Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Робототехника и комплексная автоматизация»

КАФЕДРА «Компьютерные системы автоматизации производства»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ НА ТЕМУ:

«Разработка подсистемы прореживания графиков для системы имитационного моделирования Rao X»

Студент РК9-81Б _____ (Подпись, дата) В.В. Рахматулин

Руководитель ВКР _____ (Подпись, дата) А.В. Урусов

Нормоконтролер _____ (Подпись, дата) М.Н. Святкина

2017 г.

АННОТАЦИЯ

Данная выпускная квалификационная работа посвящена теме: «Разработка подсистемы прореживания графиков для системы имитационного моделирования Rao X».

Цель работы: создание подсистемы прореживания графиков для системы имитационного моделирования Rao X.

Поставленная цель достигается за счет анализа исходного кода имеющейся кафедральной системы имитационного моделирования Rao X, выделения компонентов графической подсистемы, разработка метода прореживания графиков с последующей интеграцией в существующую подсистему.

В расчетно-пояснительной записке проведено предпроектное исследование подсистемы вывода графиков Rao X, выполнены концептуальное и техническое проектирование, а также разработан программный код и доведена до реализации подсистема прореживания. В последнем разделе проведено апробирование подсистемы.

Расчетно-пояснительная записка содержит 47 страниц, 16 рисунков, 3 графика, 2 гистограммы, 3 приложения (на 35 страницах) и ссылается на 15 источников.

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ИМ – Имитационное Моделирование

СДС – Сложная Дискретная Система

IDE – Integrated Development Environment (Интегрированная Среда Разработки)

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

Рендеринг – процесс получения изображения графиков программным путем

Рендерер – компонент, выполняющий рендеринг данных

Dataset – набор данных графика, состоит из серий

Серия – последовательность точек, которые хранятся для отображения конкретной кривой на графике

Подсистема визуализации – подсистема среды RAO X, 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

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.

ИМ является методом, опирающимся на программно-аппаратное обеспечение. Современные программные комплексы невозможно представить без графического интерфейса, который должен быть удобным для восприятия пользователем. Визуализация результатов моделирования дает наглядное представление о состоянии системы в требуемый для исследования интервал времени.

Разработка интеллектуальной среды имитационного моделирования РДО выполнена в Московском Государственном Техническом Университете (МГТУ им. Н. Э. Баумана) на кафедре "Компьютерные системы автоматизации производства" под именем Rao X. Rao X включает в себя компонент, отвечающий за визуализацию данных в формате графиков. Данная квалификационная работа использует наработки из моего курсового проекта, посвященному доработкам подсистемы вывода графиков, и является его продолжением. Текущая версия подсистемы плохо работает с большим количеством значений и требует разработки дополнительного модуля, способного прореживать данные.

1. Предпроектное исследование

1.1. Основные положения языка РДО

Основные положения системы РДО могут быть сформулированы следующим образом [2]:

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

1.2. Графики в системе имитационного моделирования RAO X

Имеется возможность визуально отображать процессы, происходящие в модели, в виде графиков [3]. 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 «Plot». 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 можно использовать выпадающее меню при правом щелчке мышки по области графика, а также комбинации ctrl или shift с колесиком мыши. 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.

Прямо в окне построенного графика можно просмотреть значения точек. При прохождении курсора мыши по графику, высвечивается окошко с координатами ближайшей к курсору точки.

Как взаимодействуют между собой компоненты подсистемы показано на Рисунок 1.

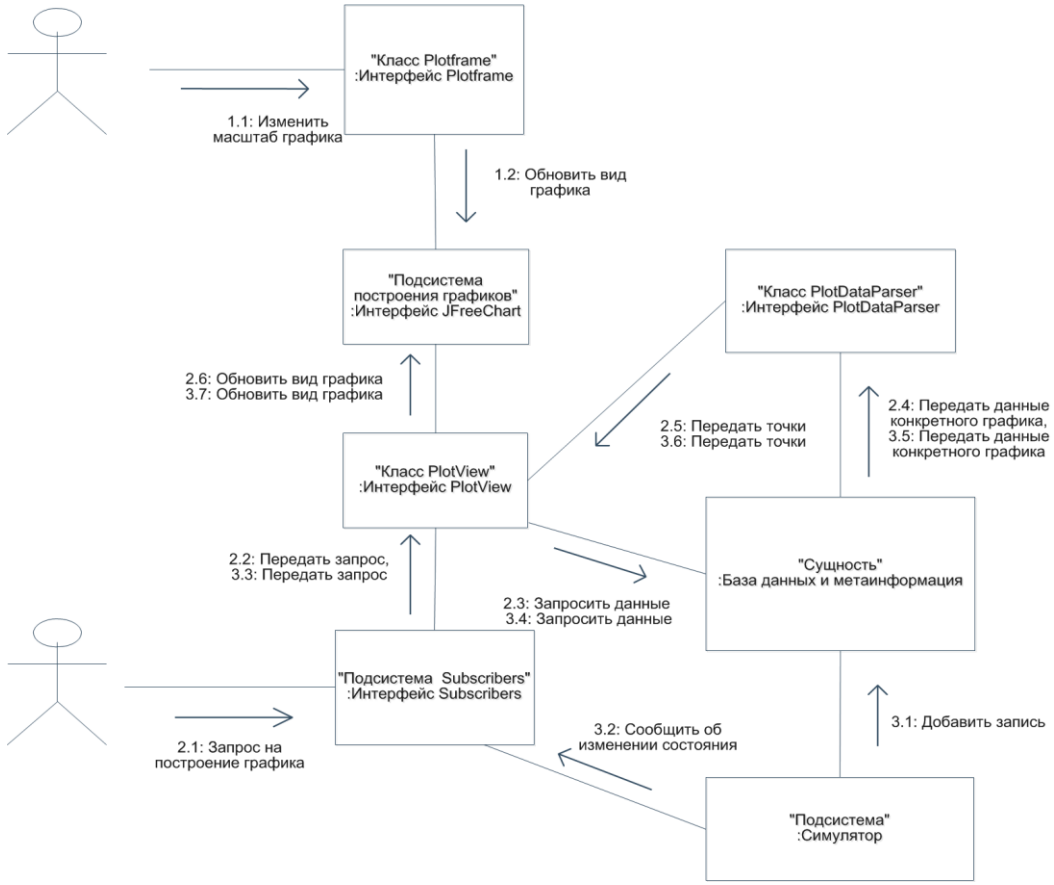


Рисунок 1. Диаграмма коммуникаций участников процесса формирования графика

Текущая подсистема вывода графиков обладает рядом существенных недостатков. Подсистема существенно тормозит при попытке вывести на график большое количество точек. Например, вывод около 150 000 значений сильно нагружает процессор и необходимое время для выполнения операции доходит до 10 минут. Зависимость времени построения графиков от времени в текущей версии подсистемы изображена на График 1. Зависимость времени построения графика от количества точек. Также имеется порог вывода до 200000 значений одновременно (размер кучи – 1900 Мбайт), после чего вся система Rao X падает

и выдает ошибку нехватки памяти. Вдобавок, построенные графики выглядят нагроможденными, что затрудняет восприятие информации пользователем.

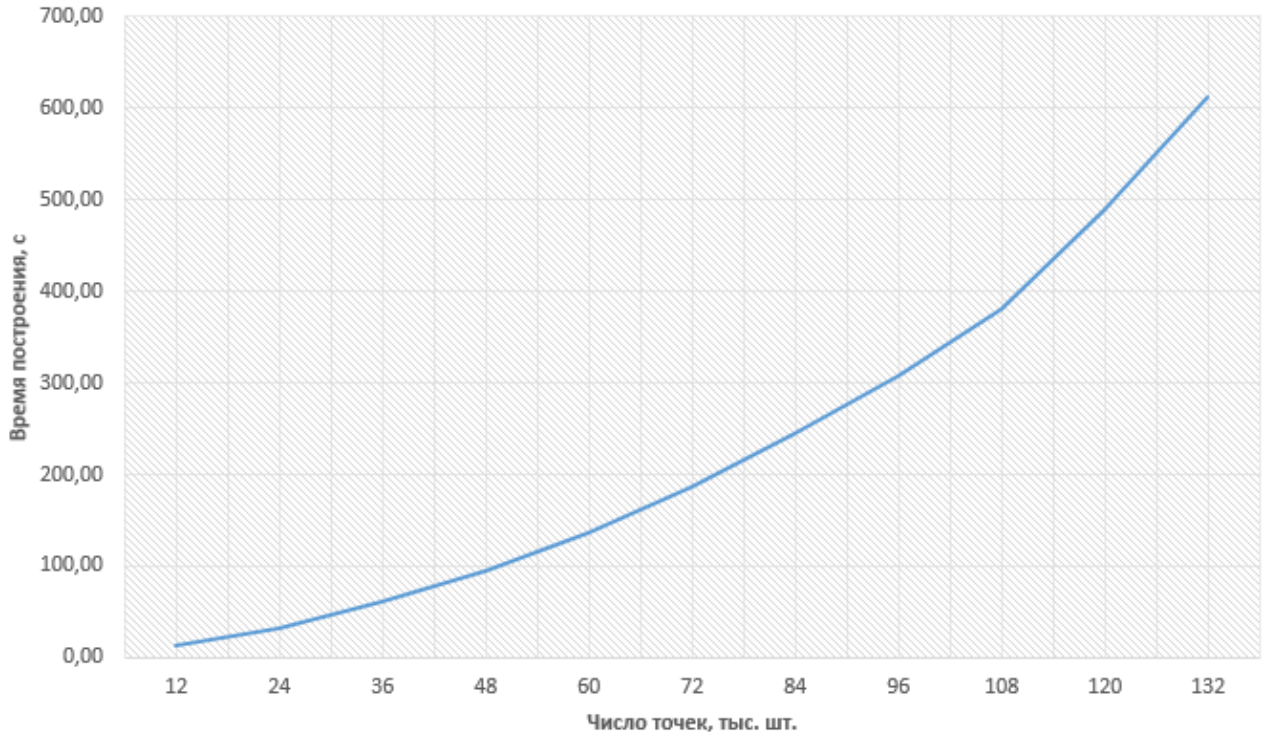
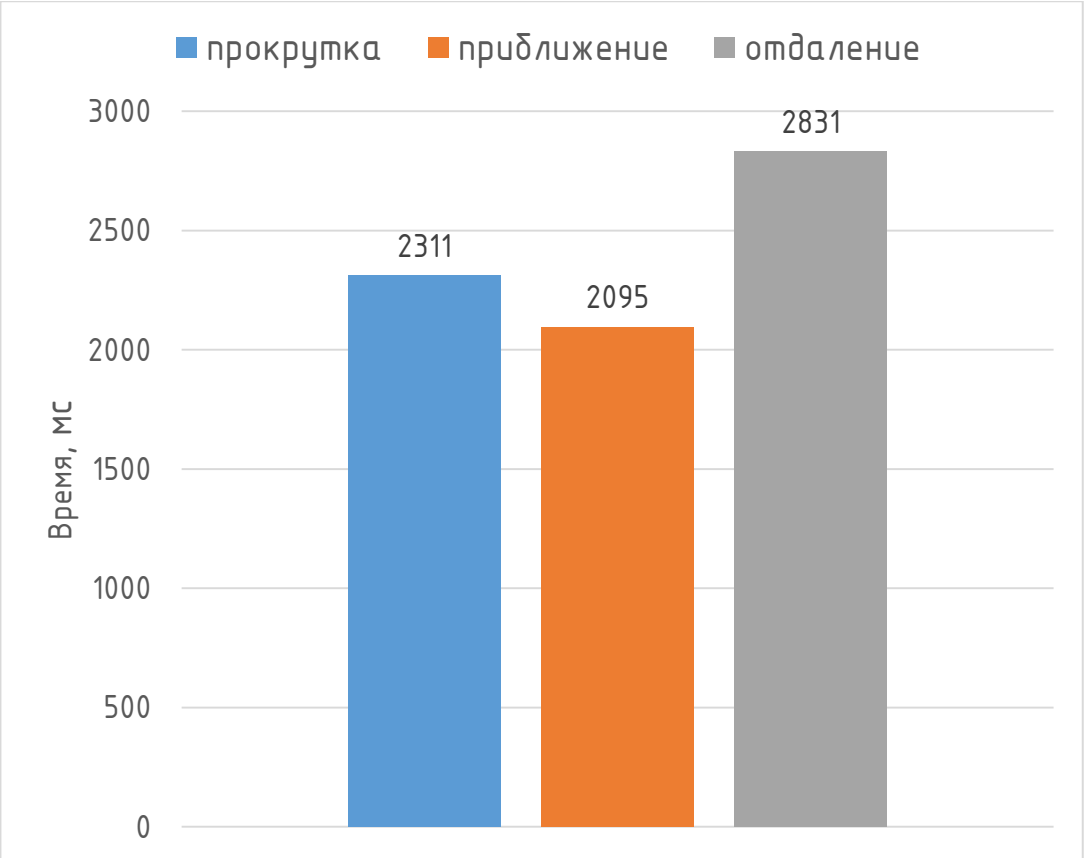


График 1. Зависимость времени построения графика от количества точек

Также подтормаживает обновления вида при проведении операций масштабирования графика и навигации по нему. Среднее время обновление вида на графике с максимально допустимым количеством одновременно выводимых точек для каждой операции отражено на Гистограмма 1. Разработка подсистемы прореживания будет производиться на языке родителя системы ИМ Rao X – Java [4].



Гистограмма 1. Среднее время обновления вида графика

2. Концептуальный этап проектирования подсистемы

2.1. Выбор общесистемной методологии проектирования

Недостатки, выявленные в предпроектном исследовании, могут быть решены на основе следующих концепций:

- Модульность;
- Объектная ориентированность.

Модульное программирование [5] — это организация программы как совокупности небольших независимых блоков, называемых модулями, структура и поведение которых подчиняются определённым правилам. Использование модульного программирования позволяет упростить тестирование программы и обнаружение ошибок. Аппаратно-зависимые подзадачи могут быть строго отделены от других подзадач, что улучшает мобильность создаваемых программ.

Модуль — функционально законченный фрагмент программы. Во многих языках (но далеко не обязательно) оформляется в виде отдельного файла с исходным кодом или поименованной непрерывной её части. Некоторые языки предусматривают объединение модулей в пакеты.

Объектно-ориентированное программирование [6] (ООП) методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования. Объект — это сущность, которой можно посылать сообщения, и которая может на них реагировать, используя свои данные.

Объект — это экземпляр класса. Данные объекта скрыты от остальной программы. Сокрытие данных называется инкапсуляцией. Наличие инкапсуляции достаточно для объектности языка программирования, но ещё не означает его объектной ориентированности — для этого требуется наличие наследования.

Но даже наличие инкапсуляции и наследования не делает язык программирования в полной мере объектным с точки зрения ООП. Основные преимущества ООП проявляются только в том случае, когда в языке программирования реализован полиморфизм; то есть возможность объектов с одинаковой спецификацией иметь различную реализацию.

Проектирование структуры подсистемы, работы методов, взаимодействия компонентов будет производиться на основе методологии UML[7].

2.2. Выделение системы из среды

Система имитационного моделирования RAO X является плагином для IDE Eclipse, позволяющий вести разработку имитационных моделей на языке РДО. Система написана на языке Java [8] и состоит из четырех основных компонентов. На Рисунок 2 представлена упрощенная диаграмма пакетов системы Rao X, на которой указаны основные зависимости от сторонних библиотек. В приложении к записке представлена диаграмма классов подсистемы вывода графиков.

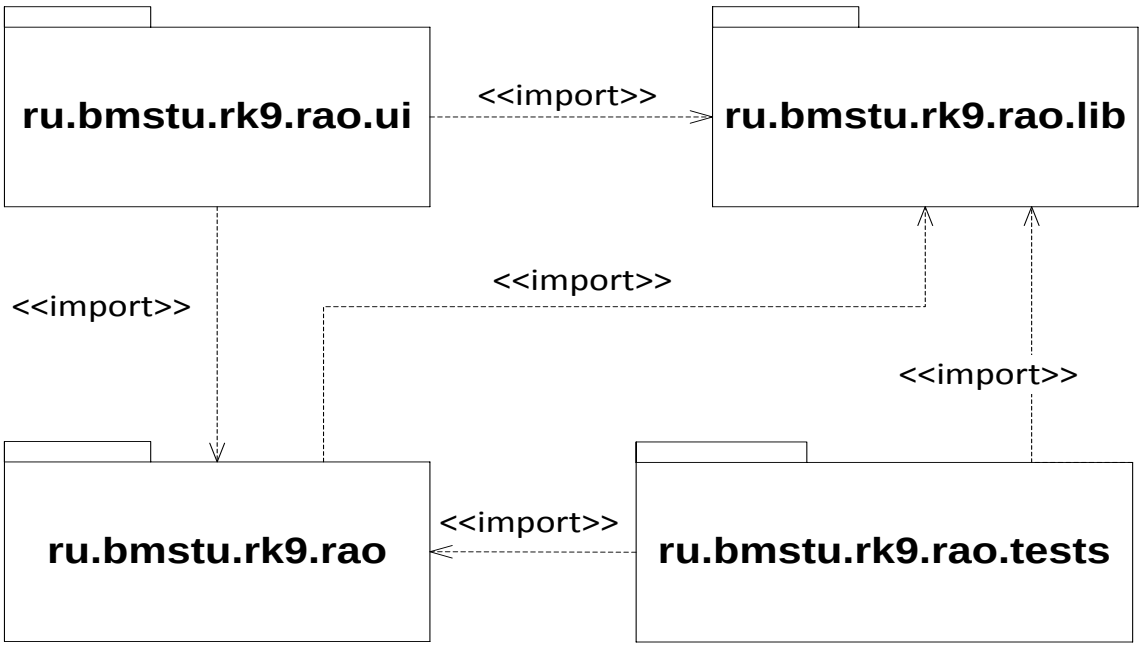


Рисунок 2. Диаграмма коммуникаций участников процесса формирования графика

ru.bmstu.rk9.rao – Пакет, содержащий модули, необходимые для работы реализация языка РДО, включает в себя описание грамматики и ее парсер

ru.bmstu.rk9.rao.ui – Пакет, содержащий модули, отвечающие за пользовательский интерфейс

ru.bmstu.rk9.rao.lib – Пакет, содержащий модули, необходимые для работы имитационных моделей

ru.bmstu.rk9.rao.tests – Пакет, содержащий модули, необходимые для тестирования

ru.bmstu.rk9.rao.ui состоит из следующих пакетов

- ru.bmstu.rk9.rao.ui
- ru.bmstu.rk9.rao.ui.animation
- ru.bmstu.rk9.rao.ui.console
- ru.bmstu.rk9.rao.ui.contentassist
- ru.bmstu.rk9.rao.ui.execution
- ru.bmstu.rk9.rao.ui.graph
- ru.bmstu.rk9.rao.ui.highlightning
- ru.bmstu.rk9.rao.ui.labeling
- ru.bmstu.rk9.rao.ui.notification
- ru.bmstu.rk9.rao.ui.outline
- ru.bmstu.rk9.rao.ui.plot
- ru.bmstu.rk9.rao.ui.quickfix
- ru.bmstu.rk9.rao.ui.results
- ru.bmstu.rk9.rao.ui.serialization
- ru.bmstu.rk9.rao.ui.simulation
- ru.bmstu.rk9.rao.ui.toolbar
- ru.bmstu.rk9.rao.ui.trace
- ru.bmstu.rk9.rao.ui.wizard

ru.bmstu.rk9.rao.ui.plot- пакет, в котором лежат классы, отвечающие за построение графиков.. Для подсистемы необходимо будет разработать ряд

классов, расширяющих или меняющих функционал исходных. Предполагается в этом случае использовать наследование. Нужно разработать и внедрить метод прореживания, который устраняет проблемы исходной подсистемы и найти причину медленного построения графиков.

