Оглавление

[1. Введение 3](#_Toc472892737)

[2. Предпроектное исследование 3](#_Toc472892738)

[2.1. Постановка задачи 3](#_Toc472892739)

[3. Концептуальный этап проектирования 3](#_Toc472892740)

[3.1. Выбор общесистемной методологии проектирования 3](#_Toc472892741)

[3.2. Выделение системы из среды 3](#_Toc472892742)

[4. Формирование технического задания 3](#_Toc472892743)

[4.1. Введение 3](#_Toc472892744)

[4.2. Общие сведения 3](#_Toc472892745)

[4.3. Назначение разработки 3](#_Toc472892746)

[4.4. Требования к программе или программному изделию 3](#_Toc472892747)

[4.5. Стадии и этапы разработки 3](#_Toc472892748)

[4.6. Порядок контроля и приемки 3](#_Toc472892749)

[5. Технический этап проектирования 3](#_Toc472892750)

[5.1. Разработка грамматики собираемых показателей 3](#_Toc472892751)

[5.2. Разработка абстрактного класса AbstractResult 3](#_Toc472892752)

[5.3. Разработка абстрактного класса AbstractDataSource 3](#_Toc472892753)

[6. Рабочий этап проектирование 3](#_Toc472892754)

[6.1. Диаграмма классов 3](#_Toc472892755)

[7. Апробирование разработанной системы для модельных условий 23](#_Toc472892756)

[8. Заключение 24](#_Toc472892757)

[Список литературы 25](#_Toc472892758)

[Приложение 1. Листинг модели парикмахерской 26](#_Toc472892759)

[Приложение 2. Листинг новой грамматики результатов 28](#_Toc472892760)

# 1. Введение

Имитационное моделирование (ИМ) 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, транспортные системы (морские порты, аэропорты) и комплексы перекачки нефти и газа, программное обеспечение сложных систем управления и вычислительные сети, а также 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

1. Синтезировать и исследовать стратегии управления
2. Прогнозировать и планировать функционирование системы в будущем
3. Обучать и тренировать управленческий персонал и т.д.

Разработка интеллектуальной среды имитационного моделирования РДО выполнена в Московском Государственном Техническом Университете (МГТУ им.Н.Э. Баумана) на кафедре "Компьютерные системы автоматизации производства". Причинами ее проведения и создания РДО явились требования универсальности ИМ относительно классов моделируемых систем и процессов, легкости модификации моделей, моделирования сложных систем управления совместно с управляемым объектом (включая использование ИМ в управлении в реальном масштабе времени) и ряд других, сформировавшихся у разработчиков при выполнении работ, связанных с системным анализом и организационным управлением сложными системами различной природы.

В настоящий момент РДО обретает второе рождение под именем Rao X. Благодаря переносу исходного кода с C++ на Java с использованием библиотеки Xtext, достигнут качественный рост производительности и удобства использования среды. В прошлом семестре мной была разработана подсистема собираемых показателей. После введения в ученый план лабораторных работ в Rao X, было отмечено удобство новой подсистемы и было решено провести дальнейшее ее развитие.

# 2. Предпроектное исследование

Конечная цель моделирования – анализ результатов. Для этой цели в РДО существует подсистема собираемых показателей, которая позволяет пользователю описать набор данных для подсчета системой.

Результаты моделирования (собираемые показатели) описываются с помощью ключевого слова result.

**result** *имя\_результата* = **new** *Имя\_счетчика*(*параметры\_счетчика*);

Рассмотрим сбор показателей на примере модели парикмахерской, текст которой приведен в Приложении 1

Значение в конце моделирования

Простейший счетчик. Вычисляет значение выражения в конце моделирования. Общий вид:

**result** *имя\_результата* = **new** Value([*арифметическое\_выражение*]);

Результаты из примера:

Сбор статистики по целочисленным и вещественным значениям

Счетчик наблюдает за значениями арифметического выражения. В конце моделирования выдает статистику наблюдений.

resultType *Имя\_счетчика*<**Integer**>(*параметры\_счетчика*) {

set **evaluate**() {

**return** *арифметическое\_выражение*;

}

}

**result** *имя\_результата* = **new** *Имя\_счетчика*(параметры\_счетчика);

Наблюдение за перечислимым типом

Счетчик наблюдает за значениями перечислимого типа. В конце моделирования выдает статистику наблюдений за каждым значением.

