Оглавление

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

График – диаграмма, изображающая при помощи кривых количественные показатели развития, состояния модели

Подсистема визуализации – подсистема среды RAO-XT, отвечающая за построение графиков изменения состояния модели

Парсинг – процесс анализа последовательности входных данных и преобразование их в необходимых формат

Парсер – компонент, выполняющий парсинг данных

Сериализация – процесс перевода какой-либо структуры данных в последовательность битов

Слайдер – элемент графического интерфейса, позволяющий отображать область окна, соответствующую его положению

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.

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. Графики в системе имитационного моделирования RAO-studio[2]

Имеется возможность визуально отображать процессы, происходящие в модели, в виде графиков. Добавление графиков состояния ресурса становится возможным только после запуска модели. Графики состояния ресурса можно добавлять как в процессе работы модели в режиме анимации, так и после завершения моделирования. График состояния ресурса может быть отображен только в том случае, если конкретный ресурс трассируется. Для добавления графика во вкладке Графики окна объектов в дереве модели следует найти необходимый вам параметр ресурса. Далее, чтобы создать новое окно графика, нужно щелкнуть два раза по выбранному параметру ресурса левой кнопкой мыши или один раз правой и в выпадающем меню выбрать команду Добавить на новый график. По оси абсцисс графика откладывается время наступления событий, при которых изменяется значение параметра выбранного ресурса.

При первом построении графика, масштаб по оси ординат устанавливается автоматически. Для изменения масштаба по оси абсцисс можно воспользоваться панелью инструментов Масштаб или использовать выпадающее меню при правом щелчке мышки по области графика. Основными функциями масштабирования являются:

- Увеличить масштаб – увеличение масштаба
- Уменьшить масштаб – уменьшение масштаба
- Автоматический масштаб – автоматический подбор масштаба по ширине области графика
- Восстановить масштаб – восстановление начального масштаба

Имеется возможность добавлять на уже существующий график другие. Для этого необходимо мышкой перетащить имя параметра ресурса на область уже построенного графика.

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 строить графики.

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.

При проведении имитационных экспериментов необходимо иметь возможность следить за изменениями состояния модели. Одним из предпочтительных способов визуализации изменений является построение графиков по параметрам модели.

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» Зудина О.В.

Наименование темы разработки: «Разработка подсистемы визуализации параметров модели для системы имитационного моделирования RAO-XT»

3.3. Назначение разработки

Разработать подсистему визуализации параметров модели для системы имитационного моделирования RAO-XT.

3.4. Требования к программе или программному изделию

3.4.1. Требования к функциональным характеристикам

Подсистема визуализации должна удовлетворять следующим требованиям:

- Построение по различным параметрам модели:
  - по параметрам ресурсов
  - выполняемым образцам

    - результатам
- Построение графиков в процессе прогона
- Построение ступенчатых графиков
- Построение по различным типам данных
    - *integer*
    - *real*
    - *enum*
- Масштабирование
- Независимость от модуля трассировки

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 2 Гб
- объем жесткого диска не менее 50 Гб
- микропроцессор с тактовой частотой не менее 2ГГц
- монитор с разрешением от 1366*768 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 – 7 февраля 2015г.

Плановый срок окончания разработки – 25 мая 2015г.

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 ручного тестирования подсистемы визуализации.

4. Концептуальный этап проектирования подсистемы

На концептуальном этапе проектирования требовалось:

- подобрать графическую библиотеку, удовлетворяющую требованиям
- разработать пользовательский интерфейс вызова подсистемы визуализации из среды RAO-XT
- обеспечить возможность построения графиков по определённым параметрам модели

4.1. Графическая библиотека

4.1.1. Выбор графической библиотеки

Были рассмотрены следующие графические библиотеки:

- JavaFX
- JFreeChart
- JGraphX
- SWTChart

Таблица 4.1. Сравнение графических библиотек

	JavaFX	JFreeChart	JGraphX	SWTChart
Совместимость с Java 8	-	+	+	+
Встраивание в среду разработки Eclipse	+	+	-	+
Построение графиков в "реальном времени"	+	+	-	-
Прореживание данных	+	+	+	+
Масштабирование	+	+	+	+
Работа с большими объёмами данных	+	+	+	+
Отображение нескольких графиков в одном окне	+	+	-	+

По результатам сравнения была выбрана графическая библиотека JFreeChart.