Содержимое пакета **ru.bmstu.rk9.rao.ui.plot:**

PlotDataParser - класс, отвечающий за преобразование данных из базы данных в формат точек графика. А **PlotDataParserException** обрабатывает возникающие исключения.

PlotView – класс, отвечающий построение за отображение графика на экране. Включает в себя внутренний класс **RealTimeUpdateRunnable** с методом run(), который добавляет результаты парсинга в dataset графика в режиме real-time.

PlotFrame содержит в себе настройки масштабирования и слайдеры, а также метод, отвечающий за возникновение окошка с координатами ближайшей к курсору точки.

Класс **PlotView** обращается к функционалу библиотеки **JFreeChart** [9] для создания графиков по имеющемуся dataset. Обновление графиков при добавлении данных или изменении их масштаба или области просмотра основано на системе подписчиков, реализующих паттерн **Observer [**10**].** Прореживание требует переработки не только классов внутри пакета, но и в самой библиотеке-отрисовщике.

2.3. Разработка укрупненной функциональной структуры подсистемы

В качестве CASE-средства для разработки функциональной структуры была использована методология ARIS [11]. На Рисунок 3 изображено дерево целей разрабатываемой подсистемы.

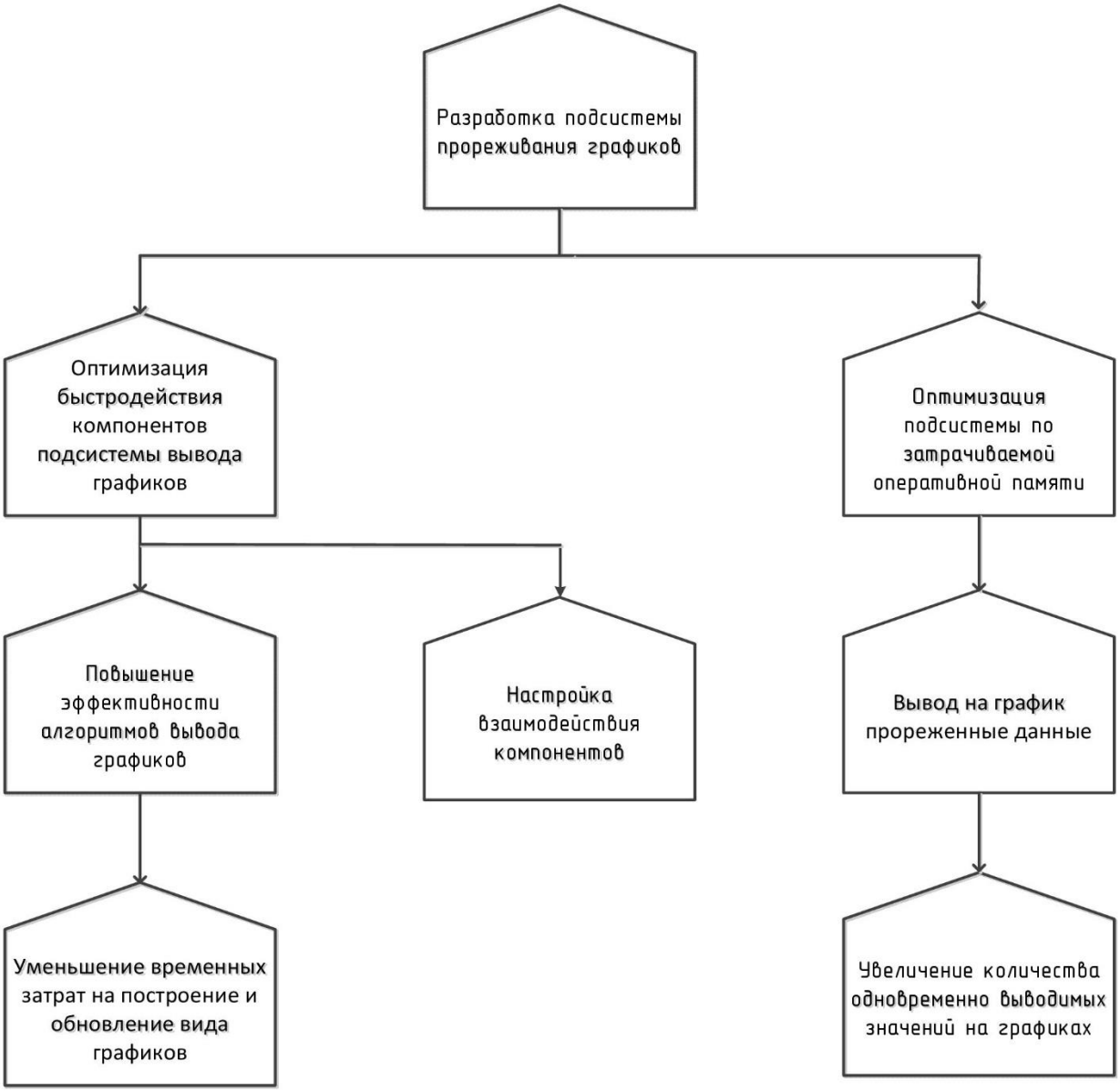


Рисунок 3. Дерево целей подсистемы прореживания

3. Формирование технического задания на подсистему

3.1. Общие сведения

Основание для разработки: задание на выполнение ВКР.

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

Разработчик: студент кафедры «Компьютерные системы автоматизации производства» Рахматулин В.В.

Наименование темы разработки: «Разработка подсистемы прореживания графиков для системы имитационного моделирования RAO X»

3.2. Назначение и цели разработки

Ускорить работу графической подсистемы, увеличить число одновременно выводимых значений на графике.

3.3. Требования к программе или программному изделию

3.3.1. Требования к функциональным характеристикам

Улучшенная подсистема должна обеспечить:

- Быстрое построение графиков;
- Вывод графиков в прореженном виде при граничных условиях;
- Увеличение количества одновременно выводимых значений на графиках;
- Быструю навигацию и обновление вида графиков при масштабировании или изменении размеров экрана;
- Быструю перестройку графика при добавлении новых точек в real-time режиме;
- Вывод всплывающей подсказки с координатами ближайшей к курсору точки.

3.3.2. Требования к надежности

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

3.3.3. Условия эксплуатации

- Эксплуатация должна производиться на оборудовании, отвечающем требованиями к составу и параметрам технических средств, и с применением программных средств, отвечающим требованиям к программной совместимости.
- Аппаратные средства должны эксплуатироваться в помещениях с выделенной розеточной электросетью 220В ±10%, 50 Гц с защитным заземлением.

3.3.4. Требования к составу и параметрам технических средств

Программный продукт должен работать на компьютерах со следующими характеристиками:

- объем ОЗУ не менее 1024 Мб;
- микропроцессор с тактовой частотой не менее 1600 МГц;
- требуемое свободное место на жестком диске – 4 Гб;
- монитор с разрешением от 1366*768 и выше.

3.3.5. Требования к информационной и программной совместимости

- операционная система Windows Server 2003 и старше или Ubuntu 15.10 и старше;
- наличие в операционной системе ПО Eclipse DSL Tools Mars 2 и новее.

3.3.6. Требования к маркировке и упаковке

Требования к маркировке и упаковке не предъявляются

3.3.7. Требования к транспортированию и хранению

Требования к транспортированию и хранению не предъявляются.

3.4. Требования к программной документации

Требования к программной документации не предъявляются.

3.5. Стадии и этапы разработки

Плановый срок начала разработки – 4 марта 2017 г.

Плановый срок окончания разработки – 20 мая 2017 г.

Этапы разработки:

- Концептуальный этап проектирования системы;
- Технический этап проектирования системы;
- Рабочий этап проектирования системы.

3.6. Порядок контроля и приемки

Контроль и приемка работоспособности системы автоматизированной сборки, тестирования и развертывания должны осуществляться в процессе проверки функциональности (апробирования) системы в целом, а также в процессе проверки функциональности (апробирования) полученной в результате его работы системы имитационного моделирования Rao X путем многократных тестов в соответствии с требованиями к функциональным характеристикам системы.

4. Технический этап проектирования

4.1. Разработка диаграмм активности подсистемы

Как упоминалось выше, подсистема визуализации графиков работает с библиотекой **JFreeChart** [9]. Были найдены классы, также участвующие в создании графиков: **XYPlotFactory**, **XYPlot**. В системе строятся ступенчатые графики, поэтому в качестве отрисовщика используется класс **XYStepRenderer**.

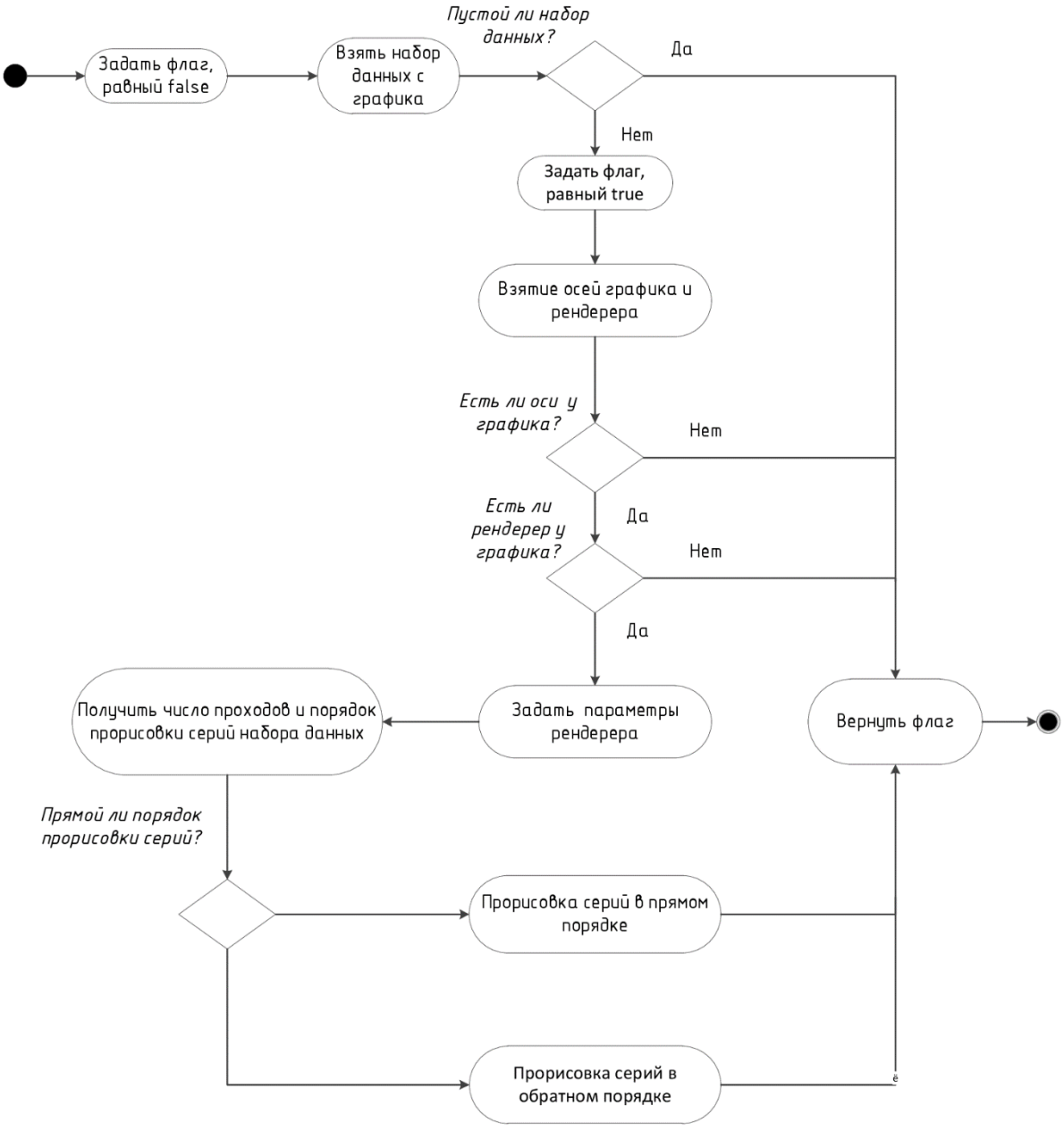


Рисунок 4. Диаграмма активности визуализации графиков

На Рисунок 4 приведена диаграмма активности [12], отражающая работу метода **render(input variables)** в **XYPlot**, возвращающий **true**, если на графике имеются данные, и **false** в противном случае. Данный алгоритм обобщает процесс вывода графиков на экран и уже имелся в подсистеме. Поскольку библиотека **JFreeChart** способна выводить несколько кривых на одном графике, то любой набор данных **dataset** состоит из серий, в которых и хранятся значения точек.

4.1.1. Разработка диаграммы активности прореживания графиков

Была разработан диаграмма активности прореживания данных, приведенная на Рисунок 5. За параметры, от которых зависит работа прореживателя были взяты ширина окошка с графиком в пикселях и количество выводимых точек.

Рисунок 5. Диаграмма активности прореживания графиков

Из соотношения этих двух параметров считается количество групп, на которые разбиваются все выводимые точки графика. В каждой из них вычисляется среднее арифметическое значение и ищется самая близкая к этому значению точка, которая будет представлять всю группу на графике.

4.1.2. Разработка диаграммы активности задания количества выводимых точек

Чтобы прореженные графики не были слепленными, но в то же время были как можно ближе к реальным, было принято выводить по одной точке на каждые 2 пикселя ширины графика, что и отражено на Рисунок 6.

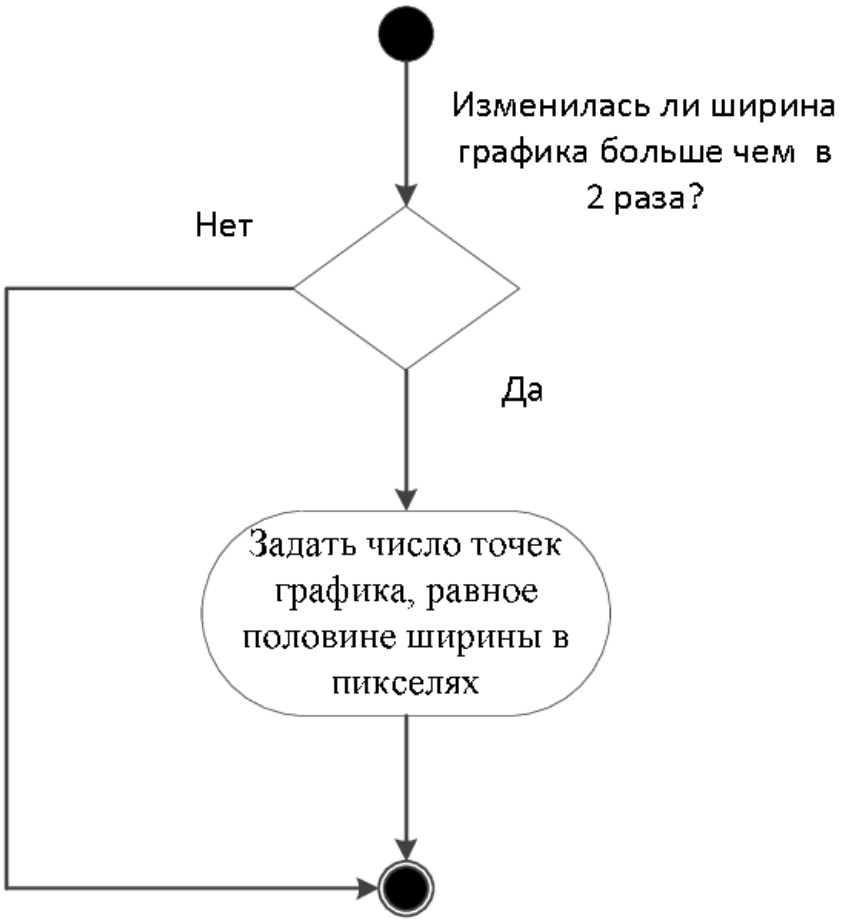


Рисунок 6. Диаграмма активности задания количества выводимых точек

4.1.3. Разработка диаграммы активности отрисовки точек в серии прореживаемого графика

На диаграмме активности, отображенной на Рисунок 7, детализирован процесс отрисовки точек из серии графика.

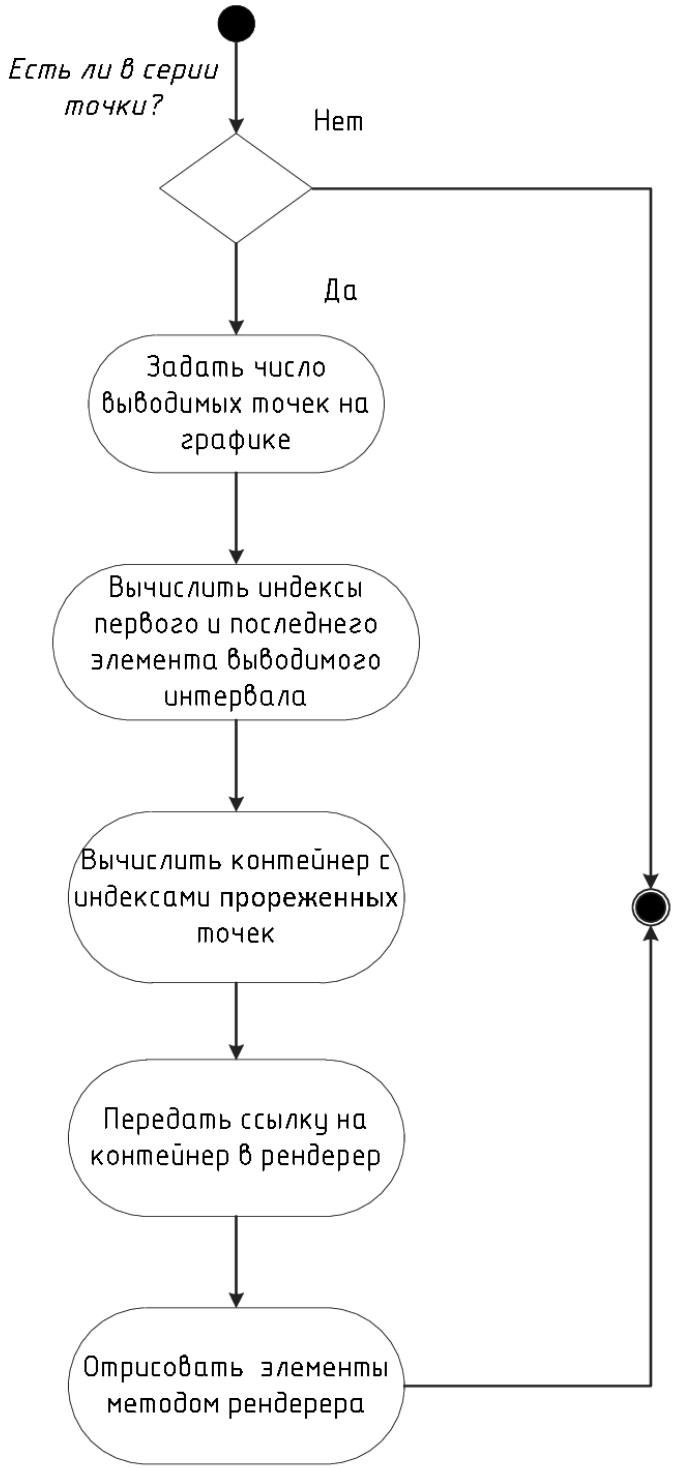


Рисунок 7. Диаграмма активности отрисовки точек в серии прореживаемого графика

Метод **drawItem()** лежит в классе **XYStepRenderer**. Если обьект данного класса не обнаружит прореженный набор точек, то он должен строить данные по всем точкам серии.

