;
- построение графиков изменения интересующих разработчика характеристик в режиме реального времени;
- обработка синтаксических ошибок при запуске процесса моделирования;
- обработка ошибок во время выполнения модели;
- обеспечение пользователя справочной информацией.

Программный комплекс состоит из двух частей:

- среды разработки (файл RAO-studio.exe под Windows и RAO-studio под Linux);
- файлов справок (rdo_lang_rus.qch - справка по языку РДО, rdo_studio_rus.qch - справка по программному комплексу, RAO-help.qhc - объединяет два последних для справочной системы).

2.3. Трассировка в системе имитационного моделирования RAO-studio[2]

Трассировка содержит информацию четырех видов: трассировку событий, трассировку состояния ресурсов, трассировку показателей и трассировку точек принятия решений. Трассировка выдается только если текущее модельное время больше или равно времени начала трассировки и меньше или равно времени окончания трассировки, а также если в соответствующем объекте (ресурсе, образце, показателе или точке принятия решений) явно указан признак трассировки.

В РДО можно производить трассировку:

- событий и образцов
- точек принятия решений
- ресурсов
- показателей

Для указанных объектов может быть указан признак трассировки.

Для событий, образцов, ресурсов, показателей и точек принятия решений типа some и prior признак задают одним из двух зарезервированных слов:

- trace - производить трассировку объекта
- no_trace - не производить трассировку объекта

Для точек принятия решений типа search признак трассировки может быть одним из следующих:

- no_trace - не производить трассировку точки
- trace_stat - выдавать в объект трассировки только статистическую информацию по процессу поиска на графе
- trace_tops - выдавать в объект трассировки статистическую информацию по процессу поиска и информацию о всех вершинах графа поиска
- trace_all - выдавать в объект трассировки статистическую информацию по процессу поиска, информацию о всех вершинах графа поиска и для каждой вершины новое состояние всех ресурсов, изменивших свое состояние при применении правила, породившего эту вершину

Значением признака трассировки по умолчанию является значение no_trace. Остальные значения признака трассировки имеют смысл только для точек типа search.

2.4. Система имитационного моделирования RAO-XT

Система имитационного моделирования RAO-XT представляет собой плагин для интегрированной среды разработки Eclipse, позволяющий вести разработку имитационных моделей на языке РДО. Система написана на языке Java[3] 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[4].

На момент начала выполнения курсового проекта, система не имела возможности выводить для пользователя трассировочную информацию, однако позволяла сохранять в бинарном формате сериализованные данные.

3. Формирование ТЗ

3.1. Введение

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.

3.2. 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» Богачев П.А.

Наименование темы разработки: «Проектирование модуля трассировки для системы имитационного моделирования RAO-XT»

3.3. Назначение разработки

Разработать модуль трассировки для системы имитационного моделирования RAO-XT.

3.4. Требования к программе или программному изделию

3.4.1. Требования к функциональным характеристикам

Модуль трассировки должен удовлетворять следующим требованиям:

- Обеспечивать корректное отображения изменения состояния системы в процессе прогона модели в текстовом формате.
- Предоставлять возможность настраивать элементы модели, для которых необходимо предоставлять трассировочный вывод.
- Предоставлять удобный для восприятия и анализа формат вывода.
- Иметь возможность работать с большими объемами данных.
- Предоставлять трассировочную информацию в реальном времени в процессе прогона модели.
- Предоставлять трассировку в формате, совпадающим с форматом трассировки системы имитационного моделирования RAO-studio, для обеспечения возможности проведения автоматического тестирования.

3.4.2. Требования к надежности

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

3.4.3. 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.

3.4.4. Требования к составу и параметрам технических средств

Программный продукт должен работать на компьютерах со следующими характеристиками:

- объем ОЗУ не менее 1 Гб;

- объем жесткого диска не менее 50 Гб;

- микропроцессор с тактовой частотой не менее 1ГГц;

- монитор с разрешением от 800*600 и выше.

3.4.5. Требования к информационной и программной совместимости

Система должна работать под управлением следующих ОС: Windows 7, Ubuntu 14.10.

3.4.6. Требования к маркировке и упаковке

Требования к маркировке и упаковке не предъявляются.

3.4.7. Требования к транспортированию и хранению

Требования к транспортированию и хранению не предъявляются.

3.5. Требования к программной документации

Требования к программной документации не предъявляются.

3.6. Стадии и этапы разработки

Плановый срок начала разработки – 6 сентября 2014г.

Плановый срок окончания разработки – 29 декабря 2014г.

Этапы разработки:

- Концептуальный этап проектирования системы;

- Технический этап проектирования системы;

- Рабочий этап проектирования системы.

3.7. Порядок контроля и приемки

Контроль и приемка работоспособности системы осуществляются с помощью следующих методов:

- Опрос экспертов. Используется для оценки качества нового формата трассировки.

- Автоматизированное сравнение с трассировкой RAO-studio с помощью утилит сравнения файлов.

4. Концептуальный этап проектирования системы

На концептуальном этапе проектирования требовалось:

- разработать новый формат трассировки, который отвечал бы требованиям компактности, читаемости и минимизации избыточных данных
- разработать пользовательский интерфейс, позволяющий настраивать, какие элементы системы необходимо трассировать в процессе прогона
- разработать общую структуру модуля трассировки и схему его взаимодействия с другими компонентнами системы
- разработать схему, по которой будет осуществляться тестирование корректности выполнения моделей с использованием трассировки

4.1. Формат трассировки

Было разработано и предложено на рассмотрение экспертам несколько вариантов формата трассировки, из которых, в результате опроса и обсуждения, был выбран финальный формат.

4.1.1. Формат трассировки ресурсов

Было принято решение внести следующие изменения:

- Для ресурсов, которые имеют имена, выводить имена ресурсов вместо номера типа и номера ресурса, т.к. имя ресурса уникально, и этой информации достаточно для однозначного его определения.
- Для ресурсов, не имеющих имен (временных ресурсов, созданных в процессе прогона модели), выводить имя их типа, а так же порядковый номер ресурса данного типа в модели в квадратных скобках (при этом используется сквозная нумерация).
- Параметры ресурсов выводить в фигурных скобках через запятую после имена ресурса и знака равенства. Это позволяет отразить физический смысл ресурса, как некоторой структуры.
- Для ресурсов, созданных перед началом прогона (т.е. ресурсов, описанных во вкладке $Resources) выводить первую запись с символом статуса C вместо K, т.е. выводить запись создания ресурса вместо записи изменения ресурса.

Таким образом окончательный формат вывода ресурсов можно описать следующим образом:

```
R<символ_статуса>
    <модельное_время>
    <уникальный_идентификатор_ресурса>
    = {<новые_значения_параметров>}
```

Где уникальный_идентификатор_ресурса имеет следующий формат:

```
<имя_ресурса>
```

для ресурсов, имеющих имена, и

```
<имя_типа>[<номер_ресурса>]
```

для ресурсов, не имеющи имен.

Примеры вывода трассировки состояния ресурсов:

```
RC 80.6 Клиенты[1] = {Тип2, Пришел}
RK 80.6 Парикмахер_3 = {Занят, 0, 30, 60, Тип2}
RE 100.6 Клиенты[1] = {Тип2, Закончил_Обслуживание}
```

4.1.2. Формат трассировки образцов

Было принято решение внести следующие изменения:

- Ввести сквозную нумерацию всех типов образцов: операций, продукционных правил и событий. Это позволит оценить порядковый номер данного события в системе, не просматривая другие записи трассироки.
- Для операций опустить номер образца, так как он уникально идентифицируется номером активности. Вместо номера активности выводить её имя, а затем в квадратных скобках порядковый номер выполненного образца в процессе прогона.
- Для событий выводить имя образца события вместо его номера, а затем в квадратных скобках порядковый номер выполненного события в процессе прогона.
- Для продукционных правил опустить номер образца, так как он уникально идентифицируется номером активности. Вместо номера активности выводить её имя, а затем в квадратных скобках порядковый номер выполненного правила в процессе прогона.
- Вместо номеров релевантных ресурсов выводить их имена (см. 4.1.2). Имена подобранных релевантных ресурсов выводить в круглых скобках через запятую. Это позволяет отразить физический смысл образца, как некоторой функции над ресурсами.
- Опустить число релевантных ресурсов, так как они и так перечислены.

