Оглавление

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
2. Синтезировать и исследовать стратегии управления
3. Прогнозировать и планировать функционирование системы в будущем
4. 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. Однако, подсистема собираемых показателей не была переработана и осталась рудементарной.

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

```
result имя_результата = тип_результата(параметры_результата);
```

Определены результаты пяти типов, которые указаны в Таблица 1

Таблица 1. Типы результатов

Тип результата	Описание
getValue (арифметическое_выражение)	Вычисление значения переданного в качестве параметра выражения в момент окончания моделирования.
watchParameter (полное_имя_параметра_ресурса)	Сбор статистических данных по изменению значения параметра ресурса.
watchState (логическое_выражение)	Сбор статистических данных по изменению состояния системы, описываемого логическим выражением.
watchQuantity (имя_типа_ресурса, логическое_выражение)	Сбор статистических данных по изменению количества ресурсов определенного типа, удовлетворяющих определенным условиям. Сбор статистики производится только по тем ресурсам, для которых значение логического выражения является истинным. Для сбора данных по всем ресурсам данного типа в качестве второго параметра следует указать ключевое слово any.
watchValue (имя_типа_ресурса, логическое_выражение, арифметическое_выражение)	Сбор статистических данных по произвольному арифметическому выражению. Сбор статистики происходит в момент уничтожения ресурса указанного типа, удовлетворяющего указанному условию.

Рассмотри модель колл-центра, в котором работают 3 оператора. Клиенты условно делятся на 2 типа. Соответсвенно один из операторов обслуживает только клиентов первого типа, остальные принимают звонки второго типа.

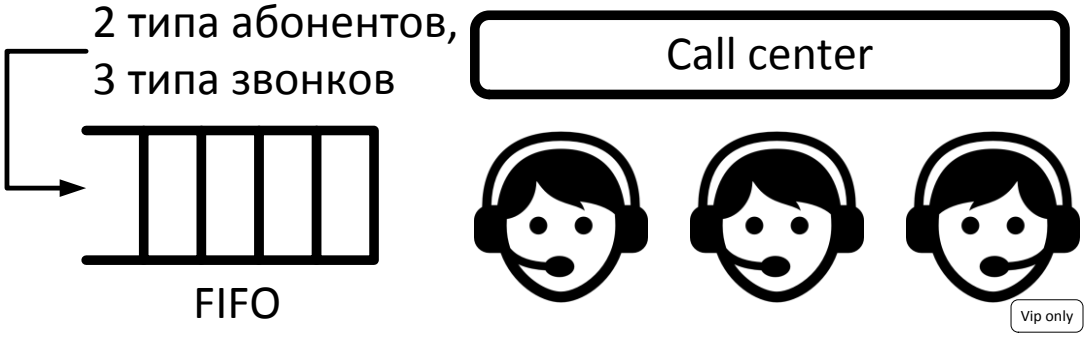

Рисунок 1. Модель колл-центра

Текст модели на языке РДО представлен в приложении 1. В качестве результатов моделирования необходимо собрать следующие показатели:

- Количество абонентов обслуженных каждым оператором и полное количество поступивших звонков
- Статистика по очереди
- Коэффициент загрузки каждого оператора
- Статистика по типам звонков
- Статистика по продолжительности звонков

С помощью типов результатов, используемых в текущей системе собираемых показателей, описанных ранее невозможно решить эту задачу целиком. И если первые два пункта с различной степень сложности решаются с помощью getValue, watchParameter и watchQuantity, то с коэффициентом загрузки дела обстоят хуже. Если состояние оператора имеет только 2 альтернативы {Свободен, Занят}, то задача решается с помощью watchState, если усложнить задачу, определив три состояния {Свободен, Разговаривает_с_абонентом, Решает_вопрос}, то задача теряет элегантное решение, так как в описанных типах результатов нет статистики по перечислимым типам, статистики по категориям.

Два последних пункта вообще не могут быть решены в текущих терминах, так как такую статистику необходимо взвешивать по количеству вхождений, а не по времени, как это сделано сейчас.

Так или иначе каждый тип результатов имеет свои недостатки, некоторые выходят из определения, хотя и не выглядя естественными, и другие более неочевидные.