4.1.4. Разработка диаграммы активности отрисовки точки рендерером с прореживанием

На Рисунок 8 приведена диаграмма активности обновленного метода рендерера ступенчатых графиков.

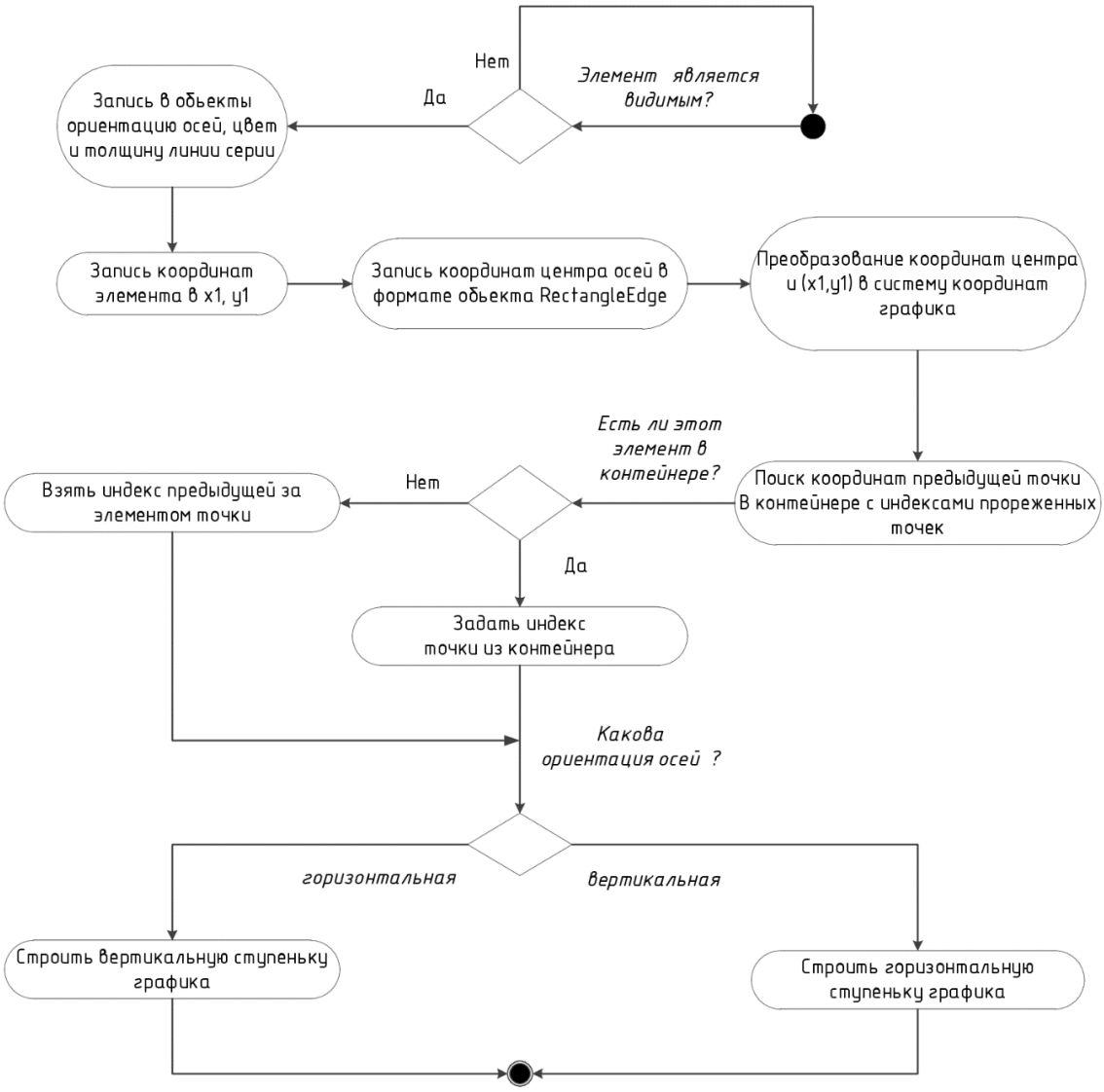


Рисунок 8. Диаграмма активности отрисовки точки рендерером с прореживанием

На вход рендерер получает индекс элемента. Для построения ступеньки необходимо 2 точки: текущая и предыдущая. В зависимости от наличия индекса в **filteredMap** и направления осей графика, рендерер отрисовывает ступеньку пришедшего элемента.

4.1.5. Разработка диаграмма активности добавления точек в серию графика

Объект класса **XYSeries**, который использовался для хранения серии на графике, содержит в себе метод добавления точек **add**(itemX, itemY, flag). ItemX и ItemY – координаты добавляемой точки, а последний параметр посылает в систему сообщения типа **SeriesChangedEvent** при значении **true.** На Рисунок 9 продемонстрирована разработанная диаграмма активности для измененного метода.

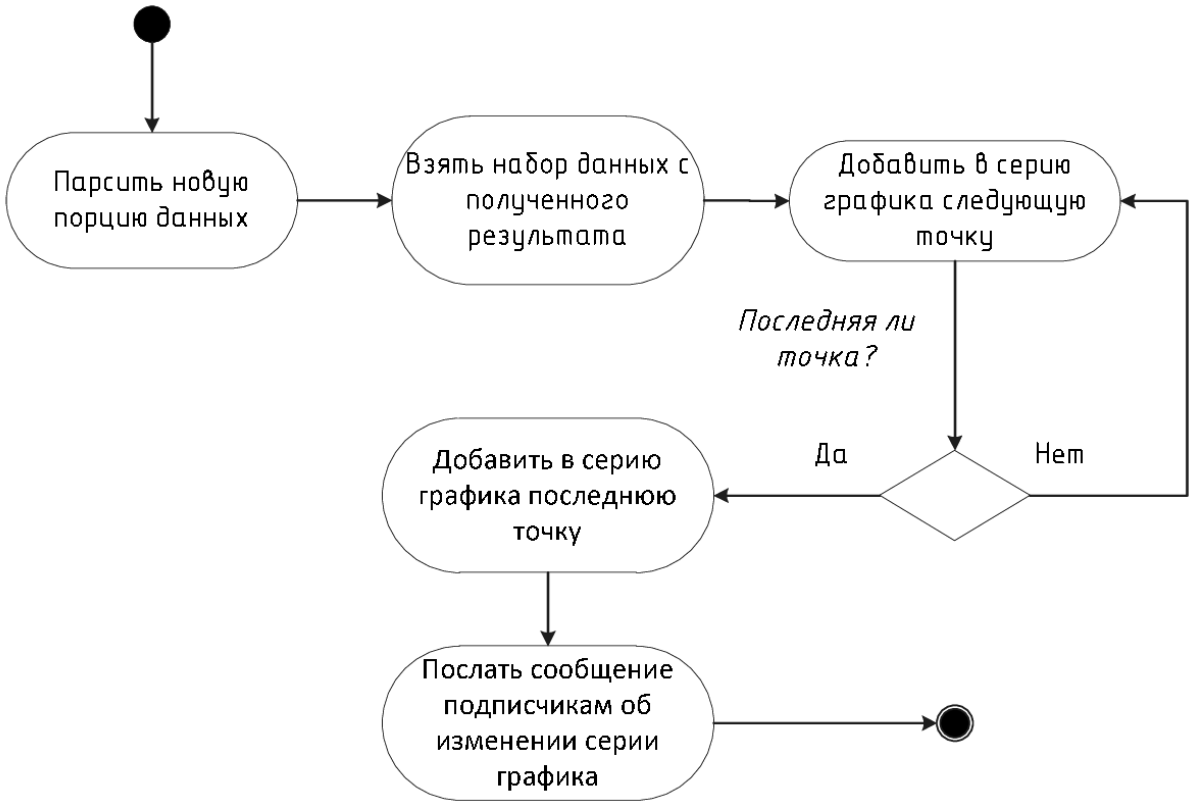


Рисунок 9. Диаграмма активности добавления точек в серию графика

Поскольку система построения и обновления графиков использует систему подписчиков, то каждое сообщение **SeriesChangedEvent** запускает процесс полной перестройки графика, что и является причиной сильного торможения работы подсистемы. Было принять решение дробить порцию добавляемых данных и запускать перестройку лишь с добавлением последнего элемента.

4.1.6. Разработка диаграммы активности вывода точек во всплывающем окошке

В методе **mouseMove(MouseEvent e)** класса **Plotframe** реализована возможность выводить окошко с координатами ближайшей к курсору точки. Поиск значений производился из всех имеющихся точек серии графика. Разработанный алгоритм проверяет: является ли объект прореженным графиком, есть ли в прореженном контейнере данные? В утвердительном случае процесс поиска точек для вывода на график осуществляется среди индексов прореженных точек. Принцип работы отражен на Рисунок 10.

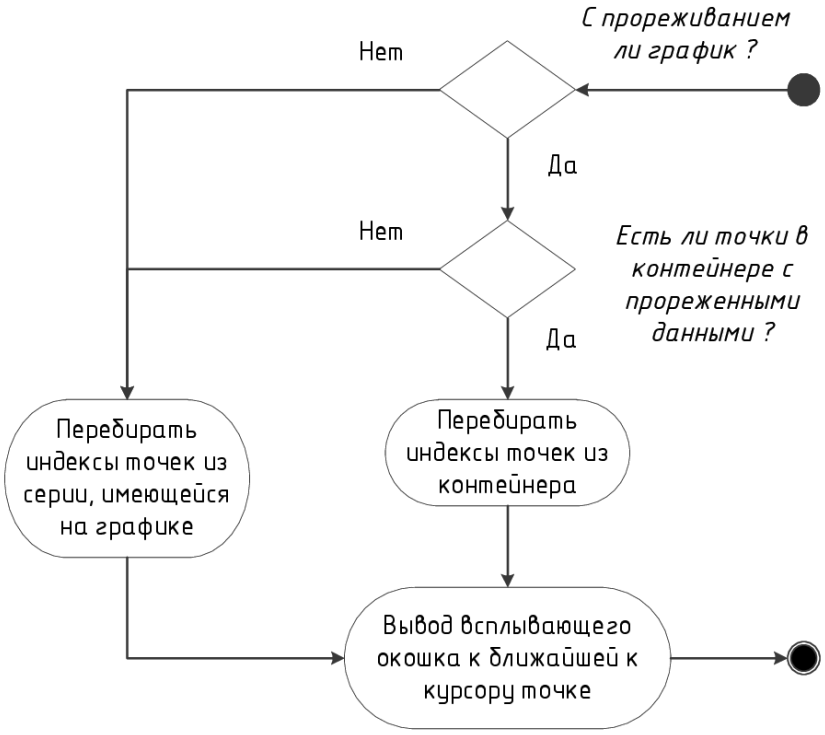


Рисунок 10. Диаграмма активности всплывающей подсказки

4.1.7. Разработка диаграммы активности фабричного метода для графиков с прореживанием

Разработана диаграмма активности фабричного метода [14], который отвечает за сборку всех объектов и создание графика, работающего с прореживанием с использованием классов библиотеки **JFreeChart**. На Рисунок 11 продемонстрирована работа метода в нотации UML.

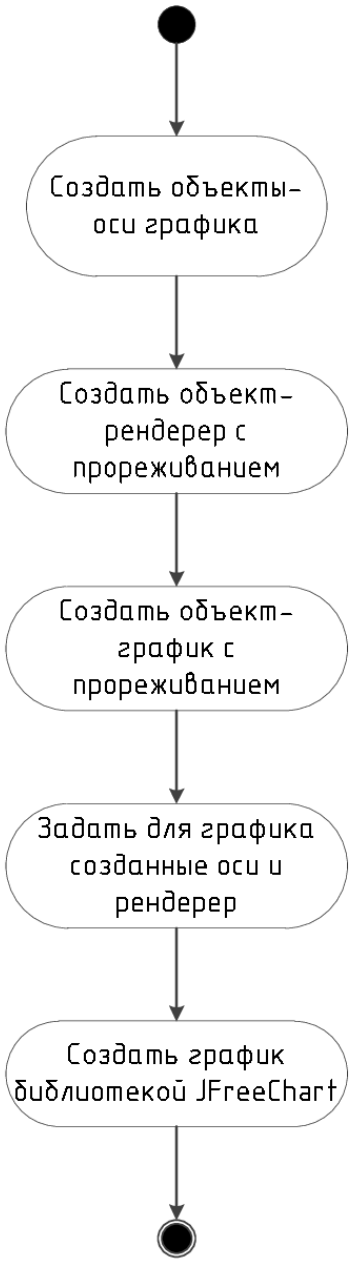


Рисунок 11. Диаграмма активности фабричного метода

4.2. Разработка классов подсистемы прореживания

4.2.1. Разработка класса ProxyDataSet

Метод прореживания графиков, контейнер для хранения данных и методы для его передачи были вынесены в отдельный класс, содержимое которого указано на Рисунок 12.

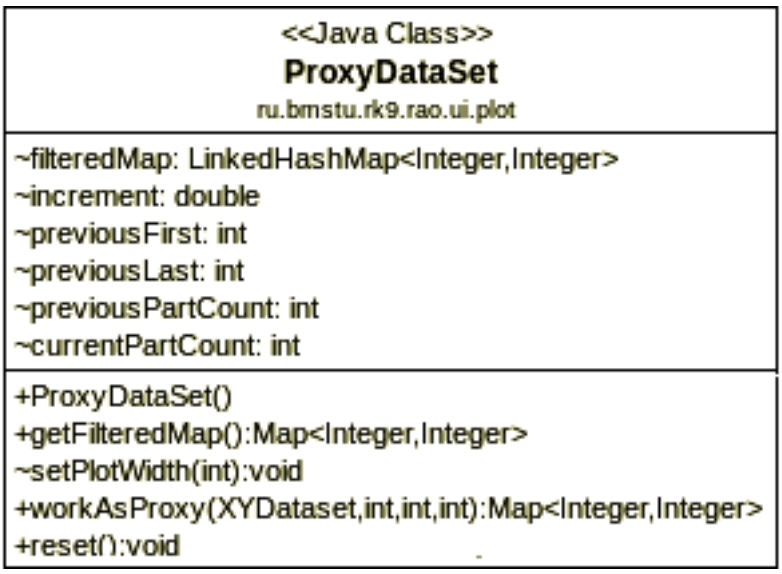


Рисунок 12. Класс **ProxyDataSet**

- **filteredMap** – хранилище для индексов прореженных точек;
- **increment** – шаг прореживания;
- **partCount** – количество групп, на которые разбивается график;
- Переменные с приставкой **previous** хранят в себе данные предыдущего прореживания.

Метод **setPlotWidth(int)** задает количество выводимых точек для прореживаемого графика и работает так, как указано на Рисунок 6.

Метод **workAsProxy (input variables)** является прореживателем, наполняющим и возвращающим **filteredMap** и работает по в соответствии с диаграммой, отраженной на Рисунок 5.

getFiltered () возвращает **filteredMap** запрашиваемому обьекту, а **reset()** очищает контейнер**.**

4.2.2. Разработка класса XYPlotWithFiltering

Необходимо было добавить в имеющиеся классы возможность обращаться к методу прореживания и работать с посчитанным контейнером. На Рисунок 13 изображен класс, наследующий функционал от **XYPlot** и переопределивший метод базового класса **render()**.

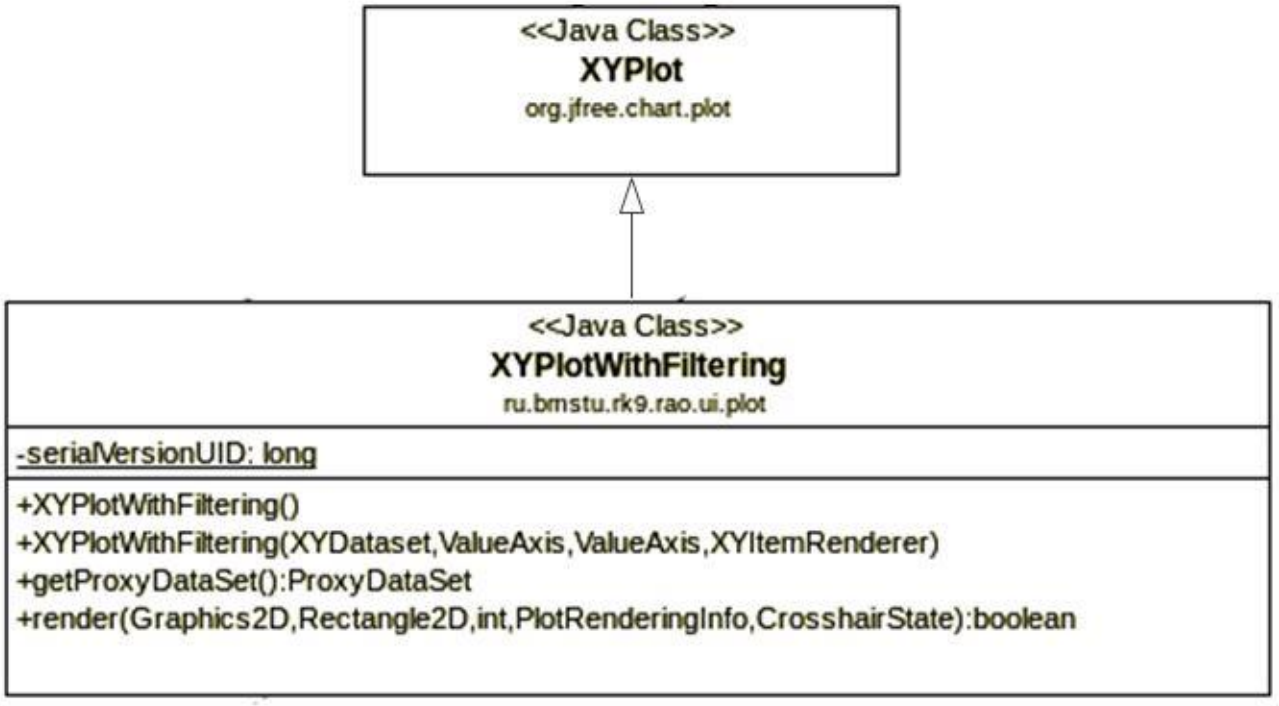


Рисунок 13. Класс **XYPlotWithFiltering**

- Класс содержит 2 конструктора объектов, которые обращаются к аналогичным конструкторам базового класса, использую ключевое слово super;
- Метод **getProxyDataSet()** возвращает объект класса **ProxyDataSet**;
- Метод **render()** переопределен и работает с прореженными индексами. Работа метода отражена на Рисунок 4. Отрисовка каждой отдельной серии происходит согласно диаграмме, приведенной на Рисунок 7.

4.2.3. Разработка класса XYFilteringStepRenderer

Класс **XYFilteringStepRenderer** унаследован от класса **XYStepRenderer,** участвующем в отрисовке ступенчатых графиков. Новые поля и методы класса показаны на Рисунок 14.

