Государственное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

Р А С Ч Ё Т Н О - П О Я С Н И Т Е Л Ь Н А Я З А П И С К А

к дипломной работе на тему:

Разработка системы имитационного моделирования в форме библиотеки языка Haskell

---

Студент _____ _____ **И. В. Миникс**
(Подпись, дата) (И.О.Фамилия)

Руководитель дипломной работы _____ _____ **В. П. Степанов**
(Подпись, дата) (И.О.Фамилия)

Консультант по исследовательской части __ _____ **В. П. Степанов**
(Подпись, дата) (И.О.Фамилия)

Консультант по конструкторско-технологической части __ _____ **В. П. Степанов**
(Подпись, дата) (И.О.Фамилия)

Консультант по организационно-экономической части __ _____ **О. Н. Мельников**
(Подпись, дата) (И.О.Фамилия)

Консультант по охране труда и экологии ___ _____ **О. В. Кирикова**
(Подпись, дата) (И.О.Фамилия)

Москва, 2013

Государственное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой _____
(Индекс)

_____ _____
(И.О.Фамилия)

« ___ » _____ 20 __ г.

З А Д А Н И Е
на выполнение дипломной работы

Студент __Миникс Игорь Владимирович _____
(Фамилия, имя, отчество)

Разработка системы имитационного моделирования в форме библиотеки языка Haskell _____
(Тема дипломной работы)

_____

_____

Источник тематики (НИР кафедры, заказ организаций и т.п.) __НИР кафедры_____

_____

_____

_____

Тема дипломной работы утверждена распоряжением по факультету № ____
от « ___ » _____ 20__ г.

1. Исходные данные

Техническое задание, содержащее следующие требования: _____
__- разработать библиотеку языка Haskell, позволяющую осуществлять имитационное моделирование систем массового обслуживания; _____
__- разработать механизм трансляции описания систем из формата GPSS в формат разработанной библиотеки; _____
__- обеспечить возможность распространения моделей, разработанных при помощи библиотеки, в виде самостоятельных приложений._____

_____

2. Технико-экономическое обоснование

Существующие системы имитационного моделирования либо являются коммерческими, либо имеют серьезные функциональные ограничения (ограничено максимальное число используемых блоков, поддерживаются не все операционные системы и др.) _____

_____

_____

_____

_____

(обзор и анализ альтернативных решений; выбор вариантов для сравнения;
конкретные улучшаемые характеристики или параметры; возможный технико-экономический эффект и т.п.)

3. Научно-исследовательская часть

Сравнить характеристики системы массового обслуживания, полученные теоретически, с помощью разработанного ПО и с помощью одной из существующих реализаций GPSS. _____

_____

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

4. Проектно-конструкторская часть

Определить синтаксис описания систем массового обслуживания. Разработать структуры данных для хранения описания систем и методы и алгоритмы имитации и определения характеристик систем. _____

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

5. Технологическая часть

Осуществить выбор конкретных технологий и сторонних библиотек, позволяющих реализовать спроектированную библиотеку. Провести тестирование разработанного ПО на предмет корректности его работы и соответствия заданию._____

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

6. Организационно-экономическая часть

_____

_____

_____

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

7. Охрана труда и экология

_____

_____

_____

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

8. Оформление дипломной работы

8.1. Расчетно-пояснительная записка на ____ листах формата А4.

8.2. Перечень графического материала (плакаты, схемы, чертежи и т.п.) _____

_____

_____

Дата выдачи задания « ___ » _____ 20__ г.

В соответствии с учебным планом дипломную работу выполнить в полном объеме в срок до « ___ » _____ 20__ г.

Руководитель дипломной работы _____ _____
(Подпись, дата) (И.О.Фамилия)

Студент _____ _____
(Подпись, дата) (И.О.Фамилия)

Примечание:

1. Задание оформляется в двух экземплярах; один выдаётся студенту, второй хранится на кафедре.

Государственное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой _____
(Индекс)

_____ _____
(И.О.Фамилия)

« ___ » _____ 20 __ г.

К А Л Е Н Д А Р Н Ы Й П Л А Н
выполнения дипломной работы

Студент _Миникс Игорь Владимирович _____
(Фамилия, имя, отчество)

_Разработка системы имитационного моделирования в форме библиотеки языка Haskell_____
(Тема дипломной работы)

_____

_____

№ п/п	Наименование этапов дипломной работы	Выполнение этапов		Примечание
		Срок	Объем, %	
1.	Разработка структур данных и выбор методов и алгоритмов.	17.02.2012	15%	
2.	Определение синтаксиса описания систем.	24.02.2012	20%	
3.	Написание программной части.	31.03.2012	55%	
4.	Тестирование и отладка.	7.04.2012	60%	
5.	Исследовательская часть.	14.04.2012	65%	
6.	Подготовка расчетно-пояснительной записки.	30.04.2012	80%	
7.	Оформление организационно-экономической и экологической части.	12.05.2012	85%	
8.	Оформление графической части.	19.05.2012	90%	
9.	Подготовка к защите.	26.05.2012	100%	

Руководитель дипломной работы _____ _____
(Подпись, дата) (И.О.Фамилия)

Студент _____ _____
(Подпись, дата) (И.О.Фамилия)

Содержание

Введение

Зародившаяся в начале прошлого века с целью упорядочить работу телефонных станций, теория массового обслуживания нашла применения в моделировании самых разнообразных систем, таких как системы связи, обработки информации, снабжения, производства и др.