Таким образом окончательный формат вывода трассировки образцов можно описать следующим образом:

```
E<тип_образца > <модельное_время>
<имя_активности_или_события>[<порядковый_номер>]
(<список_релевантных_ресурсов>)
```

Примеры:

```
EB 40.6 Обслуживание_клиента[0] (Парикмахерская, Клиенты[0],
Парикмахер_2)
EI 80.6 Образец_прихода_клиента[1] (Парикмахерская, Клиенты[1])
EF 82.6 Обслуживание_клиента[0] (Парикмахерская, Клиенты[0],
Парикмахер_2)
ER 100.6 Тестовое_правило[3] (Тестовый_ресурс)
```

4.1.3. Формат трассировки результатов

Было принято решение внести следующие изменения:

- Вместо номера результата выводить его имя.
- Значение результата выводить после знака равенства. Это позволит отразить физический смысл результата как некоторой единичной величины, в отличие от ресурсов, которые представляют собой структуры.

Таким образом окончательный формат вывода трассировки результатов можно описать следующим образом:

```
V <модельное_время> <имя _показателя> = <значение>
```

Примеры:

```
V 80.6 Занятость_парикмахера = false
V 80.6 Длина_очереди = 0.0
```

4.1.4. Формат трассировки точек принятия решений

Было принято решение внести следующие изменения:

- В начале поиска по графу выводить имя точки принятия решения вместо её номера.
- При порождении вершины выводить номер порожденной вершины и номер родителя в квадратных скобках со стрелкой, следующей от родителя к потомку.
- При раскрытии вершины выводить только номер раскрываемой вершины в квадратных скобках, для облегчения трассировки и избежания дублирования данных.
- Применяемые правила в процессе порождения вершины выводить в формате трассировки образцов, описанном в пункте 4.1.3.
- Перед значением стоимости правила выводить слово «cost», затем двоеточие и стоимость пути в формате [f = g + h].
- Числовые значения результатов поиска пояснять текстом (solution cost, nodes opened, nodes total, nodes added, nodes spawned).
- Опустить неинформативные записи порождения корневой вершины.
- Опустить неинформативные записи в выводе результата (затраченная память и т.п.)

Таким образом окончательный формат вывода трассировки точек принятия решений можно описать следующим образом:

Начало поиска:

```
SB <модельное_время> <имя_точки_принятия_решения>
```

Раскрытие вершины:

```
SO [<номер_вершины>]
```

Информация о порожденн:

```
ST<признак_вершины> [<номер_вершины-родителя>]->[<номер_вершины>]
    <имя_активности_правила>(<имена_релевантных_ресурсов>)
```

```
    cost <стоимость_применения_правила>:
<оценочная_стоимость_полного_пути_f> =
<фактическая_стоимость_пройденного_пути_g> +
<оценочная_стоимость_оставшегося_пути_h>
```

Найденное решение:

```
SD [<номер_вершины>]
<имя_активности_правила>(<имена_релевантных_ресурсов>)
```

Окончание поиска:

```
SE<признак> <модельное_время>
solution cost = <значение_стоимости_решения>,
nodes opened = <количество_раскрытых_вершин>,
nodes total =  <количество_вершин_в_графе>,
nodes added = <количество_включавшихся_в_граф_вершин>,
nodes spawned =  <количество_порожденных_вершин>
```

Примеры:

```
SB 0.0 model
SO  [1]
STN [1]->[3] Перемещение_вправо(Фишка1, Дырка) cost 1.0: 3.0 = 3.0 +
0.0
SD  [3] Перемещение_вправо(Фишка1, Дырка)
SES 0.0 solution cost = 3.0, nodes opened = 8, nodes total = 12, nodes
added = 13, nodes spawned = 19
```

4.1.5. Формат системных событий

Было принято решение внести следующие изменения:

- Вместо кода системного события выводить его текстовое описание.
- Упразднить вывод системного события 4 («Завершение процесса поиска на графе состояний в точке принятия решений») и добавить новое системное событие «User interrupt».

Таким образом окончательный формат вывода системных событий можно описать следующим образом:

```
ES <модельное_время> <название_системного_события>
```

Примеры:

```
ES 0.0 Tracing started
ES 0.0 Simulation started
ES 1.0 User interrupt
```

4.1.6. Формат вывода параметров

Для всех записей, которые содержат параметры стандартных типов, также внесены изменения в вывод этих параметров:

- Для параметров типа enumeration выводить имя значения вместо его номера.

- Для параметров типа boolean выводить true/false вместо TRUE/FALSE.

4.2. Пользовательский интерфейс конфигурации трассировки

Задание трассировки элементам модели через грамматику имеет следующие недостатки:

- Концептуально трассировочная информация и конфигурация не является неотъемлемой частью модели, и наличие в грамматике языка конструкций для трассировки противоречит этому.
- Конструкции для задания трассировки переусложняют грамматику языка.
- Место написания ключевых слов для задания трассировки не всегда очевидно и отличается для различных элементов модели.
- Включение/отключение трассировки для части элементов модели требует поиска нужных элементов в коде модели, что может быть достаточно трудоемким в случае больших моделей.

В связи с этими недостатками было принято решение исключить трассировочные конструкции из грамматики и вынести конфигурацию трассировки в отдельное окно пользовательского интерфейса среды RAO-XT.

Данный пользовательский интерфейс имеет древовидную структуру, отражающую все элементы модели, и позволяющую отметить флажком те элементы, которые необхожимо трассировать.

Схематически данную древовидную структуру можно изобразить следующим образом:

- Resources
  - Resource_1
  - …
  - Resource_n
- Patterns
  - Pattern_1
    - Relevant_resource_1
    - …
    - Relevant_resource_n
  - …
- Decision points
  - Decision_point_1 (some/prior)
  - Decision_point_2 (search)
    - all
    - tops
    - decision
    - result
  - …
- Results
  - Result_1
  - …

Такая структура позволяет задать все конфигурации трассировки, которые возможны были в RAO-studio, а также может быть легко изменена для добавления новых вариантов конфигурации без необходимости изменять грамматику языка.

4.3. Модуль трассировки в системе

На этапе концептуального проектирования ключевой задачей является разработка правильной схемы взаимодействия модуля трассировки с остальными частями системы RAO-XT. Ниже приведены основные требования к модулю трассировки 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 трассировки должен формировать выходные данные на основе сериализованных данных и преобразовывать их в формат, удобный для восприятия человеком.

- Модуль трассировки должен предоставлять выходные данные исключительно для пользователя. Никакие другие компоненты системы не должны основывать свою работу на выходных данных модуля трассировки.

- Модуль трассировки должен содержать в себе два подмодуля: модуль вывода трассировочной информации и модуль конфигурации трассировки.

- Библиотечная часть модуля вывода трассировки должна заниматься исключительно преобразованием бинарных данных в текстовый формат. Далее с полученным текстом работает часть модуля, расположенная в пакете UI. Эта часть модуля занимается отображением трассировочного вывода для пользователя, работой по отображению больших объемов данных, т.е. реализует удобный пользовательский интерфейс. При этом данных, которые отображает графическая часть модуля трассировки должны полностью соответствовать тем данным, которые на данный момент преобразовал трассировщик в библиотечной части модуля.

- Модуль конфигурации трассировки должен быть активен только до и после (но никак не во время) прогона модели. Модуль конфигурации трассировки должен передавать установленную пользователем конфигурацию в начале прогона модели, и в дальнейшем (в процессе прогона) данная конфигурация изменяться не должна.

4.4. Использование трассировки для тестирования

Для тестирования корректности работы ядра системы имитационного моделирования RAO-XT необходимо иметь возможность сравнить результаты всех шагов прогона с результатами уже отлаженной системы RAO-studio. Для системы RAO-studio трассировка является единственным средством отображения состояния модели в каждый момент модельного времени. Поэтому для проведения тестирования необходимо иметь возможность осуществлять вывод трассировки, идентичный выводу трассировки в RAO-studio.

Было рассмотрено три подхода к структуре подсистемы, позволяющий выводить трассировку в legacy-формате:

I. Подсистема legacy-трассировки представляет собой отдельный класс (LegacyTracer) библиотеки RAO-XT, который, как и основной трассировщик (Tracer), работает непосредственно с базой данных, и преобразовывает её содержимое в формат, идентичный формату трассировки RAO-studio.

   Достоинства данного подхода:

   - Модульность. Подобный класс не будет зависеть от основного трассировщика, а значит, изменения в формате трассировки в RAO-XT его не затронут.

   - Класс работает по тому же принципу, что и класс основного трассировщика, а значит, не требует значительных временных затрат в написании.

Недостатки данного подхода:

- Часть методов легаси-трассировщика будет абсолютно идентична методам основного трассировщика, т.е. часть кода будет продублирована.

II. Подсистема legacy-трассировки представляет собой отдельный модуль библиотеки RAO-XT, который преобразовывает вывод основного трассировщика в формат, идентичный формату трассировки RAO-studio.