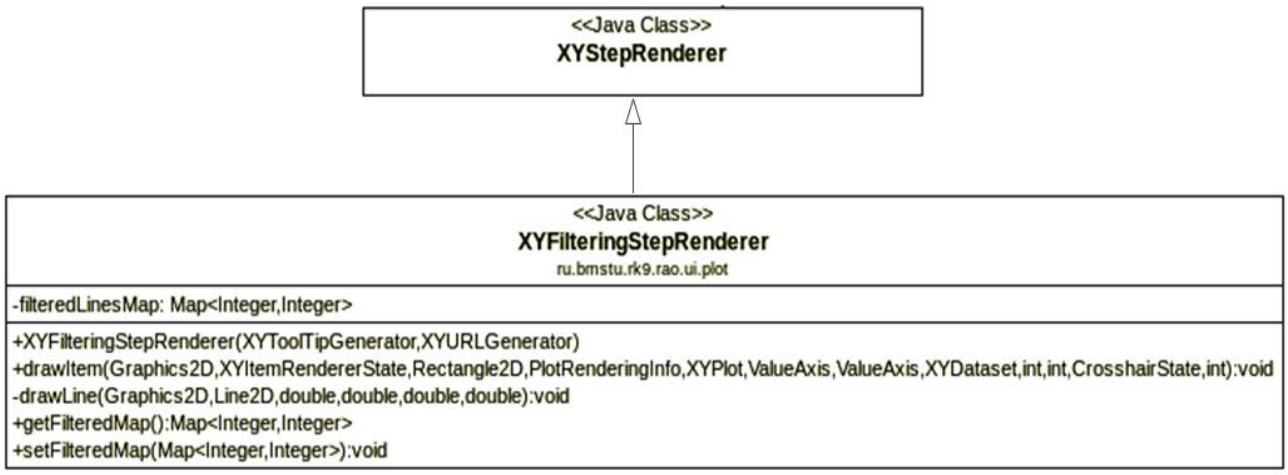


Рисунок 14. Класс **XYFilteringStepRenderer**

Поле **filteredLinesMap** – контейнер **Map** с индексами прореженных точек. В любом контейнере типа **Map** данные хранятся в виде пары ключ/значение. Добавленное поле содержит в себе пары: индекс предыдущей, индекс текущей точки. Слово **filtered** указывает, что данные в этом объекте прорежены.

Были добавлены методы:

- **getFilteredMap()** – метод, передающий **filteredLinesMap** вызывающему обьекту;
- **setFilteredMap()** – метод, задающий контейнер с прореженными точками.

И переопределен метод базового класса **drawItem(input variables)**, который занимается отрисовкой ступени каждого элемента и который теперь работает с прореженными точками. Принцип работы метода был приведен ранее на Рисунок 8.

4.2.4. Разработка класса FilteringPlotFactory

Класс **FilteringPlotFactory,** изображенный на Рисунок 15, содержит в себе фабричный метод **createXYStepChart(input variables)**. Основная задача метода – сконструировать окно графика, включающее в себя оси графика, набор данных, рендерер и являющееся объектом типа **JFreeChart**.

Рисунок 15 Класс **FilteringPlotFactory**

createXYStepChart(input variables) – метод, который конструирует прореженный график, используя все связанные с прореживанием классы. Работа метода соответствует диаграмме активностей, приведенной выше, на Рисунок 11.

5. Рабочее проектирование подсистемы

5.1. Вычисление количества выводимых точек для прореженного графика

Был реализован метод **setPlotWidth(int width)** [13] в соответствии с Рисунок 6, лежащий в классе **ProxyDataSet**. В качестве аргумента посылается текущая ширина графика, а сам метод меняет количество выводимых точек **currentPartCount**, если ширина графика изменилась больше чем в 2 раза.

```
void setPlotWidth(int width) {
   int newPartCount = width / 2;
   if (((double) Math.max(currentPartCount, newPartCount))
         / ((double) Math.min(currentPartCount, newPartCount)) > 2) {
      this.currentPartCount = newPartCount;
   }
}
```

5.2. Реализация метода прореживания графиков

Был реализован метод прореживания графиков **workAsProxy(XYDataset items, int series, int first, int last)**, соответствующий диаграмме активности на Рисунок 5. В качестве аргументов принимается набор данных с графика «**XYDataset items**», номер серии «**int series**» и номера первого и последнего элементов «**int first**», «**int last**». Метод возвращает контейнер типа Map [15] с индексами прореженных точек

Если параметры прореживания не изменились или число элементов меньше количества выводимых точек, то прореживание не запускается:

```
if (previousFirst == first && previousLast == last && previousPartCount
== currentPartCount) {
   return filteredMap;
}
filteredMap.clear();
if (last - first < currentPartCount) {
   return filteredMap;
}
```

Вычисление размера групп, на которые будет разбиваться набор данных:

```
double groupSize = (last - first) / (double) currentPartCount;
```

Расчет среднего арифметического значения группы:

```
for (int i = 0; i < currentPartCount; i++) {
   int innerBegin = (int) (first + (groupSize * i));
   int innerEnd = (int) (first + (groupSize * (i + 1)));
   double sumVal = 0.0;
   for (int inner = innerBegin; inner < innerEnd; inner++) {
      sumVal += items.getYValue(series, inner);
   }
```

Поиск ближайшего к среднему значению индекса точки:

```
for (int inner = innerBegin; inner < innerEnd; inner++) {
   double nearestCandidate = Math.abs(items.getYValue(series, inner) -
sumVal);
   if (nearestCandidate < nearest) {
      nearestIndex = inner;
      nearest = nearestCandidate;
   }
}
```

Добавление в контейнер индекса новой точки в порядке (индекс предыдущей точки, индекс текущей точки):

```
filteredMap.put(nearestIndex, prevFiltered);
```

Код метода приведен в ПРИЛОЖЕНИЕ Б.

5.3. Обращение к индексам прореженных точек при отрисовке серии

Отрисовка одной серии из набора данных является частью метода **render(input variables)**, принадлежащего классу **XYPlotWithFiltering**. Работа метода отражена на Рисунок 7. В переопределенном методе появляется возможность отправлять на отрисовку индексы прореженных точек.

В первую очередь метод пытается получить контейнер с индексами прореженных точек:

```
Map<Integer, Integer> filteredMap = proxy.workAsProxy(dataset, series,
firstItem, lastItem);
```

Затем, если контейнер не пустой, то методом **drawItem(input variables)** класса **XYFilteringStepRenderer** отрисовываются только индексы прореженных точек. В противном случае, серию отрисовывает стандартный рендерер – **XYStepRenderer:**

```
if (renderer instanceof XYFilteringStepRenderer) {
   ((XYFilteringStepRenderer) renderer).setFilteredMap(filteredMap);
}
if (filteredMap.size() == 0 || !(renderer instanceof
XYFilteringStepRenderer)) {
   for (int item = firstItem; item <= lastItem; item++) {
      renderer.drawItem(g2, state, dataArea, info, this, xAxis, yAxis,
dataset, series, item,
            crosshairState, pass);
   }
} else {
   for (Integer item : filteredMap.keySet()) {
      renderer.drawItem(g2, state, dataArea, info, this, xAxis, yAxis,
dataset, series, item,
            crosshairState, pass);
   }
}
```

Код метода приведен в ПРИЛОЖЕНИЕ Б.

5.4. Реализация метода отрисовки отдельного элемента рендерером с прореживанием

Разработан метод **drawItem(input variables)** в соответствии с диаграммой активности на Рисунок 8. В отличии от метода базового класса, разработанный может строить точки, взятые с контейнера с индексами прореженных данных. На вход подается индекс текущего элемента. Для построения ступеньки требуется найти индекс парной точки. Работа

Поиск парной точки для построения:

```
 Integer previousPoint = filteredLinesMap.size() > 0 ?
filteredLinesMap.get(item) : null;
if (previousPoint == null) {
   previousPoint = item - 1;
}
```

Код метода приведен в ПРИЛОЖЕНИЕ Б.

5.5. Появление окошка с координатами ближайшей к курсору мыши точки

Был переопределен и доработан имеющийся в классе **PlotFrame** метод **mouseMove().** Принцип работы был показан на Рисунок 10. Текущая версия метода поддерживает вывод координат прореженных точек.

Теперь в методе присутствует проверка, на соответствие графика типу **XYPlotWithFiltering**:

```
XYPlotWithFiltering filteredPlot = plot instanceof XYPlotWithFiltering ?
(XYPlotWithFiltering) plot      : null;
```

В положительном случае поиск ближайшей к курсору точки производится по индексам прореженных точек.

```
if (filteredPlot != null &&
filteredPlot.getProxyDataSet().getFilteredMap().size() > 0) {
   Collection<Integer> filteredPoints =
filteredPlot.getProxyDataSet().getFilteredMap().keySet();
   for (Integer index : filteredPoints) {
      valueX = dataset.getXValue(0, index);
      valueY = dataset.getYValue(0, index);
      widgetPoint = plotToSwt(valueX, valueY);
      distanceToMouse = getDistance(widgetPoint, mousePoint);

      if (distanceToMouse < previousDistance) {
         previousDistance = distanceToMouse;
         currentIndex = index;
      }
   }
```

В противном случае метод работает с набором данных, полученным напрямую с графика:

```
else {
   for (int index = 0; index < itemsCount; index++) {
      valueX = dataset.getXValue(0, index);
      valueY = dataset.getYValue(0, index);
      widgetPoint = plotToSwt(valueX, valueY);
      distanceToMouse = getDistance(widgetPoint, mousePoint);

      if (distanceToMouse < previousDistance) {
         previousDistance = distanceToMouse;
         currentIndex = index;
      }
   }
```

Код метода приведен в ПРИЛОЖЕНИЕ Б.

5.6. Изменение порядка добавления точек в серию графика

Был изменен фрагмент кода, имеющийся в методе **run()** класса **RealTimeUpdateRunnable**. Подсистема вывода графиков основана на подписчиках, и **run()** вызывается всякий раз, когда симулятор добавляет новые записи в базу данных. Метод **add(item.x, item.y, boolean flag)** класса **XYSeries** в качестве последнего параметра имеет флаг, отвечающий за отправку сообщения, вызывающего полную перестройку графика. В предыдущей версии метода **run()** при каждом добавлении точки отправлялось сообщение **SeriesChangedEvent()**, что и являлось причиной сильных задержек при первом построении графика. В текущей версии сообщение посылается только один раз вместе с последней добавленной точкой:

```
List<PlotItem> startItems = items.subList(0, items.size() - 1);

for (PlotItem item : startItems) {
   newSeries.add(item.x, item.y, false);
}

PlotItem lastItem = items.get(items.size() - 1);
newSeries.add(lastItem.x, lastItem.y, true);
```

Принцип работы был продемонстрирован на Рисунок 9. Код метода приведен в ПРИЛОЖЕНИЕ Б.

5.7. Реализация фабричного метода для графиков с прореживанием

Разработан программный код метода, собирающего все объекты классов с прореживанием в соответствии с интерфейсом библиотеки JFreeChart, отвечающую за работу подсистемы вывода графиков. Метод разработан в соответствии с Рисунок 11.

Создание осей и меток на них:

```
DateAxis xAxis = new DateAxis(xAxisLabel);
NumberAxis yAxis = new NumberAxis(yAxisLabel);
```

Создание рендерера и объекта-графика:

```
XYItemRenderer renderer = new XYFilteringStepRenderer(toolTipGenerator,
urlGenerator);
XYPlotWithFiltering plot = new XYPlotWithFiltering(dataset, xAxis, yAxis,
null);
```

Задание параметров, состоящих из созданных прежде объектов, графику:

```
plot.setRenderer(renderer);
plot.setOrientation(orientation);
```

Создание конечного объекта типа **JFreeChart**:

```
JFreeChart chart = new JFreeChart(title, JFreeChart.DEFAULT_TITLE_FONT,
plot, legend);
```

5.8. Разработка кода для тестирования подсистемы

Поскольку необходимо получить объективные данные для оценки работоспособности разработанной подсистемы (время построения графика и обновления его вида), были разработаны фрагменты-вставки.

Время в мс. измеряет встроенная функция: **System.currentTimeMillis()**.

За начало отсчета при тестировании времени вывода графика на экран принимается инициирование пользователем построение графика: в методе **show(CollectedDataNode node),** лежащим внутри класса **PlotMenuItem**:

```
@Override
public void show(CollectedDataNode node) {
   try {
      System.out.println("Initialization Started" +
System.currentTimeMillis());
      if (PlotView.getOpenedPlotMap().containsKey(node)) {

PlatformUI.getWorkbench().getActiveWorkbenchWindow().getActivePage().show
View(PlotView.ID,
                String.valueOf(PlotView.getOpenedPlotMap().get(node)),
IWorkbenchPage.VIEW_ACTIVATE);
      } else {
         PlotView newView = (PlotView)
PlatformUI.getWorkbench().getActiveWorkbenchWindow().getActivePage()
                .showView(PlotView.ID, String.valueOf(secondaryID),
IWorkbenchPage.VIEW_ACTIVATE);
         PlotView.addToOpenedPlotMap(node, secondaryID);
         secondaryID++;

         newView.initialize(node);
      }
   } catch (PartInitException e) {
      e.printStackTrace();
   }
}
```

Конец измеряемого отрезка лежит внутри класса **XYPlotWithFiltering**, в методе **render(input variables)**. Собираем время после отрисовки рендерером всех точек:

```
state.endSeriesPass(dataset, series, firstItem, lastItem, pass,
passCount);
      System.out.println("Rendering finished" +
System.currentTimeMillis());
```

Код метода приведен в ПРИЛОЖЕНИЕ Б без функции измерения времени.

Для оценки скорости работы подсистемы было выбран шаг для замера в 3 обращения к стандартным методам масштабирования как **zoomInDomain(input variables)** и **zoomOutDomain(input variables)**. Для удобства проверка начинается по нажатию на клавиши «стрелка вниз» и «стрелка вверх». Был доработан код метода класса **KeyListener**: **keyPressed(KeyEvent e)**.

Для проверки уменьшения масштаба:

```
case SWT.ARROW_UP:
   System.out.println("ZoomingStarted" + System.currentTimeMillis());
   zoomOutDomain(getChart().getXYPlot().getDomainAxis().getUpperBound() /
2, 0);
   zoomOutDomain(getChart().getXYPlot().getDomainAxis().getUpperBound() /
2, 0);
   zoomOutDomain(getChart().getXYPlot().getDomainAxis().getUpperBound() /
2, 0);
      return;
```

Для проверки увеличения масштаба:

```
case SWT.ARROW_DOWN:
   System.out.println("ZoomingStarted" + System.currentTimeMillis());
   zoomInDomain(getChart().getXYPlot().getDomainAxis().getUpperBound() /
2, 0);
   zoomInDomain(getChart().getXYPlot().getDomainAxis().getUpperBound() /
2, 0);
   zoomInDomain(getChart().getXYPlot().getDomainAxis().getUpperBound() /
2, 0);
      return;
```

Для проверки навигации по графику:

```
case SWT.ARROW_LEFT:
   System.out.println("ScroolStarted" + System.currentTimeMillis());
   double upper = getChart().getXYPlot().getDomainAxis().getUpperBound();
   double lower = getChart().getXYPlot().getDomainAxis().getLowerBound();
   double interval = (upper - lower) / 4;
   getChart().getXYPlot().getDomainAxis().setLowerBound(lower -
interval);
   getChart().getXYPlot().getDomainAxis().setUpperBound(upper -
interval);
   return;
```

Разработанный метод использовался для формирования графиков и гистограмм с результатами в разделе апробирование.

6. Апробирование подсистемы

Апробирование разработанной подсистемы осуществлялось при помощи многократного тестирования функционала. В ходе тестирования были построены графики по всем необходимым параметрам модели (параметры ресурсов, результаты, образцы). Количество точек, выводимое на графике, регулировалось изменением модельного времени. Модели, использованные для тестирования, приведены в ПРИЛОЖЕНИЕ В.

На График 2 приведена зависимость времени построения графиков подсистемы от количества одновременно выводимых точек. 631 мс занимает построение графика с 10 точками, что демонстрирует издержки подсистемы на загрузку классов при первом построении любого графика. Самый объемный график в 385000 значений строится всего за 1521 мс.

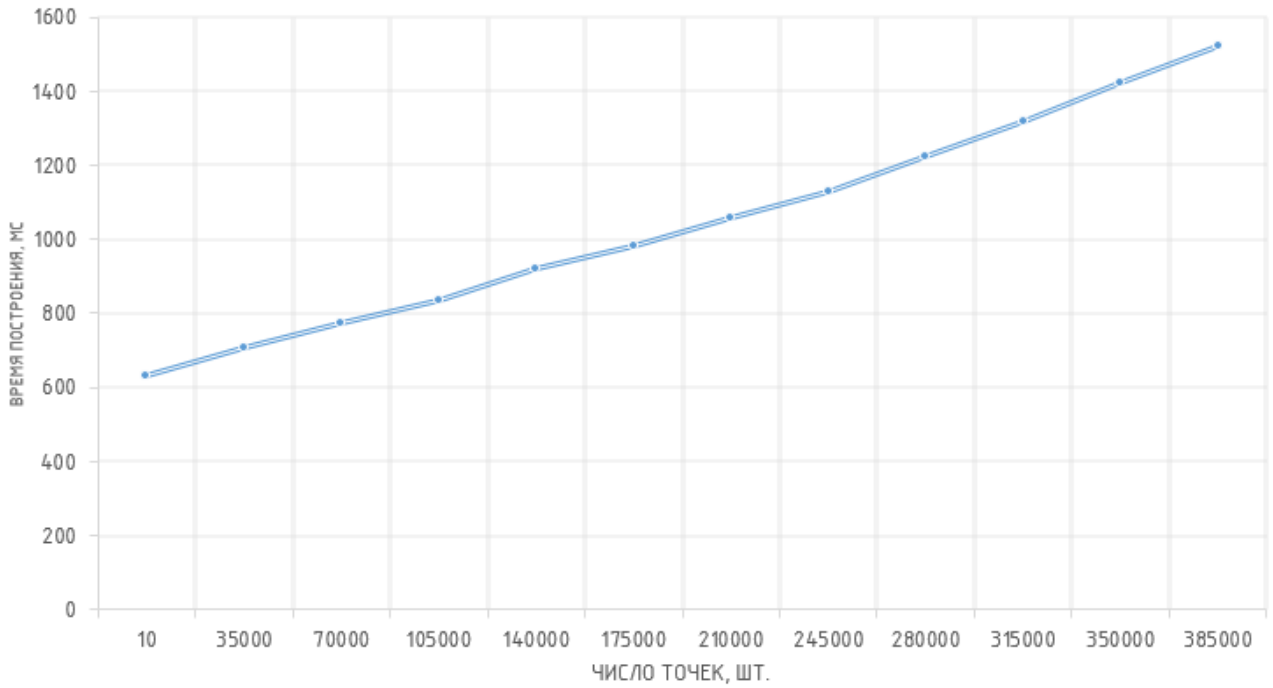


График 2. Зависимости времени построения графиков подсистемы от числа точек

На График 3 сравнивается производительность построения графиков с и без подсистемы прореживания графиков.