4.1.2. Выбор оболочки для графической библиотеки

Графическая библиотека JFreeChart имеет возможность использования с различными оболочками:

- SWT[4]
- AWT[5]
- Bridge

Ниже представлены результаты измерений быстродействия JFreeChart с различными оболочками на различных операционных системах. Данные представлены в секундах.

Таблица 4.2. Измерение быстродействия без прореживания данных

Количество точек графика	**Оболочка**	**Операционная система**					
		Ubuntu 14.10			**Windows 7**		
1000000	SWT	28.33	27.80	27.54	41.95	41.04	42.64
	AWT	9.54	8.79	8.89	11.03	11.25	11.16
	Bridge	7.78	8.18	8.67	5.93	8.73	7.66
2000000	SWT	55.86	58.42	55.00	x	x	x
	AWT	19.79	19.61	19.56	22.84	23.46	23.43
	Bridge	15.47	15.15	14.91	13.49	16.08	18.06
2500000	SWT	67.31	70.08	69.45	-	-	-
	AWT	24.85	24.99	25.09	-	-	-
	Bridge	25.41	20.93	19.72	-	-	-
3000000	SWT	79.30	84.24	83.49	-	-	-
	AWT	30.71	30.48	30.20	-	-	-
	Bridge	22.51	22.90	22.37	-	-	-
3500000	SWT	88.89	85.03	89.96	-	-	-
	AWT	34.47	34.55	34.66	-	-	-
	Bridge	24.85	24.62	27.40	-	-	-
5000000	SWT	x	x	x	-	-	-
	AWT	x	x	x	-	-	-
	Bridge	318.85	159.09	164.86	-	-	-

Таблица 4.3. Измерение быстродействия с прореживанием данных

Количество точек графика	Оболочка	Операционная система					
		Ubuntu 14.10			Windows 7		
1000000	SWT	2.45	2.50	2.45	2.99	2.97	3.16
	AWT	2.43	2.44	2.45	2.73	2.70	2.75
	Bridge	2.12	2.17	2.15	3.33	3.54	3.40
2000000	SWT	4.17	4.03	4.13	5.27	5.03	5.35
	AWT	4.24	4.02	4.08	4.65	4.63	4.65
	Bridge	4.29	4.02	4.19	5.40	5.38	5.43
5000000	SWT	5.83	5.93	5.90	7.21	6.96	7.18
	AWT	5.98	5.89	6.04	6.65	10.45	6.66
	Bridge	8.76	8.76	8.78	7.42	7.29	7.28
10000000	SWT	21.30	22.03	21.69	20.70	26.69	24.64
	AWT	21.27	21.17	21.26	24.49	23.21	25.48
	Bridge	21.46	21.18	20.72	25.62	24.77	25.77

Результаты измерений показали, что ни одна из оболочек не обеспечивает значительного преимущества по быстродействию по сравнению с остальными. В связи с этим была выбрана оболочка SWT, позволяющая обеспечить наиболее тесную интеграцию со средой разработки Eclipse.

4.2. Пользовательский интерфейс вызова подсистемы визуализации

Был разработан пользовательский интерфейс вызова подсистемы визуализации из среды RAO-XT. Вызов подсистемы производится с использованием дерева сериализованных объектов. В дереве перечислены все параметры модели, о которых есть записи в базе данных. При клике правой кнопкой мыши по выбранному элементу дерева происходит вызов выпадающего меню. В зависимости от типа параметра модели происходит вывод активного или неактивного меню построения графика.