resultType *Имя\_счетчика*<*Имя\_перечислимого\_типа*>(*параметры\_счетчика*) {

set **evaluate**() {

**return** *выражение\_возвращающее\_значение\_перечислимого\_типа*;

}

}

**result** *имя\_результата* = **new** *Имя\_счетчика*(параметры\_счетчика);

Кроме того счетчики имеют метод *condition()*, в котором описывается условие его срабатывания, а сам результат имеет модификатор *ResultMode.Manual* для ручного сбора показателей с помощью метода *update()*

Текущая версия была разработана, чтобы разрешить следующие недостатки:

* Отсутствие ручного подсчета статистики
* Ограничение доступных типов данных
* Отсутствие возможности выбора взвешенной по количеству статистики
* Не следует единому стилю
* Перегружена ключевыми словами
* Имеет большое количество искусственных ограничений
* Недостаточно гибкая

За счет возможности выбора различных типов статистики для одного источника данных, возможность описания и использования пользовательских типов статистики была достигнута необходимая гибкость, однако это привнесло сложность и некую перегруженность лексических конструкций.

Система следует применяемому java-подобному стилю, но все еще изобилует ключевыми словами.

В ходе эксплуатации новой системы появилась необходимость в отладке и визуализации собираемых показателей. На текущий момент в Rao X доступны анимация, трассировка, логирование и построение графиков, как средства отладки, однако, анимация и логирование не применимы в полном смысле к системе собираемых показателей, а вот добавление возможности трассировки и построения графиков по собираемым показателям удобный и необходимый мезанизм, разработка которого и является одной из моих задач

## 2.1. Постановка задачи

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

В предпроектном исследовании была описана текущая система собираемых показателей и доступные типы результатов, а так же выявлены ее проблемы. Необходимо доработать подсистему собираемых показателей, так, чтобы она отвечала следующим требованиям:

* Лаконичность при сохраненной гибкости
* Наличие механизмов отладки и визуализации, а именно трассировки и возможности построения графиков
* Модернизация ручного сбора статистики, отвязав его от источника данных, что даст возможность собирать статистику по любым данным не описывая их в источнике данных

# 3. Концептуальный этап проектирования

## 3.1. Выбор общесистемной методологии проектирования

Задача, поставленная на этапе предпроектного обеспечения, может быть решена на основе следующих концепций:

* Модульность
* Объектная ориентированность

Модульность — это свойство системы, связанное с возможностью ее декомпозиции на ряд внутренне связанных между собой модулей. Применительно к конструированию технических систем модульность — принцип, согласно которому функционально связанные части группируются в законченные узлы — модули. В свою очередь модульность в программировании — принцип, согласно которому программное средство (ПС) разделяется на отдельные именованные сущности, называемые модулями. Модульность часто является средством упрощения задачи проектирования ПС и распределения процесса разработки ПС между группами разработчиков. При разбиении ПС на модули для каждого модуля указывается реализуемая им функциональность, а также связи с другими модулями.

Объектно-ориентированное программирование (ООП) — парадигма программирования, в которой основными концепциями являются понятия объектов и классов. 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.

Как итог, необходимо разработать абстрактные классы AbstractDataSource и AbstractResult, для определения интерфейса взаимодействия, описываемыми ими сущностей источника данных и непосредственно результатов с остальной системой и между собой.

Абстрактный класс в объектно-ориентированном программировании — базовый класс, который не предполагает создания экземпляров. Абстрактные классы реализуют на практике один из принципов ООП — полиморфизм. Абстрактный класс может содержать (и не содержать) абстрактные методы и свойства. Абстрактный метод не реализуется для класса, в котором описан, однако должен быть реализован для его неабстрактных потомков. Абстрактные классы представляют собой наиболее общие абстракции, то есть имеющие наибольший объём и наименьшее содержание

Интерфейс определяет границу взаимодействия между классами или компонентами, специфицируя определенную абстракцию, которую осуществляет реализующая сторона. В отличие от концепции интерфейсов во многих других областях, интерфейс в ООП является строго формализованным элементом объектно-ориентированного языка и в качестве семантической конструкции широко используется кодом программы.

