содержание

[**1 Описание проблемы 3**](#_Toc169519533)

[**2 Описание класса генерируемых сетей**](#_Toc169519533) **5**

[**3 Описание требований к решению проблемы**](#_Toc169519533) **8**

[**4 Общее описание потенциальной системы**](#_Toc169519533) **9**

[**5 Описание библиотеки**](#_Toc169519533)**…………………………………………………………..11**

[**6 Описание проектирования классов…………………………………………**](#_Toc169519533)**..16**

[**6.1. Семантическое описание классов.................................................................**](#_Toc169519538)**23**

[**6.2. Диаграммы классов………………………………………………………….**](#_Toc169519539)**34**

[**7. Описание “ручного” режима и авто-каркаса для моделирования сетей из Библиотеки…………………………………………………………………………**](#_Toc169519541)**……45**

**[8. Описание генератора моделей………………………………………………..47](#_Toc169519542)**

**[9.Результаты тестирования на базе библиотеки……………………………...](#_Toc169519542)50**

**10.**[**Инструкция по установке, проверке, использованию ПО**](#_Toc169519542)**……………….53 11.Подведение итогов……………………………………………………………..54**

**12.Список используемой литературы…………………………………………..55**

**ПРИЛОЖЕНИЕ 1: КОД АВТО-КАРКАСА**

1.**Описание проблемы**

Моделирование используется в различных сферах человеческой жизни повсеместно начиная от принятия решений и заканчивая прикладными науками.Моделирование — исследование объектов познания на их моделях. Это означает что при моделировании исходный объект подменяется моделью, которая с некоторой погрешностью отражает характеристики исходного. Это бывает полезно, когда работать с объектом напрямую долго, трудоемко, опасно и т.д., например, модель солнечной системы может быть построена на ЭВМ, что проще и дешевле чем изучать ее напрямую. Можно считать моделирование универсальным методом описания физических, технических, организационных и других систем.

До появления вычислительной техники в основном исследовались и применялись аналитические модели. Аналитические модели основаны на описании поведения объекта в виде систем уравнений. Такие модели очень сложны в построении, использовании. Примером аналитической модели является, например, уравнение Клапейрона-Менделеева. Для простоты исследователи абстрагируются от частных или несущественных характеристик, и заостряют внимание на значимых. Абстрагирование снижает точность модели, но значительно может ее упростить. Невозможно идеально точно перенести объект на модель, в следствии чего всегда существует погрешность при моделировании. Для ранее упомянутого уравнения оговаривается, что оно применимо только к идеальному газу, которого в природе не существует в принципе. Аналитические модели хороши тем, что по входных параметрам они сразу же возвращают результат, и в общем случае результат работы модели зависит только от входных параметров.

Однако в практических задачах сложно использовать аналитическое моделирование. Например, сложно представить систему уравнений, которая сможет рассчитать поведение системы, которая изменяется случайным образов в следствие влияния внешней среды. Для решения такого рода задач применяется имитационное моделирование.

В настоящее время имитационное моделирование является общепризнанным методом для исследований в различных областях науки, производства, бизнеса и т.д.

Моделирование объекта производится для выявления его свойств:

* прогнозирования будущего состояния или поведения объекта;
* количественной оценки эффективности объекта;
* нахождения аналитической зависимости между характеристиками и параметрами, отыскания оптимальных значений параметров объектов;
* обучения специалистов.

Это типичные задачи анализа, прогнозирования и синтеза систем. При этом исходный объект заменяется другим, подобным, аналогичным ему. Все действия по анализу и синтезу производят на новом объекте, а полученные результаты (значения характеристик, выявленные закономерности) переносят на исходный объект. Таким образом, моделирование – процесс замещения одного объекта другим, с целью исследования его свойств с последующим переносом их на исходный объект. А модель – замещающий объект; результат отображения одной структуры на другую.

Имитационная модель отображает стохастический процесс смены дискретных состояний сетей массового обслуживания (СеМО) в непрерывном времени в форме моделирующе­го алгоритма. При его реализации на ЭВМ производится накопление статистических данных по тем атрибутам модели, характеристики кото­рых являются предметом исследований. По окончании моделирования на­копленная статистика обрабатывается, и результаты моделирования по­лучаются в виде выборочных распределений исследуемых величин или их выборочных моментов. Таким образом, при имитационном моделировании СеМО речь всегда идет о статистическом имитационном моделировании.

На практике имитационные модели реализуются при помощи ЭВМ. Существуют специализированные инструменты моделирования, которые позволяют моделировать в конкретных предметных областях (например, транспортные системы). Более сложным для освоения средством являются языки моделирования, которые предоставляют пользователю весь необходимый функционал для сбора моделей, запуска и генерации отчетов. Примером такого языка является GPSS. Это средство позволяет собирать модели практически неограниченной сложностью, быстро прогонять их и получать статистику. Работа пользователя заключается в задании входных характеристик (количество заявок, законы распределения и т.д.) и установки точек измерения времени. Однако зачастую описывать модели на GPSS трудоемко и большая часть функционала не используется.

Более сложный способ моделирования заключается в использовании языков программирования высокого уровня (C++, C# и т.д). Плюсом данного подхода является неограниченная функциональность. Модель будет настолько сложна, насколько пользователь сможет реализовать. Минусом же является то, что в общем случае весь функционал пользователю придется реализовывать самостоятельно.

Целью данной работы является снижение трудоемкости обучения моделированию и повышение эффективности за счетавтоматизации процессов моделирования, что снимет массу головной боли пользователю, которому необходимо обучится моделированию. Например, пользователю не придется заботится о том, как добавлять события в очередь событий, как получать из списка все произошедшие события, или как рассчитать путь заявки по сети. С другой стороны, ручной каркас позволит конфигурировать и организовывать процесс моделирования произвольным образом, что в конечном счете ускорит и упростит процесс обучения моделированию

Если пользователю необходимо средство для запуска моделей, ему следует предоставить упрощенный способ описания моделей по сравнению с языком GPSS.

[2. **Описание класса генерируемых сетей**](#_Toc169519533)

Стохастические сетевые модели (ССМ) содержат в себе статичную часть (узлы) и динамические объекты (заявки). Стохастические сетевые модели переходят в СеМО, если:

из обслуживающих узлов используются только устройства;

из маршрутных узлов используются только вероятностные или вообще не используются.

Сети массового обслуживания подразделяются на линейные и нелинейные. Сеть массового обслуживания является линейной, если входной поток представляет собой линейную суперпозицию выходных потоков других узлов. Заявки не могут теряться, размножаться и т.д.

Линейные сети масового обслуживания в свою очередь подразделяются на разомкнутые, замкнутые и смешанные сети. В разомкнутой СеМО есть только одиночные генераторы. В замкнутой СеМО могут быть только групповые генераторы. В смешанной СеМО могут быть, как одиночные, так и групповые генераторы и возможна обратная связь между обслуживающими узлами.

Разомкнутые, замкнутые и смешанные сети массового обслуживания могут быть как однородными, так и неоднородными. В однородных сетях может быть только один тип заявок. В неоднородных сетях соответственно может быть несколько типов заявок.

Однородные и неоднородные сети массового обслуживания подразделяются на экспоненциальные и неэкспоненциальные.В экспоненциальных сетях входные потоки Пуасоновские, законы распределения времени обслуживания в каждом устройстве экспоненциальные. В неэкспоненциальных сетях массового обслуживания закон паступления транзактов или закон распределения времени обслуживания в каждом устройстве, или и то и другое в каждом устройстве не является экспоненциальным.

Сеть считается заданной, если задана структура сети и потоки заявок. Потоки заявок считаются заданными, если известно количество заявок, циркулирующих в сети, и матрица вероятностей перехода. Структура сети определяется через задание количества устройств, количества каналов в них и скорости обслуживания поступившей заявки каналом устройства.

В качестве класса генерируемых сетей были выбраны не экспоненциальные однородные разомкнутые линейные сети массового обслуживания.

Для построения структурной схемы будем применять следующие типы узлов:

генератор входных потоков заявок:

Рисунок 2.1. Условное графическое обозначение генератора входных потоков заявок

Генератор генерирует числа, которые соответствуют временным отрезкам, через которое появляется очередная заявка.

узел типа устройство:



Рисунок 2.2. Условное графическое обозначение узла типа устройство.

Узел типа устройство вносит задержку на обслуживание заявки.

очередь:



Рисунок 2.3. Условное графическое обозначение очереди

Узел типа очередь выполняет функцию накопления поступающих заявок, при занятости узла типа устройство.

приемник:

Рисунок 2.4. Условное графическое обозначение приемника.

Приемник поглощает заявку и отмечает статистику по ней.

маршрутный (вероятностный) узел:

Рисунок 2.5. Условное графическое обозначение маршрутного (вероятностного) узла.

*Параметры элементов сети*

По описанию системы строится первоначальная схема Q-модели. Схема изображается графом G: - точки (вершины) представляют узлы системы; - стрелки (направленные дуги, ребра) представляют возможные направления перемещения заявок в процессе обслуживания в системе. Граф взвешенный - направлениям переходов (дугам) ставятся в соответствие условия перехода, в т.ч. задаваемые вероятностями.

Для описания устройств используются 2 основных параметра:

* канальность (К) — количество каналов обработки (обрабатывающих приборов в СМО) поступающих транзактов, в каждом из которых одновременно может обрабатываться ровно одна заявка;
* Закон распределения и его аргументы для генератора задержек

Источник запросов является точкой входа заявок в систему. Его параметры – входные параметры СМО.

Для описания источников запросов используются 2 основных параметра:

* Закон распределения и его аргументы для генератора заявок

Параметр функционирования модели есть совокупность вышеописанных численных параметров узлов СМО.

**3.Описание требований к решению проблемы**

Система должна обеспечивать:

1. Типовую функциональность имитационных моделей.