Таблица 2. Недостатки текущих типов результатов

getValue	Вычисление только в конце прогона
watchParaneter	Ограничение доступных типов данных, взвешенная по времени статистика
watchState	Только взвешенная по времени статистика
watchQuantity	Только взвешенная по времени статистика
watchValue	Ограничение доступных типов данных, вычисление только в момент удаления

Общие недостатки:

- Отсутствие ручного подсчета статистики
- Ограничение доступных типов данных
- Отсутствие возможности выбора взвешенной по количеству статистики

Таким образом можно выделить проблемы в системе собираемых показателей:

- Не следует единому стилю
- Перегружена ключевыми словами
- Имеет большое количество искусственных ограничений
- Недостаточно гибкая

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 разработать новую подсистему собираемых показателей, которая будет отвечать следующим требованиям:

- Гибкость, а именно возможность выбора различных типов статистики для одного источника данных, возможность описания и использования пользовательских типов статистики
- Лаконичность, система не должна быть перегружена ключевыми словами, и следовать применяемому java-подобному стилю

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 Results и Statistics, для определения интерфейса взаимодействия, описываемыми ими

сущностей результатов и непосредственно статистики 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 2. Диаграмма компонентов системы Rao XРисунок 2 представлена упрощенная диаграмма пакетов системы Rao X, на которой указаны основные зависимости от сторонних библиотек. Более подробная схема вынесена на лист 2 графического материала.

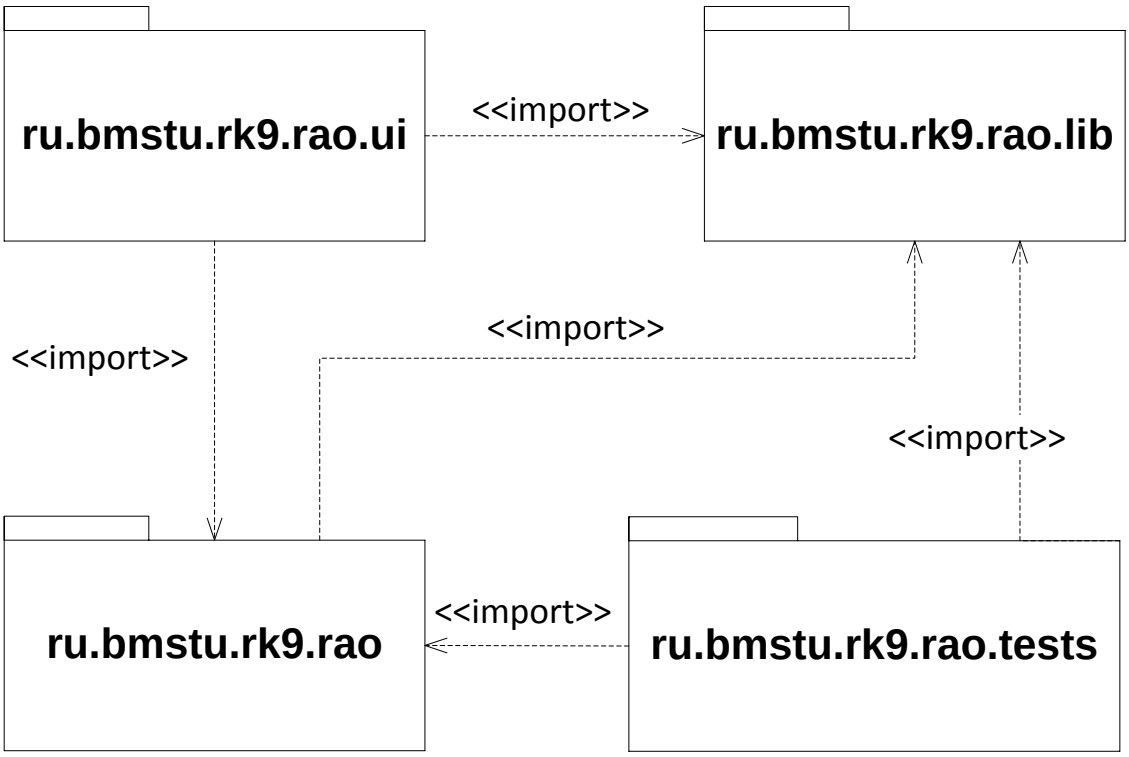

Рисунок 2. Диаграмма компонентов системы Rao X

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 граматики 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 **Result** и **Statistics,** 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.execution, пакет в котором описан запуск модели, в него необходимо внести изменения, необходимые для сбора показателей во время прогона

ru.bmstu.rk9.rao.ui.results, пакет в котором необходимо описать классы, позволяющие отобразить собираемы епоказатели

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: 13 февраля 2016г.
- Плановые сроки окончания работы по созданию системы: 9 июня 2016г.