4.3. Построение графиков по определённым параметрам модели

Были определены параметры модели, по которым необходимо построение графиков:

- параметры ресурсов типов:
  - *integer*
  - *real*
  - *enum*

- результаты типов:
    - *watch_par*
    - *watch_state*
    - *watch_quant*
    - *watch_value*
- образцы типа *operation* – график строится по количеству одновременно запущенных образцов

5. Технический этап проектирования подсистемы

5.1. Построение графика по сериализованным данным

Библиотечная часть подсистемы визуализации, основным компонентом которой является класс парсера (*PlotDataParser*), взаимодействует непосредственно с содержимым базы данных и создает массив точек графика.

В качестве входных данных парсер имеет элемент дерева сериализованных объектов и содержимое базы данных. В элементе дерева хранятся номера записей в базе данных, которые соответствуют параметру, график которого должен быть построен. Зная правила сериализации данных, парсер считывает нужное количество байт из записи, преобразовывает их в переменную требуемого типа, подготавливает координаты точки и создает массив на основе считанных данных.

5.1.1. Получение данных графика параметра ресурса

Парсер находит запись ресурса, которому принадлежит данный параметр. Формат бинарной сериализации позволяет получить значения любого параметра без необходимости парсить всю запись целиком. В элементе дерева хранится смещение, в соответствии с которым расположено значение данного параметра, и число байт, которое данный параметр занимает. Это происходит в методе *parseResourceParameter()*.

5.1.2. Получение данных графика результата

В методе *parseResult()* парсер считывает только значение результата по его номеру без необходимости парсить всю запись целиком.

5.1.3. Получение данных графика образца

В методе *parsePattern()* заводится счётчик одновременно запущенных операций. Парсер считывает только тип записи и в зависимости от типа изменяет значение счётчика. Для записей начала операции (*operation_begin*) счётчик увеличивается на единицу, для записей окончания операции (*operation_end*) – уменьшается на единицу.

5.2. Построение графика в процессе моделирования

Для обновления в процессе моделирования используется таймер. Каждый раз при срабатывании таймера для каждого открытого графика проверяется наличие в базе данных новых записей. Если появились новые записи, то происходит вызов метода *UpdateAllOpenedCharts()*. Парсер считывает только

новые записи для соответствующего параметра, для чего хранится номер последней считанной записи.

5.3. Построение графика данных перечислимого типа

В графической части подсистемы визуализации вызывается метод *getEnumNames()* библиотечной части подсистемы. Если данные, по которым строится график, имеют тип *enum*, то имена значений сохраняются в массив. На основе массива в методе *createChart()* графической части подсистемы назначается тип вертикальной оси графика, значениями шкалы которой являются имена.

5.4. Масштабирование и навигация по графику

5.4.1. Масштабирование графика

Изменение масштаба по горизонтальной оси производится при выделении требуемой области графика слева направо.

Изменение масштаба по вертикальной и горизонтальной осям производится при одновременной нажатии клавиши Shift и выделении требуемой области графика слева направо.

Для отображения графика целиком необходимо выделить любую область графика справа налево.

5.4.2. Навигация по графику

Навигация по графику происходит с помощью вертикального и горизонтального слайдеров.

Размеры и положение слайдеров зависят от масштабирования, поэтому при каждом изменении масштаба графика необходимо переопределять данные характеристики слайдеров. Это выполняется в методах *setSliders()* и *setSlidersMaximum()* в графической части подсистемы визуализации.

5.5. Жизненный цикл окна графика

5.5.1. Создание окна

При построении нового графика создается объект *RDOPlotView*, идентификационный номер которого вместе с информацией о параметре сохраняется в контейнер *openedPlotMap* типа Map[6].

При попытке построить график проверяется наличие информации об этом графике в контейнере *openedPlotMap*. Если информация найдена, то объект

RDOPlotView не создается, а уже созданное соответствующее окно становится активным.

5.5.2. Закрытие окна

При закрытии окна графика информация о нём удаляется из контейнера *openedPlotMap*. Также удаляется информация о последней построенной точке графика.