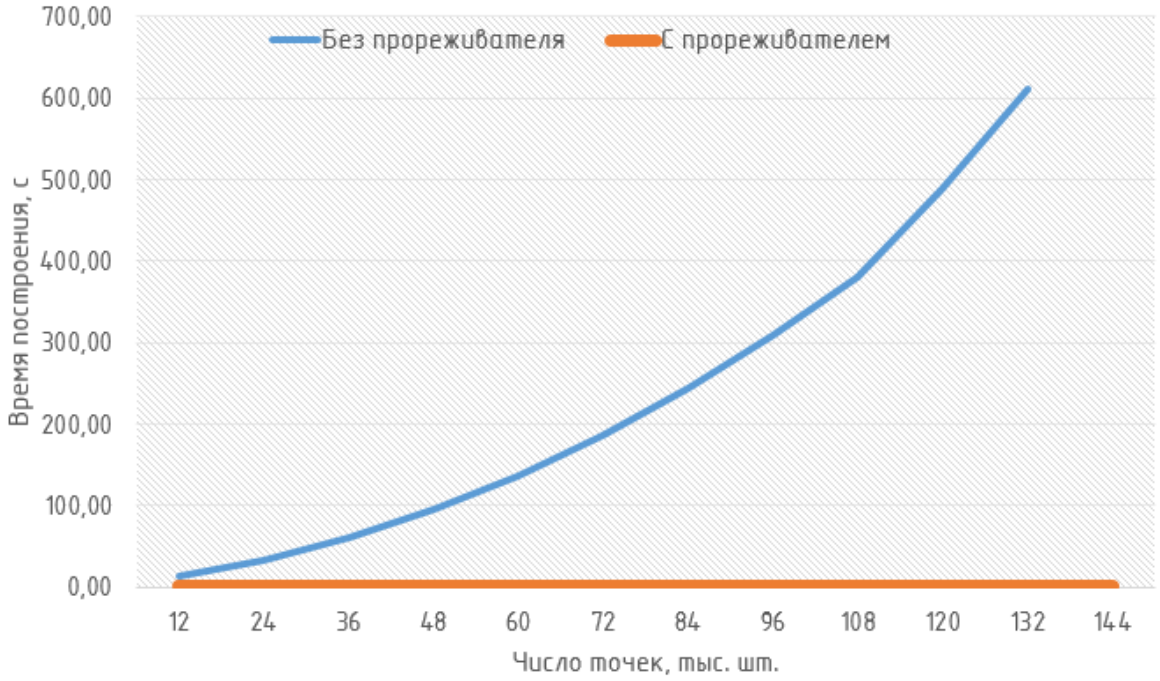
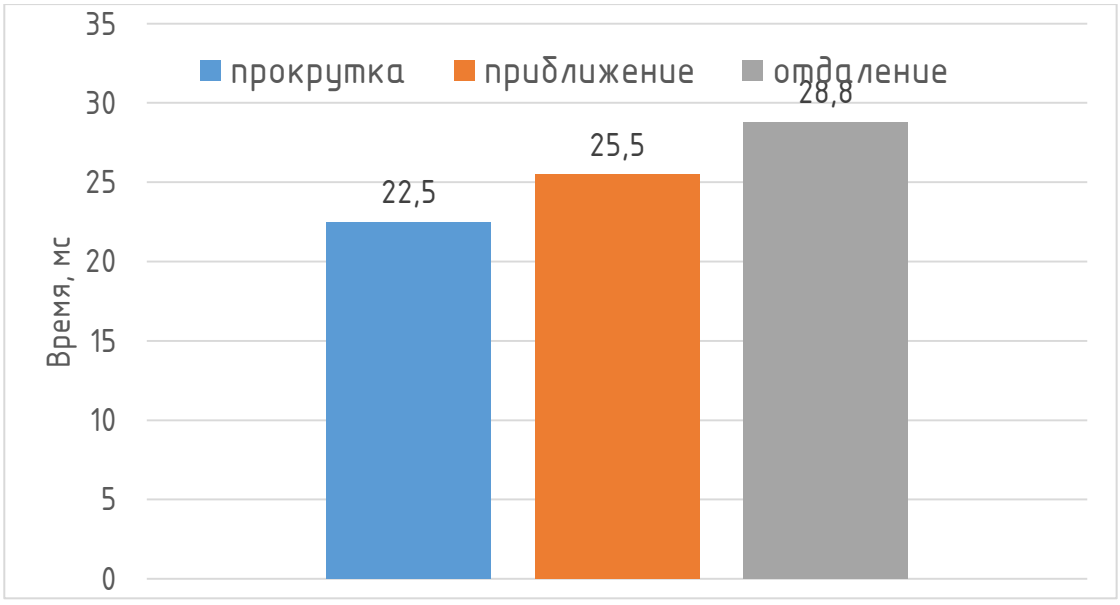


График 3. Сравнение скорости построения графиков

Также изменилась скорость перерисовки графиков. Требуемое время для обновления вида графика с 385000 точками при выполнении операций увеличения или уменьшения масштаба графика, а также при прокрутке окна с данными при помощи слайдеров отражено на Гистограмма 2.



Гистограмма 2. Время обновления графика

Поскольку на Гистограмма 2 отображены зависимости для графиков с предельным количеством одновременно выводимых точек, то можно заключить,

что обновление вида графиков с меньшим числом данных происходит также достаточно быстро.

На Рисунок 16 сравнивается скорость обновления вида графиков до и после реализации подсистемы прореживания.

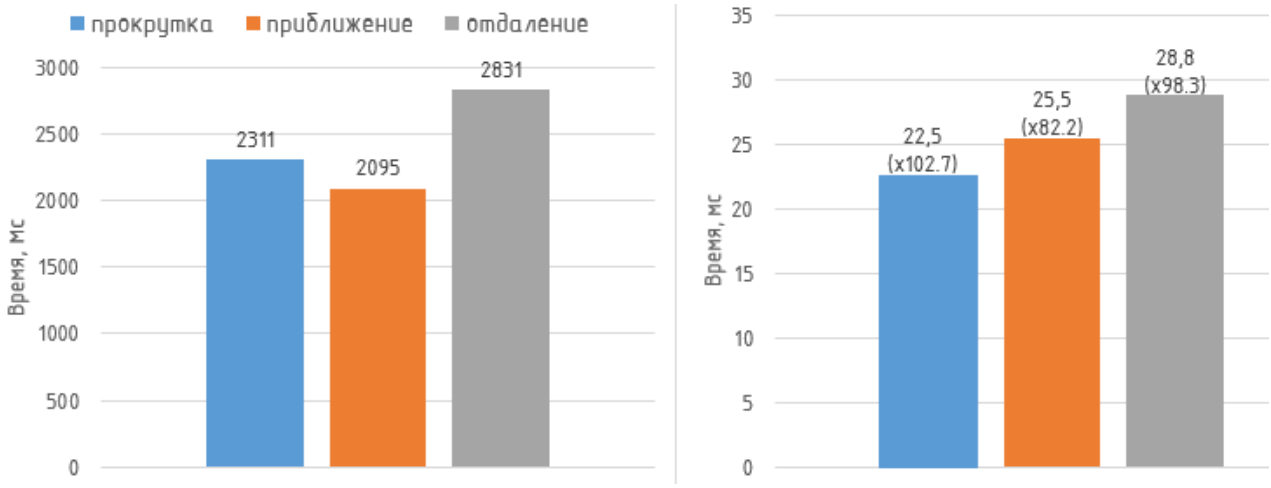


Рисунок 16. Сравнение времени обновления вида графиков с и без прореживателя

В итоге была разработана и доведена до реализации подсистема прореживания графиков. В ПРИЛОЖЕНИЕ А прикреплены диаграмма классов, которая отражает структуру подсистемы и отношениями между ее компонентами, и диаграмма последовательности прореживания графиков.

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

ЗАКЛЮЧЕНИЕ

В рамках данной выпускной квалификационной работы были получены следующие результаты:

- Проведено предпроектное исследование системы ИМ Rao X и определены недостатки существующей подсистемы вывода графиков;
- На этапе концептуального проектирования была выбрана общая методология проектирования, выделена подсистема из среды, и было сформировано техническое задание на проектирование подсистемы;
- На этапе технического проектирования были разработаны диаграммы активностей для методов разработанной подсистемы и определен функционал разрабатываемых классов;
- На рабочем этапе проектирования был разработан программный код для классов подсистемы и были составлены диаграмма классов подсистемы прореживания и диаграмма последовательности прореживания графика;
- Апробирование тестируемой подсистемы показало, что подсистема удовлетворяет всем указанным в техническом задании требованиям.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Емельянов В.В., Ясиновский С.И. Имитационное моделирование систем: Учебн. пособие. – М.: Издательство МГТУ им. Н.Э. Баумана, 2009. – 584 с.
2. Документация по языку РДО [Электронный ресурс] // Справка по языку РДО URL: http://www.rdostudio.com/help/index.html (дата обращения: 04.03.2017).
3. Рахматулин В.В., Доработка подсистемы вывода графиков. Курсовой [Электронный ресурс] // Rao X – URL: http://raox.ru/?p=1367
4. Герберт Шилдт. [Herbert Schildt] Java 8: руководство для начинающих, 6-е изд. Пер. с англ. – М. ООО "И.Д. Вильяме", 2015. – 720 с.
5. Википедия [Электронный ресурс] // Модульное программирование URL: https://en.wikipedia.org/wiki/Modular_programming (дата обращения: 06.03.2017).
6. Википедия [Электронный ресурс] // Обзор технологии проектирования программ с помощью ООП. URL: https://en.wikipedia.org/wiki/Object-oriented_programming (дата обращения: 06.03.2017).
7. Википедия [Электронный ресурс] // Обзор диаграмм на языке UML. URL: https://ru.wikipedia.org/wiki/UML (дата обращения: 10.03.2017).
8. Java™ Platform, Standard Edition 8 API Specification [Электронный ресурс] // Документация по Java 8 в открытом доступе URL: https://docs.oracle.com/javase/8/docs/api/ (дата обращения: 06.05.2017).
9. SWT Documentation [Электронный ресурс] // Документация по библиотеке SWT в открытом доступе URL: https://www.eclipse.org/swt/docs.php (дата обращения: 04.04.2017).
10. Википедия [Электронный ресурс] // Примеры применения паттерна наблюдатель на языке Java. URL: https://en.wikipedia.org/wiki/Observer_pattern (дата обращения: 12.03.2017).

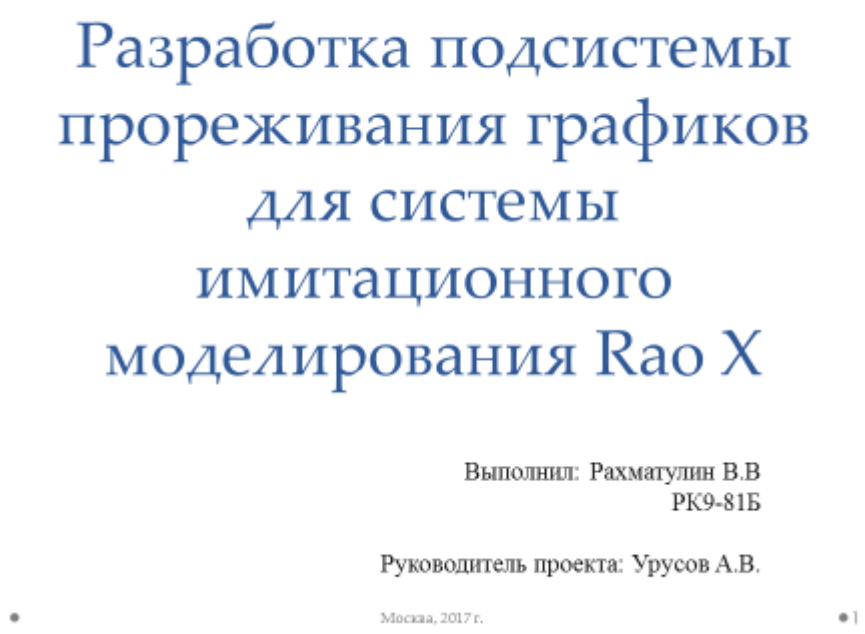
11. Дроговоз П. А. Методические указания по использованию программного продукта ARIS; МГТУ им. Н. Э. Баумана. – М. : Изд-во МГТУ им. Н. Э. Баумана, 2007. – 23 с.
12. Буч Г., Рамбо Дж., Джекобсон А. Язык UML. Руководство пользователя : пер. с англ. / Буч Г., Рамбо Дж., Джекобсон А. ; пер. Слинкин А. А. – 2-е изд., стер. – М. : ДМК Пресс ; СПб. : Питер, 2004. – 429 с. : ил. – (Объектно-ориентированные технологии в программировании)
13. Мартин Р. Чистый код. Создание, анализ и рефакторинг / пер. с анг. Е. Матвеев – СПб.: Питер, 2010. – 464 с.
14. Википедия [Электронный ресурс] // Паттерн проектирования фабричный метод URL: https://en.wikipedia.org/wiki/Factory_method_pattern (дата обращения: 08.05.2017).
15. Map Documentation [Электронный ресурс] // Документация по классам Map URL: https://docs.oracle.com/javase/8/docs/api (дата обращения: 04.04.2017).

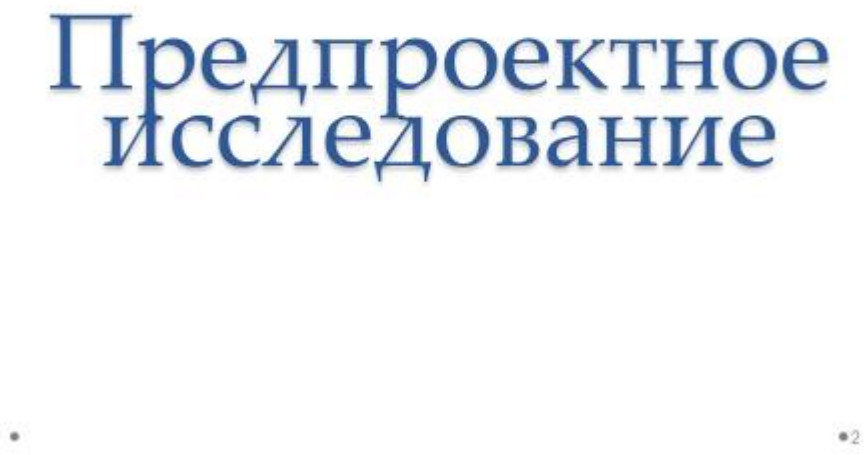
СПИСОК ИСПОЛЬЗОВАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

1. RAO X
2. Eclipse IDE for Java Developers Mars 2.0
3. openjdk version "1.8.0_40-internal"
4. UMLet 14.2
5. Microsoft® Office Word 2013
6. Microsoft® Office Exel 2013
7. Microsoft® Visio 2010

ПРИЛОЖЕНИЕ А

Презентация





Графики в системе RaoX

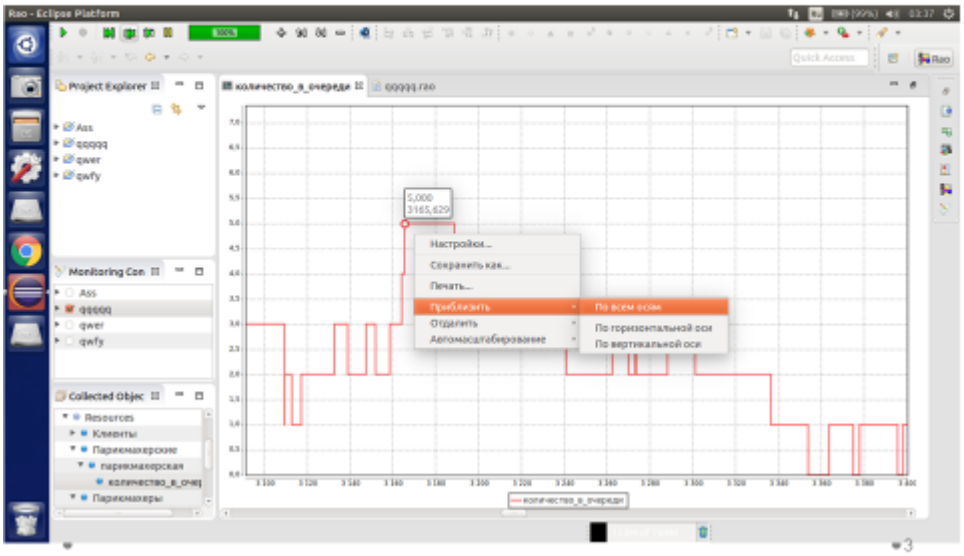


Диаграмма коммуникаций участников процесса формирования графика

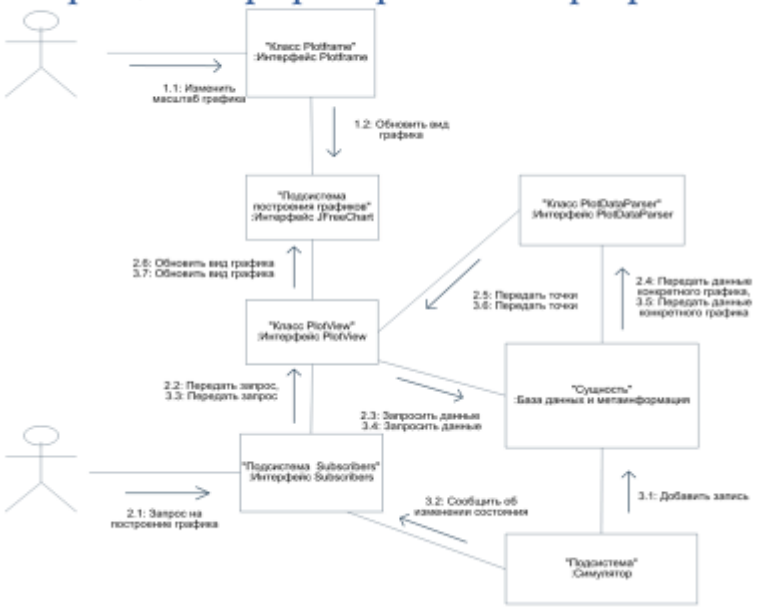
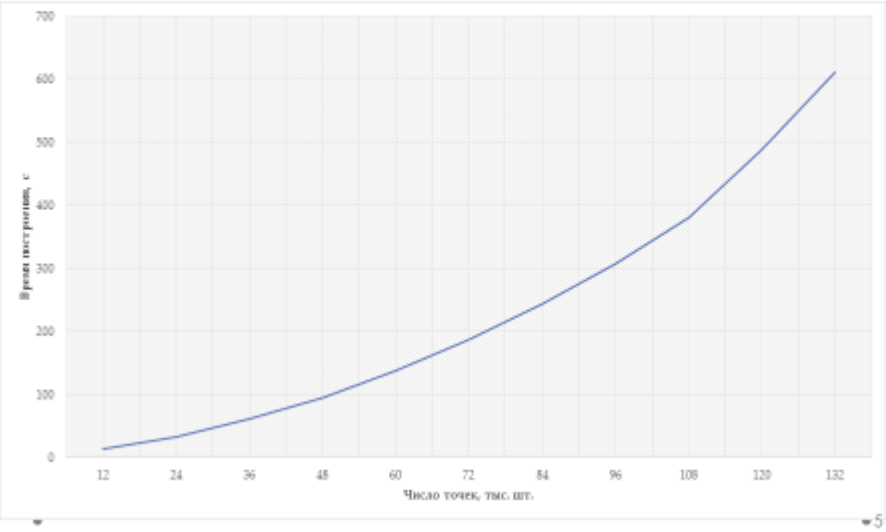
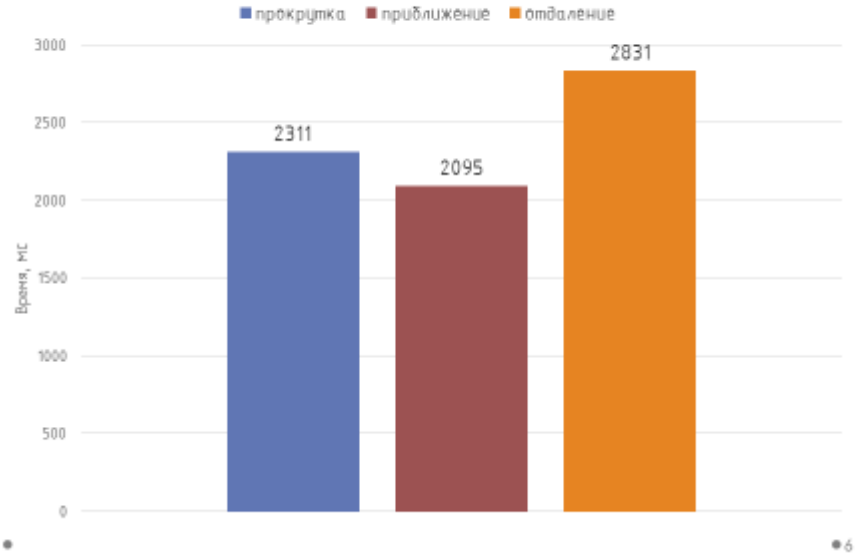