Это означает что система должна:

1. Инициализироваться посредством файлового описания
2. Загружать необходимое количество заявок в сеть
3. Управлять продвижением заявок
4. Управлять работой генераторов
5. Управлять работой очередей
6. Накапливать статистику
7. По завершению процесса моделирования печать отчета

То есть предоставлять весь необходимый функционал для автоматического

запуска моделей на основании текстового описания, а на выходе печатать отчет по накопленным данным. Работа с системой должна быть проста, от пользователя требуется минимальное количество усилий для получения результата.

1. Моделирование сетей в ручном режиме

Это означает что должна быть возможность используя средства системы писать собственные модели на языке высокого уровня. Система в таком случае выступает как фрейм, предоставляющий структуры данных и функционал, который поможет пользователю собирать собственные модели. В таком случае фрейм должен быть расширяемым и простым для использования. Необходимо наличие аналогий между названиями структур данных и теорией массового обслуживания.

Требования к средствам реализации:

* **Объектно-ориентированный подход.**
* **Принципы динамического полиморфизма.**
* **Средство описания проекта - язык (диаграммы) UML.**
* **MS Visual Studio, C++, библиотека STL**

**4.Общее описание потенциальной системы**

Основываясь на предыдущем пункте можно заключить что система должна быть проста, и позволять работать с ней как в автоматическом, так и в ручном режиме. Проблема заключается в том, что для обучения иерархия должна быть максимально простая, без оптимизаций, но для выполнения моделей оптимизации нужны. Поэтому систему условно необходимо разбить на части:

* Автоматическая работа с системой

Быстрый запуск моделирования на основании текстового описания. От пользователя требуется описать модель и запустить приложение.

* Ручная работа с системой

Подробное, но тем не менее автоматизированное описание работы моделирования. От пользователя требуется ввести, используя библиотеку классов, код моделирования (возможно использование каркаса).

В первом случае классы должны позволять быстрое выполнение модели, оптимальный расход ресурсов компьютера. Во втором случае требуется прозрачно, просто, наглядно предоставить возможность пользователю моделировать в ручном режиме. Отсюда появляется требования в двух иерархиях классов. Первая – закрытая, для генератора моделей. Вторая – открытая, для пользователя. Есть и такие классы, которые используется в рамках каждой из иерархий, например, классы STL, генераторы, очереди, модули формирования отчетов и т.д.

Также необходима информационная база системы, которая предполагает наличие документации (в том числе и для кода, каркаса) и тестовых примеров, которые позволяют наглядно ознакомится с работой генератора моделей.

Помимо описанного также предполагается наличие вспомогательного программного обеспечения (графический клиент, средство визуализации и рисования моделей, средство отладки). Вспомогательное программное обеспечение строится около основного, что еще раз подчеркивает необходимость иерархий классов и системы в целом к расширению. В прочем это выходит за рамки работы.

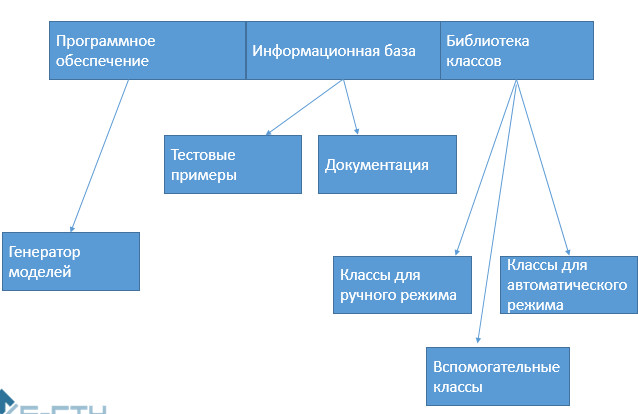


Рисунок 3.1 Схема проекта системы

**5 Описание библиотеки**

Перед тем как перейти к иерархии классов, необходимо поговорить о библиотеке классов. Библиотека классов состоит из модулей, взаимодействующих между собой. Для поддержки возможности к расширению система должна быть слабосвязанная. Начнем с описания общих модулей для обеих иерархий.

1)Модуль Heap(Куча)

Куча - это специализированная структура данных типа дерево. Кучи используются для реализации очередей с приоритетом. Поскольку система должна быть гибкой и расширяемой, имеет смысл начинать с низкоуровневое описания очередей. На данном этапе для очередей ставятся следующие требования:

* Вариант кучи, обеспечивающий экономное использование ресурсов
* Вариант кучи, обеспечивающий самый простой способ организации для расширения возможностей манипуляции с очередями
* Вариант кучи, работающий быстрее всего.

Таким образом было выбрано три варианта куч:

1)Бинарная куча : обеспечивает самое экономное использование ресурсов и самый простой способ организации, но медленнее всего

2)Куча Фибоначчи: обеспечивает хорошее быстродействие, но не надежна из-за возможности вырождения в список

3)2-3 Куча: как и куча Фибоначчи, только немного быстрее и нет ситуаций вырождения в список, но самый сложный способ организации.

Данный модуль описывает концептуальный и структурный уровни очередей

2)Модуль Queue(Очереди):

Минусом модуля Heap является отсутствие общего интерфейса. Каждая структура данных работает по своему принципу, прототипы методов из-за этого отличаются, и пользоваться модулем неудобно. Целью модуля Queue стоит создание общего интерфейса для работы с выбранной кучей. Более широко стоит вопрос о наличии различных способов задания приоритета (FIFO, LIFO, и т.д.).

Обязательно наличие следующих операций:

* Добавить в очередь
* Удалить из очереди
* Получить длину очереди
* Получить значение без удаления из очереди
* Получить тип очереди
* Очистить очередь

Следующее требование, это удобность работы. Пользователю необходимо предоставить сервисные функции для создания очередей унифицировано. Также необходимо наличие специализированных очередей, например, очередь событий. Специализированный очереди должны ещё сильнее упростить работу системой, автоматизируя некоторые процессы, например, получить все ближайшие события.

Данный модуль описывает интерфейсный и уровни реализации, фабричный и частный

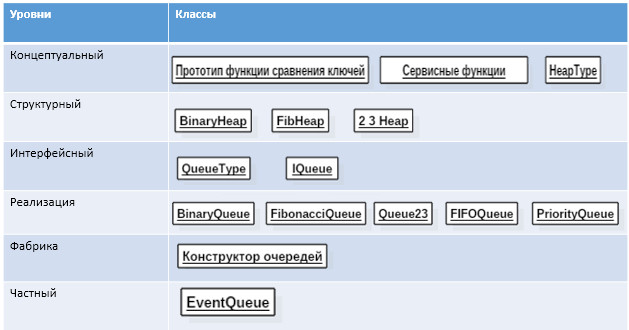


Рисунок 5.1 :Уровни организации очередей в системе.

1. Модуль Random(случайные величины)

Предназначение модуля: предоставить средство генерации псевдослучайных чисел в том числе и по законам распределений.

Генератор СВ должен выдавать по требованию целые и вещественные числа из диапазона. Реализация в библиотеке основана на генераторе Мерсена Твистера, реализация взята из STL. Является основой для распределения. Распределение обеспечивает:

* Генерацию числа
* Получение типа распределения
* Получение аргументов распределения
* Создание копии данного распределения с новыми аргументами.

Реализованы наиболее распространенные распределения:

* Бернулли
* Биномиальное
* Экспоненциальное
* Нормальное
* Равномерное
* А также различные адаптированные варианты.

1. Модуль Report( отчет)

Его базовой функция является печать отчета в файл используя различные способы компоновки, локализации и вывода.

По умолчанию реализован один вариант компоновки, один вариант локализации(русский) и три варианта вывода(строковая переменная, терминал, файл).

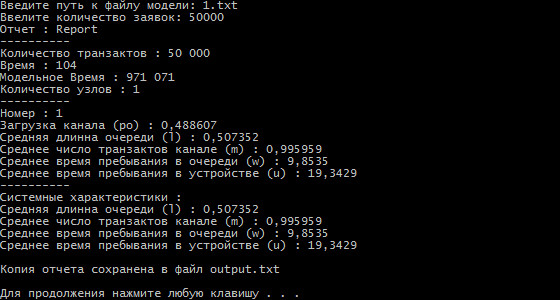


Рисунок 5.2: Пример работы модуля Report

Остальные модули для автоматического и ручного режима отличаются. Начнем с автоматического режима:

1)Модуль Statistic.

В данном варианте статистика тесно переплетена с рабочими компонентами модели. Это связано с тем, что статистика накапливается напрямую компонентами модели, по факту принадлежит им. Было бы расточительно с точки зрения скорости работы постоянно вызывать методы для работы со статистикой, будь она обособлена от модели. Вся статистика собирается автоматически. Накапливается информация о времени появления заявки в модели, о приходе в каждую из очередей, в каждое из устройств, о времени выхода из системы. На основании этих данных все необходимые параметры могут быть расcчитаны:

* Загрузка канала (ро)
* Средняя длинна очереди (l)
* Среднее число транзактов канале (m)
* Среднее время пребывания в очереди (w)
* Среднее время пребывания в устройстве (u)

А также системные эквиваленты.

Статистику условно можно разделить на следующие части:

1. Статистика транзакта. Хранит информацию о времени появления, и о времени окончания обработки транзакта.
2. Статистика работы устройства. Хранит информацию о времени появления заявок в канале, начала захвата канала и освобождения.
3. Статистика состояний канала. Хранит информацию о всех состояниях устройства (количество свободных каналов, длинна очереди)
4. Статистика модели. Может хранить общие характеристики модели, такие как реальное время работы, количество тактов, необходимых для обработки всех заявок и т.д.
5. Полная статистика. Хранит все данные, которые необходимы для расчета характеристик системы.

6) Обработанная статистика. Хранит результат обработки данных статистики

2) Модуль Model

Отвечает за протекание процесса моделирования. Является ядром системы. Именно здесь происходит диспетчеризация очередей, заявок, генераторов. Именно здесь происходит обработка собранной статистики. Здесь также описаны все структуры данных, необходимые для работы моделирования. Все, что связано непосредственно с моделирование сидит в одном месте в целях оптимизации операций доступа, что повышает быстродействие системы.