Несмотря на имеющиеся достижения в области математического исследования характеристик систем массового обслуживания, наиболее универсальным подходом по прежнему остается имитационное моделирование.

Язык имитационного моделирования GPSS (General Purpose Simulation System — система моделирования общего назначения) создан специально для моделирования систем массового обслуживания и на данный момент является доминирующим в этой области. Однако, существующие версии систем имитационного моделирования на основе языка GPSS либо слишком дороги, либо ограничены в возможностях и не позволяют провести все необходимые исследования.[1] Помимо этого, на данный момент затруднено интегрирование моделей, разработанных при помощи GPSS в другие программные средства (например, в целях оптимизации параметров исследуемой системы).

Целью данной работы является создание системы имитационного моделирования, основанной на принципах и синтаксисе GPSS, однако позволяющей разрабатывать модели как часть более крупной программы.

В качестве языка разработки используется Haskell. Haskell является динамично развивающимся функциональным языком программирования, который получает все больше сторонников во всем мире, в том числе и в России. [2]. Для Haskell характерны строгая статическая типизация, модульность, строгое разделение функций на чистые и не чистые, ленивые вычисления, функции высших порядков и др.[3] Помимо этого использование языка Haskell позволит производить описание систем при помощи синтаксиса схожего с синтаксисом GPSS, при этом разработанные модели будут являться объектами первого класса, что позволит, например, передать модель как параметр в функцию оптимизации.

Для достижения поставленной цели необходимо решить следующие задачи:

— изучить принципы функционирования и синтаксис описания моделей в GPSS;

— разработать синтаксис описания моделей схожий с синтаксисом GPSS, но при этом позволяющий составлять модели в виде функций языка Haskell;

— выбрать подмножество блоков GPSS, которые следует реализовать в системе;

— реализовать алгоритмы описания моделей и имитационного моделирования;

— провести тестирование разработанного программного обеспечения;

— провести моделирование некоторой эталонной системы массового обслуживания в разработанной системе и аналитически и убедиться в совпадении полученных результатов.

1 Аналитический раздел

В данном разделе проводится обзор принципов функционирования и синтаксиса системы GPSS, а также производится выбор подмножества возможностей GPSS, которые следует реализовать в разрабатываемой системе.

1.1 Краткий обзор GPSS

GPSS стал одним из первых языков моделирования, облегчающих процесс написания имитационных программ. Он был создан в виде конечного продукта Джефри Гордоном в фирме IBM в 1962 г.[4] В свое время он входил в десятку лучших языков программирования и по сей день широко используется для решения практических задач. Наиболее современной версией GPSS для персональных компьютеров на данный момент является пакет GPSS World, разработанный компанией Minuteman Software.

Описание системы на GPSS представляет собой последовательность блоков, каждый из которых соответствует некоторому оператору (подпрограмме). Каждый блок имеет определенное количество параметров (полей).

Основой имитационных алгоритмов GPSS является дискретно-событийный подход — моделирование системы в дискретные моменты времени, когда происходят события, отражающие последовательность изменения состояний системы во времени.[4]

1.2 Объекты языка GPSS

Основными Объектами языка GPSS являются транзакты и блоки, которые отображают соответственно динамические и статические объекты моделируемой системы.

Транзакты — динамические элементы GPSS-модели. В реальной системе транзактам могут соответствовать такие элементы как заявка, покупатель автомобиль и др. Состояние транзакта в процессе моделирования характеризуется следующими атрибутами:

а) параметры — набор значений связанных с транзактом. Каждый транзакт может иметь произвольное число параметров. Каждый параметр имеет уникальный номер, по которому на него можно сослаться;

б) приоритет — определяет порядок продвижения транзактов при конкурировании за общий ресурс;

в) текущий блок — номер блока, в котором транзакт находится в данный момент;

г) следующий блок — номер блока, в который транзакт попытается войти;

д) время появления транзакта — момент времени в который транзакт был создан;

е) состояние — состояние, показывающее в каких списках транзакт находится в данный момент. Транзакт может находиться в одном из следующих состояний:

1) активен — транзакт находится в списке текущих событий и имеет наивысший приоритет;
2) приостановлен — транзакт находится в списке будущих событий либо в списке текущих событий, но с меньшим приоритетом;
3) пассивен — транзакт находится в списке прерываний, списке синхронизации, списке блокировок или списке пользователя;
4) завершен — транзакт уничтожен и больше не участвует в модели.

Диаграмма состояний транзакта показана на Рисунке 1.1.

Блоки — статические элементы GPSS-модели. Модель в GPSS может быть представлена как диаграмма блоков, т.е. ориентированный граф, узлами которого являются блоки, а дугам — направления движения транзактов. с каждым блоком связано некоторое действие, изменяющее состояние прочих элементов модели. Транзакты проходят блоки один за другим, до тех пор пока не достигнут блока TERMINATE. В ряде случаев транзакт может быть остановлен в одном из блоков до наступления некоторого события.

Помимо транзактов и блоков в GPSS используются следующие объекты: устройства, многоканальные устройства (хранилища, памяти), ключи, очереди, списки пользователя и др.