График зависимости времени построения графиков подсистемы от числа точек



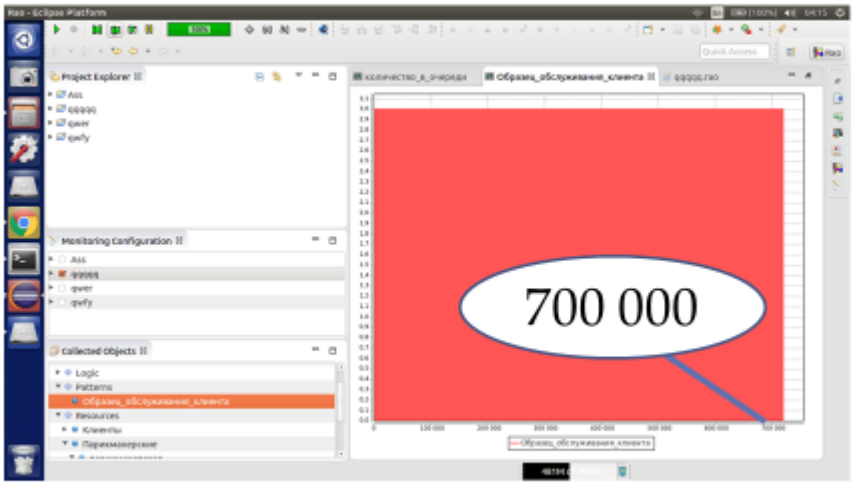
5

Диаграмма времени выполнения операций



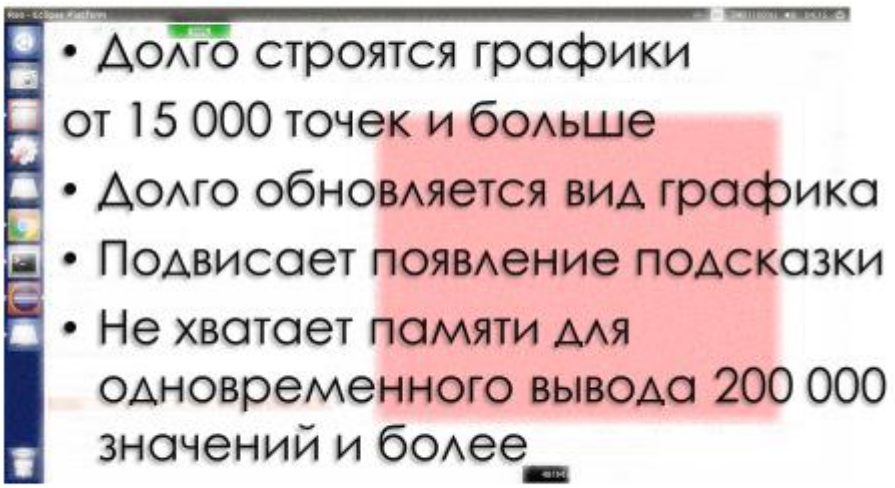
6

Недостатки:



6

Недостатки:



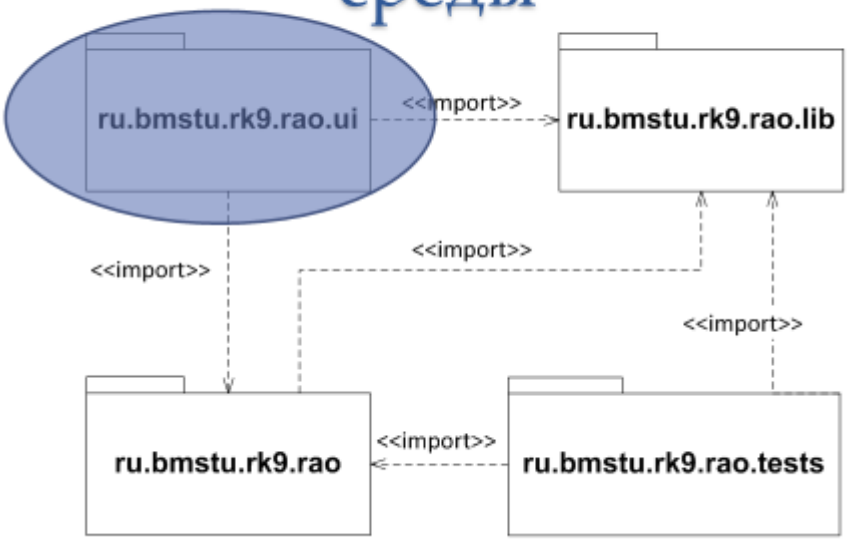
10

Концептуальное проектирование

- Язык программирования Java
- Следование принципам ООП
- Использование системы контроля версий Git
- Модульность

11

Выделение системы из среды

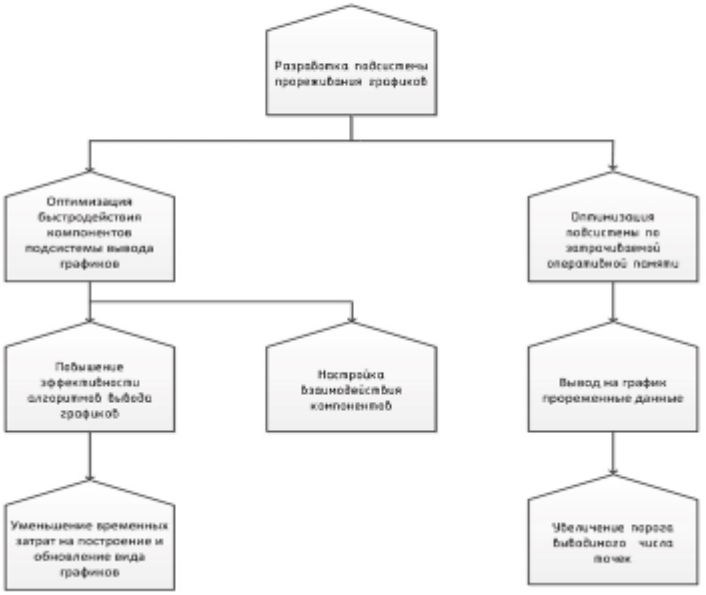


13

Техническое проектирование

15

Дерево целей



14

Диаграмма активности добавления точек в серию графика

17

Диаграмма активности прореживания графиков

16

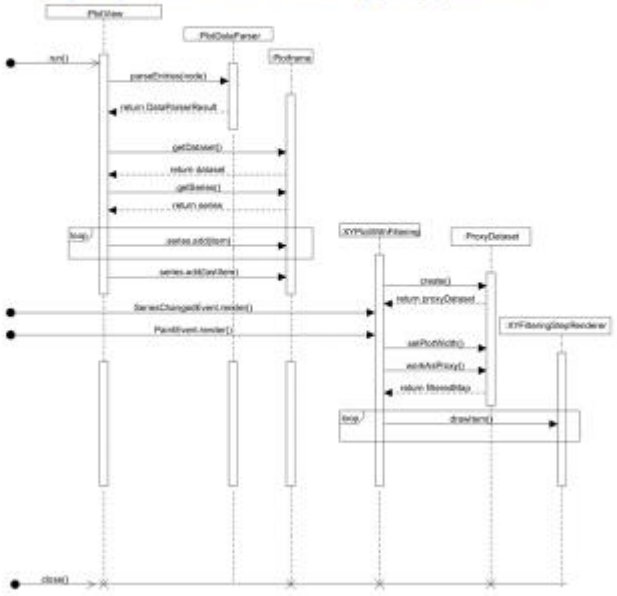
Диаграмма активности всплывающей подсказки

18

Диаграмма классов

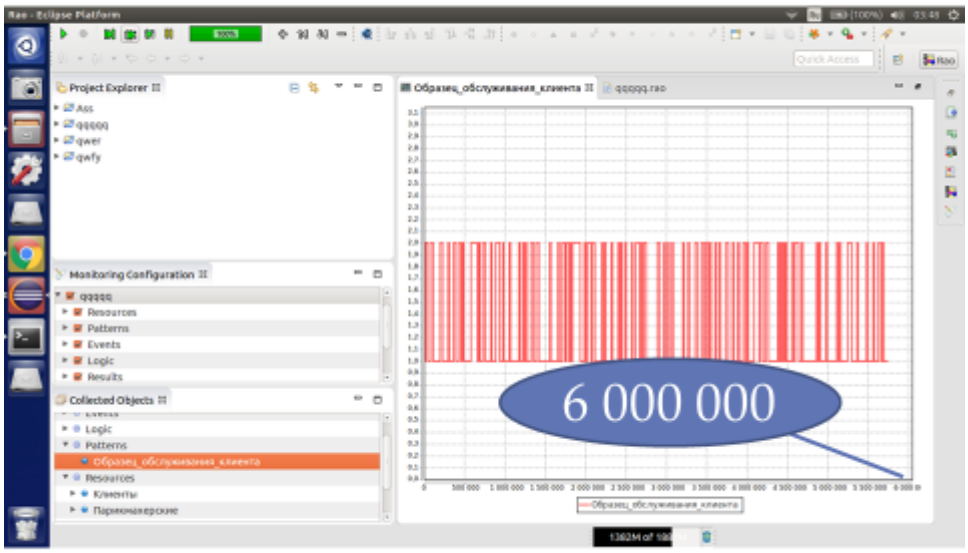
19

Диаграмма последовательности прореживания графиков



20

Результаты



21

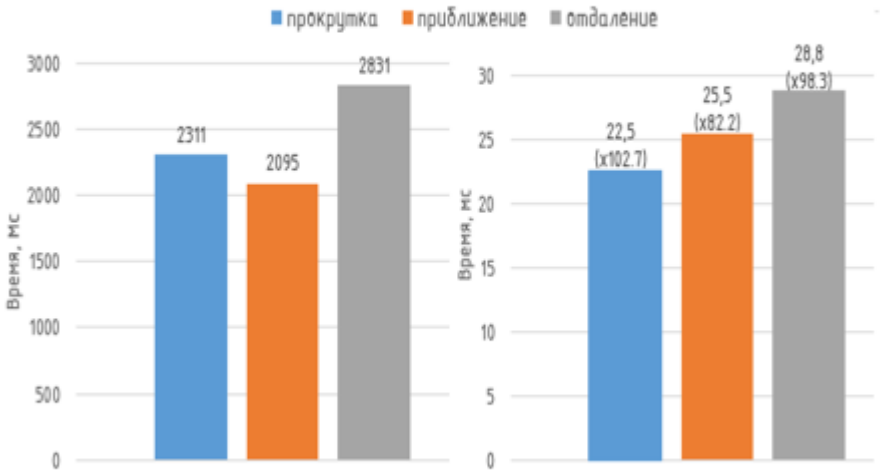

Длительность построения графиков
ВРЕМЯ ПОСТРОЕНИЯ, МС
1600
1400
1200
1000
800
600
400
200
0
10
35000
70000
105000
140000
175000
210000
245000
280000
315000
350000
385000
ЧИСЛО ТОЧЕК, ШТ.

24


Сравнение времени построения графиков
Без прореживателя
С прореживателем
ВРЕМЯ ПОСТРОЕНИЯ, С
700,00
600,00
500,00
400,00
300,00
200,00
100,00
0,00
0,1
12
24
36
48
60
72
84
96
108
120
132
144
ЧИСЛО ТОЧЕК, ТЫС. ШТ.

25

Сравнение длительности обновления вида



27

Результаты

- Вывод на график одновременно до 380 000 значений
- Существенное ускорение построения графиков
- Быстрое обновление вида графиков при навигации и масштабировании
- Вывод всплывающей подсказки

27

Недостатки

- Невозможно выводить несколько кривых на одной временной оси
- Невозможность настройки прореживания пользователем