3)Модуль Descriptor

Основная идея создания данного модуля заключается в предоставлении возможности быстрой сериализации состояния объекта. При этом не нарушается инкапсуляция. Например, для генератора чисел будет сериализован тип и аргументы, но не будет сериализовано внутренне состояние работы. Это позволяет делать безопасные конструкторы моделей, которые выполняют функции сериализации и десериализации модели не нарушая инкапсуляции. Также это предоставляет возможность конструирования объектов по описанию, что позволяет сериализовать не только модель, но и генераторы, компоненты модели.

4) Вспомогательный функционал

Содержит такие функции как:

* Сериализация и Десереализация модели
* Фабричные функции для упрощенного сбора распределений и очередей
* Вспомогательные функции лексического разбора(токенизация и т.д)

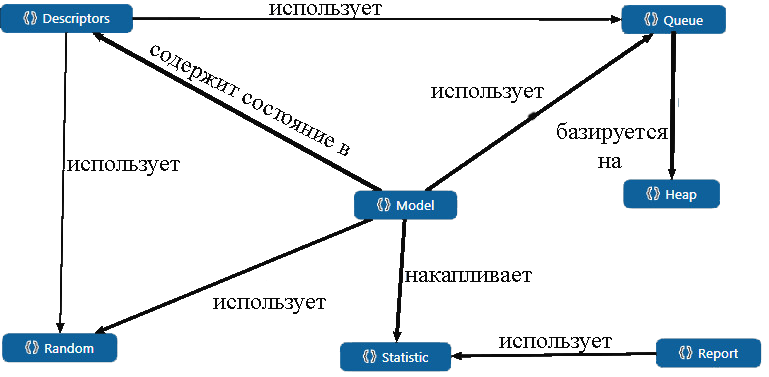


Рисунок 5.3: Взаимодействие модулей для автоматического выполнения моделей

Отличие ручного режима от автоматического в общих чертах заключается в том, что ручной более соответствует терминам предметной области, а в автоматическом применяются различные оптимизации. Ключевые отличия:

* Ручной режим не поддерживает сериализацию и функционал модуля Descriptors. Это связано с тем что ручной режим не предполагает расширения и создания инфраструктуры. Отсутствие данного уровня абстракции делает иерархию более прозрачной и простой для типичного пользователя.
* Статистика откреплена от ядра моделирования (модуль Model). Пользователю проще работать с системой, когда у него есть конкретный объект типа таблица статистики, а поскольку при ручном моделировании вся модель, как правило, расположена в одном модуле, снижаются затраты на перенос данных из модели в таблицу статистики.
* Ручной режим содержит больше побочных классов (классы, которые присутствуют чисто формально) и вспомогательных оберток. Таким образом алгоритм проще перенести с блок-схемы на код, а раз нам не нужно беспокоится о быстродействии, можно использовать дополнительные уровни абстракции для упрощения работы с системой( например вместо понятия очередь использовать очередь канала, а вместо целого числа понятие время)
* Автоматический режим более адаптирован для расширения.

Ручной режим:

Взаимодействие модулей, как и у автоматического режима за исключением описанного выше.

6. **Описание проектирования классов**

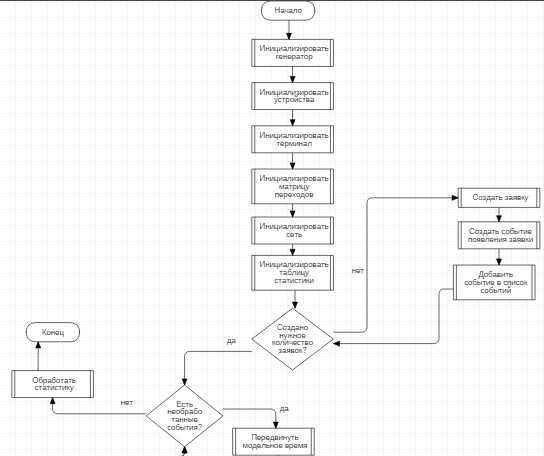
Описание классов проще всего начать с ручного режима, так как он создан быть наиболее интуитивно понятным. Поскольку иерархия должна обеспечивать работу алгоритма моделирования, будем выявлять сущности из блок-схемы алгоритма.

Рисунок 6.1: Подробная блок-схема алгоритма моделирования часть 1

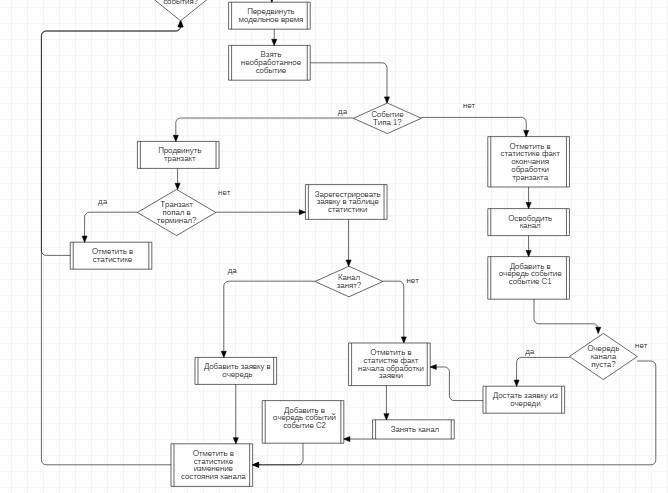


Рисунок 6.2: Подробная блок-схема алгоритма моделирования часть 2

Теперь выделим кандидаты в классы из текстового описания:

1.Инициализация

1.1.инициализировать ГЕНЕРАТОР

1.2.инициализировать УСТРОЙСТВА

1.3.инициализировать ТЕРМИНАЛ

1.4.инициализировать МАТРИЦУ ПЕРЕХОДОВ

1.5 инициализировать СЕТЬ

1.6.инициализировать ТАБЛИЦУ СТАТИСТИКИ

2.Пока не создано нужное количество ЗАЯВОК

2.1.Создать ТРАНЗАКТ

2.2.Создать СОБЫТИЕ появления заявки

2.3.добавить СОБЫТИЕ в СПИСОК СОБЫТИЙ

3.Пока есть необработанные СОБЫТИЯ

3.1.Продвинуть МОДЕЛЬНОЕ ВРЕМЯ

3.2.Взять необработанное СОБЫТИЕ

3.3.Если СОБЫТИЕ типа 1 на 3.3.1 иначе на 3.3.3

3.3.1.Продвинуть ТРАНЗАКТ

3.3.2.Если ТРАНЗАКТ попал в ТЕРМИНАЛ то 3.3.2.1 иначе 3.3.2.2

3.3.2.1.Отметить в СТАТИСТИКЕ смену состояния модели и перейти на 3

3.3.2.2. Зарегистрировать заявку в СТАТИСТИКЕ

3.3.2.3. Если КАНАЛ занят, то перейти на 3.3.2.3.1 иначе 3.3.2.3.3

3.3.2.3.1. Добавить заявку в ОЧЕРЕДЬ СОБЫТИЙ

3.3.2.3.2. Отметить в СТАТИСТИКЕ смену состояния модели и перейти на 3

3.3.2.3.3. Отметить в СТАТИСТИКЕ факт начала обработки заявки

3.3.2.3.4. Занять КАНАЛ

3.3.2.3.5. Добавить в ОЧЕРЕДЬ СОБЫТИЙ СОБЫТИЕ С2

3.3.2.3.6. Отметить в СТАТИСТИКЕ смену состояния модели и перейти на 3

3.3.3. Отметить в СТАТИСТИКЕ факт окончания обработки ТРАНЗАКТА.

3.3.4. Освободить КАНАЛ

3.3.5.Добавить в ОЧЕРЕДЬ СОБЫТИЙ СОБЫТИЕ С1

3.3.6.Если ОЧЕРЕДЬ КАНАЛА пуста, то 3.3.7 иначе 3.3.8

3.3.7. Отметить в СТАТИСТИКЕ смену состояния модели и перейти на 3

3.3.8.Перейти на 3.3.2.3.3.

4. Обработать СТАТИСТИКУ.

Из описания можно извлечь следующие сущности и поведения:

1)ГЕНЕРАТОР

-Генерирует транзакты

2)УСТРОЙСТВО = КАНАЛ (для простоты канальность примем числом)

-Занимается

-Освобождается

3)ТЕРМИНАЛ

-Освобождает транзакты

4)МАТРИЦА ПЕРЕХОДОВ = МАТРИЦА

5)СЕТЬ = МОДЕЛЬ

-Хранит 1-4

-Двигает время

6)ТАБЛИЦА СТАТИСТИКИ = СТАТИСТИКА

-Регистрирует транзакт

-Обновляет состояние

-Отмечает занятие канала

-Отмечает освобождение канала

-Обрабатывает данные

7)ЗАЯВКА = ТРАНЗАКТ

-Двигается

8)СОБЫТИЕ

9)СПИСОК СОБЫТИЙ

-Возвращает произошедшие события

10)МОДЕЛЬНОЕ ВРЕМЯ = ВРЕМЯ.

Алгоритм можно представить, как машину состояний. Это представление позволит взглянуть на проблему, с другой стороны.

с1 запуск прецедента начала моделирования

д1 очитска статистики,очистка очередей событий.

с2 модель инициализирована или необходимо сгенерировать очередной транзакт

д2 создать объект траназакт, поместить связанное с ним событие(С1) в очередь событий.

с3 был загружен последний транзакт

д3 сдвинуть модельное время на ближайшее событие

с5 список ожидаемых событий пуст

д5 остановить моделирование

с7 событие С1(новый транзакт)

д7 достать из события объект транзакт

с8 появление возможности работы с транзактом

д8 пометить в статистике модели что транзакт появился в заданный такт времени

с10 транзакт достиг терминала

д10 удалить траназкт из сети. изменить статистику.