Достоинства данного подхода:

- В случае небольших отличий формата трассировки RAO-XT от формата трассировки RAO-stuido, модуль будет компактным и достаточно простым.

Недостатки данного подхода:

- Модуль зависит от основного трассировщика и его необходимо корректировать при любых изменениях в формате трассировки.
- В случае серьезных изменений в формате трассировки, модуль будет громоздким и сложным.
- Некоторые изменения в формате могут оказаться невосстановимыми. В этом случае модуль перестанет функционировать вообще.

III. Режим legacy-трассировки представляет собой особенный режим работы основного трассировщика и не является отдельным модулем. Весь код по формированию трассировки в legacy-формате находится в классе основного трассировщика.

Достоинства данного подхода:

- Минимизация количества дополнительного кода.
- Возможность практически полностью исключить дублирование кода.

Недостатки данного подхода:

- Класс трассировщика получится громоздним и трудночитаемым.

В результате рассмотрения вышеперечисленных вариантов, был сформирован компромиссный подход, основывающийся на вариантах I и III, согласно которому legacy-трассировщик представляет собой отдельный класс, однако этот класс является потомком класса основного трассировщика (Tracer), в котором переопределены методы по формированию строк вывода трассировочной информации. Такой подход позволяет сохранить модульность системы и независимость от формата вывода основного трассировщика, не переусложнять класс основного трассировщика и, в то же время, минимизировать дублирование кода.

5. Технический этап проектирования системы

5.1. Перевод сериализованных бинарных данных в текстовый формат

Библиотечная часть модуля трассировки, основным компонентом которой является класс трассировщика (Tracer) взаимодействует непосредственно с содержимым базы данных и переводит это содержимое в текстовый формат. Для этого, трассировщику необходимо, зная правила сериализации данных, считать нужное количество байт из записи, преобразовать их в переменную корректного типа, а затем сформировать на основе считанных данных строку, которая и будет отображена пользователю.

Трассировщик должен проверить тип записи, считанной из базы данных, а затем провести чтение записи в соответствии с правилами её сериализации. 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 преобразования бинарных данных в читаемый формат, трассировщику необходимо иметь возможность получить информацию о структуре того или иного элемента модели по его номеру. Структура модели хранится в формате JSON[6]. Однако частое обращение к структуре в данном формате может значительно снизить производительность трассировки, поэтому считывание структуры модели следует производить один раз перед началом трассировки, а затем закэшировать только необходимые трассировщику данные о структуре модели в быстрые и удобные контейнеры, такие как ArrayList или HashMap. Для осуществления подобного кэширования созданы вспомогательные классы, хранящие необходимые данные в своих полях, а также вспомогательный класс ModelStructureHelper, способный заполнить поля данных классов.

5.2. Работа с большими объемами данных

Одним из требований к модулю трассировки для RAO-XT является возможность работы с большими объемами данных. Библиотека SWT, используемая в RAO-XT для отображения графического интерфейса предоставляет возможность использовать ленивые вычисления при отображении табличных данных. Для этого источник данных для таблицы (ContentProvider) должен реализовывать интерфейс ILazyContentProvider[5], имеющий следующий вид:

```
public interface ILazyContentProvider extends IContentProvider
{
   public void updateElement(int index);
}
```

Метод updateElement() должен быть переопределен в классе, реализующем данный интерфейс. В данном методе указать, какой элемент должен находиться в таблице по данному индексу. Таким образом, вместо того, чтобы сразу наполнять таблицу всеми элементами и выводить их по мере перемещения по таблице, каждый раз расчитываются заново и выводятся только те

элементы, которые попадают в видимую область таблицы на данный момент. Подобный подход ведет к значительному увеличению производительности при работе с большими объемами данных.

5.3. Схема работы в реальном времени

Система имитационного моделирования RAO-studio предоставляла два режима вывода трассировки:

- В реальном времени. В случае прогона модели с некоторым конечным масштабом времени, трассировочная информация выводилась в окно трассировки по мере выполнения модели.

- В конце моделирования. В случае прогона модели в режиме без анимации, вся трассировочная информация выводилась по окончании прогона. При этом необходимо заметить, что формирование строк трассировки всё равно выполнялось в процессе прогона, однако пользователю они не выводились.

Для модуля трассировки RAO-XT было решено сохранить оба режима, однако для режима вывода в конце моделирования, формировать вывода трассировки не в процессе прогона, а по его окончании.

Реализация обоих режимов осуществляется с помощью системы оповещений библиотеки RAO-XT. Для этого класс, которому необходимо получать оповещения о том, что произошли некоторые события, должен реализовывать интерфейс Subscriber, имеющий следующий вид:

```
public interface Subscriber
{
   public void fireChange();
}
```

Общая схема работы такова: трассировщик (Tracer) имеет два подписчика: подписчик реального времени (realTimeSubscriber) и подписчик, который получает оповещения только о начале и конце прогона (commonSubscriber). Объект commonSubscriber получает оповещения от симулятора, а объект realTimeSubscriber – от базы данных при добавлении новых записей. Таким образом в режиме моделирования без анимации никаких трассировочных расчетов не производится вплоть до окончания моделирования, а в режиме реального времени все расчеты производятся сразу как только новые данные были сериализованы.

5.4. Вывод трассировки в формате, идентичном формату RAO-studio

Формат вывода трассировки в RAO-studio в достаточной степени соответствует формату бинарных данных, которые хранятся в базе данных RAO-XT. Тем не менее, для получения совершенно идентичного вывода необходимо сделать ряд преобразований для каждого из типов записей.

5.4.1. Преобразование системных записей

В случае системных записей необходимо преобразовать номера кодов системных событий RAO-XT к кодам RAO-studio. Кроме того, системное событие «Завершение процесса поиска на графе состояний в точке принятия решений» в RAO-XT не используется, и потому его вывод должен добавляться автоматически после вывода статистики поиска по графу.

5.4.2. Преобразование записей ресурсов

Для записей ресурсов необходимо выполнить следующие преобразования:

- Ресурсы, созданные до начала прогона, в RAO-XT имеют статус ‘CREATED’ и выводятся с конвертором RC. Для данных ресурсов необходимо выводить конвертор RK.

- Нумерация типов ресурсов в RAO-XT начинается с 0, а в RAO-studio с 1, поэтому необходимо добавлять 1 к номеру типа при его выводе.

- Нумерация ресурсов в RAO-XT координальным образом отличается от нумерации ресурсов в RAO-studio. В RAO-studio все ресурсы имели общую нумерацию, начинающуюся с 1, при этом каждый новый ресурс получал минимально возможный незанятый номер. В RAO-XT нумерация ресурсов отдельная для каждого типа, начинается с 0, и вместо наименьшего незанятого номера новому ресурсу всегда назначается номер на 1 больше предыдущего.

  Для выполнения преобразований номеров ресурсов необходимо вести в процессе трассировки два списка: список соответствия пар <тип, номер> legacy-номерам и список занятых legacy-номеров, чтобы выбирать наименьший номер, не содержащийся в этом списке, по аналогии с алгоритмом, реализованным в RAO-studio.

5.4.3. Преобразование записей образцов

Для записей образцов необходимо выполнить следующие преобразования:

- Нумерация точек принятия решений, активностей и образцов в RAO-XT начинается с 0, а в RAO-studio с 1, поэтому необходимо добавлять к данным номерам 1 при их выводе.

- В RAO-XT все типы образцов имеют нумерацию, в то время как в RAO-studio её имеют только образцы типа ‘operation’, поэтому нумерацию для образцов остальных типов необходимо игнорировать.

- Нумерация образцов в RAO-XT служит не только для того, чтобы связать начало и конец операции, но и как информативный вывод абсолютного номера выполнения образца в процессе прогона (номер действия). В RAO-studio для номера действия выбирался наименьший незанятый на данный момент другими действиями номер.

  Для выполнения преобразований номеров операций необходимо вести в процессе трассировки список свободных номеров действий и выводить номер действия из этого списка, а не из сериализованных данных.

- Число релевантных ресурсов паттернов в RAO-XT не сериализуется, поэтому необходимо получить эти данные из структуры модели.

5.4.4. Преобразование записей поиска по графу

Для записей поиска по графу необходимо выполнить следующие преобразования:

- Нумерация точек принятия решений, активностей и образцов в RAO-XT начинается с 0, а в RAO-studio с 1, поэтому необходимо добавлять к данным номерам 1 при их выводе.

- Нумерация вершин при поиске по графу в RAO-XT начинается с 0, а в RAO-studio с 1, поэтому необходимо добавлять к данным номерам 1 при их выводе.

- В начале поиска по графу в RAO-XT не сериализуется порождение и раскрытие корневой вершины, поэтому необходимо вручную добавить записи об этом

(которые всегда имеют одинаковые значения) после записи о начале поиска по графу.