## 3.2. Выделение системы из среды

На Рисунок 1. Диаграмма компонентов системы Rao X Рисунок 1 представлена упрощенная диаграмма пакетов системы Rao X, на которой указаны основные зависимости от сторонних библиотек. Более подробная схема вынесена на лист 2 графического материала.



Рисунок 1. Диаграмма компонентов системы Rao X

**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 теститрования

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

* ru.bmstu.rk9.rao
* ru.bmstu.rk9.rao.compiler
* ru.bmstu.rk9.rao.formatting2
* ru.bmstu.rk9.rao.jvmmodel
* ru.bmstu.rk9.rao.naming
* ru.bmstu.rk9.rao.typesystem
* ru.bmstu.rk9.rao.validation

Из них интересны пакеты

**ru.bmstu.rk9.rao**, в котором необходимо описать грамматику результатов

**ru.bmstu.rk9.rao.validation**, в котором необходимо описать правила проверки грамматики

**ru.bmstu.rk9.rao.jvmmodel,** в котором необходимо описать соответствие грамматики и реализации из **ru.bmstu.rk9.rao.lib**

**ru.bmstu.rk9.rao.lib** состоит из следующих пакетов

* ru.bmstu.rk9.rao.lib.animation
* ru.bmstu.rk9.rao.lib.database
* ru.bmstu.rk9.rao.lib.dpt
* ru.bmstu.rk9.rao.lib.event
* ru.bmstu.rk9.rao.lib.exception
* ru.bmstu.rk9.rao.lib.json
* ru.bmstu.rk9.rao.lib.modeldata
* ru.bmstu.rk9.rao.lib.naming
* ru.bmstu.rk9.rao.lib.notification
* ru.bmstu.rk9.rao.lib.pattern
* ru.bmstu.rk9.rao.lib.process
* ru.bmstu.rk9.rao.lib.resource
* ru.bmstu.rk9.rao.lib.result
* ru.bmstu.rk9.rao.lib.runtime
* ru.bmstu.rk9.rao.lib.sequence
* ru.bmstu.rk9.rao.lib.simulator
* ru.bmstu.rk9.rao.lib.thirdparty
* ru.bmstu.rk9.rao.lib.type

**ru.bmstu.rk9.rao.lib.result –** в этом пакете необходимо описать абстрактные классы **AbstractResult** и **AbstractDataSource,** о которых говорилось на этапе концептуального проектирования ранее, а так же другие классы, необходимые для работы подсистемы

**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 необходимо произвести изменения, позволяющие строить и отображать графики по собираемым показателям

# 4. Формирование технического задания

## 4.1. Введение

Программный комплекс Rao X предназначен для разработки и отладки имитационных моделей на языке РДО. Конечная цель любого моделирования – анализ результатов. Подсистема собираемых показателей позволяет получить необходимые от модели результаты.

## 4.2. Общие сведения

* Полное наименование темы разработки: «Подсистема собираемых показателей»
* Заказчик: Кафедра "Компьютерные системы автоматизации производства " МГТУ им. Н.Э.Баумана"
* Разработчик: студент кафедры "Компьютерные системы автоматизации производства" Чернов А.О.
* Основание для разработки: Задание на курсовой проект
* Плановые сроки начала работы: 1 сентября 2016г.
* Плановые сроки окончания работы по созданию системы: 25 января 2017г.

## 4.3. Назначение разработки

Функциональным назначением объекта разработки является предоставление гибкого и лаконичного инструмента для сбора необходимых показателей дискретной имитационной модели

## 4.4. Требования к программе или программному изделию

**Требования к функциональным характеристикам:**

Система должна реализовывать:

* Возможность выбора различных типов статистики для одного источника данных
* Сбор статистики
* Возможность построения графиков
* Возможность трассировки

**Требования к надежности:**

Основное требование к надежности направлено на поддержание в исправном и работоспособном состоянии системы Rao X.

**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

**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

**Требования к информационной и программной совместимости:**

* операционная система Windows Server 2003 и старше или Ubuntu 15.10 и старше[[1]](#footnote-1)
* наличие в операционной системе ПО Eclipse DSL Tools Mars 2 и новее

**Требования к маркировке и упаковке:**

Не предъявляются.