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
- Возможность описания и использования пользовательских типов статистики
- Сбор статистики
- Отображение на экране результатов

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]
- 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

[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.

- 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 Result и Statistics.

5.1. Разработка грамматики собираемых показателей

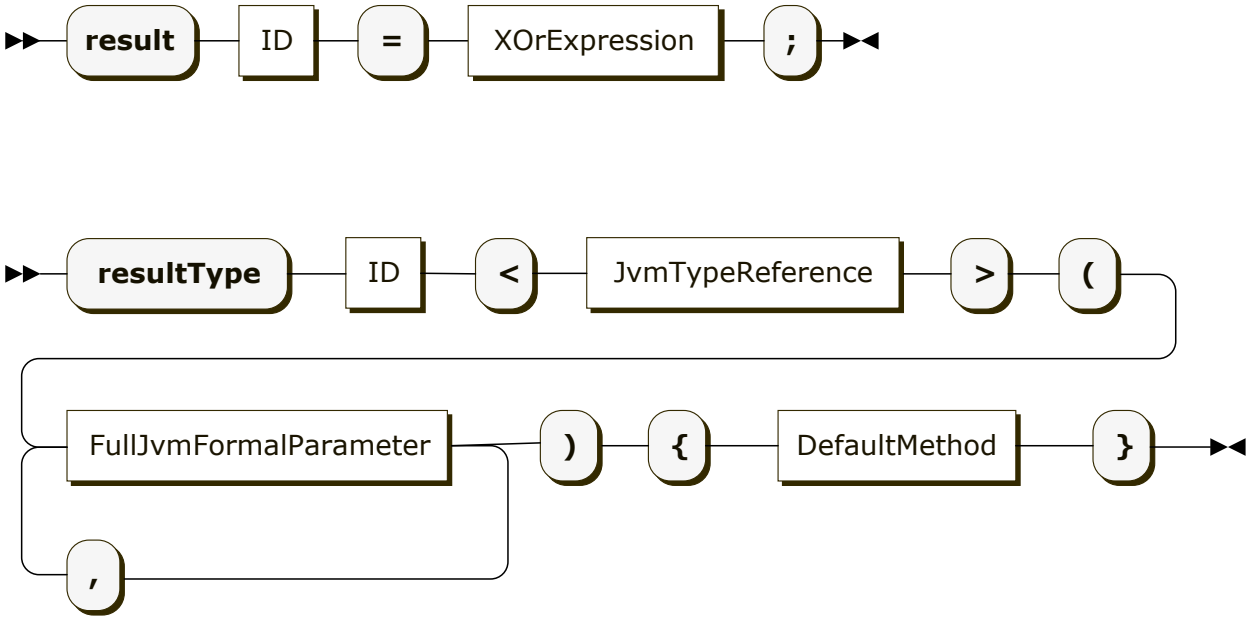

Рисунок 3. Синтаксическая диаграмма собираемых показателей

Рисунок 3 изображает синтаксическую диаграмму грамматики определения нового типа результатов и создания экземпляра этого класса.

Рассмотрим пример, соответствующий этой грамматике:

```
resultType Состояние_оператора_2<Состояние_оператора>() {
     set evaluate() {
          return оператор_2.состояние_оператора
     }
}
result состояние_оператора_1 = new Состояние_оператора_2();
```

Эти строчки создадут класс **Состояние_оператора_2**, реализующий абстрактный класс **Result** и экземпляр этого класса состояние_оператора_1, который будет являться непосредственно результатом