5.4.5. Преобразование записей результатов

Для записей результатов необходимо лишь добавить единицу к номеру результата, т.к. в RAO-XT результаты нумеруются начиная с 0, а в RAO-studio – с 1.

5.5. Схема работы пользовательского интерфейса конфигурации трассировки

Конфигурация трассировки задается с помощью древовидной структуры, имеющей 4 фиксированных категории: Resources, Patterns, Decision points и Results, потомки которых, в свою очередь, наполняются именами соответствующего содержимого модели. Древовидная структура с помощью класса CheckboxTreeViewer выводится в пользовательском интерфейсе, где разработчик модели может задать, какие именно элементы модели нужно трассировать в данном прогоне.

Перед прогоном модели имена всех элементов, для которых включена трассировка, передаются в базу данных в список HashSet<String> sensitivityList. В процессе прогона перед добавлением в базу данных каждая запись проверяется на наличие в этом списке, и, если соответствующее имя в списке не обнаружено, запись не сериализуется.

Дерево конфигурации трассировки должно в каждый момент времени корректно отображать все элементы модели. Чтобы обеспечить это, необходимо, чтобы класс, отвечающий за отображение получал постоянные оповещения об изменении в модели.

Получение подобных оповещений реализуется с помощью анонимного класса, реализующего интерфейс IxtextModelListener[7], имеющий следующий вид:

```
public interface IXtextModelListener
{
   void modelChanged(XtextResource resource);
}
```

В методе `modelChanged` необходимо перестраивать дерево конфигурации трассировки в соответствии с актуальным содержанием модели.

В процессе редактирования модели разработчик может временно удалять и перемещать записи, что вызовет формирование для них новых элементов в дереве. Если одна и та же запись будет сначала удалена, а затем снова добавлена, то она будет считать новой записью и, по умолчанию, трассировка для неё будет отключена. Чтобы избежать потери желаемой конфигурации в процессе редактирования модели, была предложена следующая система:

- Каждый раз, когда дерево конфигурации трассировки перестраивается, т.е. при любом изменении в коде модели, все элементы прошлого дерева не удаляются, а становятся скрытыми.

- При формировании нового дерева имена элементов сначала проверяются на наличие в списке скрытых элементов дерева и, если они там есть, то вместо формирования нового элемента дерева, восстанавливается старый.

- При сохранении модели все скрытые элементы дерева удаляются, т.к. ожидается, что разработчик модели закончил текущие изменения. Таким образом обеспечивается удаление элементов, которые разработчик и не собирался восстанавливать.

Подобная система позволяет сохранять конфигурацию трассировки в процессе редактирования модели и, при этом, не хранить объекты, более не нужные.

6. Рабочий этап проектирования системы

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

6.1. Реализация читаемого вывода трассировки

Для реализации читаемого вывода перед началом трассировки считывается структура модели и необходимые для трассировки данные кэшируются в следующие контейнеры:

```
protected final HashMap<Integer, HashMap<Integer, String>> resourceNames =
   new HashMap<Integer, HashMap<Integer, String>>();
protected final ArrayList<ResourceTypeInfo> resourceTypesInfo =
   new ArrayList<ResourceTypeInfo>();
protected final ArrayList<ResultInfo> resultsInfo =
   new ArrayList<ResultInfo>();
protected final ArrayList<PatternInfo> patternsInfo =
   new ArrayList<PatternInfo>();
protected final ArrayList<DecisionPointInfo> decisionPointsInfo =
         new ArrayList<DecisionPointInfo>();
```

Данные контейнеры хранят информацию о структуре модели, которые позволяют определять, с одной стороны, данные необходимые для чтения сериализованных записей базы данных (число и типы релевантных ресурсов образца, связь образцов и активностей, типы параметров ресурса и т.п.), а с другой, обеспечивать читаемый вывод трассировочной информации (получать имена ресурсов, активностей, событий и др. по их номерам). Данные контейнеры заполняются методами класса ModelStructureHelper в момент создания трассировщика.

```
Tracer()
{
   ModelStructureHelper.fillResourceNames(resourceNames);
   ModelStructureHelper.fillResourceTypesInfo(resourceTypesInfo);
   ModelStructureHelper.fillResultsInfo(resultsInfo);
   ModelStructureHelper.fillPatternsInfo(patternsInfo);
   ModelStructureHelper.fillDecisionPointsInfo(decisionPointsInfo);
}
```

Для удобного формирования строк был написан класс RDOLibStringJoiner, позволяющий добавлять произвольно количество элементов во временную строку вывода, а затем получить эту строку при завершении добавления элементов, а также предоставляющий несколько встроенных форматов вывода строк (в виде структуры, в виде функции, в виде массива, в виде простого перечисления).

Встроенные форматы вывода определены в следующем enumerative классе:

```
enum StringFormat
{
   FUNCTION(", ", "(", ")"),
   STRUCTURE(", ", "{", "}"),
   ARRAY(", ", "[", "]"),
   ENUMERATION(", ", "", "");

   StringFormat(String delimiter, String prefix, String suffix)
   {
         this.delimiter = delimiter;
         this.prefix = prefix;
         this.suffix = suffix;
   }
```

```
    private final String delimiter;
    private final String prefix;
    private final String suffix;
}
```

Ниже приведен пример использования кэшированных данных и класса RDOLibStringJoiner при формировании вывода описания релевантных ресурсов образцов:

```
final RDOLibStringJoiner relResStringJoiner =
    new RDOLibStringJoiner(RDOLibStringJoiner.StringFormat.FUNCTION);

int numberOfRelevantResources = data.getInt();
for(int num = 0; num < numberOfRelevantResources; num++)
{
    final int resNum = data.getInt();
    final int typeNum =
            patternsInfo.get(patternNumber).relResTypes.get(num);
    final String typeName =
            resourceTypesInfo.get(typeNum).name;

    final String name = resourceNames.get(typeNum).get(resNum);
    final String resourceName =
            name != null ?
                    name :
                    typeName + encloseIndex(resNum);

    relResStringJoiner.add(resourceName);
}
```

В результате данных преобразовании из компактно сериализованных данных, состоящих из одного 4-байтового числа, в котором записано число релевантных ресурсов паттерна, и нескольких 4-байтовых чисел с номерами подобранных ресурсов, формируется подробная и удобная для восприятия человеком строка вывода, формат которой описан в пункте 4.1.2.

6.2. Реализация вывода трассировки в реальном времени

Для поддержания различных режимов моделирования трассировщик должен реализовывать интерфейс Subscriber и быть подписанным на изменения в базе данных или в состоянии прогона. Реализация метода fireChange() выглядит следующим образом:

```
@Override
public void fireChange()
{
    if (paused)
            return;

    parseNewEntries();
    notifyRealTimeSubscriber();
}
```

После обработки новых данных трассировщик должен оповестить об этом графический интерфейс отрисовки трассировочных данных, для чего трассировщик имеет указатели на два объекта типа Subscriber: один для вывода в реальном времени, и один для обновления таблицы в начале и в конце моделирования.

```
private Subscriber realTimeSubscriber = null;

public final void setRealTimeSubscriber(Subscriber subscriber)
```

```
{
    this.realTimeSubscriber = subscriber;
}

private final void notifyRealTimeSubscriber()
{
    if (realTimeSubscriber != null)
        realTimeSubscriber.fireChange();
}

private Subscriber commonSubscriber = null;

public final void setCommonSubscriber(Subscriber subscriber)
{
    this.commonSubscriber = subscriber;
}

public final void notifyCommonSubscriber()
{
    parseNewEntries();
    if (commonSubscriber != null)
        commonSubscriber.fireChange();
}
```

Для обеспечения возможности приостановить вывод трассировки было добавлено поле paused, значение которого проверяется перед началом чтения новых записей из базы данных и, в случае если трассировщик приостановлен, не производить это чтение. В случае, если трассировщик был снят с паузы, необходимо прочитать все новые записи в базе данных.

```
private boolean paused = true;

public final synchronized void setPaused(boolean paused)
{
    if (this.paused == paused)
        return;

    this.paused = paused;
    fireChange();
}
```

После оповещения графического интерфейса, последний производит обновление таблицы в соответствии с новыми данными трассировки.

6.3. Реализация вывода трассировки в формате RAO-studio

Одной из наиболее сложных задач описанного в пункте 5.4 класса LegacyTracer является приведение номеров ресурсов и активностей в соответствие с системой нумерации RAO-studio.