**Требования к транспортированию и хранению:**

Не предъявляются.

## 4.5. Стадии и этапы разработки

Разработка должна быть проведена в три стадии:

* техническое задание
* технический и рабочий проекты
* внедрение

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

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

* разработка системы
* апробирование системы

## 4.6. 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.

# 5. Технический этап проектирования

В рамках технического этапа проектирования была разработана требуемая для работы подсистемы грамматика, интерфейсы абстрактных классов AbstractResult и AbstractDataSource.

## 5.1. Разработка грамматики собираемых показателей



Рисунок 2. Синтаксическая диаграмма собираемых показателей

Рисунок 2 изображает синтаксическую диаграмму грамматики определения нового типа результатов и создания экземпляра этого класса.

Рассмотрим пример, соответствующий этой грамматике:

**result** занятость\_парикмахера = **Result**.**create**([парикмахерская.состояние\_парикмахера])

**result** длина\_очереди = **Result**.**create**([парикмахерская.количество\_в\_очереди])

**result** всего\_обслужено = **Result**.**create**([парикмахерская.количество\_обслуженных], **new** LastValueStatistics())

**result** пропускная\_способность = **Result**.**create**([парикмахерская.количество\_обслуженных / **currentTime** \* 60])

**result** длительность\_работы = **Result**.**create**([**currentTime** / 60], **new** LastValueStatistics())

Эти строчки создадут необходимые классы, реализующие абстрактный класс **AbstractDataSource**, а так же экземпляры классов, реализующие абстрактный класс **AbstractResult**, однако для пользователя это выглядит как вызов метода **create()** с нужными параметрами.

## 5.2. Разработка абстрактного класса AbstractResult



Рисунок 3. Класс Result

Рисунок 3 показывает, какой интерфейс должны реализовывать классы используемые для сбора показателей.

Класс Result – шаблонный, так называемый jeneric. С помощью специального синтаксиса в описании классов и методов можно указать параметры-типы, которые внутри описания могут использоваться в качестве типов полей, параметров и возвращаемых значений методов. Таким образом тип собираемых показателей определяется пользователем в момент создания типа результата, что обеспечивает необходимую гибкость

**getDate()** – метод возвращающий данные для отображения

**update(T,double) –** метод обновляющий статистику и результат

**statistics –** атрибутсоответственно назначает и возвращает используемую статистику – экземпляр класса, реализующий абстрактный класс **Statistics<T>**

Как видно из рисунка 4 общий родитель всех результатов имеет только статистику и интерфейс для отображения, а классы, реализующие этот абстрактный класс, могут иметь источник данных – класс реализующий абстрактный класс **AbstractDataSource**

## 5.3. Разработка абстрактного класса AbstractDataSource



Рисунок 4. Класс **AbstractDataSource**

Как и **AbstractResult,** класс **AbstractDataSource** – шаблонный**.**

Абстрактный метод **evaluate()** – функция, возвращающая интересующее нас значение

**condition() –** метод возвращающий булево выражение, определяющее условие по которому **evaluate()** возвращает значение

# 6. Рабочий этап проектирование

На рабочем этапе проектирования была реализована грамматика, разработанная на техническом этапе проектирования. Для это были внесены соответствующие правки в файл грамматики Rao.xtext, а так же в парсер RaoJvmModelInferrer.xtend и ResultCompiler.xtend

Так же внесены необходимые изменения в **ru.bmstu.rk9.rao.lib**

## 6.1. Диаграмма классов

Рисунок 5 изображает упрощенную диаграмму классов, разработанных для работы подсистемы собираемых показателей, реализованных в пакете **ru.bmstu.rk9.rao.lib**

**ResultMode –** Enum, в котором перечислены доступные режимы работы результатов. **AUTO –** автоматический, в этом режиме, статистика результатов обновляется по каждому изменению в системе. **MANUAL –** ручной, в этом режиме статистика результатов обновляется только по вызову метода **update()** в ручную.

**ResultManager –** класс, управляющий результатами. Экземпляр этого класса получает уведомления от системы о изменении состояния и вызывает метод обновления статистики у результатов, работающих в автономном режиме. Кроме того этот класс используется для отображения всех результатов.