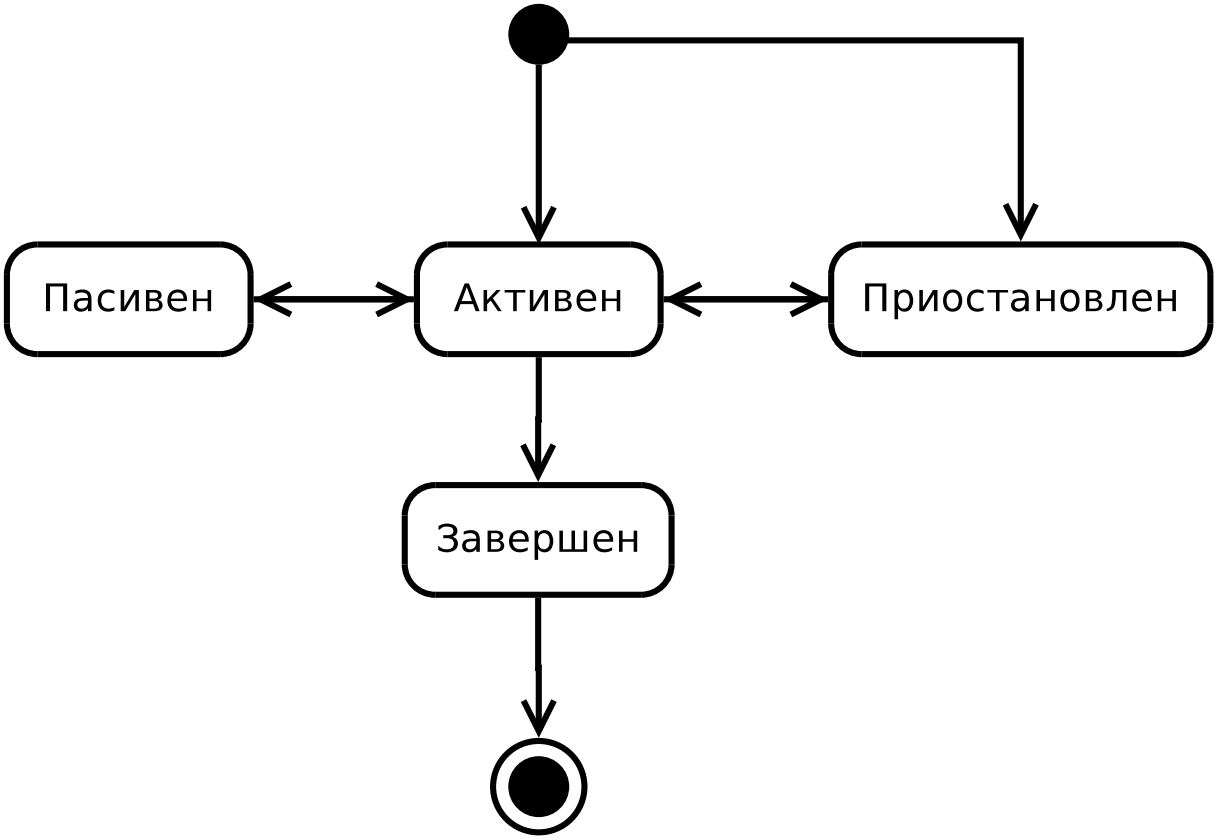

Рисунок 1.1 — Състояния транзакта

1.3 Управления процессом моделирования в GPSS

В системе GPSS интерпретатор поддерживает сложные структуры организации списков (см. Рисунок 1.2).[4] Два основных из них — список текущих событий (СТС) и список будущих событий (СБС).

В СТС входят все события запланированные на текущий момент модельного времени. Интерпретатор в первую очередь просматривает этот список и перемещает по модели те транзакты, для которых выполнены все условия. Если таких транзактов в списке не оказалось интерпретатор обращается к СБС. Он переносит все события, запланированные на ближайший момент времени и вновь возвращается к просмотру СТС. Перенос также осуществляется в случае совпадения текущего момента времени с моментом наступления ближайшего события из СБС.

В целях эффективной организации просмотра транзактов, движение которых заблокировано (например, из-за занятости некоторого ресурса), используются следующие вспомогательные списки:

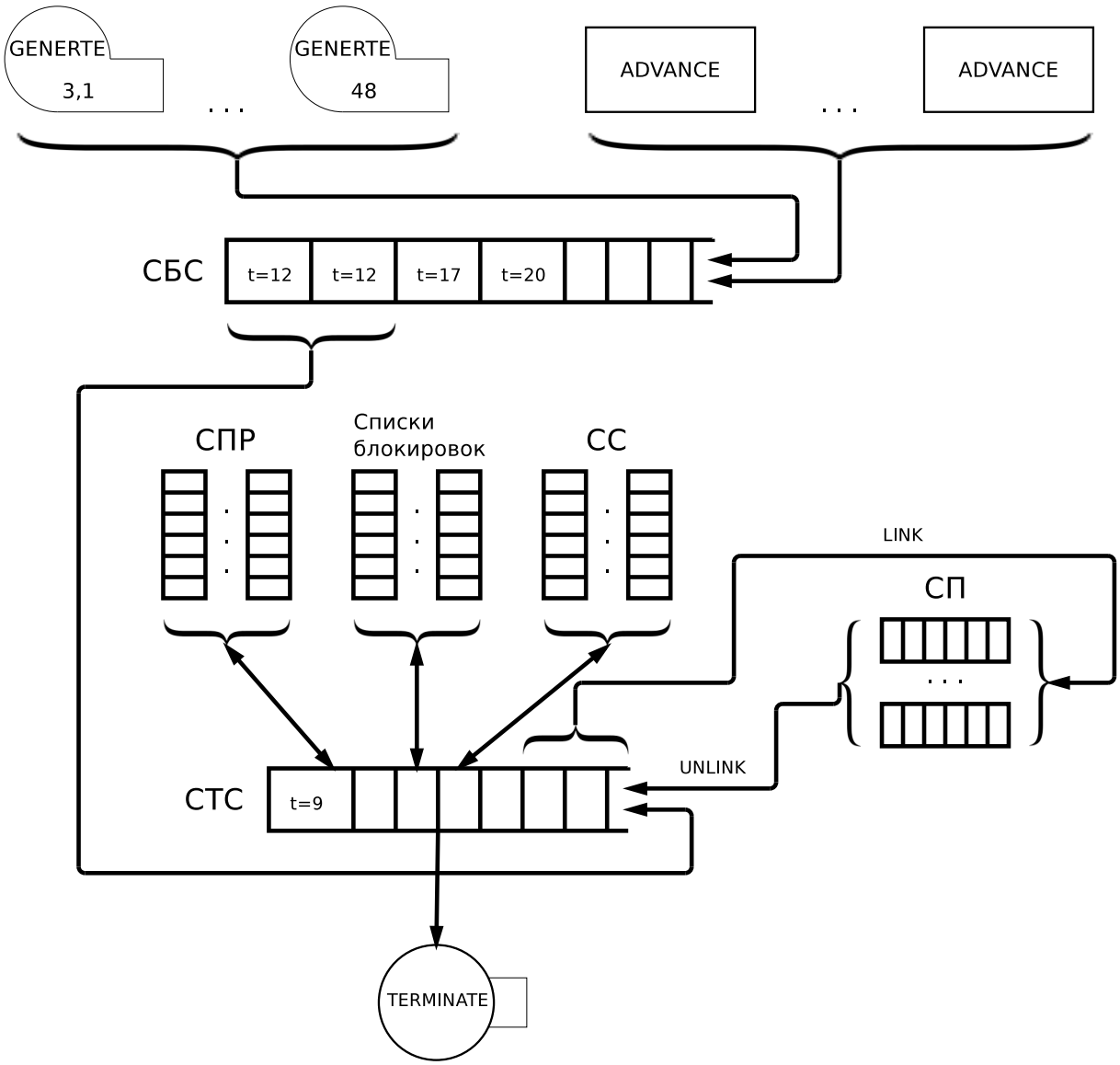

Рисунок 1.2 — Списки GPSS

— списки блокировок — списки транзактов, которые ожидают освобождения некоторого ресурса;

— список прерываний — содержит транзакты, прерванные во время обслуживания. Используется для организации обслуживания одноканальных устройств с абсолютным приоритетом;

— списки синхронизации — содержат транзакты одного семейства (созданные блоком SPLIT), которые ожидают синхронизации в блоках (MATCH, ASSEMBLE или GATHER);

— списки пользователя — содержат транзакты, выведенные пользователем из СТС с помощью блока LINK. Транзакты могут быть возвращены в СТС помощью блока UNLINK.

1.4 Выбор подмножества реализуемых блоков

В современной версии языка GPSS (входящей в пакет GPSS World) поддерживается 53 различных блока.[5] В рамках данной работы не представляется возможным реализовать аналоги каждого из них. Поэтому следует выделить некоторое подмножество блоков, которое с одной не будет слишком обширным, а с другой — позволит решать практические или по крайней мере учебные задачи.

В качестве примера рассмотрим задачу из курса Модели оценки качества аппаратно программных комплексов:

В вычислительной системе, содержащей N процессоров и M каналов обмена данными, постоянно находятся K задач. Разработать модель, оценивающую производительность системы с учетом отказов и восстановлений процессоров и каналов. Имеется не более L ремонтных бригад, которые ремонтируют отказывающие устройства с бесприоритетной дисциплиной. Интенсивность отказов, восстановлений, средние времена обработки сообщения и среднее время обдумывания также известны.