с11 транзакт дошел до канала и канал не занят

д11 занять канал, добавить в очередь событий событие освобождения канала(С2). Изменить данные в статистике данного канала

с12 транзакт дошел до канала и канал занят

д12 добавить транзакт в конец очереди

изменить данные статистики

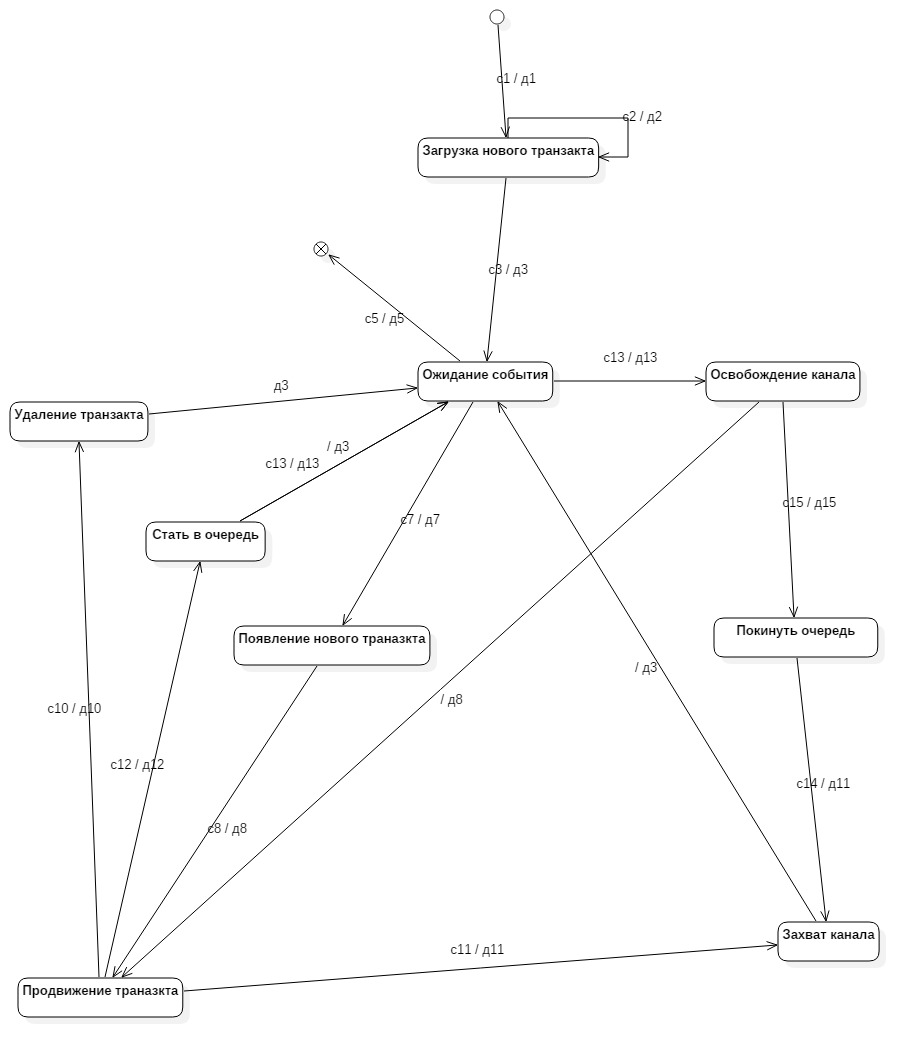
с13 возникновение события освобождения канала (С2)

д13 пометить канал как не занят. обновить статистику

с14 транзакт покинул очередь

с15 в очереди к каналу присутсвовали транзакты

д15 достать из очереди первый транзакт. изменить статистику

Рисунок 6.3: Диаграмма состояний

Для автоматического режима:

Объединим полученные знания с требованиями к модулям из предыдущего пункта и породим диаграмму первичных сущностей для автоматического режима:

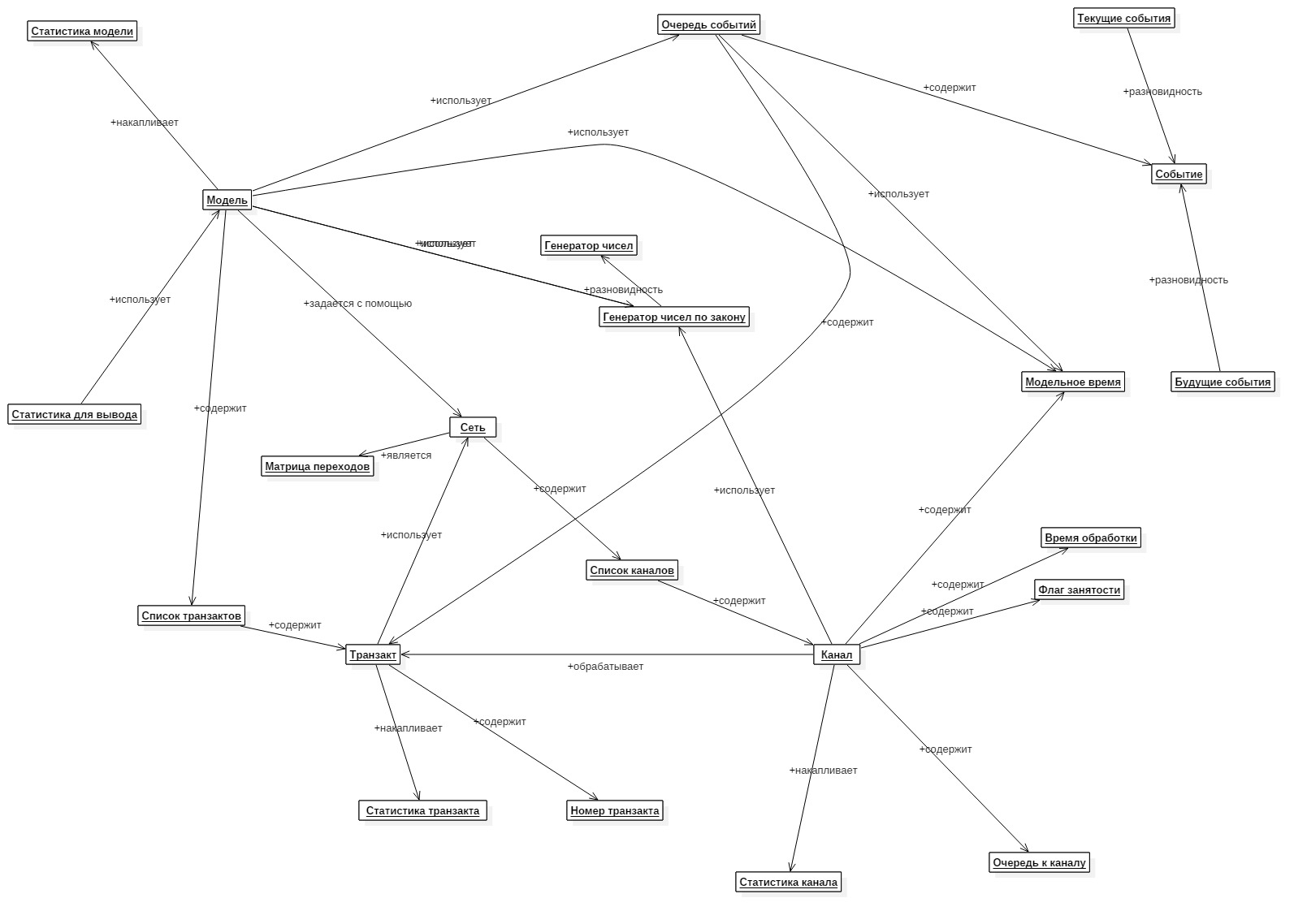


Рисунок 5.3: Диаграмма первичных сущностей.

Поскольку автоматический режим должен быть оптимизированным, можно применить следующие модификации:

1. Расширить понятие сеть. То есть необходимо перенести весь алгоритм управления моделированием и данные в одно место для ускорения работы
2. Перенести обязанности класса терминал и таблица статистики на сеть. Задачи этих классов не велики, и их можно не описывать. Терминал можно заменить счетчиком обработанных заявок. Как говорилось ранее перенос статистики на сеть делает систему быстрее.
3. Убрать “маленькие классы”. Например, класс время хранит лишь номер такта. Можно заменить класс на переменную типа целое число.

6.1 **Семантическое описание классов**

Начнем с общих модулей:

1)Модуль Heap

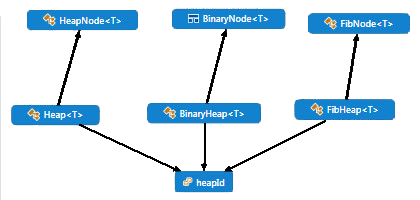


Рисунок 6.1.1: Семантика модуля Heap

|  |  |
| --- | --- |
| **Класс** | **Понятие** |
| HeapNode<T> | Узел 2-3 кучи |
| Heap<T> | 2-3 куча |
| BinaryNode<T> | Узел бинарной кучи |
| BinaryHeap<T> | Бинарная куча |
| FibNode<T> | Узел кучи фибоначчи |
| FibHeap<T> | Куча Фибоначчи |
| heapId | Идентификаторы куч |

Таблица 6.1.1: Описание классов модуля Heap

2)Модуль Queue

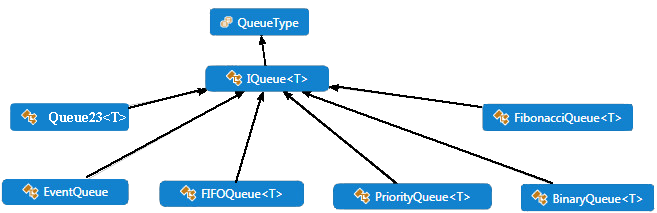


Рисунок 6.1.2: Семантика модуля Queue

|  |  |
| --- | --- |
| **Класс** | **Понятие** |
| QueueType | Идентификаторы очередей |
| IQueue<T> | Интерфейс очередей |
| Queue23<T> | Очередь, основанная на Heap<K> |
| EventQueue | Очередь событий |
| FIFOQueue<T> | Очередь по дисциплине FIFO |
| PriorityQueue<T> | Очередь с приоритетом из STL |
| BinaryQueue<T> | Очередь на основе BinaryHeap<K> |
| FibonacciQueue<T> | Очередь на основе FibHeap<K> |