При закрытии среды RAO-XT закрываются все открытые окна графиков.

6. Рабочий этап проектирования подсистемы

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

6.1. Реализация построения графика по сериализованным данным

Точка графика представлена в виде объекта класса *PlotItem*, имеющего два поля – x и y координаты точки.

```
public final static class PlotItem {
	PlotItem(final double x, final double y) {
		this.x = x;
		this.y = y;
	}

	final public double x;
	final public double y;
}
```

Для создания массива значений графика необходимо получить информацию о выделенном элементе дерева сериализуемых объектов:

```
final CollectedDataNode node = (CollectedDataNode) serializedObjectsTreeViewer
		.getTree().getSelection()[0].getData();
final AbstractIndex index = node.getIndex();
```

В объекте типа *AbstractIndex* хранится номер и тип элемента дерева. В зависимости от типа элемента вызывается один из методов парсинга данных:

```
case RESOURCE_PARAMETER:
	final ResourceParameterIndex resourceParameterIndex = (ResourceParameterIndex)
index;
	final CollectedDataNode resourceNode = node.getParent();
	final ResourceIndex resourceIndex = (ResourceIndex) resourceNode
			.getIndex();
	return parseResourceParameter(resourceParameterIndex,
			resourceIndex, startItemNumber);
```

В каждом из методов создается массив для записи точек графика и считываются номера всех записей из базы данных, относящихся к данному элементу:

```
final List<Integer> entriesNumbers = index.getEntryNumbers();
final List<Entry> allEntries = Simulator.getDatabase().getAllEntries();
```

Для получения координаты x считывается значение модельного времени:

```
final double time = header.getDouble();
```

В случае считывания значения результата необходимо получить его номер:

```
final int resultNum = header.getInt();
```

По номеру результата получить тип значения и в зависимости от типа значения считать нужное количество байт для получения координаты y:

```
case INTEGER:
	item = new PlotItem(time, data.getInt());
	break;
```

Считывания значения параметра ресурса происходит подобным образом. В случае получения данных для образцов в качестве координаты y используется значение счётчика одновременно запущенных операций.

В результате преобразований из компактно сериализованных данных, состоящих из одного 4-байтового числа формируется массив координат точек графика.

6.2. Реализация построения графика в процессе моделирования

Для оптимизации получения координат точек графика в процессе моделирования необходимо хранить в контейнере номер последней считанной записи из базы данных. Это позволяет не перестраивать каждый раз график с начальной точки, а продолжать построение с последующей точки:

```
private static final Map<AbstractIndex, Integer> lastItemMap = new HashMap<AbstractIndex, Integer>();
```

В случае построения графика образца также в контейнере хранится текущее значение счётчика одновременно запущенных операций, так как число одновременно запущенных операций не хранится в базе данных:

```
private static final Map<AbstractIndex, Integer> lastPatternCountMap = new HashMap<AbstractIndex, Integer>();
```

Если информация о графике хранится в контейнере, то построение происходит с последней записанной точки:

```
if (lastItemMap.containsKey(index)) {
	startItemNumber = lastItemMap.get(index);
}
```

6.3. Реализация масштабирования и навигации по графику

6.3.1. Реализация масштабирования графика

Функция масштабирования реализуется переопределением двух стандартных классов графической библиотеки JFreeChart. По умолчанию масштабирование происходит только по горизонтальной оси. При нажатии клавиши Shift становится возможным масштабирование по двум осям, для чего в классе *RDOChartComposite* переопределяется класс *keyPressed:*

```
@Override
public void keyPressed(KeyEvent e) {
	if (e.keyCode == SWT.SHIFT) {
		this.setRangeZoomable(true);
	}
}
```

В момент, когда клавиша Shift не нажата, масштабирование по вертикальной оси снова становится невозможным:

```
@Override
public void keyReleased(KeyEvent e) {
	if (e.keyCode == SWT.SHIFT) {
		this.setRangeZoomable(false);
	}
}
```