6.3.1. Приведение номеров ресурсов

Для реализации описанной в пункте 5.4 схемы приведения номеров ресурсов класс LegacyTracer хранит два списка:

```
private final HashMap<Integer, HashMap<Integer, Integer>> legacyResourceIds =
        new HashMap<Integer, HashMap<Integer, Integer>>();
private final TreeSet<Integer> takenResourceIds =
        new TreeSet<Integer>();
```

В данных списках хранятся, соответственно, данные о соответствии номеров ресурсов в базе данных и legacy-номеров и уже занятые legacy-номера.

При создании нового ресурса, ищется новый допустимый legacy-номер, который затем добавляется в оба списка.

```
private final int getNewResourceId(int typeNum, int resNum)
{
	int current;
	int legacyId = 1;
	Iterator<Integer> it = takenResourceIds.iterator();
	while (it.hasNext())
	{
		current = it.next();
		if (current != legacyId)
			break;
		legacyId++;
	}
	legacyResourceIds.get(typeNum).put(resNum, legacyId);
	takenResourceIds.add(legacyId);
	return legacyId;
}
```

При удалении ресурса, его legacy-номер также удаляется из обоих списков.

```
private final void freeResourceId(int typeNum, int resNum)
{
	int legacyId = legacyResourceIds.get(typeNum).get(resNum);
	legacyResourceIds.get(typeNum).remove(resNum);
	takenResourceIds.remove(legacyId);
}
```

При изменении значения ресурса для вывода используется его legacy-номер, получаемый из legacyResourceIds.

```
legacyId = legacyResourceIds.get(typeNum).get(resNum);
```

Таким образом для всех ресурсов выводятся именно такие номера, какие они имели бы в RAO-studio.

6.3.2. Приведение номеров выполнения активностей

Для реализации описанной в пункте 5.4 схемы приведения номеров выполнения активностей (номеров действий) класс LegacyTracer хранит два списка:

```
private final PriorityQueue<Integer> vacantActionNumbers =
	new PriorityQueue<Integer>();
private final HashMap<Integer, HashMap<
	Integer, HashMap<Integer, Integer>>> legacyActionNumbers =
	new HashMap<Integer, HashMap<Integer, HashMap<Integer, Integer>>>();
```

В данных списках хранятся, соответственно, номера не занятых действий и данные о соотвествии номеров действий базы данных и legacy-номеров действий.

В случае получения записи о начале выполнения операции ищется наименьший незанятый номер действия и найденный номер добавляется в список соответствия номеров.

```
HashMap<Integer, Integer> activityActions =
	legacyActionNumbers
	.get(dptNumber)
```

```
        .get(activityNumber);

int legacyNumber;
if (vacantActionNumbers.isEmpty())
        legacyNumber = activityActions.size();
else
        legacyNumber = vacantActionNumbers.poll();

activityActions.put(actionNumber, legacyNumber);
```

В случае получения записи об окончании операции legacy-номер ищется в списке соответствия но номеру из базы данных, а затем добавляется в список вакантных номеров.

```
HashMap<Integer, Integer> activityActions =
              legacyActionNumbers
              .get(dptNumber)
              .get(activityNumber);
int legacyNumber = activityActions.remove(actionNumber);
        vacantActionNumbers.add(legacyNumber);
```

Таким образом осуществляется вывод таких номеров действий для операций, каким он было бы в RAO-studio.

7. Апробирование разработанной системы в модельных условиях

7.1. Сравнение вывода трассировки RAO-XT в legacy-режиме и вывода трассировки RAO-studio

Для автоматизированного тестирования моделей необходимо в первую очередь проверить, что legacy-режим трассировки RAO-XT действительно выводит данные в правильном формате. Для этого была написана тестовая модель (приведена в Приложении 1), которая была запущена в обеих системах имитационного моделирования.

В результате было получено два файла: trace-output-RAO-studio.trc и trace-output-RAO-XT.trc. Затем файлы с трассировочной информации сравнивались с использованием средства сравнения файлов diff.

Т.к. файл трассировки, которую генерирует RAO-studio содержит помимо непосредственно вывода трассировки еще и дополнительню поясняющую информацию для пользователя, необходимо было сначала выбрать из него только те строки, которые относятся к трассировки. Это было выполнено с помощью программы sed следующей командой:

```
sed -n -e '1,/$Tracing/d;/$Status/q;p' trace-output-RAO-studio.trc > trace-output-RAO-studio-traceonly.trc
```

После чего сравнение файлов было выполнено следующей командой:

```
diff -s trace-output-RAO-XT.trc trace-output-RAO-studio-traceonly.trc
```

Результат выполнения команды:

```
Files trace-output-RAO-XT.trc and trace-output-RAO-studio-traceonly.trc are identical
```

Что свидетельствует о том, что оба файла трассировки посимвольно совпадают.

7.2. Сравнение производительности

Для оценки производительности нового модуля трассировки было произведено два теста:

- Тест на скорость добавления данных в вывод трассировки.
- Тест на скорость обновления в окне вывода трассировки.

Особый интерес представляли результаты обоих тестов в случае работы с большими объемами данных (от 100 000 записей).

7.2.1. Тест на скорость добавления данных в вывод трассировки.

Тест актуален для режима моделирования без анимации. Однако из-за отличий в схеме формирования трассировки в этом режиме (в RAO-studio формируется в процессе прогона, а в RAO-XT по окончани прогона), а также из-за различий в скорости прогона моделей, измерения было решено производить следующим образом:

- Для RAO-studio запустить простую модель с коротким временем прогона и малым количеством записей трассировки. Изменить код вывода трассировочной строки, означающей окончание трассировки, таким образом, чтобы последняя строка выводилась в цикле N раз. Время выполнения данного цикла и будет измеряться. Таким образом достигается независимость измерения скорости работы трассировки от скорости прогона моделей.

- Для RAO-XT поступить аналогично, с тем лишь исключением, что в замер необходимо включить не только время добавления записей в список, но и время перерисовки таблицы, т.к. при добавлении записей в трассировку RAO-studio это происходит сразу.

Таким образом для обоих систем производится замер скорости от момента, когда записи только начинают добавляться в трассировку и до момента, когда пользователь эти записи сможет увидеть.

Оба теста производились для 1000, 10000, 100000 и 1000000 (миллиона) записей.

Для проведения измерения скорости в RAO-studio код метода `RDOTrace::writeTraceEnd` был изменен следующим образом:

```
void RDOTrace::writeTraceEnd(const LPRDORuntime& pRuntime)
{
    if (isNull()) return;

    auto before = std::chrono::high_resolution_clock::now();
    unsigned count = 1000;
    for (unsigned i = 0; i < count; i++)
    {
        getOStream() << "ES " << pRuntime->getCurrentTime()
          << " 2" << std::endl << getEOL();
    }
    auto after = std::chrono::high_resolution_clock::now();
    std::cout << "diff = " <<
       std::chrono::duration_cast<std::chrono::nanoseconds>(
            after - before).count() << std::endl;
}
```

Значение переменной count для различных тестов изменялось, соответственно, на 1000, 10000, 100000 и 1000000.

Для проведения измерения скорости в RAO-XT в код трассировщика был добавлен метод makeFakeList(), имеющий следующий вид:

```
public final synchronized void makeFakeList()
{
    int count = 1000;
    for (int i = 0; i < count; i++)
    {
        traceList.add(new TraceOutput(TraceType.SYSTEM, "dummy string"));
    }
}
```

Данный метод вызывается из метода fireChange() объекта Subscriber commonUpdater окна вывода трассировки. Все измерения скорости проводятся в этом методе:

```
public static Subscriber commonUpdater =
        new Subscriber()
        {
            @Override
            public void fireChange()
            {
                long startTime = System.nanoTime();
                if (Simulator.getTracer().getTraceList().size() > 0)
                    Simulator.getTracer().makeFakeList();
                final ArrayList<TraceOutput> traceList =
                    Simulator.getTracer().getTraceList();
```

```
            final int size = traceList.size();
            PlatformUI.getWorkbench().getDisplay().asyncExec(
                new Runnable()
                {
                    @Override
                    public void run()
                    {
                        if (!readyForInput())
                            return;
                        RDOTraceView.viewer.setInput(traceList);
                        RDOTraceView.viewer.setItemCount(size);
                        viewer.refresh();
                    }
                }
            );
            long endTime = System.nanoTime();
            System.out.println("diff = " + (endTime - startTime));
        }
    };
```

Переменная count используется аналогичным образом.

После проведения ряда измерений были получены следующие результаты по времени добавления записей в таблицу (взяты средние значения по 6 измерениями и округлены до милисекунд). Результаты приведены в таблице 7.1.