Таблица 6.1.2: Описание классов модуля Queue

3)Модуль Random

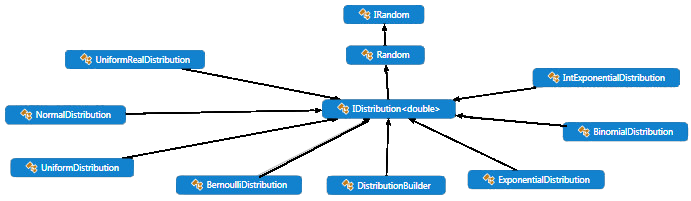


Рисунок 6.1.3: Семантика модуля Random

|  |  |
| --- | --- |
| **Класс** | **Понятие** |
| IRandom | Интерфейс генератора чисел |
| Random | Генератор чисел |
| IDistribution<double> | Интерфейс распределения |
| UniformRealDistribution | Равномерное распределение |
| NormalDistribution | Нормальное распределение |
| UniformDistribution | Равномерное распределение с округлением |
| ExponentialDistribution | Интерфейс генератора чисел |
| BinomialDistribution | Генератор чисел |
| IntExponentialDistribution | Интерфейс распределения |
| BernoulliDistribution | Распределение Бернулли |
| DistributionBuilder | Конструктор распределение |

Таблица 6.1.3: Описание классов модуля Random

4)Модуль Report



Рисунок 6.1.4: Семантика модуля Report

|  |  |
| --- | --- |
| **Класс** | **Понятие** |
| IReport | Интерфейс печати отчетов |
| Report | Печать отчета в строку |
| StreamReport | Печать отчета в поток |
| Locale | Локаль |
| RussianLocale | Локаль для русского языка |

Таблица 6.1.4: Описание классов модуля Report

Также существуют сервисные функции, которые не принадлежат к пространству имен, а также класс BasicMatrix, который является внешним компонентом и также не принадлежит к пространствам имен системы.Класс используется для описания матриц переходов

Далее опишем модули, которые уникальны для автоматического режима:

1)Модуль Statistic

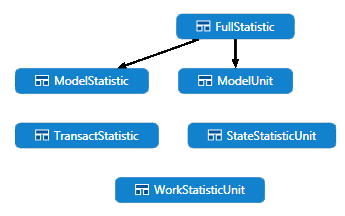


Рисунок 6.1.5: Семантика модуля Statistic для автоматического режима

|  |  |
| --- | --- |
| **Класс** | **Понятие** |
| TransactStatistic | Статистика заявки |
| WorkStatisticUnit | Страница данных о работы канала |
| StateStatisticUnit | Страница данных о состояний канала |
| ModelUnit | Страница обработанной статистики |
| ModelStatistic | Статистики модели |
| FullStatistic | Обработанная статистика |

Таблица 6.1.5: Описание классов модуля Statistic

Стоит отметить что некоторые классы не имеют связей с другими. Это означает что данные классы выступают как структуры для хранения информации и используются модулем Model о котором речь пойдет дальше.

2) Модуль Model

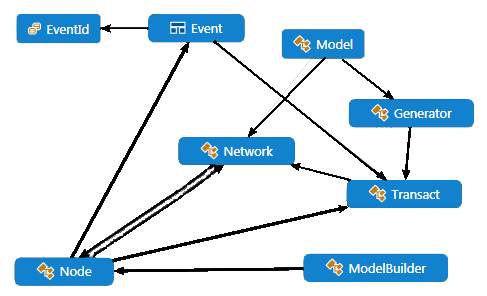


Рисунок 6.1.6: Семантика модуля Model для автоматического режима

|  |  |
| --- | --- |
| **Класс** | **Понятие** |
| EventId | Идентификаторы событий |
| Event | Событие |
| Model | АУМ + связь всех компонент модели |
| Generator | Генератор заявок |
| Node | Устройство |
| ModelBuilder | Сериализатор моделей |
| Transact | Заявка |
| Network | Сеть |

Таблица 6.1.6: Описание классов модуля Model

3)Модуль Descriptors

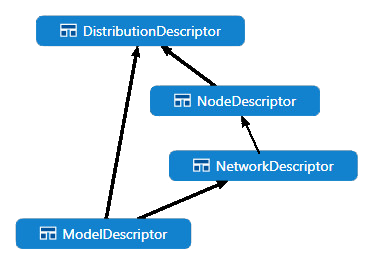


Рисунок 6.1.7: Семантика модуля Descriptors для автоматического режима

|  |  |
| --- | --- |
| **Класс** | **Понятие** |
| DistributionDescriptor | Хранение состояния распределения |
| NodeDescriptor | Хранение состояния устройства |
| NetworkDescriptor | Хранение состояния сети |
| ModelDescriptor | Хранение состояния модели |

Таблица 6.1.7: Описание классов модуля Descriptors

Теперь опишем модули, уникальные для ручного режима.

1)Модуль Statistic

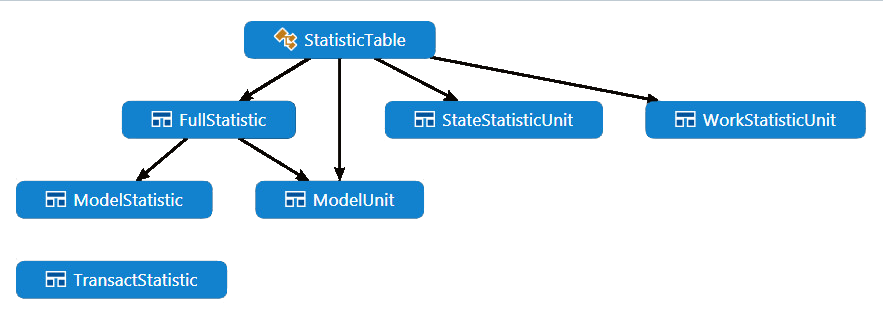


Рисунок 6.1.8: Семантика модуля Statistic для ручного режима

|  |  |
| --- | --- |
| **Класс** | **Понятие** |
| TransactStatistic | Статистика заявки |
| WorkStatisticUnit | Страница данных о работы канала |
| StateStatisticUnit | Страница данных о состояний канала |
| ModelUnit | Страница обработанной статистики |
| ModelStatistic | Статистики модели |
| FullStatistic | Обработанная статистика |
| StatisticTable | Статистическая таблица |

Таблица 6.1.8: Описание классов модуля Statistic

2)Модуль Model

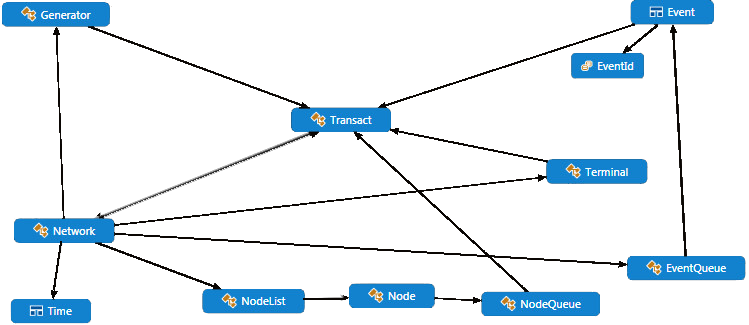


Рисунок 6.1.9: Семантика модуля Model для ручного режима

|  |  |
| --- | --- |
| **Класс** | **Понятие** |
| EventId | Идентификаторы событий |
| Event | Событие |
| Generator | Генератор заявок |
| Node | Устройство |
| NodeList | Список устройство. |
| NodeQueue | Очередь устройства |
| Transact | Заявка |
| Terminal | Терминальный узел |
| Time | Время(модельное) |
| Network | Сеть. Также содержит вспомогательные методы для организации процесса моделирования |
| EventQueue | Очередь событий |

Таблица 6.1.9: Описание классов модуля Model

Семантика показывает какие из классов взаимодействуют между собой, но не отражает характер взаимодействия. Для отображения характера взаимодействия между классами необходима диаграмма отношений. На диаграммах отношений будут отображены только значимые части иерархий для большей наглядности.

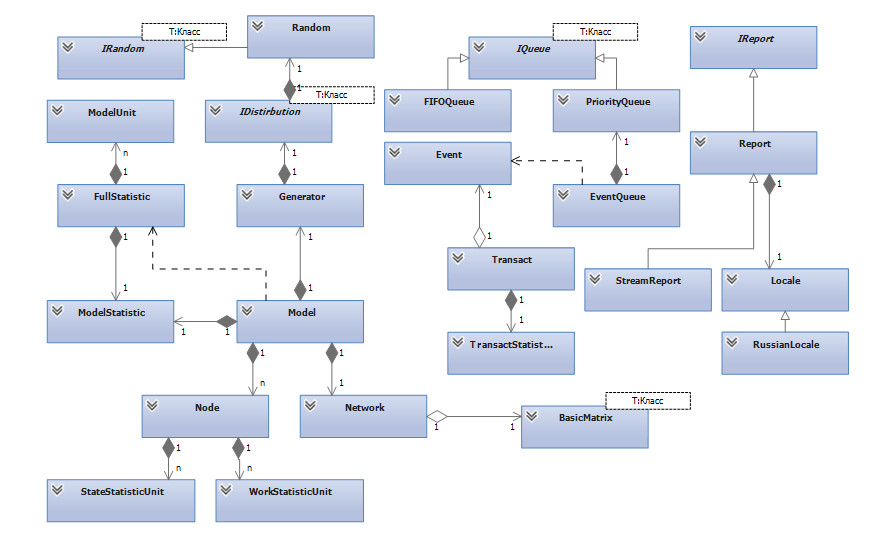


Рисунок 6.1.10 Диаграмма отношений для автоматического режима

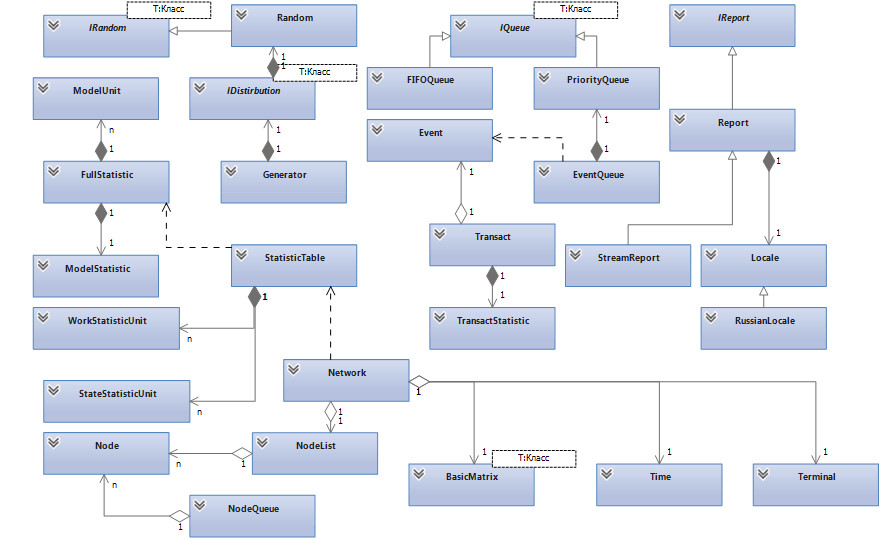


Рисунок 6.1.11: Диаграмма отношений для ручного режима.