**CategoricalStatistics, StorelessNumericStatistics, ValueStatistics WeightedStorelessNumericStatistics –** классы статистики, именно от выбора класса зависит собираемая статистика. Эти классы реализуют абстрактный класс **Statistics**

**CategoricalStatistics –** класс статистики для категорий, для каждого нового значения собирает статистику по общему времени этого значения и процентному соотношению нахождения в этом значении, минимальному/максимальному времени вхождения. Назначается по умолчанию для результатов типа **Boolean**, **String**, пользовательских типов данных и перечислений(**Enum**)

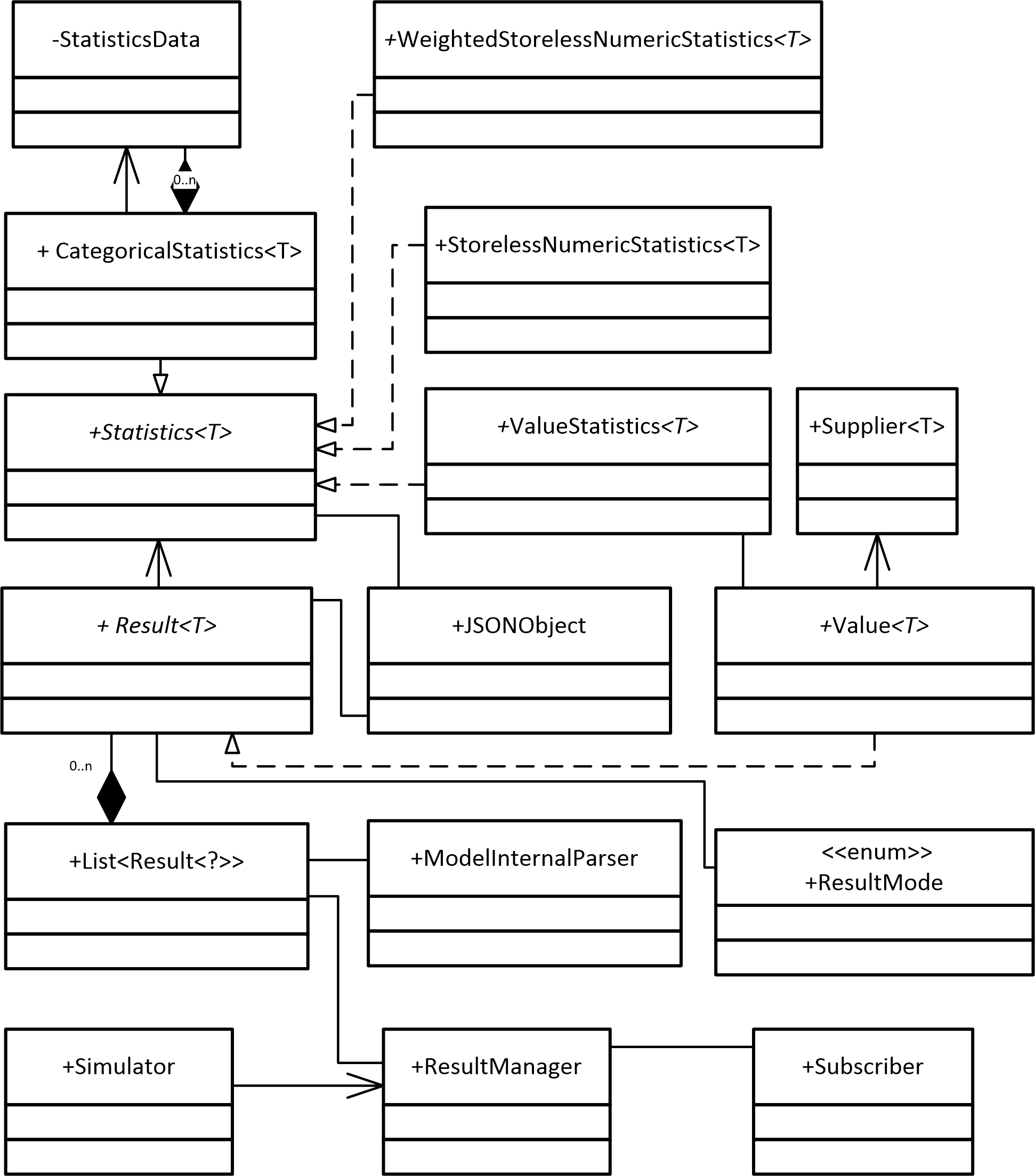


Рисунок 5

**StorelessNumericStatistics –** классстатистики для числовых значений, подсчитывает среднее значение, стандартное отклонение, коэффициент вариации и медиану. Статистика не взвешена по времени и обновляется по каждому вхождению, поэтому она не годится для сбора статистики по очереди в СМО. С помощью этой статистики можно подсчитать среднее значение времени обслуживания, средний интервал прихода клиентов и другие данные взвешенные по количеству вхождений, а не времени

**WeightedStorelessNumericStatistics –** классстатистики для числовых значений**,** взвешенной по времени. Подсчитывает среднее значение, стандартное отклонение, коэффициент вариации и медиану, взвешенные по времени. Применяется для сбора статистики по очереди и другим показателям, которым необходима взвешенная по времени статистика. Применяется по умолчанию для результатов типа **Double**, **Integer** и других реализующих класс **Number**

**ValueStatistics –** статистика для единичного значения, возвращает последнее значение, для автоматического режима это будет значение на момент остановки модели

**LogStatistics** – статистика, которая записывает значение из метода **update()** в файл с меткой модельного времени.

Подробнее взаимодействие классов рассмотрено на диаграмме последовательностей, которая вынесена на лист 6 графического материала