Схема модели данной системы показана на Рисунке 1.3

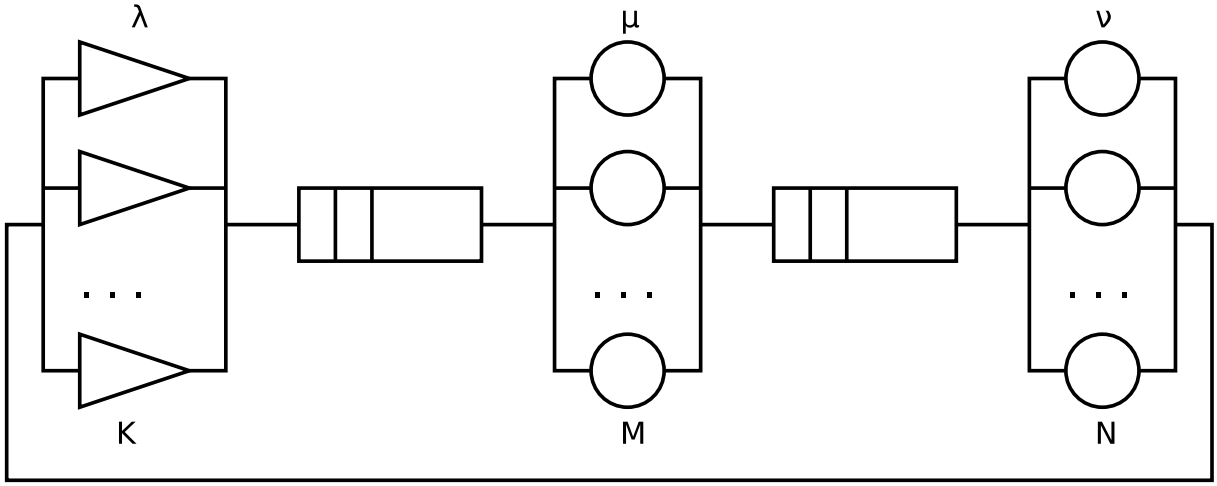

Рисунок 1.3 — Схема моделируемой системы

Как и подавляющее большинство других задач, данная задача, безусловно, не может быть решена без использования блоков GENERATE, TERMINATE и ADVANCE. Так как моделируемая система является замкнутой, при описании модели не обойтись без блока TRANSFER.

К сожалению, не представляется возможным реализовать процессоры и каналы как многоканальные устройства, т.к. многоканальные устройства в GPSS не поддерживают абсолютные приоритеты и не позволяют смоделировать выход из строя отдельных каналов устройства. Однако, требуемую систему можно описать при помощи множества одноканальных устройств и блока TRANSFER в режиме ALL. Таким образом, также понадобятся блоки SEIZE и RELEASE. Для моделирования отказов устройств можно воспользоваться блоками FAVAIL и FUNAVAIL либо блоками PREEMPT и RETURN.

Наконец, доступные ремонтные бригады можно смоделировать с помощью многоканального устройства. Соответственно, понадобятся блоки ENTER и LEAVE.

Приблизительная модель системы показана в Листинге 1.1

Листинг 1.1 — Приблизительная модель системы

```
1  ;Доступное число ремонтных бригад
2  REPAIRERS STORAGE 5
3  ;Общее время моделирования
4  GENERATE ,,,1
5  ADVANCE 1000
6  TERMINATE 1
7
8  ;Основная часть модели
9  GENERATE ,,,10
10
11 ;Фаза обдумывания
12 LUSER ADVANCE 4,1
13       TRANSFER ALL,LCPU1,LCPUN,4
14
15 ;Первая фаза обработки
16 LCPU1 SEIZE CPU1
17       ADVANCE 2,1
18       RELEASE CPU1
19       TRANSFER ,LPHASE2
20 ...
21 LCPUN SEIZE CPUN
22       ADVANCE 2,1
23       RELEASE CPUN
24       TRANSFER ,LPHASE2
25
26 LPHASE2 TRANSFER ALL,LCHAN1,LCHANM,4
27 ;Вторая фаза обработки
```

```
28 LCHAN1  SEIZE  CHAN1
29           ADVANCE  2,1
30           RELEASE  CHAN1
31           TRANSFER  ,LUSER
32  ...
33 LCHANM  SEIZE  CHANM
34          ADVANCE  2,1
35          RELEASE  CHANM
36          TRANSFER  ,LUSER
37
38  ;Моделирование отказов и восстановлений.
39               GENERATE  ,,,1,10
40 CPUBROKE1  ADVANCE  20,4
41               PREEMPT  CPU1
42               ENTER  REPAIRERS
43               ADVANCE  10,4
44               LEAVE  REPAIRERS
45               RETURN  CPU1
46               TRANSFER  ,CPUBROKE1
47  ...
48               GENERATE  ,,,1,10
49 CPUBROKEN  ADVANCE  20,4
50               PREEMPT  CPUN
51               ENTER  REPAIRERS
52               ADVANCE  10,4
53               LEAVE  REPAIRERS
54               RETURN  CPUN
55               TRANSFER  ,CPUBROKEN
56
57                GENERATE  ,,,1,10
58 CHANBROKE1  ADVANCE  20,4
59                PREEMPT  CHAN1
60                ENTER  REPAIRERS
61                ADVANCE  10,4
62                LEAVE  REPAIRERS
63                RETURN  CHAN1
64                TRANSFER  ,CHANBROKE1
65  ...
66                GENERATE  ,,,1,10
67 CHANBROKEN  ADVANCE  20,4
68                PREEMPT  CHANM
69                ENTER  REPAIRERS
70                ADVANCE  10,4
71                LEAVE  REPAIRERS
72                RETURN  CHANM
73                TRANSFER  ,CHANBROKEM
```

Таким образом, разрабатываемая система имитационного моделирования должна поддерживать аналоги по крайней мере следующих блоков: ADVANCE, ENTER, GENERATE, LEAVE, PREEMPT, RELEASE, RETURN, SEIZE, TERMINATE и TRANSFER.

1.5 Описание выбранных блоков

Ниже представлено описание выбранных блоков в соответствии со справочным руководством GPSS World.[5]

ADVANCE A,B

Блок ADVANCE осуществляет задержку продвижения транзактов на заданный промежуток времени.