	RAO-studio	RAO-XT
1000	10	1
10 000	109	7
100 000	1095	12
1 000 000	10818	59

Таблица 7.1 – Время добавления записей в окно вывода трассировки

Таким образом можно наблюдать, что для обоих систем характерна линейная сложность добавления данных в таблицу, однако скорость работы RAO-XT приблизительно на порядок выше для малого числа записей (до 100 000) и на два порядка выше для большого числа записей (от 100 000).

Данный тест показывает существенный прирост производительности в процессе добавления новых записей в трассировку для модуля трассировки RAO-XT по сравнению с трассировкой в RAO-studio.

7.2.2. Тест на скорость обновления данных в таблице

Тест проводится после окончания вывода трассировки и направлен в первую очередь на графический интерфейс и его производительность при отрисовке записей. В данном тесте сравнивается скорость обновления данных при прокрутке таблицы.

Для измерения скорости обновления в RAO-studio метод LogView::scrollVertically окна отрисовки трассировки был изменен следующим образом:

```
bool LogView::scrollVertically(int inc)
{
    if (!m_SM_Y.applyInc(inc))
```

```
    {
        return false;
    }
    auto before = std::chrono::high_resolution_clock::now();
    m_strings.setCursor(m_SM_Y.position, m_SM_Y.posMax);
    getVertScrollBar().setValue(m_SM_Y.position);
    update();
    auto after = std::chrono::high_resolution_clock::now();
    std::cout << "diff = " <<
        std::chrono::duration_cast<std::chrono::nanoseconds>(
            after - before).count() << std::endl;
    return true;
}
```

Для измерения скорости обновления в RAO-XT метод updateElement() класса RDOTraceViewContentProvider был изменен следующим образом:

```
@Override
public void updateElement(int index)
{
    long startTime = System.nanoTime();
    if (traceList != null && index < traceList.size())
        RDOTraceView.viewer.replace(traceList.get(index), index);
    long endTime = System.nanoTime();
    System.out.println("diff = " + (endTime - startTime));
}
```

После проведения ряда измерений были получены следующие результаты (взяты средние значения по 6 измерениями и округлены до микросекунд). Результаты приведены в таблице 7.2.

	RAO-studio	RAO-XT
1000	3	33
10 000	20	35
100 000	124	57
1 000 000	1147	46

Таблица 7.2. – Время обновления записей в окне вывода трассировки

Таким образом, можно наблюдать, что для RAO-studio характерная линейная сложность обновления данных в таблице, а для RAO-XT – константная. Выигрыш по алгоритмической сложности обеспечивает значительное преимущество модуля трассировки RAO-XT над системой трассировки в RAO-studio для больших объемов данных. При этом худшие показатели для малых объемов данных не играют большой роли, т.к. время, затрачиваемое на обновление данных в таблице измеряется в данном случае в десятках микросекунд.

Данный тест показывает существенный прирост производительности при обновлении записи в таблице при работе с большими объемами данных в систему RAO-XT по сравнению в RAO-studio.

8. Заключение

В рамках данного курсового проекта были получены следующие результаты:

- Был разработан и реализован модуль трассировки для системы имитационного моделирования RAO-XT. Модуль реализован таким образом, что другим компонентам системы не требуется использовать его вывод для своего функционирования.

- Был разработан и внедрен читаемый формат вывода трассировки.

- Было реализовано два режима вывода трассировки: в реальном времени и в конце моделирования.

- Был разработан и реализован графический интерфейс конфигурации трассировки. Конструкции для конфигурации трассировки через грамматику были удалены.

- Была реализована возможность эффективной работы модуля с большими объемами данных. Были проведены тесты производительности, результаты которых показали лучшие характеристики модуля трассировки RAO-XT по сравнению с системой трассировки в RAO-studio.

- Была разработана схема проведения автоматизированного тестирования моделей на основе трассировки. В систему был добавлен модуль, позволяющий реализовать данную схему. Была написана тестовая модель и проведено сравнение вывода трассировки RAO-studio и RAO-XT в legacy-режиме. Сравнение показало, что вывод обоих систем абсолютно идентичен.

Список используемых источников

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

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

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

4. **SWT Documentation** [https://www.eclipse.org/swt/docs.php]

5. **SWT Javadoc** [https://www.eclipse.org/swt/javadoc.php]

6. **ECMA-404 The JSON Data Interchange Standard** [http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf]

7. **Xtext Documentation** [http://eclipse.org/Xtext/documentation.html]

Список использованного программного обеспечения

1. RAO-Studio v2.3.1
2. Eclipse IDE for Java Developers Luna Service Release 1 (4.4.1)
3. openjdk version "1.8.0_40-internal"
4. NetBeans IDE 8.0.
5. ArgoUml v0.34
6. UMLet 13.1
7. Inkscape 0.48.4
8. Microsoft® Office Word 2010
9. Microsoft® Visio 2010
10. sed (GNU sed) 4.2.2
11. diff (GNU diffutils) 3.3

Приложение 1 – Исходный код модели для тестирования трассировки

```
$Resource_type enumOrigin: permanent
$Parameters
    origin : ( Тип1, Тип2 )
$End

$Resource_type Парикмахерские: permanent
$Parameters
    количество_в_очереди: real
$End

$Resource_type Клиенты : temporary
$Parameters
    тип      : such_as enumOrigin.origin
    состояние: ( Пришел, Начал_стрижку )
$End

$Resource_type Парикмахеры: permanent
$Parameters
    состояние_парикмахера : ( Свободен, Занят ) = Свободен
    количество_обслуженных: integer
    длительность_min      : integer
    длительность_max      : integer
    тип_клиента           : such_as Клиенты.тип
$End

$Resources
    Парикмахерская = Парикмахерские(0.0);
    Парикмахер_1 = Парикмахеры(*, 0, 20, 40, Тип1);
    Парикмахер_2 = Парикмахеры(*, 0, 25, 70, Тип2);
    Парикмахер_3 = Парикмахеры(*, 0, 30, 60, Тип2);
$End

$Pattern Образец_прихода_клиента : event
$Relevant_resources
    _Парикмахерская: Парикмахерская Keep
    _Клиент        : Клиенты        Create
$Body
_Парикмахерская:
    Convert_event
        Образец_прихода_клиента.planning( Time_now + 40 );
        количество_в_очереди++;

_Клиент:
    Convert_event
        if (_Парикмахерская.количество_в_очереди > 3)
            тип       = Тип1;
        else
            тип = Тип2;
        состояние = Пришел;
$End

$Pattern Образец_обслуживания_клиента : operation
$Relevant_resources
    _Парикмахерская: Парикмахерская Keep NoChange
    _Клиент        : Клиенты        Keep Erase
    _Парикмахер    : Парикмахеры    Keep Keep
$Time = 42
```

```
$Body
_Парикмахерская:
    Choice from _Парикмахерская.количество_в_очереди > 0
    Convert_begin
        количество_в_очереди--;

_Клиент:
    Choice from _Клиент.состояние == Пришел
    Convert_begin
        состояние = Начал_стрижку;

_Парикмахер:
    Choice from _Парикмахер.состояние_парикмахера == Свободен and
_Парикмахер.тип_клиента == _Клиент.тип
    with_min( _Парикмахер.количество_обслуженных )
    Convert_begin
        состояние_парикмахера  = Занят;
    Convert_end
        состояние_парикмахера  = Свободен;
        количество_обслуженных++;
$End

$Decision_point model: some
$Condition NoCheck
$Activities
    Обслуживание_клиента: Образец_обслуживания_клиента;
$End

$Simulation_run
    Образец_прихода_клиента.planning( Time_now + 40.6 );
    Terminate_if Time_now >= 8 * 60;
$End
```