6.2 **Диаграммы классов**

Диаграммы класс для общих классов:

Будут приведены только диаграммы тех классов, которые используются напрямую. Например, класс Heap не предназначен для использования пользователем, так как является не безопасным. Для пользователя существует обертка, называемая Queue23. Поэтому диаграммы классов модуля Heap не будут приводится.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | Перечисление основ для очередей, поддерживаемых системой. Необходим для упрощения создания системных очередей. Также позволяет очередям хранить метаданные, и сохранять в компактном виде информацию об используемой очереди. | | |
|  | | | IQueue – Шаблонный интерфейс очередей. Все очереди, используемые в системе реализуют этот интерфейс. Содержит минимальный набор операций, необходимый для работы очередей. Реализуя его, пользователь может добавлять собственные варианты очередей. Очереди содержат строго целочисленные ключи, однако порядок сравнения ключей может быть задан произвольно. Для системных очередей, основанных на Heap ключ задается при помощи функций приоритетности. Поле \_type содержит тип очереди из перечисления QueueType. Перечисление используется для определения основы реализации очереди. | | |
|  | | | Queue23 – Реализация очереди, основанная на Heap. Является безопасной оберткой. Heap использует список 2-3 деревьев, называемый 2-3 кучей для управление очередью с приоритетом. Конструктор требует наличия функции приоритетности, которая задает правило сравнения ключей. По умолчанию используется приоритет меньшему. Данная очередь является самой быстрой, но самой тяжелой как в плане внутренней структуры, так и в плане потребления ресурсов. Из-за этого ее рекомендуется использовать только в случае необходимости выжать максимум производительности. Преимущества и недостатки остальных куч были описаны выше, работа с ними ничем не отличается от работы с этим классом, посему нет причин добавлять диаграммы классов BinaryQueue и FibonacciQueue. | | |
|  | | | | | FIFOQueue - Очередь по дисциплине FIFO. Основана на очереди STL.Особенностью является то, что ключ при добавлении в очередь вычисляется автоматически на основе длинны очереди, поэтому при использовании метода Add первый параметр можно задавать произвольным. PriorityQueue основана на очереди с приоритетом из STL.Поддерживает только функцию сравнения ключей по умолчанию. Самый надежный вариант из всех очередей. |
|  | | | | Очередь событий является оберткой над системными реализациями интерфейса IQueue. Эта очередь уже настроена на хранение событий, а конструктор позволяет при желании задать структуру, на которой будет базироваться очередь. Метод добавления упрощен. Ключ автоматически извлечется на основании данных события. Метод GetHappened возвращает список событий, которые произошли до заданного времени, а GetNextEvents вернет все верхние события, произошедшие в один такт времени. | |
|  | | IRandom – интерфейс генератора СВ. Все генераторы в системе реализуют этот интерфейс. | | | |
|  | | Random – реализация генератора СВ, основанная на генераторе Mersenne Twister из STL. Является основой генерации чисел в системе. | | | |
|  | | IDistribution – абстрактный класс, который хранит в себе Random и на основании его выхода генерирует числа по заданному распределению. Распределение задается на основе аргументов. Метод Argument возвращает список аргументов. Метод Clone создает копию данного генератора, но с другими аргументами. Метод Type возвращает название генератора и используется в системе для управления генераторами, упрощенного создания генераторов Все генераторы по закону распределения в системе наследуются от этого класса | | | |
|  | |  | | | |
|  | IReport – абстрактный класс, реализация метода GetReport которого позволит печатать статистику в строку. Используются длинные двухбайтовые строки. Также есть возможность задания разделителя. | | | | |
|  | Класс для печати отчета в строку. Печать формируется по заранее заданному шаблону и с использованием локализации. Наследование от класс позволяет определить свой шаблон для печати. Метод GetReport печатает статистику в строку | | | | |
|  | Класс для печати отчета в поток. Поддерживает как терминал, так и файловые потоки. Метод Bind связывает объект с потоком для печати. Метод Print печатает отчет. Существует также перегрузка конструктора, которая принимает в качестве параметра поток для печати. | | | | |
|  | Локализация, используемая для печати отчетов. Наследование от класса позволяет определять собственные локализации. Суть локализации заключается в присвоении названия каждому термину, которой потом будет использован при печати отчета Таким образом можно изменять локализацию без переопределения класса печать отчетов. Также существует класс RussianLocale с реализованной русской локализацией. | | | | |
|  |  | | | | |
|  | Представляет базовую функциональность для работы с матрицами. Используется для задания матрицы переходов. | | | | |

|  |  |
| --- | --- |
|  | StateStatisticUnit хранит страницу статистики состояния устройства. Хранит данные о количестве занятых каналов, о длине очереди и такте, в которой происходило измерение.  WorkStatisticUnit хранит страницу статистики работы устройства. Фиксирует такт прибытия заявки, захвата ею канала и освобождения его |
|  | TransactStatistic хранит информацию о такте появления заявки в сети, и такте завершения обработки заявки.  ModelUnit хранит страницу с результатами обработки статистики. Поля соответствуют характеристикам, которые могут быть рассчитаны в результате обработки данных, накопленных в процессе моделирования |
|  | Model Statistic это информация модели. Хранит данные, необходимые для расчета реального времени выполнения моделирования, а также количество тактов и обработанных транзактов.FullStatistic это обработанная статистика, которая может быть использована для печати отчетов. Хранит модельные данные, статистику по каждому устройству и системную статистику. |

Диаграммы классов для автоматического режима:

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | Перечисляет возможные идентификаторы событий. Возможно два события: Событие появления(освобождения) заявки и событие окончания обработки события.  Event хранит информацию о событии. Она включает ид события, указатель на связанную заявку и время возникновения события. |
|  | | Transact представляет из себя заявку, которая перемещается по сети. Заявка хранит свой идентификатор. Уникальный идентификатор выдается автоматически при создании транзакта. Транзакт хранит собственные статистические данные и служебную информацию. Служебная информация — это номер узла, в котором в данный момент находится транзакт и ячейка статистической таблицы устройства с которой он связан если заявка в данный момент обрабатывается каналом.  Кроме методов получения информации транзакт также может передвигается по сети вызовом метода Move. Методы OnStart и OnStop необходимы для модификации статистических данных транзакта. В ручном режиме вызов этих методов скрыт и выполняется системой. | |
|  | | Класс, который отвечает за Узел типа Генератор заявок. Класс полностью соответствует эквиваленту из ручного режима. Используется для упрощения создания заявок и улучшает читаемость кода. Для создания требует Генератор распределения, который задает темп появления заявок. | |
|  | | Узел типа устройство. Опишем подробнее:  Capacity – текущая свободная емкость  Channel – общая емкость  Generator – генератор задержек канала  Queue – очередь устройства  Statistic и WorkStatistic – таблицы статистики канала.  Методы:  BeginCapture – начало процеса захвата канала.  Register – регистрирует пришедший транзакт  IsBusy – проверяет занято ли устройства  OnEnter – вызывается при попадании заявки в канал  OnStart – вызывается при попадании заявки в канал  OnRelease – вызывается при освобождении канала  TryCapture – попытка захвата канала.  UpdateState – записывает в статистику текущее состояние устройства.  Для правильного захвата канала сначала выхывается метод BeginCapture, которые зарегистрирует заявку и вызовет метод Trycaptyre.Если все каналы заняты, то заявка станет в очередь канала, иначе метод вызовет Caprute для захвата канала. Метод Capture вернет указатель на событие освобождение канала. Когда оно наступит, вызов метода Release правильно освободит канал и если надо займет его следующей заявкой | |
|  | | Сеть представляет собой контейнер для хранения всех устройств типа обрабатывающее и матрицы переходов.  Основной задачей сети является не только хранение данных, но и предоставление сервисных функция для диспетчеризации заявок в сети на основе матрицы переходов. | |
|  | Ключевой класс системы. Является аналогом АУМ, но в добавок к этому хранит данные о сети. Здесь также расположен генератор заявок, сеть, очередь событий, модельное время. Для пользователя доступны методы:  Start - запуск моделирования  GetCurrentTime – узнать текущее модельное время  ComputeStatistic – рассчитать статистику на основе накопленных данных  GetDesсriptor – используется стерилизаторами для сохранения модели в файл.  Закрытыми методами являются методы управления процессом моделирования, изменить которые можно только унаследованием от класса:  AddTransact – аналог метода генератора  EventEnter – обработка события первого типа  EventRelease – обработка события второго типа  HandleEvent - обработка события. Вызывает обработку нужного типа в зависимости от пришедшего события.  StartModeling – запускает цикл обработки событий. | | |
|  | Сериализатор моделей. Описывает способ записи модели в текстовое представление, и восстановление ее обратно. На основе этого класс устроена работа Генератора моделей.  Наследование от этого класса позволяет определять собственные языки описания моделей. Основные методы это Load и Save, которые соответственно загружает модель из текстового описания в объект и из объекта в текстовое описание.  Описание этого класса подробнее будет дано в следующих разделах. | | |