# 7. Апробирование разработанной системы для модельных условий

Апробирование разработанной системы осуществлялось при помощи многократного тестирования функционала. Выявленные в процессе тестирования ошибки и недочеты были исправлены на этапе рабочего проектирования. Результаты, которые реализуемы в старой грамматике были сверены с результатами на новой грамматике

На листе результатов представлены снимки работающей системы, а именно окно трассировки, окно графика, построенного по собираемым показателям и окно справки с документацией по новому синтаксису и новым возможностям подсистемы собираемых показателей

# 8. Заключение

В рамках данного курсового проекта были получены следующие результаты:

1. Проведено предпроектное исследование системы имитационного моделирования Rao X и выделены недостатки подсистемы собираемых показателей из системы
2. На этапе концептуального проектирования системы был сделан выбор общесистемной методологии проектирования, с помощью диаграммы пакетов нотации UML определены зависимости. Сформулировано техническое задание
3. На этапе технического проектирования выбраны необходимые технические средства, уточнена топология системы. Разработана грамматика и соответствующие синтаксические диаграммы, разработаны необходимые абстрактные классы AbstractResult и AbstractDataSource
4. На рабочем этапе проектирования был разработан программный код, реализующий разработанную грамматику, а также ее парсер. диаграмма классов, а так же диаграмма последовательностей подсистемы собираемых показателей
5. Результаты апробирования позволяют сделать вывод об адекватной работе разрабатываемой системы

# Список литературы

1. Емельянов В.В., Ясиновский С.И. Имитационное моделирование систем, язык и среда РДО. М.: МГТУ им. Н. Э. Баумана, 2009. -583с.
2. Мартин Р. Чистый код. Создание, анализ и рефакторинг / пер. с анг. Е. Матвеев – CПб.: Питер, 2010. – 464 стр.
3. Программирование на Java. Java Maven. [http://javaxblog.ru/article/java-maven-1/]
4. Подбельский В. В. Глава 10.3 Виртуальные функции и абстрактные классы \\ Абстрактные классы. // Язык Си++ / рец. Дадаев Ю. Г.. — 4. — М.: Финансы и статистика, 2003. — С. 365-373. — 560 с.