5.2. Разработка абстрактного класса Result

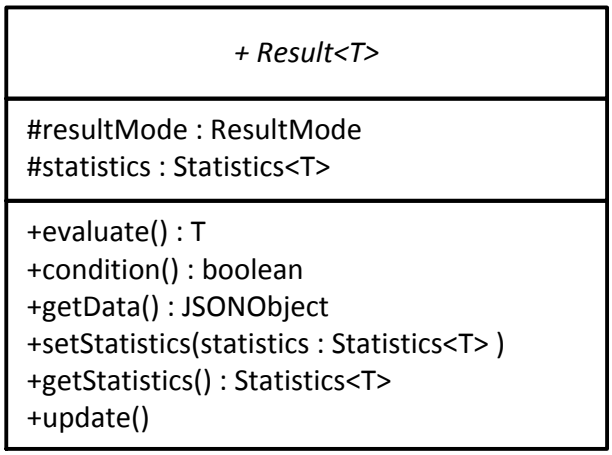

Рисунок 4. Класс Result

Рисунок 4 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

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

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

getDate() – метод возвращающий данные для отображения

update() – метод обновляющий статистику и результат

resultMode – атрибут**,** необходимый для гибкости, позволяет собирать показатели в ручном режиме

setStatistics() и **getStatistics()** – соответственно назначает и возвращает используемую статистику – экземпляр класса, реализующий абстрактный класс **Statistics()**

5.3. Разработка абстрактного класса Statistics

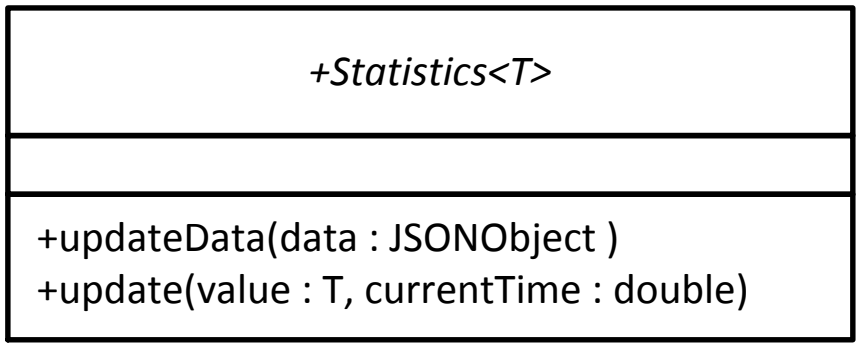

Рисунок 5. Класс Statistics

Как и **Result,** класс **Statistics** – шаблонный, что логично ведь экземпляр этого класса является атрибутом класса **Result.**

Метод **updateDate()** вызывается из **Result**.**getData()**, и принимает в качестве параметра JSONObject, который был в нем создан и дописывает в него необходимую информацию.

Метод **update()** вызывается из **Result.update()** и обновляет статистическую информацию, которая зависит от реализации

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. Для повышения удобства использования и уменьшения количества потенциальных ошибок были внесены исправления в DefaultMethodsHelper.java и RaoHelper.xtend.

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

6.1. Диаграмма классов

Рисунок 6 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.

Value – класс, реализующий абстрактный класс **Result,** создан для удобства пользования подсистемой сбора показателя, как предопределённый тип результатов. Использует статистику **ValueStatistics.**

CategoricalStatistics, StorelessNumericStatistics, ValueStatistics WeightedStorelessNumericStatistics – классы статистики, именно от выбора класса зависит собираемая статистика, так как **Result** является лишь источником данных. 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**)