6.3.2. Реализация навигации по графику

Для реализации навигации по графику объекту класса RDOChartComposite необходимо назначить объекты класса Slider. В зависимости от текущего масштаба объектам Slider назначаются координата, интервал перемещения, и, соответственно, ширина или высота:

```
horizontalSlider
		.setThumb((int) (plot.getDomainAxis().getUpperBound() - plot
			.getDomainAxis().getLowerBound()));
horizontalSlider.setSelection((int) (plot.getDomainAxis()
		.getLowerBound()));
```

При перемещении слайдера необходимо задавать отображаемый интервал графика:

```
getChart().getXYPlot().getDomainAxis()
	.setLowerBound(horizontalSlider.getSelection());
getChart()
	.getXYPlot()
	.getDomainAxis()
	.setUpperBound(
		horizontalSlider.getThumb()
			+ horizontalSlider.getSelection());
```

6.4. Реализация создания окна графика

При открытии окна графика информация о нём заносится в контейнер *openedPlotMap* и вызывается метод *showView()*, при этом значение *SecondaryID* увеличивается на единицу. *SecondaryID* – уникальный номер открытого окна. Также окну присваивается имя, соответствующее имени параметра, по которому строится график:

```
final RDOPlotView newView = (RDOPlotView) PlatformUI
	.getWorkbench()
	.getActiveWorkbenchWindow()
	.getActivePage()
	.showView(RDOPlotView.ID, String.valueOf(secondaryID),
```

```
                IWorkbenchPage.VIEW_ACTIVATE);
newView.setName(String.valueOf(dataset.getSeriesKey(0)));
RDOPlotView.addToOpenedPlotMap(node, secondaryID);
secondaryID++;
```

При наличии в контейнере информации о графике новое окно не создается, а становится активным уже открытое:

```
if (RDOPlotView.getOpenedPlotMap().containsKey(node)) {
      PlatformUI
                  .getWorkbench()
                  .getActiveWorkbenchWindow()
                  .getActivePage()
                  .showView(
                              RDOPlotView.ID,
                              String.valueOf(RDOPlotView
                                    .getOpenedPlotMap().get(node)),
                              IWorkbenchPage.VIEW_ACTIVATE);
}
```

Удаление из контейнера информации о графике реализуется переопределением метода *widgetDisposed()*:

```
@Override
public void widgetDisposed(DisposeEvent event) {
      if (!openedPlotMap.isEmpty()
                  && openedPlotMap.containsKey(partNode)) {
            openedPlotMap.remove(partNode);
      }
      if (partNode != null) {
            PlotDataParser.removeIndexFromMaps(partNode.getIndex());
      }
}
```

7. Апробирование разработанной подсистемы в модельных условиях

Апробирование разработанной подсистемы визуализации осуществлялось при помощи многократного тестирования функционала. В ходе тестирования были построены графики: по всем необходимым параметрам модели (параметры ресурсов, результаты, образцы), по всем допустимым типам данных (integer, real, boolean, enum), данных перечислимого типа (enum) с большим количеством значений. Модели, использованные для тестирования приведены в Приложении 1 и Приложении 2. Также тестировалось построение графиков по окончанию моделирования и в процессе прогона модели.

Выявленные в процессе тестирования ошибки и недочеты были исправлены на этапе рабочего проектирования.

8. Заключение

В рамках данного курсового проекта были получены следующие результаты:

- На этапе концептуального проектирования был сделан обзор существующих графических библиотек и выбрана библиотека JFreeChart, в наиболее полной мере удовлетворяющая всем требованиям к подсистеме.
- Была разработана и реализована подсистема визуализации параметров модели для системы имитационного моделирования RAO-XT.
- Было реализовано два режима построения графиков: по окончанию моделирования и в процессе прогона модели.
- Было реализовано построение графиков по всем необходимым параметрам.
- Было реализовано построение графиков по всем допустимым типам данных.
- Были реализованы масштабирование и навигация по графику.
- Был разработан и реализован графический интерфейс для запуска подсистемы визуализации из среды RAO-XT.