Диаграммы классов для ручного режима:

Несмотря на различие строения модулей с автоматическим режимом, часть функционала ручного режима соответствует автоматическому. Классы, которые соответствуют точь-в-точь на диаграммах представлены **не будут**. Их можно посмотреть в **автоматическом режиме.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | | | Предоставляет упрощенный способ задания распределений. Distrubution является словарем, который позволяет классу понимать текстовое описание. Например удобнее написать “EXP” чем ExponentialDistribution.  Метод Create получает строку, описывающую распределение и список аргументов, и возвращает созданное распределение. Иными словами этот класс это фабрика, которая конструирует распределения. По умолчанию содержит:  "EXP",IntExponentialDistribution  "NORMAL",NormalDistribution  "UNIFORM"UniformDistribution  У класс есть псевдоним Distribution  Класс Event в отличие от автоматического режима содержит статический метод, который упрощает процедуру создания события появления заявки. | |
|  | | | | | | Сеть хранит Генератор заявок, Список устройств, Терминальный узел, Модельное время, матрицу Переходов и Очередь событий. Методы аналогичны как в автоматическом режиме |
|  | | | | Класс узла типа устройство. Опишем подробнее:  Capacity – текущая свободная емкость  Channel – общая емкость  Distribution – генератор задержек канала  Queue – очередь устройства  Id – идентификатор узла. Назначается автоматически при добавлении в NodeList.  Методы:  IsBusy – проверяет занято ли устройства  выхывается метод  Capture – захват канала  Release – освобождение канала.  Перед захватом и после освобождения канала необходимо фиксировать изменения в статистической таблице | | |
|  | | | | | NodeList хранит список устройств. Предоставляет все необходимые методы для удобного добавления,получения,удаления устройств. А также автоматически назначает уникальные идентификаторы узлам. Единственное ограничение заключается в том, что сеть не должна меняться после начала процесса моделирования, так как это может нарушить индексацию. | |
|  | | | Терминальный узел. Необходим для подсчета обработанных заявок. Добавлен в ручной режим исключительно в целях большего соответствия иерархии классов тексту алгоритма. | | | |
|  | Класс Time описывает время, а точнее текущий такт. Соаместим с int. Добавлен в ручной режим исключительно в целях большего соответствия иерархии классов тексту алгоритма. | | | | | |
|  | | Очередь канала является оберткой над системными реализациями интерфейса IQueue. Эта очередь уже настроена на хранение транзактов, а конструктор позволяет при желании задать структуру, на которой будет базироваться очередь. Метод добавления упрощен. Ключ автоматически извлечется на основании данных транзакта. | | | | |
|  | | | | | | Статистическая таблица необходима для хранения и обработки накопленных данных. Вся статистика хранится в виде двух таблиц: таблице состояний каналов и таблице работы каналов. В первой отмечается изменения состояний устройств. Во второй отмечается движение заявок в устройстве. Методы:  Register регистрирует в статистике информацию о приходе заявки в канал  OnStart отмечает в статистике начало захвата канала  OnRelease отмечает в статистике освобождение канала. |

7.**Описание “ручного” режима и авто-каркаса для моделирования сетей из Библиотеки**

Ручной режим предназначен для описания сетей используя структуры данных, их методы и сервисные функции из библиотеки классов. Библиотека классов хранит реализации типовых структур и алгоритмов моделирования, что значительно упрощает процесс описания моделей на языке высокого уровня. При ручном режиме пользователь использует иерархию классов как конструктор, который обладает достаточным функционалом для поддержания работы сетей. Однако существует общая последовательность действия, которая должна выполнятся для того, чтобы собранный конструктор заработал.

1. Инициализация модели
   1. Инициализация генератора заявок(Generator)
   2. Инициализация узлов(Node)
   3. Инициализация очереди событий(EventQueue)
   4. Инициализация матрицы переходов(Matrix<double>)
   5. Инициализация сети
   6. Инициализация терминала
   7. …

Для того чтобы можно было моделировать, в первую очередь должна быть собрана структура сети. Не обязательно использовать все типовые блоки для сборки сети, но не проинициализировав один из блоков, пользователю необходимо реализовывать работу связанных блоков самому. Например, если пользователь откажется от такого компонента как генератор заявок, ему придется заботится самому о создании заявок. К счастью большинство структур данных имеют простую структуру, а исходный код открыт, что увеличивает возможности модификации системы. С другой стороны пользователь может переопределять часть функционала наследованием.

1. Загрузка модели заявками(Generator.CreateTransact())

Очевидно, модель не будет работать, если в ней нет событий. Один из способов добавить события, это загрузка модели заявками. Использование класс Generator для этих целей не панацея, так как есть возможность создавать события “руками”

1. Цикл обработки событий

Собственно, это и есть процесс моделирования. Важно следить чтобы сеть работала правильно, что не всегда бывает просто. Также необходимо позаботиться о накоплении данных в таблицу статистики, иначе зачем вообще нужно моделирование?

1. Обработка статистики

В результате моделирования мы накопим огромные массивы информации, глядя на которую могут разболеться глаза и голова. В общем случае эта информация не принесет нам пользы если ее не обработать. Система умеет обрабатывать информацию, что позволит получать из больших массивов, данных конкретные числа, описывающие результаты моделирования.

1. Печать статистики

Обработанная статистика хранится в оперативной памяти, и было бы не плохо вывести ее на экран, или сохранить в файл, или напечатать на принтере. Классы из библиотеки легко позволяют это сделать.

Используя иерархию классов можно легко строить модели. Для этого необходимо быть знакомым с документацией, знать алгоритмы управления моделированием. важно понимать, что большинству пользователей, которые на знакомы с моделированием, необходимо чтобы раз и работало. Чтобы модель запускалась, и отчетики на экран печатались, и для этого не приходилось залипать в увлекательную документацию. Для таких целей и был реализован авто-каркас. Суть авто-каркаса в том, чтобы предоставить пользователю типовой код, который будет создавать простую модель, организовывать процесс моделирования, сбора данных, обрабатывать их в и оконцовке печатать хоть в терминал, хоть в файл. Самое интересное что код можно модифицировать, и если это сделать правильно, то можно в два счета получать произвольные модели, работающие по классическому алгоритму моделирования почти также быстро как если бы оно само сделалось. Код Авто-каркаса расположен в файле carcas.cpp. Авто-каркас реализует алгоритм моделирования в полной мере, что также позволяет студентам и молодым специалистам учится азам моделирования на рабочем примере, на котором можно дергать за разные ‘рычажки’ и сразу же получать результат. Если сильно не изменять код авто-каркаса, можно даже избежать пару десятков ударов граблями по голове.

8.**Описание генератора моделей**

Авто-каркас или все писать самому это конечно же хорошо, но порой пользователю нужно за пару минут получить результаты моделирования, С++ он не знает и не хочет, Visual Studio ему не интересна. Для таких пользователей существуют такие полнофункциональные среды моделирования как например GPSS World. Но синтаксис описание моделей на этом языке для неокрепших умов выглядит отчасти пугающим. Хочется, чтобы в файле можно было построить матрицу переходов, и по-простому за пять минут описать что и как должно генерировать числа, а на выход красивый оформленный отчет. Для таких целей служит генератор моделей. Идея генератора моделей вытекает из правила Парето. 20% функционала GPSS World используется в 80% случаев, а остальное 80% в 20%. Идея упростить язык описания моделей, чтобы он умел описывать только 20% функционала, но очень просто.

Для пользователя генератор моделей — это черный ящик. На вход он отдает код модели, а на выход получает отчет. Генератор моделей выполнен в консольном варианте. Интерфейс предельно прост и понятен, а сам генератор моделей работает максимально быстро. Применялся рад оптимизаций, в том числе специализированный структуры данных. Шаблоны описания моделей могут быть доопределены пользователем, что позволит реализовать не только 20% функционала, но даже 25%. Для описания своего языка, необходимо у наследоваться от класс ModelBuilder.В таком случае придется реализовывать сериализацию и десериализацию самому.

Общий алгоритм сериализации:

1)Считать файл

2)Разбить на токены

3)Каждый токен обработать, проверить корректность

4)Полученную из токенов информацию использовать для создания объекта модели

Общий алгоритм десериализации:

1)Получить состояния сериализуемых объектов, например через метод GetDescriptor()

2)Сформировать синтаксическую структуру файла

3)Заполнить структуру данными, полученными из дескрипторов

4)Сохранить в файл

Общий алгоритм лишь дает рекомендацию к релизации, но не обязыет делать именно так. В качестве примера языка описания моделей приведем язык, который использует генератор моделей. Общая структура описания:

Количество устройств

Матрица переходов

Для каждого устройства: Закон распределения задержек узлов : аргументы

Закон распределения задержек модели : аргументы

Пример:

1

0 1 0

0 0 1

0 0 0

EXP : 10

EXP : 20

Соответствует сети:

Матрица переходов:

|  |  |  |  |
| --- | --- | --- | --- |
|  | S1 | b1 | S2 |
| S1 |  | 1 |  |
| b1 |  |  | 1 |
| S2 |  |  |  |

Параметры системы:

|  |  |
| --- | --- |
| Потоки | Устройства |
| № mт | № K mt |
| 1 20 | 1 1  10 |

Схема сети:



Пользоваться генератором моделей предельно просто если есть описание как на примере выше.