# Приложение 1. Листинг модели парикмахерской

enum Состояние\_парикмахера {Свободен, Занят}

type Парикмахерские {

Состояние\_парикмахера состояние\_парикмахера;

int количество\_в\_очереди;

int количество\_обслуженных;

}

resource парикмахерская = Парикмахерские.create(Состояние\_парикмахера.Свободен, 0, 0);

event Событие\_прихода\_клиента() {

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 [

состояние\_парикмахера == Состояние\_парикмахера.Свободен && количество\_в\_очереди > 0

];

set begin() {

\_Парикмахерская.количество\_в\_очереди = \_Парикмахерская.количество\_в\_очереди - 1;

\_Парикмахерская.состояние\_парикмахера = Состояние\_парикмахера.Занят;

}

set duration() {

return длительность\_обслуживания.next();

}

set end() {

\_Парикмахерская.состояние\_парикмахера = Состояние\_парикмахера.Свободен;

\_Парикмахерская.количество\_обслуженных = \_Парикмахерская.количество\_обслуженных + 1;

}

}

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, 20, 40);

set init() {

Событие\_прихода\_клиента.plan(интервал\_прихода.next());

}

set terminateCondition() {

return currentTime >= 7 \* 12 \* 60;

}

resultType Занятость\_парикмахера<Состояние\_парикмахера>(Парикмахерские парикмахерская) {

set evaluate() {

return парикмахерская.состояние\_парикмахера;

}

}

resultType Длина\_очереди<Integer>(Парикмахерские парикмахерская) {

set evaluate() {

return парикмахерская.количество\_в\_очереди;

}

}

result занятость\_парикмахера = new Занятость\_парикмахера(парикмахерская);

result длина\_очереди = new Длина\_очереди(парикмахерская);

result калькулятор = new Value([2\*2]);

result всего\_обслужено = new Value([парикмахерская.количество\_обслуженных]);

result пропускная\_способность = new Value([парикмахерская.количество\_обслуженных / currentTime \* 60]);

result длительность\_работы = new Value([currentTime / 60]);

# Приложение 2. Листинг новой грамматики результатов

enum Состояние\_парикмахера {Свободен, Занят}

type Парикмахерские {

Состояние\_парикмахера состояние\_парикмахера

int количество\_в\_очереди

int количество\_обслуженных

}

resource парикмахерская = Парикмахерские.create(Состояние\_парикмахера.Свободен, 0, 0)

event Приход\_клиента() {

Приход\_клиента.plan(currentTime + интервал\_прихода.next())

парикмахерская.количество\_в\_очереди=парикмахерская.количество\_в\_очереди+1

}

operation Обслуживание\_клиента() {

relevant парикмахерская\_ = парикмахерская.onlyif [

состояние\_парикмахера == Состояние\_парикмахера.Свободен && количество\_в\_очереди > 0

]

def begin() {

парикмахерская\_.количество\_в\_очереди = парикмахерская\_.количество\_в\_очереди - 1

парикмахерская\_.состояние\_парикмахера = Состояние\_парикмахера.Занят

}

def duration() {

return длительность\_обслуживания.next()

}

def end() {

парикмахерская\_.состояние\_парикмахера = Состояние\_парикмахера.Свободен

парикмахерская\_.количество\_обслуженных = парикмахерская\_.количество\_обслуженных + 1

}

}

logic Model {

activity обслуживание\_клиента = new Activity(Обслуживание\_клиента.create())

}

sequence интервал\_прихода = new Exponential(123456789, 1 / 30.0)

sequence длительность\_обслуживания = new Uniform(123456789, 20, 40)

def init() {

Приход\_клиента.plan(currentTime + интервал\_прихода.next())

}

def terminateCondition() {

return currentTime >= 7 \* 12 \* 60

}

result занятость\_парикмахера = Result.create([парикмахерская.состояние\_парикмахера])

result длина\_очереди = Result.create([парикмахерская.количество\_в\_очереди])

result всего\_обслужено = Result.create([парикмахерская.количество\_обслуженных], new LastValueStatistics())

result пропускная\_способность = Result.create([парикмахерская.количество\_обслуженных / currentTime \* 60])

result длительность\_работы = Result.create([currentTime / 60], new LastValueStatistics())

dataSource Занятость\_парикмахера<Состояние\_парикмахера>() {

def evaluate() {

return парикмахерская.состояние\_парикмахера

}

}

result занятость\_парикмахера\_ds = Result.create(new Занятость\_парикмахера())

1. Microsoft, Windows являются зарегистрированными торговыми марками или торговыми марками Microsoft Corporation (в США и/или других странах).

   Ubuntu является зарегистрированной торговой маркой Canonical Ltd.

   Названия реальных компаний и продуктов, упомянутых в данной пояснительной записке, могут быть торговыми марками соответствующих владельцев. [↑](#footnote-ref-1)