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. **AWT Documentation** [https://docs.oracle.com/javase/8/docs/api/]
6. **Map Documentation** [ps://docs.oracle.com/javase/8/docs/api]

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. UMLet 13.1
5. Inkscape 0.48.4
6. Microsoft® Office Word 2010
7. Microsoft® Office Excel 2010
8. Microsoft® Visio 2013

Приложение 1 – Исходный код модели для тестирования построения графиков

```
$Resource_type Парикмахерские: permanent
$Parameters
	количество_в_очереди: integer
$End

$Resource_type Клиенты : temporary
$Parameters
	тип	: ( Тип1, Тип2 )
	состояние: ( Пришел, Начал_стрижку )
$End

$Resource_type Парикмахеры: permanent
$Parameters
	состояние_парикмахера : ( Свободен, t1, t2, Занят ) = Свободен
	количество_обслуженных: integer
	длительность_min	: integer
	длительность_max	: integer
	тип_клиента		: such_as Клиенты.тип
$End

$Resources
	Парикмахерская = Парикмахерские(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 + Интервал_прихода( 30 ) );
		количество_в_очереди++;

_Клиент:
	Convert_event
		тип	= Тип_клиента;
		состояние = Пришел;
$End

$Pattern Образец_обслуживания_клиента : operation
$Relevant_resources
	_Парикмахерская: Парикмахерская Keep NoChange
	_Клиент		: Клиенты	Keep Erase
	_Парикмахер	: Парикмахеры	Keep Keep
$Time = Длительность_обслуживания( _Парикмахер.длительность_min,
_Парикмахер.длительность_max )
$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

$Sequence Интервал_прихода : real
$Type = exponential 123456789 legacy
$End

$Sequence Длительность_обслуживания : real
$Type = uniform 123456789 legacy
$End

$Sequence Тип_клиента : such_as Клиенты.тип
$Type = by_hist 123456789 legacy
$Body
      Тип1 1.0
      Тип2 5.0
$End

$Simulation_run
      Образец_прихода_клиента.planning( time_now + Интервал_прихода( 30 ) );
      Terminate_if Time_now >= 1000;
$End

$Results
      Занятость_парикмахера_1 : watch_state Парикмахер_1.состояние_парикмахера ==
Занят
      Занятость_парикмахера_2 : watch_state Парикмахер_2.состояние_парикмахера ==
Занят
      Занятость_парикмахера_3 : watch_state Парикмахер_3.состояние_парикмахера ==
Занят
      Обслужено_парикмахером_1: get_value Парикмахер_1.количество_обслуженных
      Обслужено_парикмахером_2: get_value Парикмахер_2.количество_обслуженных
      Обслужено_парикмахером_3: get_value Парикмахер_3.количество_обслуженных
$End
```