A — Среднее время задержки. Не обязательный параметр. Значение по умолчанию — 0.

B — Максимально допустимое отклонение времени задержки либо функция-модификатор.

ENTER A,B

При входе в блок ENTER транзакт либо занимает заданное количество каналов указанного многоканального устройства либо блокируется до его освобождения.

A — Имя или номер многоканального устройства. Обязательный параметр.

B — Число требуемых каналов. Не обязательный параметр. Значение по умолчанию — 1.

GENERATE A,B,C,D,E

Блок GENERATE предназначен для создания новых транзактов.

A — Среднее время между генерацией последовательных заявок. Не обязательный параметр.

B — Максимальное допустимое отклонение времени генерации либо функция-модификатор. Не обязательный параметр.

C — Задержка до начала генерации первого транзакта. Не обязательный параметр.

D — Ограничение на максимальное допустимое число созданных транзактов. Не обязательный параметр. Пол умолчанию ограничение отсутствует.

E — Уровень приоритета создаваемых заявок. Не обязательный параметр. Значение по умолчанию — 0.

LEAVE A,B

При входе в блок LEAVE транзакт освобождает заданное число каналов указанного многоканального устройства.

A — Имя или номер многоканального устройства. Обязательный параметр.

B — Число требуемых каналов. Не обязательный параметр. Значение по умолчанию — 1.

PREEMPT A,B,C,D,E

Блок PREEMPT подобен блоку SEIZE и вошедший в него транзакт также пытается занять указанное одноканальное устройство. Однако, данный блок позволяет транзакту занять устройство, даже если в данный момент оно уже занято другим транзактом, при соблюдении ряда условий, определяемых параметрами блока.

A — Имя или номер одноканального устройства. Обязательный параметр.

B — задает режим работы блока. PR — режим приоритетов. По умолчанию — режим прерываний. В режиме прерываний транзакт может вытеснить из устройства любой другой транзакт, если тот в свою очередь не захватил устройство через блок PREEMPT. В режиме приоритетов транзакт может вытеснить любой транзакт с меньшим приоритетом.

C — задает номер блока, куда будет направлен транзакт вытесненный и устройства в результате действия блока PREEMPT.

D — задает номер параметра вытесненного транзакта, в котором будет сохранено время, которое осталось транзакту до окончания обработки а устройстве.

E — задает режим удаления вытесненного транзакта. RE — вытесненное сообщение удаляется из устройства и более не претендует на владение им. Требует обязательного указания параметра C. Значение по умолчанию — вытесненный транзакт будет вновь пытаться занять устройство.

RELEASE A

Блок RELEASE освобождает одноканальное устройство.

A — Имя или номер одноканального устройства. Обязательный параметр.

RETURN A

Блок RELEASE освобождает одноканальное устройство.

A — Имя или номер одноканального устройства. Обязательный параметр.

SEIZE A

При входе в блок SEIZE транзакт занимает указанное одноканальное устройство либо блокируется до его освобождения.

A — Имя или номер одноканального устройства. Обязательный параметр.

TERMINATE A

Блок TERMINATE завершает поступивший в него транзакт. И опционально уменьшает счетчик завершенных транзактов. Когда счетчик достигает нуля имитация останавливается.

A — Значение, на которое следует уменьшить счетчик завершенных транзактов. Не обязательный параметр. Значение по умолчанию — 0.

TRANSFER A,B,C,D

Блок TRANSFER является основным средством позволяющим изменить маршрут транзакта и перенаправить его к произвольному блоку модели. Параметр А определяет режим работы блока. Смысл остальных параметров меняется в зависимости от выбранного режима.

Безусловный режим . Если параметр А пропущен, то блок TRANSFER работает в безусловном режиме. Входящий в блок TRANSFER транзакт переходит к блоку, указанному в поле В.

Статистический режим . Параметр А является числом от 0 до 1, показывающим какая доля транзактов перейдет к блоку, указанному в параметре С. Остальные транзакты переходят к блоку,ь указанному в параметре В.

Режим BOTH . Если параметр А равен BOTH, то блок TRANSFER работает в одноименном режиме.В этом режиме каждый входящий транзакт сначала пытается перейти к блоку, указанному в поле В. Если это сделать не удается, транзакт пытается перейти к блоку, указанному в поле С. Если транзакт не сможет перейти ни к тому, ни к другому блоку, он остается в блоке TRANSFER и будет повторять в том порядке попытки перехода при каждом просмотре списка текущих событий, до тех пор, пока не сможет выйти из блока TRANSFER.

Режим ALL . Если параметр А равен ALL, то блок TRANSFER работает в одноименном режиме.В этом режиме каждый входящий транзакт1 прежде всего пытается перейти к блоку, указанному в поле В. Если транзакт не может войти в этот блок, то последовательно проверяются все блоки в определенном ряду в поисках первого, способного принять это сообщение, включая последний блок, указанный операндом С. Номер каждого проверяемого блока вычисляется как сумма номера предыдущего блока и шага, заданного операндом D. По умолчанию значение операнда D принимается равным 1.

Режим PICK . Если параметр А равен PICK, то блок TRANSFER работает в одноименном режиме. Этот режим подобен режиму ALL, за тем

исключением, что блок назначения выбирается случайным образом с одинаковой вероятностью.