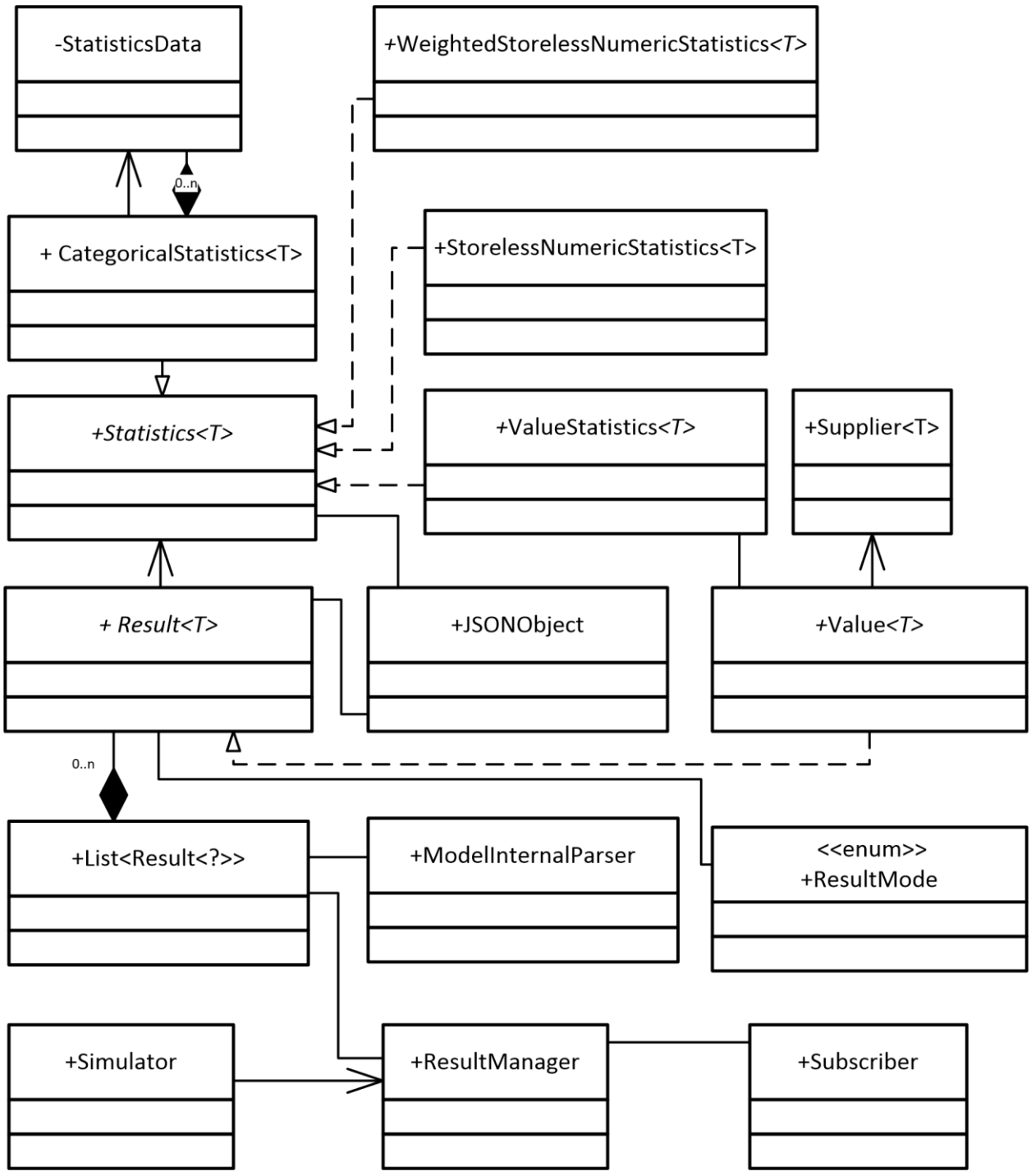

Рисунок 6

StorelessNumericStatistics – класс статистики 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 – класс статистики 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

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 редактора модели, в которой написана модель с использованием новой грамматики собираемых показателей. Листинг результатов представлен в приложении 2. Окно редактора Java кода, где можно описать собственный класс статистики, который реализует интерфейс из абстрактного класса **Statistics;** Окна с различным набором ошибок, демонстрирующие работу валидатора; Окно с выводом результатов на экран