28

Спасибо за внимание

29

ПРИЛОЖЕНИЕ Б

Исходный код разработанных классов

1. PlotFrame

```
package ru.bmstu.rk9.rao.ui.plot;

import java.awt.BasicStroke;
import java.awt.Paint;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Point2D;
import java.util.Collection;

import org.eclipse.jface.window.DefaultToolTip;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.KeyEvent;
import org.eclipse.swt.events.KeyListener;
import org.eclipse.swt.events.MouseEvent;
import org.eclipse.swt.events.MouseMoveListener;
import org.eclipse.swt.events.MouseWheelListener;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.events.SelectionListener;
import org.eclipse.swt.graphics.Color;
import org.eclipse.swt.graphics.Point;
import org.eclipse.swt.graphics.Rectangle;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;
import org.eclipse.swt.widgets.Slider;
import org.jfree.chart.annotations.XYShapeAnnotation;
import org.jfree.chart.axis.SymbolAxis;
import org.jfree.chart.axis.ValueAxis;
import org.jfree.chart.plot.XYPlot;
import org.jfree.data.xy.XYDataset;
import org.jfree.experimental.chart.swt.ChartComposite;

public class PlotFrame extends ChartComposite {
   private Slider horizontalSlider;
   private Slider verticalSlider;
   private double horizontalMaximum;
   private double verticalMaximum;
   private double horizontalRatio;
   private double verticalRatio;
   private final ExtendedToolTip toolTip;

   public PlotFrame(final Composite comp, final int style) {
      super(comp, style, null, ChartComposite.DEFAULT_WIDTH,
ChartComposite.DEFAULT_HEIGHT, 0, 0, Integer.MAX_VALUE,
            Integer.MAX_VALUE, ChartComposite.DEFAULT_BUFFER_USED, true,
true, true, true, true);
      addSWTListener(new PlotKeyListener());
      addMouseWheelListener(new PlotMouseWheelListener());
      addSWTListener(new PlotMouseMoveListener());
      setZoomInFactor(0.75);
      setZoomOutFactor(1.25);
      toolTip = new ExtendedToolTip(this, org.eclipse.swt.SWT.BALLOON,
false);
      toolTip.setBackgroundColor(new Color(comp.getDisplay(), 255, 255,
255));
      toolTip.setForegroundColor(new Color(comp.getDisplay(), 128, 128,
```

```
128));
      toolTip.setRespectDisplayBounds(false);
      toolTip.setShift(new Point(0, -50));
      toolTip.setHideDelay(0);
      toolTip.setHideOnMouseDown(false);
      toolTip.setPopupDelay(0);
   }

   private final Point plotToSwt(double x, double y) {
      Rectangle rectangle = PlotFrame.this.getScreenDataArea();
      final ValueAxis domainAxis = getChart().getXYPlot().getDomainAxis();
      final ValueAxis rangeAxis = getChart().getXYPlot().getRangeAxis();
      double widthInAxisUnits = domainAxis.getUpperBound() -
domainAxis.getLowerBound();
      double heightInAxisUnits = rangeAxis.getUpperBound() -
rangeAxis.getLowerBound();
      double relativeX = (x - domainAxis.getLowerBound()) /
widthInAxisUnits;
      double relativeY = (rangeAxis.getUpperBound() - y) /
heightInAxisUnits;
      double screenX = relativeX * rectangle.width + rectangle.x;
      double screenY = relativeY * rectangle.height + rectangle.y;

      return new Point((int) Math.round(screenX), (int)
Math.round(screenY));
   }

   class ExtendedToolTip extends DefaultToolTip {
      private boolean isActive = false;

      public ExtendedToolTip(Control control, int style, boolean
manualActivation) {
         super(control, style, manualActivation);
      }
   }

   private final Point2D swtToPlot(int x, int y) {
      Rectangle rectangle = PlotFrame.this.getScreenDataArea();
      final ValueAxis domainAxis = getChart().getXYPlot().getDomainAxis();
      final ValueAxis rangeAxis = getChart().getXYPlot().getRangeAxis();
      final double widthInAxisUnits = domainAxis.getUpperBound() -
domainAxis.getLowerBound();
      final double heightInAxisUnits = rangeAxis.getUpperBound() -
rangeAxis.getLowerBound();
      double relativeX = ((double) x - rectangle.x) / (rectangle.width);
      double relativeY = ((double) y - rectangle.y) / (rectangle.height);
      double xPlot = domainAxis.getLowerBound() + relativeX *
widthInAxisUnits;
      double yPlot = rangeAxis.getUpperBound() - relativeY *
heightInAxisUnits;

      return new Point2D.Double(xPlot, yPlot);
   }

   class PlotMouseMoveListener implements MouseMoveListener {
      final static int MAX_HINT_DISTANCE = 50;
      int previousIndex = -1;
      int distanceToMouse = 0;
      Point widgetPoint = null;
      double valueX;
      double valueY;
      final static int BORDER = 1;

      final private int getDistance(Point x1, Point x2) {
```

```
            return (int) Math.sqrt(Math.pow((x1.x - x2.x), 2) +
Math.pow((x1.y - x2.y), 2));
        }

        @Override
        public void mouseMove(MouseEvent e) {
            Point mousePoint = new Point(Math.round(e.x), Math.round(e.y));
            if (getChart() == null) {
                return;
            }
            XYPlot plot = getChart().getXYPlot();
            if (plot == null) {
                return;
            }

            XYDataset dataset = getChart().getXYPlot().getDataset();
            int itemsCount = dataset.getItemCount(0);
            java.awt.Color backgroundColor = (java.awt.Color)
getChart().getXYPlot().getBackgroundPaint();

            if (itemsCount > 0) {
                double previousDistance = Double.MAX_VALUE;
                int currentIndex = -1;
                XYPlotWithFiltering filteredPlot = plot instanceof
XYPlotWithFiltering ? (XYPlotWithFiltering) plot
                        : null;
                if (filteredPlot != null &&
filteredPlot.getProxyDataSet().getFilteredMap().size() > 0) {
                    Collection<Integer> filteredPoints =
filteredPlot.getProxyDataSet().getFilteredMap().keySet();
                    for (Integer index : filteredPoints) {
                        valueX = dataset.getXValue(0, index);
                        valueY = dataset.getYValue(0, index);
                        widgetPoint = plotToSwt(valueX, valueY);
                        distanceToMouse = getDistance(widgetPoint, mousePoint);

                        if (distanceToMouse < previousDistance) {
                            previousDistance = distanceToMouse;
                            currentIndex = index;
                        }
                    }
                } else {
                    for (int index = 0; index < itemsCount; index++) {
                        valueX = dataset.getXValue(0, index);
                        valueY = dataset.getYValue(0, index);
                        widgetPoint = plotToSwt(valueX, valueY);
                        distanceToMouse = getDistance(widgetPoint, mousePoint);

                        if (distanceToMouse < previousDistance) {
                            previousDistance = distanceToMouse;
                            currentIndex = index;
                        }
                    }
                }
                if (currentIndex >= 0) {
                    mousePoint = new Point(Math.round(e.x), Math.round(e.y));
                    valueX = dataset.getXValue(0, currentIndex);
                    valueY = dataset.getYValue(0, currentIndex);
                    widgetPoint = plotToSwt(valueX, valueY);
                    distanceToMouse = getDistance(widgetPoint, mousePoint);
                    Rectangle screenDataArea = getScreenDataArea();
                    Rectangle realAreaOfPlot = new Rectangle(screenDataArea.x -
BORDER, screenDataArea.y - BORDER,
                            screenDataArea.width + 2 * BORDER,
```

```
screenDataArea.height + 2 * BORDER);

                if (realAreaOfPlot.contains(mousePoint) && distanceToMouse
< MAX_HINT_DISTANCE
                        && (toolTip.isActive == false || previousIndex !=
currentIndex)) {
                    getChart().getXYPlot().clearAnnotations();
                    toolTip.isActive = true;
                    previousIndex = currentIndex;
                    Rectangle rectangle =
PlotFrame.this.getScreenDataArea();
                    final ValueAxis domainAxis =
getChart().getXYPlot().getDomainAxis();
                    final ValueAxis rangeAxis =
getChart().getXYPlot().getRangeAxis();
                    double widthInAxisUnits = domainAxis.getUpperBound() -
domainAxis.getLowerBound();
                    double heightInAxisUnits = rangeAxis.getUpperBound() -
rangeAxis.getLowerBound();
                    double radiusOfCircleX = 5 * widthInAxisUnits /
rectangle.width;
                    double radiusOfCircleY = 5 * heightInAxisUnits /
rectangle.height;

                    Ellipse2D Circle = new Ellipse2D.Float((float) (valueX -
radiusOfCircleX),
                            (float) (valueY - radiusOfCircleY), (float) (2 *
radiusOfCircleX),
                            (float) (2 * radiusOfCircleY));
                    Paint seriesColor =
getChart().getXYPlot().getRenderer().getSeriesPaint(0);
                    XYShapeAnnotation annotation = new
XYShapeAnnotation(Circle, new BasicStroke(3.0f), seriesColor,
                            backgroundColor);
                    getChart().getXYPlot().addAnnotation(annotation, false);
                    final String tooltipText;

                    if (rangeAxis instanceof SymbolAxis) {
                        int valueYId = (int) Math.round(valueY);
                        final SymbolAxis symbolRangeAxis = (SymbolAxis)
rangeAxis;
                        tooltipText = String.format("%s%n%.3f",
symbolRangeAxis.getSymbols()[valueYId], valueX);
                    } else {
                        tooltipText = String.format("%.3f%n%.3f", valueY,
valueX);
                    }

                    toolTip.setText(tooltipText);
                    toolTip.show(widgetPoint);
                }

                if ((distanceToMouse > MAX_HINT_DISTANCE ||
!realAreaOfPlot.contains(mousePoint))
                        && toolTip.isActive == true) {
                    toolTip.hide();
                    toolTip.isActive = false;
                    getChart().getXYPlot().clearAnnotations();
                }
            }
        }
    }
}
```

```
   class PlotKeyListener implements KeyListener {
      @Override
      public void keyPressed(KeyEvent e) {
         switch (e.keyCode) {
         case SWT.SHIFT:
            setRangeZoomable(true);
            return;
         case SWT.CTRL:
            setDomainZoomable(true);
            return;
         case SWT.SPACE:
            restoreAutoBounds();
            return;
         }
      }

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

   class PlotMouseWheelListener implements MouseWheelListener {
      @Override
      public void mouseScrolled(MouseEvent e) {
         if (e.count > 0 && isRangeZoomable())
            zoomInRange(e.x, e.y);
         if (e.count < 0 && isRangeZoomable())
            zoomOutRange(e.x, e.y);
         if (e.count > 0 && isDomainZoomable())
            zoomInDomain(e.x, e.y);
         if (e.count < 0 && isDomainZoomable())
            zoomOutDomain(e.x, e.y);
         if (e.count > 0 && isDomainZoomable() && isRangeZoomable())
            zoomInBoth(e.x, e.y);
         if (e.count < 0 && isDomainZoomable() && isRangeZoomable())
            zoomOutBoth(e.x, e.y);
      }
   }

   public final void setSliders(final Slider horizontalSlider, final
Slider verticalSlider) {
      this.horizontalSlider = horizontalSlider;
      this.verticalSlider = verticalSlider;
      horizontalSlider.setMinimum(0);
      verticalSlider.setMinimum(0);
      this.horizontalSlider.addSelectionListener(new SelectionListener() {
         @Override
         public void widgetSelected(SelectionEvent e) {

getChart().getXYPlot().getDomainAxis().setLowerBound(horizontalSlider.getS
election() / horizontalRatio);
            getChart().getXYPlot().getDomainAxis().setUpperBound(
                  (horizontalSlider.getThumb() +
horizontalSlider.getSelection()) / horizontalRatio);
         }

         @Override
         public void widgetDefaultSelected(SelectionEvent e) {
         }
      });
```

```
      this.verticalSlider.addSelectionListener(new SelectionListener() {
         @Override
         public void widgetSelected(SelectionEvent e) {
            getChart().getXYPlot().getRangeAxis().setLowerBound(
                  (verticalSlider.getMaximum() -
verticalSlider.getSelection() - verticalSlider.getThumb())
                        / verticalRatio);
            getChart().getXYPlot().getRangeAxis()
                  .setUpperBound((verticalSlider.getMaximum() -
verticalSlider.getSelection()) / verticalRatio);
         }

         @Override
         public void widgetDefaultSelected(SelectionEvent e) {
         }
      });
   }

   public final void setChartMaximum(final double horizontalMaximum, final
double verticalMaximum) {
      final double freeSpaceCoefficient = 1.05;
      this.horizontalMaximum = horizontalMaximum * freeSpaceCoefficient;
      this.verticalMaximum = verticalMaximum * freeSpaceCoefficient;
   }

   @Override
   public void zoomInDomain(double x, double y) {
      super.zoomInDomain(x, y);
      updateSliders();
      getChart().getXYPlot().clearAnnotations();
   }

   @Override
   public void zoomInRange(double x, double y) {
      super.zoomInRange(x, y);
      updateSliders();
      getChart().getXYPlot().clearAnnotations();
   }

   @Override
   public void zoomOutDomain(double x, double y) {
      super.zoomOutDomain(x, y);
      updateSliders();
      getChart().getXYPlot().clearAnnotations();
   }

   @Override
   public void zoomOutRange(double x, double y) {
      super.zoomOutRange(x, y);
      updateSliders();
      getChart().getXYPlot().clearAnnotations();
   }

   public final void updateSliders() {
      getChart().getXYPlot().clearAnnotations();
      final ValueAxis domainAxis = getChart().getXYPlot().getDomainAxis();
      final ValueAxis rangeAxis = getChart().getXYPlot().getRangeAxis();
      final double SLIDER_CONST = 0.001;

      if (horizontalMaximum - domainAxis.getRange().getUpperBound() >
SLIDER_CONST) {
         horizontalSlider.setVisible(true);
         horizontalSlider.setEnabled(true);
```

```
                horizontalRatio = horizontalSlider.getMaximum() /
        horizontalMaximum;
                horizontalSlider.setThumb(
                      (int) Math.round((domainAxis.getUpperBound() -
        domainAxis.getLowerBound()) * horizontalRatio));
                horizontalSlider.setSelection((int)
        Math.round(domainAxis.getLowerBound() * horizontalRatio));
             } else {
                horizontalSlider.setVisible(false);
                horizontalSlider.setEnabled(false);
             }

             if (verticalMaximum - rangeAxis.getRange().getUpperBound() >
        SLIDER_CONST) {
                verticalSlider.setVisible(true);
                verticalSlider.setEnabled(true);
                verticalRatio = verticalSlider.getMaximum() / verticalMaximum;
                verticalSlider.setThumb(
                      (int) Math.round((rangeAxis.getUpperBound() -
        rangeAxis.getLowerBound()) * verticalRatio));
                verticalSlider
                      .setSelection((int) Math.round((verticalMaximum -
        rangeAxis.getUpperBound()) * verticalRatio));
             } else {
                verticalSlider.setVisible(false);
                verticalSlider.setEnabled(false);
             }
          }

          @Override
          public void zoom(Rectangle selection) {
             super.zoom(selection);
             horizontalSlider.setVisible(true);
             horizontalSlider.setEnabled(true);
             if (this.isRangeZoomable()) {
                verticalSlider.setVisible(true);
                verticalSlider.setEnabled(true);
             }
             updateSliders();
          }

          @Override
          public void restoreAutoBounds() {
             super.restoreAutoBounds();
             horizontalSlider.setEnabled(false);
             horizontalSlider.setVisible(false);
             verticalSlider.setEnabled(false);
             verticalSlider.setVisible(false);
          }
       }
```

2. PlotView

```
package ru.bmstu.rk9.rao.ui.plot;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Font;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
```

```
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.DisposeEvent;
import org.eclipse.swt.events.DisposeListener;
import org.eclipse.swt.layout.FormAttachment;
import org.eclipse.swt.layout.FormData;
import org.eclipse.swt.layout.FormLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Menu;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Slider;
import org.eclipse.ui.IWorkbenchPage;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.PlatformUI;
import org.eclipse.ui.part.ViewPart;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.axis.SymbolAxis;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.plot.XYPlot;
import org.jfree.data.xy.XYDataItem;
import org.jfree.data.xy.XYDataset;
import org.jfree.data.xy.XYSeries;
import org.jfree.data.xy.XYSeriesCollection;

import ru.bmstu.rk9.rao.lib.database.CollectedDataNode;
import ru.bmstu.rk9.rao.lib.database.CollectedDataNode.Index;
import ru.bmstu.rk9.rao.lib.database.CollectedDataNode.PatternIndex;
import ru.bmstu.rk9.rao.lib.modeldata.ModelStructureConstants;
import ru.bmstu.rk9.rao.lib.notification.Subscriber;
import ru.bmstu.rk9.rao.lib.simulator.CurrentSimulator;
import ru.bmstu.rk9.rao.lib.simulator.CurrentSimulator.ExecutionState;
import ru.bmstu.rk9.rao.lib.simulator.SimulatorSubscriberManager;
import
ru.bmstu.rk9.rao.lib.simulator.SimulatorSubscriberManager.SimulatorSubscriberInf
o;
import ru.bmstu.rk9.rao.ui.notification.RealTimeSubscriberManager;
import ru.bmstu.rk9.rao.ui.plot.PlotDataParser.DataParserResult;
import ru.bmstu.rk9.rao.ui.plot.PlotDataParser.PlotItem;
import
ru.bmstu.rk9.rao.ui.serialization.SerializedObjectsView.ConditionalMenuItem;

public class PlotView extends ViewPart {
   public static final String ID = "ru.bmstu.rk9.rao.ui.PlotView";
   private final static Map<CollectedDataNode, Integer> openedPlotMap = new
HashMap<CollectedDataNode, Integer>();
   private static int secondaryID = 0;
   XYSeries newSeries2 = null;

   public static Map<CollectedDataNode, Integer> getOpenedPlotMap() {
      return openedPlotMap;
   }

   public static void addToOpenedPlotMap(final CollectedDataNode node, final int
secondaryID) {
      openedPlotMap.put(node, secondaryID);
   }

   private PlotFrame plotFrame;
   private CollectedDataNode partNode;

   public PlotFrame getFrame() {
      return plotFrame;
   }
```

```
    @Override
    public void createPartControl(Composite parent) {
       plotFrame = new PlotFrame(parent, SWT.NONE);
       FormLayout layout = new FormLayout();
       parent.setLayout(layout);

       Slider verticalSlider = new Slider(parent, SWT.VERTICAL);
       FormData verticalSliderLayoutData = new FormData();

       Slider horizontalSlider = new Slider(parent, SWT.HORIZONTAL);
       FormData horizontalSliderLayoutData = new FormData();

       horizontalSliderLayoutData.left = new FormAttachment(0, 0);
       horizontalSliderLayoutData.bottom = new FormAttachment(100, 0);
       horizontalSliderLayoutData.right = new FormAttachment(verticalSlider);
       horizontalSlider.setLayoutData(horizontalSliderLayoutData);

       verticalSliderLayoutData.right = new FormAttachment(100, 0);
       verticalSliderLayoutData.bottom = new FormAttachment(horizontalSlider);
       verticalSliderLayoutData.top = new FormAttachment(0, 0);
       verticalSlider.setLayoutData(verticalSliderLayoutData);

       FormData graphLayoutData = new FormData();
       graphLayoutData.left = new FormAttachment(0, 0);
       graphLayoutData.right = new FormAttachment(verticalSlider);
       graphLayoutData.top = new FormAttachment(0, 0);
       graphLayoutData.bottom = new FormAttachment(horizontalSlider);
       plotFrame.setLayoutData(graphLayoutData);

       parent.layout(true);
       horizontalSlider.setEnabled(false);
       horizontalSlider.setVisible(false);
       verticalSlider.setEnabled(false);
       verticalSlider.setVisible(false);
       plotFrame.setSliders(horizontalSlider, verticalSlider);

       plotFrame.addDisposeListener(new DisposeListener() {
          @Override
          public void widgetDisposed(DisposeEvent event) {
             if (!openedPlotMap.isEmpty() && openedPlotMap.containsKey(partNode))
{
                openedPlotMap.remove(partNode);
             }
             deinitializeSubscribers();
          }
       });
    }

    private final void initialize(CollectedDataNode node) {
       partNode = node;
       setPartName(partNode.getName());
       plotDataParser = new PlotDataParser(partNode);

       XYSeriesCollection dataset = new XYSeriesCollection();
       XYSeries series = new XYSeries(partNode.getName());
       dataset.addSeries(series);
       plotXY(dataset);
       Shell activeShell = getSite().getWorkbenchWindow().getShell();
       initializeSubscribers();
    }

    private final void initializeSubscribers() {
       simulatorSubscriberManager.initialize(
             Arrays.asList(new SimulatorSubscriberInfo(commonSubcriber,
```

```
ExecutionState.EXECUTION_STARTED),
                  new SimulatorSubscriberInfo(endSubscriber,
ExecutionState.EXECUTION_COMPLETED)));

realTimeSubscriberManager.initialize(Arrays.asList(realTimeUpdateRunnable));
   }

   private final void deinitializeSubscribers() {
      simulatorSubscriberManager.deinitialize();
      realTimeSubscriberManager.deinitialize();
   }

   private final SimulatorSubscriberManager simulatorSubscriberManager = new
SimulatorSubscriberManager();
   private final RealTimeSubscriberManager realTimeSubscriberManager = new
RealTimeSubscriberManager();
   private PlotDataParser plotDataParser;

   private final boolean readyForInput() {
      return plotFrame != null && !plotFrame.isDisposed();
   }

   private class RealTimeUpdateRunnable implements Runnable {
      private boolean isLastEntry;

      RealTimeUpdateRunnable(boolean isLastEntry) {
         this.isLastEntry = isLastEntry;
      }

      private void changeLastValueFlag() {
         isLastEntry = true;
      }

      @Override
      public void run() {
         if (!readyForInput())
            return;

         final DataParserResult dataParserResult =
plotDataParser.parseEntries();
         if (dataParserResult.axisHelper.axisChanged) {
            XYPlot plot = (XYPlot) plotFrame.getChart().getPlot();
            SymbolAxis rangeAxis;
            String[] labels = new
String[dataParserResult.axisHelper.axisValues.size()];
            labels = dataParserResult.axisHelper.axisValues.toArray(labels);

            final String fontName =
plotFrame.getChart().getTitle().getFont().getName();

            final Font font = new Font(fontName, Font.PLAIN, 10);
            rangeAxis = new SymbolAxis("", labels);
            rangeAxis.setAutoRangeIncludesZero(true);
            plot.setRangeAxis(rangeAxis);
            rangeAxis.setTickLabelFont(font);
         }

         final List<PlotItem> items = dataParserResult.dataset;

         if (!items.isEmpty()) {
            final XYSeriesCollection newDataset = (XYSeriesCollection)
plotFrame.getChart().getXYPlot()
                  .getDataset();
            final XYSeries newSeries = newDataset.getSeries(0);
```

```
            List<PlotItem> startItems = items.subList(0, items.size() - 1);

            for (PlotItem item : startItems) {
               newSeries.add(item.x, item.y, false);
            }

            PlotItem lastItem = items.get(items.size() - 1);
            newSeries.add(lastItem.x, lastItem.y, true);

            plotFrame.setChartMaximum(newSeries.getMaxX(), newSeries.getMaxY());
            plotFrame.updateSliders();

         }

         if (isLastEntry) {

            final XYSeriesCollection newDataset = (XYSeriesCollection)
plotFrame.getChart().getXYPlot()
                  .getDataset();
            final XYSeries newSeries = newDataset.getSeries(0);
            if (newSeries2 == null) {
               newSeries2 = new XYSeries("Serrrrrrrrrrr");
               newSeries2.add(0, 0);
               newSeries2.add(Math.round(newSeries.getMaxX() / 2),
newSeries.getMaxY() / 2);
               newSeries2.add(newSeries.getMaxX(), newSeries.getMaxY());
               newDataset.addSeries(newSeries2);
            }

            if (newSeries.getItemCount() == 2) {
               @SuppressWarnings("unchecked")
               List<XYDataItem> seriesitems = newSeries.getItems();

               if (seriesitems.get(0).equals(seriesitems.get(1))) {
                  newSeries.add(CurrentSimulator.getTime(),
seriesitems.get(1).getY());
                  plotFrame.setChartMaximum(newSeries.getMaxX(),
newSeries.getMaxY());
                  plotFrame.updateSliders();
               }
            } else if (newSeries.getItemCount() == 1) {
               @SuppressWarnings("unchecked")
               List<XYDataItem> seriesitems = newSeries.getItems();
               newSeries.add(CurrentSimulator.getTime(),
seriesitems.get(0).getY());
               plotFrame.setChartMaximum(newSeries.getMaxX(),
newSeries.getMaxY());
               plotFrame.updateSliders();
            }
            System.out.println("End " + System.currentTimeMillis());
         }
      }
   };

   private final RealTimeUpdateRunnable realTimeUpdateRunnable = new
RealTimeUpdateRunnable(false);

   private final Subscriber commonSubcriber = new Subscriber() {
      @Override
      public void fireChange() {

PlatformUI.getWorkbench().getDisplay().asyncExec(realTimeUpdateRunnable);
      }
   };
```

```
   private final Subscriber endSubscriber = new Subscriber() {
      @Override
      public void fireChange() {
         realTimeUpdateRunnable.changeLastValueFlag();

PlatformUI.getWorkbench().getDisplay().asyncExec(realTimeUpdateRunnable);
      }
   };

   @Override
   public void setFocus() {
   }

   public void plotXY(final XYSeriesCollection dataset) {
      final JFreeChart chart = createChart(dataset);
      plotFrame.setChart(chart);
      plotFrame.setDomainZoomable(false);
      plotFrame.setRangeZoomable(false);
   }

   private JFreeChart createChart(final XYDataset dataset) {
      final JFreeChart chart = FilteringPlotFactory.createXYStepChart("",
"Time", "Value", dataset,
            PlotOrientation.VERTICAL, true, false, false);

      final XYPlot plot = chart.getXYPlot();

      Color white = new Color(0xFF, 0XFF, 0xFF);
      plot.setBackgroundPaint(white);
      Color grey = new Color(0x99, 0x99, 0x99);
      plot.setDomainGridlinePaint(grey);
      plot.setRangeGridlinePaint(grey);
      plot.getRenderer().setSeriesStroke(0, new BasicStroke((float) 2.5));

      NumberAxis rangeAxis = new NumberAxis();
      rangeAxis.setAutoRangeIncludesZero(true);
      plot.setRangeAxis(rangeAxis);

      final NumberAxis domainAxis = new NumberAxis();
      domainAxis.setAutoRangeIncludesZero(true);
      plot.setDomainAxis(domainAxis);

      double horizontalMaximum = domainAxis.getRange().getLength();
      double verticalMaximum = rangeAxis.getRange().getLength();
      plotFrame.setChartMaximum(horizontalMaximum, verticalMaximum);

      return chart;
   }

   static private class PlotMenuItem extends ConditionalMenuItem {
      public PlotMenuItem(Menu parent) {
         super(parent, "Plot");
      }

      @Override
      public boolean isEnabled(CollectedDataNode node) {
         Index index = node.getIndex();
         if (index == null)
            return false;

         switch (index.getType()) {
         case RESOURCE_PARAMETER:
            return true;
```

```
            case RESULT:
               return true;
            case PATTERN:
               PatternIndex patternIndex = (PatternIndex) index;
               int patternNumber = patternIndex.getNumber();
               String patternType =
CurrentSimulator.getStaticModelData().getPatternType(patternNumber);
               return patternType.equals(ModelStructureConstants.OPERATION);
            default:
               return false;
            }
         }

         @Override
         public void show(CollectedDataNode node) {
            try {
               if (PlotView.getOpenedPlotMap().containsKey(node)) {

PlatformUI.getWorkbench().getActiveWorkbenchWindow().getActivePage().showView(Pl
otView.ID,
                        String.valueOf(PlotView.getOpenedPlotMap().get(node)),
IWorkbenchPage.VIEW_ACTIVATE);
               } else {
                  System.out.println("Start " + System.currentTimeMillis());
                  PlotView newView = (PlotView)
PlatformUI.getWorkbench().getActiveWorkbenchWindow().getActivePage()
                        .showView(PlotView.ID, String.valueOf(secondaryID),
IWorkbenchPage.VIEW_ACTIVATE);
                  PlotView.addToOpenedPlotMap(node, secondaryID);
                  secondaryID++;

                  newView.initialize(node);
               }
            } catch (PartInitException e) {
               e.printStackTrace();
            }
         }
      }

      static public ConditionalMenuItem createConditionalMenuItem(Menu parent) {
         return new PlotMenuItem(parent);
      }
   }
```