1.6 Выводы

Был проведен обзор устройства системы GPSS и осуществлен выбор подмножества блоков, необходимых для моделирования не сложных систем массового обслуживания. Представлено описание назначения и параметров каждого из выбранных блоков.

2 Конструкторский раздел

В данном разделе проводится выбор синтаксиса описания моделей в разрабатываемой системе, описываются алгоритмы и структуры данных, используемые при формировании моделей и непосредственно при моделировании, а также проводится построение аналитической и имитационной модели учебной системы описанной в предыдущем разделе.

2.1 Требования к синтаксису

Синтаксис разрабатываемой системы должен быть, на сколько это возможно, схож с синтаксисом системы GPSS.

Программа на языке GPSS представляет из себя последовательность операторов, каждый из которых описывает тот или иной элемент модели (функцию, блок, устройство и др.). Этот подход естественен для императивных языков программирования, в которых программа является последовательностью команд, меняющих состояние программы. Однако Haskell относится к категории функциональных языков, программы на которых описываются как функции, значение которых вычисляется. При этом нет фиксированной, заданной программистом, последовательности операций, которые должны быть выполнены для достижения результата.

Тем не менее, в языке Haskell предусмотрен механизм, позволяющий описать конкретную последовательность вычислений — монады. В сочетании с так называемой do-нотацией, этот механизм позволит проводить описание моделей на Haskell, используя синтаксис схожий с GPSS.

2.2 Монады

Понятие монады в языке Haskell основано на теории категорий. В рамках данной теории монада может быть определена (не вполне строго) как моноид в категории эндофункторов. Однако для практического использования этого понятия в рамках языка Haskell можно обойтись менее формальным определением.

В соответствии с [3] монада — это контейнерный тип данных (то есть такой, который содержит в себе значения других типов), представляющий собой экземпляр класса Monad определенного в модуле Prelude.

Под классом в Haskell, понимается не тип данных, как в объектно-ориентированных языках, а набор методов (функций), которые применимы для работы с теми или иными типами данных, для которых объявлены экземпляры заданных классов. Наиболее близким аналогом классам в Haskell являются интерфейсы в таких языках как Java или C#. Более точно их следует называть классами типов, но т.к. в данной работе используется исключительно функциональная парадигма, в дальнейшем для краткости они будут называться просто классами.

Значения монад можно воспринимать, как значения м некоторым дополнительным контекстом. В случае монады Maybe значения обладают дополнительным контекстом того, что вычисления могли закончиться неуспешно. Монада IO добавляет контекст, указывающий что получение значений связано с действиями ввода/вывода и потому не является детерминированным и может иметь побочные эффекты. В случае списков (которые также являются монадой) контекстом является то, что значение может являться множественным или отсутствовать.

Класс Monad определен в модуле Prelude следующим образом:

Листинг 2.1 — Класс Monad

```
1 class Monad m where
2     return :: a -> m a
3     (>>=)  :: m a -> (a -> m b) -> m b
4     (>>)   :: m a -> m b -> m b
5     fail   :: String -> m a
6
7     p >> q = p >>= \_ -> q
8     fail s = error s
```

Функция **return**[1] преобразует переданное ей значение типа a в монадическое значение типа m a. Другими словами она помещает значение в

[1]Следует отметить, что название return никак нельзя назвать удачным, так как оно неизбежно вызывает ассоциации с одноименным оператором из многих императивных языков программирования, на которые она не похожа ничем кроме названия. Данная функция не завершает выполнение функции, а лишь оборачивает переданное значение в монаду.

некоторый контекст по умолчанию, в зависимости от выбранной монады. Для списка это будет список из одного элемента, для монады IO — действие ввода вывода, всегда возвращающее заданное значение и не имеющее побочных эффектов и т.д.

Функция >>= определяет операцию связывания. Она принимает монадическое значение и передает его функции, которая принимает обычное значение и возвращает монадическое. При этом сохраняется накопленный контекст и к нему добавляется новый, полученный в результате выполнения функции.

Функция >> также предназначена для связывания и используется в тех случаях, когда переданное монадическое значение не представляет интереса, а значение имеет только переданный с ним контекст вычислений. Для этой функции в классе определена реализация по умолчанию, по этому в большинстве случаев при определении экземпляра класса Monad в явном виде ее не реализуют.

Функция **fail** никогда не вызывается программистом явным образом и предназначена для обработки неуспешного окончания вычислений при сопоставлении с образцом в do-нотации, что позволяет избежать аварийного завершения программы и вернуть неудачу в контексте текущей монады.

2.3 Нотация do

Так как монады находят крайне широкое применение в программах на языке Haskell (в первую очередь, без использования монады IO невозможно осуществить ввод/вывод), в синтаксис языка было добавлено специальное ключевое слово **do**, призванное упростить написание монадических функций, сделать их более читаемыми и избавить от излишнего «синтаксического мусора».