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 Result и Statistics
4. На рабочем этапе проектирования был разработан программный код, реализующий разработанную грамматику, а также ее парсер и валидатор. Разработан программный код реализующий статистику и 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 – СПб.: 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 колл-центра

```
enum Тип_клиента {Тип1, Тип2}
enum Состояние_клиента {Ожидание, Разговор}
enum Состояние_оператора {Свободен, Занят}
enum Тип_обращения {T1, T2, T3}

type Колл_центр {
      int количество_в_очереди;
      Тип_обращения тип_последнего_звонка;
      double время_последнего_звонка;
}

type Клиенты {
      Тип_клиента тип;
      Состояние_клиента состояние;
      Тип_обращения тип_обращения;
}

type Операторы {
      Состояние_оператора состояние_оператора = Состояние_оператора.Свободен;
      int количество_обслуженных;
      int длительность_min;
      int длительность_max;
      Тип_клиента тип_клиента;
}

resource колл_центр = Колл_центр.create(0, Тип_обращения.T1, 0);
resource оператор_1 = Операторы.create(Состояние_оператора.Свободен, 0, 20,
40, Тип_клиента.Тип1);
resource оператор_2 = Операторы.create(Состояние_оператора.Свободен, 0, 25,
70, Тип_клиента.Тип2);
resource оператор_3 = Операторы.create(Состояние_оператора.Свободен, 0, 30,
60, Тип_клиента.Тип2);


event Событие_звонка_клиента() {
      Клиенты.create(случайный_тип_клиента.next(), Состояние_клиента.Ожидание,
случайный_тип_обращения.next());
      Событие_звонка_клиента.plan(currentTime + интервал_звонков.next());
      колл_центр.количество_в_очереди = колл_центр.количество_в_очереди + 1;
}

operation Образец_обслуживания_клиента() {
      relevant _Колл_центр = колл_центр.onlyif[количество_в_очереди > 0];
      relevant _Клиент =
Клиенты.all.filter[состояние.equals(Состояние_клиента.Ожидание)].any;
      relevant _Оператор =
Операторы.all.filter[состояние_оператора.equals(Состояние_оператора.Свободен)
                  &&
тип_клиента.equals(_Клиент.тип)].minBySafe[количество_обслуженных];

      set duration() {
            return длительность_обслуживания.next(_Оператор.длительность_min,
_Оператор.длительность_max);
      }

      set begin() {
            _Колл_центр.количество_в_очереди =
_Колл_центр.количество_в_очереди - 1;
            _Клиент.состояние = Состояние_клиента.Разговор;
            _Оператор.состояние_оператора = Состояние_оператора.Занят;
```

```
	}

	set end() {
		_Оператор.состояние_оператора = Состояние_оператора.Свободен;
		_Оператор.количество_обслуженных =
_Оператор.количество_обслуженных + 1;
		_Колл_центр.тип_последнего_звонка = _Клиент.тип_обращения;
		_Колл_центр.время_последнего_звонка = this.duration;
		типы_звонков.update();
		длительность_звонков.update();
		_Клиент.erase();
	}
}

logic Model {
	activity обслуживание_клиента = new
Activity(Образец_обслуживания_клиента.create());
}

sequence случайный_тип_обращения = new
DiscreteHistogram<Тип_обращения>(123456789,
	#[
		Тип_обращения.T1 -> 1.0,
		Тип_обращения.T2 -> 2.0,
		Тип_обращения.T3 -> 1.0
	]
);
sequence интервал_звонков = new Exponential(123456789, 1/30.0);
sequence длительность_обслуживания = new Uniform(123456789);
sequence случайный_тип_клиента = new
DiscreteHistogram<Тип_клиента>(123456789,
	#[
		Тип_клиента.Тип1 -> 1.0,
		Тип_клиента.Тип2 -> 5.0
	]
);

set init() {
	Событие_звонка_клиента.plan(currentTime + интервал_звонков.next());
}

set terminateCondition() {
	return currentTime >= 7 * 12 * 60;
}
```

Приложение 2. Листинг новой грамматики результатов

```
resultType Очередь<int>() {
      set evaluate() {
            return колл_центр.количество_в_очереди
      }
}

resultType Состояние_оператора_2<Состояние_оператора>() {
      set evaluate() {
            return оператор_2.состояние_оператора
      }
}

resultType Типы_звонков<Тип_обращения>() {
      set evaluate() {
            return колл_центр.тип_последнего_звонка
      }
}

resultType Длительность_звонков<Double>() {

}

result общее_кол_во_клиентов = new Value([|Клиенты.all.size]);
result обслужено_оператором_1 = new
Value([|оператор_1.количество_обслуженных]);
result очередь = new Очередь();
result состояние_оператора_1 = new Состояние_оператора_2();
result типы_звонков = new Типы_звонков(ResultMode.MANUAL);
result длительность_звонков = new Длительность_звонков(ResultMode.MANUAL, new
StorelessNumericStatistics());
```