Приложение 2 – Исходный код модели для тестирования построения графика данных перечислимого типа с большим количеством значений

```
$Resource_type
	resource : permanent
$Parameters
	type : ( t0 , t1 , t2 , t3 , t4
, t5 , t6 , t7 , t8 , t9 , t10 , t11 , t12 , t13 , t14 , t15 , t16 , t17 , t18
, t19 , t20 , t21 , t22 , t23 , t24 , t25 , t26 , t27 , t28 , t29 , t30 , t31
, t32 , t33 , t34 , t35 , t36 , t37 , t38 , t39 , t40 , t41 , t42 , t43 , t44
, t45 , t46 , t47 , t48 , t49 , t50 , t51 , t52 , t53 , t54 , t55 , t56 , t57
, t58 , t59 , t60 , t61 , t62 , t63 , t64 , t65 , t66 , t67 , t68 , t69 , t70
, t71 , t72 , t73 , t74 , t75 , t76 , t77 , t78 , t79 , t80 , t81 , t82 , t83
, t84 , t85 , t86 , t87 , t88 , t89 , t90 , t91 , t92 , t93 , t94 , t95 , t96
, t97 , t98 , t99, t100, t101, t102) = t0
$End

$Resources
	res = resource ( * );
$End

$Pattern change_type : event
$Relevant_resources
	_res : res Keep
$Body
	_res :
	Convert_event if (type == t0) type = t1; else if (type == t1) type = t2; else
if (type == t2) type = t3; else if (type == t3) type = t4; else if (type == t4) type
= t5; else if (type == t5) type = t6; else if (type == t6) type = t7; else if (type
== t7) type = t8; else if (type == t8) type = t9; else if (type == t9) type = t10;
else if (type == t10) type = t11; else if (type == t11) type = t12; else if (type ==
t12) type = t13; else if (type == t13) type = t14; else if (type == t14) type = t15;
else if (type == t15) type = t16; else if (type == t16) type = t17; else if (type ==
t17) type = t18; else if (type == t18) type = t19; else if (type == t19) type = t20;
else if (type == t20) type = t21; else if (type == t21) type = t22; else if (type ==
t22) type = t23; else if (type == t23) type = t24; else if (type == t24) type = t25;
else if (type == t25) type = t26; else if (type == t26) type = t27; else if (type ==
t27) type = t28; else if (type == t28) type = t29; else if (type == t29) type = t30;
else if (type == t30) type = t31; else if (type == t31) type = t32; else if (type ==
t32) type = t33; else if (type == t33) type = t34; else if (type == t34) type = t35;
else if (type == t35) type = t36; else if (type == t36) type = t37; else if (type ==
t37) type = t38; else if (type == t38) type = t39; else if (type == t39) type = t40;
else if (type == t40) type = t41; else if (type == t41) type = t42; else if (type ==
t42) type = t43; else if (type == t43) type = t44; else if (type == t44) type = t45;
else if (type == t45) type = t46; else if (type == t46) type = t47; else if (type ==
t47) type = t48; else if (type == t48) type = t49; else if (type == t49) type = t50;
else if (type == t50) type = t51; else if (type == t51) type = t52; else if (type ==
t52) type = t53; else if (type == t53) type = t54; else if (type == t54) type = t55;
else if (type == t55) type = t56; else if (type == t56) type = t57; else if (type ==
t57) type = t58; else if (type == t58) type = t59; else if (type == t59) type = t60;
else if (type == t60) type = t61; else if (type == t61) type = t62; else if (type ==
t62) type = t63; else if (type == t63) type = t64; else if (type == t64) type = t65;
else if (type == t65) type = t66; else if (type == t66) type = t67; else if (type ==
t67) type = t68; else if (type == t68) type = t69; else if (type == t69) type = t70;
else if (type == t70) type = t71; else if (type == t71) type = t72; else if (type ==
t72) type = t73; else if (type == t73) type = t74; else if (type == t74) type = t75;
else if (type == t75) type = t76; else if (type == t76) type = t77; else if (type ==
t77) type = t78; else if (type == t78) type = t79; else if (type == t79) type = t80;
else if (type == t80) type = t81; else if (type == t81) type = t82; else if (type ==
t82) type = t83; else if (type == t83) type = t84; else if (type == t84) type = t85;
else if (type == t85) type = t86; else if (type == t86) type = t87; else if (type ==
t87) type = t88; else if (type == t88) type = t89; else if (type == t89) type = t90;
```

```
else if (type == t90) type = t91; else if (type == t91) type = t92; else if (type ==
t92) type = t93; else if (type == t93) type = t94; else if (type == t94) type = t95;
else if (type == t95) type = t96; else if (type == t96) type = t97; else if (type ==
t97) type = t98; else if (type == t98) type = t99; else if (type == t99) type = t100;
else if (type == t100) type = t101; else if (type == t101) type = t1;
      change_type.planning(time_now +10);
$End

$Simulation_run
      change_type.planning( time_now + 10);
      Terminate_if Time_now >= 1100;
$End
```