3. XYPlotWithFiltering

```
package ru.bmstu.rk9.rao.ui.plot;

import java.awt.Graphics2D;
import java.awt.geom.Rectangle2D;
import java.util.LinkedHashMap;
import java.util.Map;

import org.jfree.chart.axis.ValueAxis;
import org.jfree.chart.plot.CrosshairState;
import org.jfree.chart.plot.PlotRenderingInfo;
import org.jfree.chart.plot.SeriesRenderingOrder;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.RendererUtilities;
import org.jfree.chart.renderer.xy.XYItemRenderer;
import org.jfree.chart.renderer.xy.XYItemRendererState;
import org.jfree.data.general.DatasetUtilities;
import org.jfree.data.xy.XYDataset;
```

```
public class XYPlotWithFiltering extends XYPlot {
   private static final long serialVersionUID =
7074889297801956906L;

   public XYPlotWithFiltering() {
      super();
   }

   public XYPlotWithFiltering(XYDataset dataset, ValueAxis
domainAxis, ValueAxis rangeAxis, XYItemRenderer renderer) {
      super(dataset, domainAxis, rangeAxis, renderer);
   }

   private ProxyDataSet proxy = new ProxyDataSet();

   public ProxyDataSet getProxyDataSet() {
      return proxy;
   }

   public class ProxyDataSet {
      final LinkedHashMap<Integer, Integer> filteredMap = new
LinkedHashMap<>();

      public Map<Integer, Integer> getFilteredMap() {
         return filteredMap;
      }

      double increment = 1.0;
      int previousFirst;
      int previousLast;
      int previousPartCount = 0;
      int currentPartCount = 1;

      void setPlotWidth(int width) {
         int newPartCount = width / 2;
         if (((double) Math.max(currentPartCount, newPartCount))
               / ((double) Math.min(currentPartCount, newPartCount))
> 2) {
            this.currentPartCount = newPartCount;
         }
      }

      public Map<Integer, Integer> workAsProxy(XYDataset items, int
series, int first, int last) {
         if (previousFirst == first && previousLast == last &&
previousPartCount == currentPartCount) {
            return filteredMap;
         }
         filteredMap.clear();
         if (last - first < currentPartCount) {
            return filteredMap;
         }
         int prevFiltered = first;
         double groupSize = (last - first) / (double)
currentPartCount;

         previousFirst = first;
         previousLast = last;
         previousPartCount = currentPartCount;

         for (int i = 0; i < currentPartCount; i++) {
            int innerBegin = (int) (first + (groupSize * i));
            int innerEnd = (int) (first + (groupSize * (i + 1)));
```

```
                double sumVal = 0.0;
                for (int inner = innerBegin; inner < innerEnd; inner++)
{
                    sumVal += items.getYValue(series, inner);
                }
                sumVal = sumVal / (innerEnd - innerBegin);
                double nearest = Double.MAX_VALUE;
                int nearestIndex = innerBegin;

                for (int inner = innerBegin; inner < innerEnd; inner++)
{
                    double nearestCandidate =
Math.abs(items.getYValue(series, inner) - sumVal);
                    if (nearestCandidate < nearest) {
                        nearestIndex = inner;
                        nearest = nearestCandidate;
                    }
                }
                filteredMap.put(nearestIndex, prevFiltered);
                prevFiltered = nearestIndex;
            }
            return filteredMap;
        }

        public void reset() {
            filteredMap.clear();
            previousFirst = -1;
            previousLast = -1;
        }
    }

    @Override
    public boolean render(Graphics2D g2, Rectangle2D dataArea, int
index, PlotRenderingInfo info,
            CrosshairState crosshairState) {

        boolean foundData = false;
        XYDataset dataset = getDataset(index);
        if (!DatasetUtilities.isEmptyOrNull(dataset)) {
            foundData = true;
            ValueAxis xAxis = getDomainAxisForDataset(index);
            ValueAxis yAxis = getRangeAxisForDataset(index);
            if (xAxis == null || yAxis == null) {
                return foundData;
            }
            XYItemRenderer renderer = getRenderer(index);
            if (renderer == null) {
                renderer = getRenderer();
                if (renderer == null) {
                    return foundData;
                }
            }
            proxy.setPlotWidth(dataArea.getBounds().width);
            XYItemRendererState state = renderer.initialise(g2,
dataArea, this, dataset, info);
            int passCount = renderer.getPassCount();
            SeriesRenderingOrder seriesOrder =
getSeriesRenderingOrder();

            if (seriesOrder == SeriesRenderingOrder.REVERSE) {
                for (int pass = 0; pass < passCount; pass++) {
                    int seriesCount = dataset.getSeriesCount();
                    for (int series = seriesCount - 1; series >= 0;
series--) {
```

```
                    int firstItem = 0;
                    int lastItem = dataset.getItemCount(series) - 1;
                    if (lastItem == -1) {
                        continue;
                    }
                    if (state.getProcessVisibleItemsOnly()) {
                        int[] itemBounds =
RendererUtilities.findLiveItems(dataset, series,
xAxis.getLowerBound(),
                                xAxis.getUpperBound());
                        firstItem = Math.max(itemBounds[0] - 1, 0);
                        lastItem = Math.min(itemBounds[1] + 1,
lastItem);
                    }
                    state.startSeriesPass(dataset, series, firstItem,
lastItem, pass, passCount);
                    Map<Integer, Integer> filteredMap =
proxy.workAsProxy(dataset, series, firstItem, lastItem);

                    if (renderer instanceof XYFilteringStepRenderer) {
                        ((XYFilteringStepRenderer)
renderer).setFilteredMap(filteredMap);
                    }
                    if (filteredMap.size() == 0 || !(renderer
instanceof XYFilteringStepRenderer)) {
                        for (int item = firstItem; item <= lastItem;
item++) {
                            renderer.drawItem(g2, state, dataArea, info,
this, xAxis, yAxis, dataset, series, item,
                                    crosshairState, pass);
                        }
                    } else {
                        for (Integer item : filteredMap.keySet()) {
                            renderer.drawItem(g2, state, dataArea, info,
this, xAxis, yAxis, dataset, series, item,
                                    crosshairState, pass);
                        }
                    }
                    state.endSeriesPass(dataset, series, firstItem,
lastItem, pass, passCount);
                }
            }
        } else {
            for (int pass = 0; pass < passCount; pass++) {
                int seriesCount = dataset.getSeriesCount();
                for (int series = 0; series < seriesCount; series++)
{
                    int firstItem = 0;
                    int lastItem = dataset.getItemCount(series) - 1;
                    if (lastItem == -1) {
                        continue;
                    }
                    if (state.getProcessVisibleItemsOnly()) {
                        int[] itemBounds =
RendererUtilities.findLiveItems(dataset, series,
xAxis.getLowerBound(),
                                xAxis.getUpperBound());
                        firstItem = Math.max(itemBounds[0] - 1, 0);
                        lastItem = Math.min(itemBounds[1] + 1,
lastItem);
                    }
                    state.startSeriesPass(dataset, series, firstItem,
lastItem, pass, passCount);
```

```
                    for (int item = firstItem; item <= lastItem;
   item++) {
                        renderer.drawItem(g2, state, dataArea, info,
   this, xAxis, yAxis, dataset, series, item,
                                crosshairState, pass);
                    }
                    state.endSeriesPass(dataset, series, firstItem,
   lastItem, pass, passCount);
                }
             }
          }
       }
       return foundData;
    }
   }
```

4. XYFilteringStepRenderer

```
package ru.bmstu.rk9.rao.ui.plot;

import java.awt.Graphics2D;
import java.awt.Paint;
import java.awt.Stroke;
import java.awt.geom.Line2D;
import java.awt.geom.Rectangle2D;
import java.util.LinkedHashMap;
import java.util.Map;

import org.jfree.chart.axis.ValueAxis;
import org.jfree.chart.entity.EntityCollection;
import org.jfree.chart.labels.XYToolTipGenerator;
import org.jfree.chart.plot.CrosshairState;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.plot.PlotRenderingInfo;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.xy.XYItemRendererState;
import org.jfree.chart.renderer.xy.XYStepRenderer;
import org.jfree.chart.urls.XYURLGenerator;
import org.jfree.data.xy.XYDataset;
import org.jfree.ui.RectangleEdge;

public class XYFilteringStepRenderer extends XYStepRenderer {
   public XYFilteringStepRenderer(XYToolTipGenerator toolTipGenerator,
XYURLGenerator urlGenerator) {
      super(toolTipGenerator, urlGenerator);
   }

   private Map<Integer, Integer> filteredLinesMap = new
LinkedHashMap<>();

   @Override
   public void drawItem(Graphics2D g2, XYItemRendererState state,
Rectangle2D dataArea, PlotRenderingInfo info,
         XYPlot plot, ValueAxis domainAxis, ValueAxis rangeAxis,
XYDataset dataset, int series, int item,
         CrosshairState crosshairState, int pass) {

      if (!getItemVisible(series, item)) {
         return;
      }

      PlotOrientation orientation = plot.getOrientation();
```

```
      Paint seriesPaint = getItemPaint(series, item);
      Stroke seriesStroke = getItemStroke(series, item);
      g2.setPaint(seriesPaint);
      g2.setStroke(seriesStroke);

      double x1 = dataset.getXValue(series, item);
      double y1 = dataset.getYValue(series, item);

      RectangleEdge xAxisLocation = plot.getDomainAxisEdge();
      RectangleEdge yAxisLocation = plot.getRangeAxisEdge();
      double transX1 = domainAxis.valueToJava2D(x1, dataArea,
xAxisLocation);
      double transY1 = (Double.isNaN(y1) ? Double.NaN :
rangeAxis.valueToJava2D(y1, dataArea, yAxisLocation));

      if (pass == 0 && item > 0) {
         Integer previousPoint = filteredLinesMap.size() > 0 ?
filteredLinesMap.get(item) : null;
         if (previousPoint == null) {
            previousPoint = item - 1;
         }
         double x0 = dataset.getXValue(series, previousPoint);
         double y0 = dataset.getYValue(series, previousPoint);
         double transX0 = domainAxis.valueToJava2D(x0, dataArea,
xAxisLocation);
         double transY0 = (Double.isNaN(y0) ? Double.NaN :
rangeAxis.valueToJava2D(y0, dataArea, yAxisLocation));

         if (orientation == PlotOrientation.HORIZONTAL) {
            if (transY0 == transY1) {
               drawLine(g2, state.workingLine, transY0, transX0,
transY1, transX1);
            } else {
               double transXs = transX0 + (getStepPoint() * (transX1 -
transX0));
               drawLine(g2, state.workingLine, transY0, transX0,
transY0, transXs);
               drawLine(g2, state.workingLine, transY0, transXs,
transY1, transXs);
               drawLine(g2, state.workingLine, transY1, transXs,
transY1, transX1);
            }
         } else if (orientation == PlotOrientation.VERTICAL) {
            if (transY0 == transY1) {
               drawLine(g2, state.workingLine, transX0, transY0,
transX1, transY1);
            } else {
               double transXs = transX0 + (getStepPoint() * (transX1 -
transX0));
               drawLine(g2, state.workingLine, transX0, transY0,
transXs, transY0);
               drawLine(g2, state.workingLine, transXs, transY0,
transXs, transY1);
               drawLine(g2, state.workingLine, transXs, transY1,
transX1, transY1);
            }
         }
         int domainAxisIndex = plot.getDomainAxisIndex(domainAxis);
         int rangeAxisIndex = plot.getRangeAxisIndex(rangeAxis);
         updateCrosshairValues(crosshairState, x1, y1, domainAxisIndex,
rangeAxisIndex, transX1, transY1,
               orientation);
```

```
                EntityCollection entities = state.getEntityCollection();
                if (entities != null) {
                   addEntity(entities, null, dataset, series, item, transX1,
        transY1);
                }
             }
             if (pass == 1) {
                if (isItemLabelVisible(series, item)) {
                   double xx = transX1;
                   double yy = transY1;
                   if (orientation == PlotOrientation.HORIZONTAL) {
                      xx = transY1;
                      yy = transX1;
                   }
                   drawItemLabel(g2, orientation, dataset, series, item, xx,
        yy, (y1 < 0.0));
                }
             }
          }

          private void drawLine(Graphics2D g2, Line2D line, double x0, double
        y0, double x1, double y1) {
             if (Double.isNaN(x0) || Double.isNaN(x1) || Double.isNaN(y0) ||
        Double.isNaN(y1)) {
                return;
             }
             line.setLine(x0, y0, x1, y1);
             g2.draw(line);
          }

          public Map<Integer, Integer> getFilteredMap() {
             return filteredLinesMap;
          }

          public void setFilteredMap(Map<Integer, Integer> filteredMap) {
             this.filteredLinesMap = filteredMap;
          }
        }
```

5. FilteringPlotFactory

```
 package ru.bmstu.rk9.rao.ui.plot;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.DateAxis;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.labels.StandardXYToolTipGenerator;
import org.jfree.chart.labels.XYToolTipGenerator;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.renderer.xy.XYItemRenderer;
import org.jfree.chart.urls.StandardXYURLGenerator;
import org.jfree.chart.urls.XYURLGenerator;
import org.jfree.chart.util.ParamChecks;
import org.jfree.data.xy.XYDataset;

public class FilteringPlotFactory {

   public static JFreeChart createXYStepChart(String title, String xAxisLabel,
String yAxisLabel, XYDataset dataset,
         PlotOrientation orientation, boolean legend, boolean tooltips, boolean
urls) {
```

```
        ParamChecks.nullNotPermitted(orientation, "orientation");
        DateAxis xAxis = new DateAxis(xAxisLabel);
        NumberAxis yAxis = new NumberAxis(yAxisLabel);
        yAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits());
        XYToolTipGenerator toolTipGenerator = null;

        if (tooltips) {
            toolTipGenerator = new StandardXYToolTipGenerator();
        }

        XYURLGenerator urlGenerator = null;

        if (urls) {
            urlGenerator = new StandardXYURLGenerator();
        }

        XYItemRenderer renderer = new XYFilteringStepRenderer(toolTipGenerator,
urlGenerator);
        XYPlotWithFiltering plot = new XYPlotWithFiltering(dataset, xAxis, yAxis,
null);

        plot.setRenderer(renderer);
        plot.setOrientation(orientation);
        plot.setDomainCrosshairVisible(false);
        plot.setRangeCrosshairVisible(false);
        JFreeChart chart = new JFreeChart(title, JFreeChart.DEFAULT_TITLE_FONT,
plot, legend);
        ChartFactory.getChartTheme().apply(chart);

        return chart;
    }
}
```

ПРИЛОЖЕНИЕ В

Код тестовой модели на языке РДО

```
enum Тип_клиента {Тип1, Тип2}

enum Состояние_клиента {Пришел, Начал_стрижку}

enum Состояние_парикмахера {Свободен, Занят}

type Парикмахерские {
	int количество_в_очереди;
}

type Клиенты {
	Тип_клиента тип;
	Состояние_клиента состояние;
}

type Парикмахеры {
	Состояние_парикмахера состояние_парикмахера = Состояние_парикмахера.Свободен;
	int количество_обслуженных;
	int длительность_min;
	int длительность_max;
	Тип_клиента тип_клиента;
}

resource парикмахерская = Парикмахерские.create(0);

resource парикмахер_1 = Парикмахеры.create(Состояние_парикмахера.Свободен, 0, 20, 40, Тип_клиента.Тип1);

resource парикмахер_2 = Парикмахеры.create(Состояние_парикмахера.Свободен, 0, 25, 70, Тип_клиента.Тип2);

resource парикмахер_3 = Парикмахеры.create(Состояние_парикмахера.Свободен, 0, 30, 60, Тип_клиента.Тип2);

event Событие_прихода_клиента() {
	Клиенты.create(случайный_тип_клиента.next(), Состояние_клиента.Пришел);
	Событие_прихода_клиента.plan(currentTime + интервал_прихода.next());
	парикмахерская.количество_в_очереди = парикмахерская.количество_в_очереди + 1;
}
```

```
operation Образец_обслуживания_клиента() {
        relevant _Парикмахерская = парикмахерская.onlyif[количество_в_очереди > 0];
        relevant _Клиент = Клиенты.all.filter[состояние == Состояние_клиента.Пришел].any;
        relevant _Парикмахер = Парикмахеры.all.filter [
                состояние_парикмахера == Состояние_парикмахера.Свободен && тип_клиента == _Клиент.тип
        ].minBySafe[количество_обслуженных];
        def duration() {
                return длительность_обслуживания.next(_Парикмахер.длительность_min,
_Парикмахер.длительность_max);
        }
        def begin() {
                _Парикмахерская.количество_в_очереди = _Парикмахерская.количество_в_очереди - 1;
                _Клиент.состояние = Состояние_клиента.Начал_стрижку;
                _Парикмахер.состояние_парикмахера = Состояние_парикмахера.Занят;
        }
        def end() {
                _Парикмахер.состояние_парикмахера = Состояние_парикмахера.Свободен;
                _Парикмахер.количество_обслуженных = _Парикмахер.количество_обслуженных + 1;
                _Клиент.erase();
        }
}
logic Model {
        activity обслуживание_клиента = new Activity(Образец_обслуживания_клиента.create());
}
sequence интервал_прихода = new Exponential(123456789, 1 / 30.0);
sequence длительность_обслуживания = new Uniform(123456789);
sequence случайный_тип_клиента = new DiscreteHistogram<Тип_клиента>(
        123456789,
        #[
                Тип_клиента.Tun1 -> 1.0,
                Тип_клиента.Tun2 -> 5.0
        ]
```

```
);

def init() {

        Событие_прихода_клиента.plan(currentTime + интервал_прихода.next());

}

def terminateCondition() {

        return currentTime >= 3000 * 12 * 60;

}

result занятость_парикмахера_1 = new Result([парикмахер_1.состояние_парикмахера]);

result занятость_парикмахера_2 = new Result([парикмахер_2.состояние_парикмахера]);

result занятость_парикмахера_3 = new Result([парикмахер_3.состояние_парикмахера]);

result обслужено_парикмахером_1 = new Result([парикмахер_1.количество_обслуженных], new LastValueStatistics());

result обслужено_парикмахером_2 = new Result([парикмахер_2.количество_обслуженных], new LastValueStatistics());

result обслужено_парикмахером_3 = new Result([парикмахер_3.количество_обслуженных], new LastValueStatistics());
```