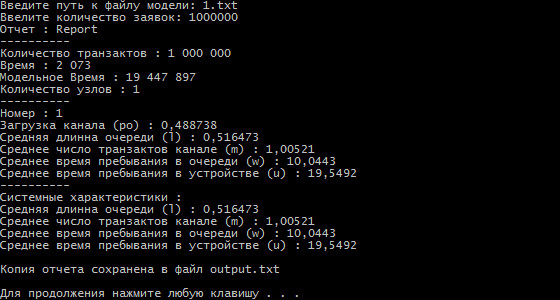


Рисунок 8.1: Пример работы генератора моделей

Идея работы генератора моделей основана на:

* Лексический разбор файла

В результате получаем промежуточное представление модели

* Конструирование модели на основе разбора

В результате получаем объект модели

* Запуск полученной модели на выполнение

В результате накапливаются данные

* Обработка статистики

В результате данные обрабатываются

* Печать отчета в файл

В результате мы видим отчет

9.**Результаты тестирования на базе библиотеки**

Библиотека тестов содержит 12 примеров сетей для тестирования и 12 отчетов по каждому из тестов для сравнения. Приведем сравнение наиболее показательных примеров:

|  |  |
| --- | --- |
| **ТЕСТ 1** | |
| Сеть |  |
| Параметры | |  |  |  |  | | --- | --- | --- | --- | | **Потоки** | **Устройства** | | | | № mт | № K mt | № K mt | № K mt | | 1 20 | 1 1 4 | 2 1 3 | 3 1 10 | |

Таблица 9.1: Описание первого теста

GPSS

|  |  |  |
| --- | --- | --- |
|  | **Характеристика** | **Значение характеристики** |
| ИМ(50000) |
| **Узел** |  |  |
| **1** | ρ1 | 0.564 |
|  | l1 | 0.507 |
|  | m1 | 1.635 |
|  | ω1 | 11.241 |
|  | u1 | 36.253 |
| **Система** |  |  |
|  | L | 0.507 |
|  | M | 1.637 |
|  | W | 11.241 |
|  | U | 57.369 |

Таблица 9.2 Прогон первого теста в GPSS

Система

|  |  |  |
| --- | --- | --- |
|  | **Характеристика** | **Значение характеристики** |
| ИМ(50000) |
| **Узел** |  |  |
| **1** | ρ1 | 0.566 |
|  | l1 | 0.53 |
|  | m1 | 1.66 |
|  | ω1 | 11.54 |
|  | u1 | 36.05 |
| **Система** |  |  |
|  | L | 0.534 |
|  | M | 1.66 |
|  | W | 11.55 |
|  | U | 57.39 |

Таблица 9.3: Прогон первого теста в системе

|  |  |
| --- | --- |
| **ТЕСТ 2** | |
| **Сеть** |  |
| **Параметры** | |  |  |  |  | | --- | --- | --- | --- | | **Потоки** | **Устройства** | | | | **№ mт** | **№ K mt** | **№ K mt** | **№ K mt** | | **1 30** | **1 1**  **4** | **2 3**  **20±14** | **3 1**  **10±8** | |
|  |  |

Таблица 9.4: Описание второго теста

GPSS

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Характе-**  **ристика** | **Значение**  **хар-ки** | **Характе-**  **ристика** | **Значение**  **хар-ки** | **Характе-**  **ристика** | **Значение хар-ки** |
| **Узел** | 1 |  | 2 |  | 3 |  |
|  | ρ1 | 0.132 | ρ2 | 0.221 | ρ3 | 0.33 |
|  | l1 | 0.02 | l2 | 0.006 | l3 | 0.093 |
|  | m1 | 0.153 | m2 | 0.668 | m3 | 0.423 |
|  | ω1 | 0.6 | ω2 | 0.178 | ω3 | 2.797 |
|  | u1 | 4.605 | u2 | 20.157 | u3 | 12.767 |
| **Система** |  |  |  | | | |
|  | R |  |
|  | L | 0.119 |
|  | M | 1.194 |
|  | W | 3.243 |
|  | U | 37.5 |

Таблица 9.5 Прогон второго теста в GPSS

Система

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Характе-**  **ристика** | **Значение**  **хар-ки** | **Характе-**  **ристика** | **Значение**  **хар-ки** | **Характе-**  **ристика** | **Значение хар-ки** |
| **Узел** | 1 |  | 2 |  | 3 |  |
|  | ρ1 | 0.12 | ρ2 | 0.223 | ρ3 | 0.33 |
|  | l1 | 0.02 | l2 | 0.0076 | l3 | 0.103 |
|  | m1 | 0.14 | m2 | 0.678 | m3 | 0.438 |
|  | ω1 | 0.61 | ω2 | 0.226 | ω3 | 3.069 |
|  | u1 | 4.16 | u2 | 20.185 | u3 | 13.03 |
| **Система** |  |  |  | | | |
|  |  |  |
|  | L | 0.131 |
|  | M | 1.256 |
|  | W | 3.905 |
|  | U | 37.38 |

Таблица 9.6: Прогон второго теста в системе

10.**Инструкция по установке, проверке, использованию ПО**

Система не имеет сложных зависимостей и ставится предельно просто. Процесс установки отсутствует как таковой. Достаточно скопировать файлы системы в рабочую папку.

Структура каталогов:

ModelRunner – содержит ModelRunner.exe. Это генератор моделей

StudyModel – содержит 65 файлов исходных кодов для использования ручного режима

И файл Karkas.cpp который содержит код авто-каркаса

Test – содержит 12 тестов и папку Report Samples который содержит 12 соответсвующих тестам отчетов.

Для проверки генератора моделей необходимо запустить ModelRunner.exe и далее следуя инструкции ввести данные, получить отчет, сравнить с эталонным если таков имеется.

Для проверки ручного режима необходима Visual Studio 13 или выше

Создайте пустой проект C++, добавьте к проекту файл Karkas.cpp, скопируйте одержимое папки StudyModel в папку с проектом.

Системные требования для ручного режима:

* Наличие Visual Studio версии 13 или выше.
* Windows 7 или выше (так как Visual Studio 13 не установится на более ранние версии ОС)
* Наличие как минимум 300 кб памяти на жестком диске(желательно 1МБ)

Для автоматического режима системные требования отсутствуют.

**11.Подведение итогов**

Мной был спроектировано, реализовано и отлажено ПО для работы с СеМО. ПО включает в себя Генератор моделей и инструмент для ручного описания моделей. Было спроектировано две системы, код одной из них закрыт, код второй открыт. В результате полученная система содержит средство для описания моделей, запуска, создания на языке С++ как в полностью ручной режиме, так и с использованием каркаса. Генератор моделей соответствует требованиям. Он требует мало ресурсов, расширяем, прост, а использовании и установке. Открытая библиотека для обучению моделированию получилось максимально прозрачной и простой, но в тоже время функциональной и расширяемой.

**12.Список используемой литературы**

1. Муравьев Г.Л., Савицкий Ю.В., Хвещук В.И. Методическое пособие "Моделирование систем. Курсовое проектирование" для студентов очного и заочного обучения по специальности АСОИ. – БрГТУ, Брест, 2003.
2. Муравьев Г.Л. Моделирование систем. Курс лекций. – БрГТУ, Брест, 2003.
3. Аверилл М. Лоу, В. Дэвид Кельтон. Имитационное моделирование. – Питер, 2004.
4. Советов, Яковлев. Моделирование систем (Книга по имитационному моделированию). – Москва: «Высшая школа», 1986

**ПРИЛОЖЕНИЕ 1: КОД АВТО-КАРКАСА**

int main()

{

//инициализация сети

/\*

Генератор -> Очередь -> Устройство -> Терминал

\*/

Generator\* generator = new Generator;

generator->Distribution = Random::Distribution::Create("EXP", { 20 });

Node\* node = new Node(1);

node->Distribution = Random::Distribution::Create("EXP", { 10 });

node->Queue = new NodeQueue;

//node->Queue = node\_queue;

//инициализация матрицы переходов

Matrix<double> transition(3,3);

transition[0][1] = 1;

transition[1][2] = 1;

Network\* network = new Network;

network->TransitionMatrix = transition;

network->Channel = new NodeList;

network->Channel->Add(node);

network->Generator = generator;

network->Terminal = new Terminal;

network->EventQueue = new EventQueue;

network->Time = new Time(0);

//инициализация таблиц статистики

StatisticTable\* Table = new StatisticTable(1);

int ammount = 50000;

//загрузка сети заявками

std::vector<Transact\*> transactList;

for (int i = 0; i < ammount; i++)

{

Transact\* t = network->Generator->CreateTransact();

transactList.push\_back(t);

Event\* e = Event::CreateFreeEvent(t,t->Statistic.Start);

network->EventQueue->Add(e);

}

while (network->EventQueue->Empty() == false)

{

network->MoveTime();

Event\* currentEvent = network->EventQueue->Remove();

//получить связанный транзакт

Transact\* transact = currentEvent->Sender;

Node\* node = nullptr;

Time time = \*network->Time;

switch (currentEvent->Id)

{

case Free:

transact->Move(network);

if (network->IsTerminator(transact->GetNode()) == true)

{

network->Terminal->Terminate(transact, time);

}

else

{

node = network->Channel->Get(transact->GetNode() - 1);

Table->Register(node, transact, time);

if (node->IsBusy() == true)

{

node->Queue->Add(transact);

}

else

{

Table->OnStart(node, transact, time);

Event\* happened = node->Capture(transact, time);

network->EventQueue->Add(happened);

}

Table->UpdateState(node, time);

}

break;

case Processed:

node = network->Channel->Get(transact->GetNode() - 1);

Table->OnRelease(node, transact, time);

node->Release();

Event\* event = Event::CreateFreeEvent(transact, time);

network->EventQueue->Add(event);

if (node->Queue->Empty() == false)

{

Transact\* newTransact = node->Queue->Remove();

Table->OnStart(node, newTransact, time);

Event\* happened = node->Capture(newTransact, time);

network->EventQueue->Add(happened);

}

Table->UpdateState(node, time);

break;

}

delete currentEvent;

}

FullStatistic state = Table->ComputeStatistic(network->Terminal);

//10) Вывод статистики на экран

Report::Locale\* loc = new Report::RussianLocale();

std::wofstream\* file = new std::wofstream("output.txt");

Report::StreamReport reporter(&std::wcout, loc);

reporter.Print(state);

file->close();

delete file;

delete loc;

**}**