Приложение 2 – Вывод трассировки модели из Приложения 1 в новом формате

```
ES 0.0 Tracing started
RC 0.0 Парикмахерская = {0.0}
RC 0.0 Парикмахер_1 = {Свободен, 0, 20, 40, Тип1}
RC 0.0 Парикмахер_2 = {Свободен, 0, 25, 70, Тип2}
RC 0.0 Парикмахер_3 = {Свободен, 0, 30, 60, Тип2}
ES 0.0 Simulation started
V  0.0 Занятость_парикмахера_1 = false
V  0.0 Занятость_парикмахера_2 = false
V  0.0 Занятость_парикмахера_3 = false
V  0.0 Длина_очереди = 0.0
EI 40.6 Образец_прихода_клиента[0](Парикмахерская, Клиенты[0])
RK 40.6 Парикмахерская = {1.0}
RC 40.6 Клиенты[0] = {Тип2, Пришел}
V  40.6 Длина_очереди = 1.0
EB 40.6 Обслуживание_клиента[0](Парикмахерская, Клиенты[0], Парикмахер_2)
RK 40.6 Парикмахерская = {0.0}
RK 40.6 Клиенты[0] = {Тип2, Начал_стрижку}
RK 40.6 Парикмахер_2 = {Занят, 0, 25, 70, Тип2}
V  40.6 Занятость_парикмахера_2 = true
V  40.6 Длина_очереди = 0.0
EI 80.6 Образец_прихода_клиента[1](Парикмахерская, Клиенты[1])
RK 80.6 Парикмахерская = {1.0}
RC 80.6 Клиенты[1] = {Тип2, Пришел}
V  80.6 Длина_очереди = 1.0
EB 80.6 Обслуживание_клиента[1](Парикмахерская, Клиенты[1], Парикмахер_3)
RK 80.6 Парикмахерская = {0.0}
RK 80.6 Клиенты[1] = {Тип2, Начал_стрижку}
RK 80.6 Парикмахер_3 = {Занят, 0, 30, 60, Тип2}
V  80.6 Занятость_парикмахера_3 = true
V  80.6 Длина_очереди = 0.0
EF 82.6 Обслуживание_клиента[0](Парикмахерская, Клиенты[0], Парикмахер_2)
RE 82.6 Клиенты[0] = {Тип2, Начал_стрижку}
RK 82.6 Парикмахер_2 = {Свободен, 1, 25, 70, Тип2}
V  82.6 Занятость_парикмахера_2 = false
EI 120.6 Образец_прихода_клиента[2](Парикмахерская, Клиенты[2])
RK 120.6 Парикмахерская = {1.0}
RC 120.6 Клиенты[2] = {Тип2, Пришел}
V  120.6 Длина_очереди = 1.0
EB 120.6 Обслуживание_клиента[2](Парикмахерская, Клиенты[2], Парикмахер_2)
RK 120.6 Парикмахерская = {0.0}
RK 120.6 Клиенты[2] = {Тип2, Начал_стрижку}
RK 120.6 Парикмахер_2 = {Занят, 1, 25, 70, Тип2}
V  120.6 Занятость_парикмахера_2 = true
V  120.6 Длина_очереди = 0.0
EF 122.6 Обслуживание_клиента[1](Парикмахерская, Клиенты[1], Парикмахер_3)
RE 122.6 Клиенты[1] = {Тип2, Начал_стрижку}
RK 122.6 Парикмахер_3 = {Свободен, 1, 30, 60, Тип2}
V  122.6 Занятость_парикмахера_3 = false
EI 160.6 Образец_прихода_клиента[3](Парикмахерская, Клиенты[3])
RK 160.6 Парикмахерская = {1.0}
RC 160.6 Клиенты[3] = {Тип2, Пришел}
V  160.6 Длина_очереди = 1.0
EB 160.6 Обслуживание_клиента[3](Парикмахерская, Клиенты[3], Парикмахер_3)
RK 160.6 Парикмахерская = {0.0}
RK 160.6 Клиенты[3] = {Тип2, Начал_стрижку}
```

```
RK 160.6 Парикмахер_3 = {Занят, 1, 30, 60, Тип2}
V  160.6 Занятость_парикмахера_3 = true
V  160.6 Длина_очереди = 0.0
EF 162.6 Обслуживание_клиента[2](Парикмахерская, Клиенты[2], Парикмахер_2)
RE 162.6 Клиенты[2] = {Тип2, Начал_стрижку}
RK 162.6 Парикмахер_2 = {Свободен, 2, 25, 70, Тип2}
V  162.6 Занятость_парикмахера_2 = false
EI 200.6 Образец_прихода_клиента[4](Парикмахерская, Клиенты[4])
RK 200.6 Парикмахерская = {1.0}
RC 200.6 Клиенты[4] = {Тип2, Пришел}
V  200.6 Длина_очереди = 1.0
EB 200.6 Обслуживание_клиента[4](Парикмахерская, Клиенты[4], Парикмахер_2)
RK 200.6 Парикмахерская = {0.0}
RK 200.6 Клиенты[4] = {Тип2, Начал_стрижку}
RK 200.6 Парикмахер_2 = {Занят, 2, 25, 70, Тип2}
V  200.6 Занятость_парикмахера_2 = true
V  200.6 Длина_очереди = 0.0
EF 202.6 Обслуживание_клиента[3](Парикмахерская, Клиенты[3], Парикмахер_3)
RE 202.6 Клиенты[3] = {Тип2, Начал_стрижку}
RK 202.6 Парикмахер_3 = {Свободен, 2, 30, 60, Тип2}
V  202.6 Занятость_парикмахера_3 = false
EI 240.6 Образец_прихода_клиента[5](Парикмахерская, Клиенты[5])
RK 240.6 Парикмахерская = {1.0}
RC 240.6 Клиенты[5] = {Тип2, Пришел}
V  240.6 Длина_очереди = 1.0
EB 240.6 Обслуживание_клиента[5](Парикмахерская, Клиенты[5], Парикмахер_3)
RK 240.6 Парикмахерская = {0.0}
RK 240.6 Клиенты[5] = {Тип2, Начал_стрижку}
RK 240.6 Парикмахер_3 = {Занят, 2, 30, 60, Тип2}
V  240.6 Занятость_парикмахера_3 = true
V  240.6 Длина_очереди = 0.0
EF 242.6 Обслуживание_клиента[4](Парикмахерская, Клиенты[4], Парикмахер_2)
RE 242.6 Клиенты[4] = {Тип2, Начал_стрижку}
RK 242.6 Парикмахер_2 = {Свободен, 3, 25, 70, Тип2}
V  242.6 Занятость_парикмахера_2 = false
EI 280.6 Образец_прихода_клиента[6](Парикмахерская, Клиенты[6])
RK 280.6 Парикмахерская = {1.0}
RC 280.6 Клиенты[6] = {Тип2, Пришел}
V  280.6 Длина_очереди = 1.0
EB 280.6 Обслуживание_клиента[6](Парикмахерская, Клиенты[6], Парикмахер_2)
RK 280.6 Парикмахерская = {0.0}
RK 280.6 Клиенты[6] = {Тип2, Начал_стрижку}
RK 280.6 Парикмахер_2 = {Занят, 3, 25, 70, Тип2}
V  280.6 Занятость_парикмахера_2 = true
V  280.6 Длина_очереди = 0.0
EF 282.6 Обслуживание_клиента[5](Парикмахерская, Клиенты[5], Парикмахер_3)
RE 282.6 Клиенты[5] = {Тип2, Начал_стрижку}
RK 282.6 Парикмахер_3 = {Свободен, 3, 30, 60, Тип2}
V  282.6 Занятость_парикмахера_3 = false
EI 320.6 Образец_прихода_клиента[7](Парикмахерская, Клиенты[7])
RK 320.6 Парикмахерская = {1.0}
RC 320.6 Клиенты[7] = {Тип2, Пришел}
V  320.6 Длина_очереди = 1.0
EB 320.6 Обслуживание_клиента[7](Парикмахерская, Клиенты[7], Парикмахер_3)
RK 320.6 Парикмахерская = {0.0}
RK 320.6 Клиенты[7] = {Тип2, Начал_стрижку}
RK 320.6 Парикмахер_3 = {Занят, 3, 30, 60, Тип2}
```

```
V  320.6 Занятость_парикмахера_3 = true
V  320.6 Длина_очереди = 0.0
EF 322.6 Обслуживание_клиента[6](Парикмахерская, Клиенты[6], Парикмахер_2)
RE 322.6 Клиенты[6] = {Тип2, Начал_стрижку}
RK 322.6 Парикмахер_2 = {Свободен, 4, 25, 70, Тип2}
V  322.6 Занятость_парикмахера_2 = false
EI 360.6 Образец_прихода_клиента[8](Парикмахерская, Клиенты[8])
RK 360.6 Парикмахерская = {1.0}
RC 360.6 Клиенты[8] = {Тип2, Пришел}
V  360.6 Длина_очереди = 1.0
EB 360.6 Обслуживание_клиента[8](Парикмахерская, Клиенты[8], Парикмахер_2)
RK 360.6 Парикмахерская = {0.0}
RK 360.6 Клиенты[8] = {Тип2, Начал_стрижку}
RK 360.6 Парикмахер_2 = {Занят, 4, 25, 70, Тип2}
V  360.6 Занятость_парикмахера_2 = true
V  360.6 Длина_очереди = 0.0
EF 362.6 Обслуживание_клиента[7](Парикмахерская, Клиенты[7], Парикмахер_3)
RE 362.6 Клиенты[7] = {Тип2, Начал_стрижку}
RK 362.6 Парикмахер_3 = {Свободен, 4, 30, 60, Тип2}
V  362.6 Занятость_парикмахера_3 = false
EI 400.6 Образец_прихода_клиента[9](Парикмахерская, Клиенты[9])
RK 400.6 Парикмахерская = {1.0}
RC 400.6 Клиенты[9] = {Тип2, Пришел}
V  400.6 Длина_очереди = 1.0
EB 400.6 Обслуживание_клиента[9](Парикмахерская, Клиенты[9], Парикмахер_3)
RK 400.6 Парикмахерская = {0.0}
RK 400.6 Клиенты[9] = {Тип2, Начал_стрижку}
RK 400.6 Парикмахер_3 = {Занят, 4, 30, 60, Тип2}
V  400.6 Занятость_парикмахера_3 = true
V  400.6 Длина_очереди = 0.0
EF 402.6 Обслуживание_клиента[8](Парикмахерская, Клиенты[8], Парикмахер_2)
RE 402.6 Клиенты[8] = {Тип2, Начал_стрижку}
RK 402.6 Парикмахер_2 = {Свободен, 5, 25, 70, Тип2}
V  402.6 Занятость_парикмахера_2 = false
EI 440.6 Образец_прихода_клиента[10](Парикмахерская, Клиенты[10])
RK 440.6 Парикмахерская = {1.0}
RC 440.6 Клиенты[10] = {Тип2, Пришел}
V  440.6 Длина_очереди = 1.0
EB 440.6 Обслуживание_клиента[10](Парикмахерская, Клиенты[10], Парикмахер_2)
RK 440.6 Парикмахерская = {0.0}
RK 440.6 Клиенты[10] = {Тип2, Начал_стрижку}
RK 440.6 Парикмахер_2 = {Занят, 5, 25, 70, Тип2}
V  440.6 Занятость_парикмахера_2 = true
V  440.6 Длина_очереди = 0.0
EF 442.6 Обслуживание_клиента[9](Парикмахерская, Клиенты[9], Парикмахер_3)
RE 442.6 Клиенты[9] = {Тип2, Начал_стрижку}
RK 442.6 Парикмахер_3 = {Свободен, 5, 30, 60, Тип2}
V  442.6 Занятость_парикмахера_3 = false
EI 480.6 Образец_прихода_клиента[11](Парикмахерская, Клиенты[11])
RK 480.6 Парикмахерская = {1.0}
RC 480.6 Клиенты[11] = {Тип2, Пришел}
V  480.6 Длина_очереди = 1.0
ES 480.6 Simulation finished: terminate condition
```

Приложение 3 – Вывод трассировки модели из Приложения 1 в legacy-формате

```
ES 0 1
RK 0 2 1 0
RK 0 4 2 0 0 20 40 0
RK 0 4 3 0 0 25 70 1
RK 0 4 4 0 0 30 60 1
ES 0 3
V  0 1  FALSE
V  0 2  FALSE
V  0 3  FALSE
V  0 7  0
EI 40.6 1 1 2  1 5
RK 40.6 2 1 1
RC 40.6 3 5 1 0
V  40.6 7  1
EB 40.6 1 1 2 3  1 5 3
RK 40.6 2 1 0
RK 40.6 3 5 1 1
RK 40.6 4 3 1 0 25 70 1
V  40.6 2  TRUE
V  40.6 7  0
EI 80.6 1 1 2  1 6
RK 80.6 2 1 1
RC 80.6 3 6 1 0
V  80.6 7  1
EB 80.6 2 1 2 3  1 6 4
RK 80.6 2 1 0
RK 80.6 3 6 1 1
RK 80.6 4 4 1 0 30 60 1
V  80.6 3  TRUE
V  80.6 7  0
EF 82.6 1 1 2 3  1 5 3
RE 82.6 3 5 1 1
RK 82.6 4 3 0 1 25 70 1
V  82.6 2  FALSE
EI 120.6 1 1 2  1 5
RK 120.6 2 1 1
RC 120.6 3 5 1 0
V  120.6 7  1
EB 120.6 1 1 2 3  1 5 3
RK 120.6 2 1 0
RK 120.6 3 5 1 1
RK 120.6 4 3 1 1 25 70 1
V  120.6 2  TRUE
V  120.6 7  0
EF 122.6 2 1 2 3  1 6 4
RE 122.6 3 6 1 1
RK 122.6 4 4 0 1 30 60 1
V  122.6 3  FALSE
EI 160.6 1 1 2  1 6
RK 160.6 2 1 1
RC 160.6 3 6 1 0
V  160.6 7  1
EB 160.6 2 1 2 3  1 6 4
RK 160.6 2 1 0
RK 160.6 3 6 1 1
```

```
RK 160.6 4 4 1 1 30 60 1
V  160.6 3  TRUE
V  160.6 7  0
EF 162.6 1 1 2 3  1 5 3
RE 162.6 3 5 1 1
RK 162.6 4 3 0 2 25 70 1
V  162.6 2  FALSE
EI 200.6 1 1 2  1 5
RK 200.6 2 1 1
RC 200.6 3 5 1 0
V  200.6 7  1
EB 200.6 1 1 2 3  1 5 3
RK 200.6 2 1 0
RK 200.6 3 5 1 1
RK 200.6 4 3 1 2 25 70 1
V  200.6 2  TRUE
V  200.6 7  0
EF 202.6 2 1 2 3  1 6 4
RE 202.6 3 6 1 1
RK 202.6 4 4 0 2 30 60 1
V  202.6 3  FALSE
EI 240.6 1 1 2  1 6
RK 240.6 2 1 1
RC 240.6 3 6 1 0
V  240.6 7  1
EB 240.6 2 1 2 3  1 6 4
RK 240.6 2 1 0
RK 240.6 3 6 1 1
RK 240.6 4 4 1 2 30 60 1
V  240.6 3  TRUE
V  240.6 7  0
EF 242.6 1 1 2 3  1 5 3
RE 242.6 3 5 1 1
RK 242.6 4 3 0 3 25 70 1
V  242.6 2  FALSE
EI 280.6 1 1 2  1 5
RK 280.6 2 1 1
RC 280.6 3 5 1 0
V  280.6 7  1
EB 280.6 1 1 2 3  1 5 3
RK 280.6 2 1 0
RK 280.6 3 5 1 1
RK 280.6 4 3 1 3 25 70 1
V  280.6 2  TRUE
V  280.6 7  0
EF 282.6 2 1 2 3  1 6 4
RE 282.6 3 6 1 1
RK 282.6 4 4 0 3 30 60 1
V  282.6 3  FALSE
EI 320.6 1 1 2  1 6
RK 320.6 2 1 1
RC 320.6 3 6 1 0
V  320.6 7  1
EB 320.6 2 1 2 3  1 6 4
RK 320.6 2 1 0
RK 320.6 3 6 1 1
RK 320.6 4 4 1 3 30 60 1
```

```
V  320.6 3  TRUE
V  320.6 7  0
EF 322.6 1 1 2 3  1 5 3
RE 322.6 3 5 1 1
RK 322.6 4 3 0 4 25 70 1
V  322.6 2  FALSE
EI 360.6 1 1 2  1 5
RK 360.6 2 1 1
RC 360.6 3 5 1 0
V  360.6 7  1
EB 360.6 1 1 2 3  1 5 3
RK 360.6 2 1 0
RK 360.6 3 5 1 1
RK 360.6 4 3 1 4 25 70 1
V  360.6 2  TRUE
V  360.6 7  0
EF 362.6 2 1 2 3  1 6 4
RE 362.6 3 6 1 1
RK 362.6 4 4 0 4 30 60 1
V  362.6 3  FALSE
EI 400.6 1 1 2  1 6
RK 400.6 2 1 1
RC 400.6 3 6 1 0
V  400.6 7  1
EB 400.6 2 1 2 3  1 6 4
RK 400.6 2 1 0
RK 400.6 3 6 1 1
RK 400.6 4 4 1 4 30 60 1
V  400.6 3  TRUE
V  400.6 7  0
EF 402.6 1 1 2 3  1 5 3
RE 402.6 3 5 1 1
RK 402.6 4 3 0 5 25 70 1
V  402.6 2  FALSE
EI 440.6 1 1 2  1 5
RK 440.6 2 1 1
RC 440.6 3 5 1 0
V  440.6 7  1
EB 440.6 1 1 2 3  1 5 3
RK 440.6 2 1 0
RK 440.6 3 5 1 1
RK 440.6 4 3 1 5 25 70 1
V  440.6 2  TRUE
V  440.6 7  0
EF 442.6 2 1 2 3  1 6 4
RE 442.6 3 6 1 1
RK 442.6 4 4 0 5 30 60 1
V  442.6 3  FALSE
EI 480.6 1 1 2  1 6
RK 480.6 2 1 1
RC 480.6 3 6 1 0
V  480.6 7  1
ES 480.6 2
```