Если в коде программы встречается конструкция с ключевым словом **do**, то транслятор выполняет следующие преобразования[1]:

[1]В приведенных преобразованиях используются управляющие символы **;**, { и }, хотя в реальных программах на языке Haskell их можно встретить довольно редко. Это связано с тем, что в Haskell используется так называемый «двумерный синтаксис»: при правильной расстановке отступов, транслятор самостоятельно расставляет точки с запятой и фигурные скобки и в большинстве случаев нет смысла загромождать ими исходный код. Тем не менее в случае необходимости их можно расставить и явным образом.

```
1. do {e} → e
2. do {e; es} → e >> do {es}
3. do {let decls; es} → let decls in do {es}
4. do {p <- e; es} → let ok p = do {es}
                         ok _ = fail "..."
                     in e >>= ok
```

При помощи нотации **do** приведенный ниже фрагмент кода

```
foo :: Maybe String
foo = Just 3 >>= (\x -> Just "!" >>= (\y -> Just (show x ++ y)))}
```

может быть записан в следующей более читаемой форме:

```
foo :: Maybe String
foo = do x <- Just 3
         y <- Just "!"
         return (show x ++ y)
```

2.4 Монада State

Часто в процессе вычислений возникает необходимость хранить и изменять некоторое состояние, в зависимости от которого результат вычислений может меняться. Haskell является чистым функциональным языком программирования функции должны быть детерминированы и не иметь побочных эффектов, поэтому текущее состояние обычно передается в функции как еще один параметр, а возвращает функция пару из собственно результата и обновленного состояния.

Для того, чтобы упростить написание функций оперирующих некоторым состоянием в Haskell была введена монада **State**. Она определена в модуле **Control.Monad.State** следующим образом:

```
newtype State s a = State {runState :: s -> (a, s)}

instance Monad (State s) where
```

```
return x = State $ \s -> (x,s)
(State h) >>= f = State $ \s -> let (a, newState) = h s
                                    (State g) = f a
                                in g newState
```

Функция **return** создает вычисление с состоянием, которое всегда возвращает один и тот же результат и оставляет переданное в него состояние без изменений. Функция $>>=$ «склеивает» два вычисления с состоянием так, что конечное состояние первого становится начальным для второго, а результат и конечное состояние второго вычисления становятся также результатом и конечным состоянием итогового, составного вычисления.

Помимо этого для работы с монадой **State** используются две вспомогательные функции **put** и **get**. Функция **put** является вычислением, которое устанавливает состояние в заданное значение не зависимо от его предыдущего значения и не возвращает никакого результата (точнее возвращает кортеж нулевой длины ()). Функция **get** возвращает текущее состояние и оставляет его без изменений.

2.5 Описание модели как вычисление с состоянием

Описание модели на языке GPSS представляет из себя последовательность блоков. В Haskell такое описание удобно представить как последовательность функций, каждая из которых добавляет к уже сформированной модели очередной блок.

Такой процесс удобно представить как вычисление с состоянием. Каждая функция, формирующая блок, помимо параметров самого блока должна принимать текущее состояние — список уже сформированных к данному моменту боков в порядке их формирования. В качестве результата функция возвращает новое состояние — модель к которой добавлен только что сформированный блок (см. Рисунок 2.1).

Для реализации такого механизма целесообразно воспользоваться монадой **State**, что позволит скрыть явную передачу состояния от одной

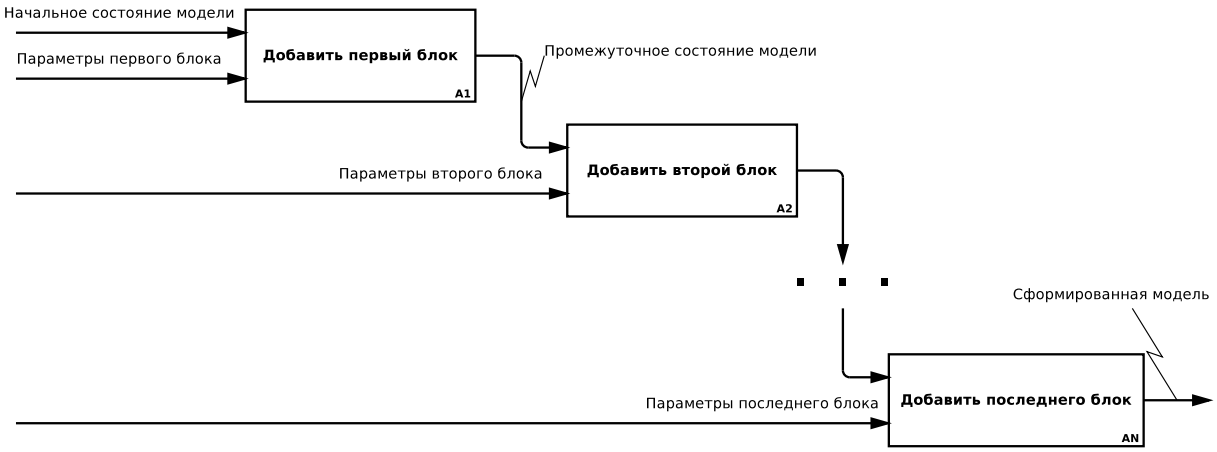

Рисунок 2.1 — Процесс формирования модели

функции к другой. А использование нотации **do** сделает описание модели почти идентичным синтаксису GPSS:

```
model =
    do generate (10,2)
       advance 3
       terminate 1
```

2.6 Функции формирования блоков

В языке GPSS имена всех блоков пишутся с заглавной буквы. Параметры отделяются от имени блока пробелом и разделяются запятыми. Синтаксис Haskell не позволяет в точности повторить эти соглашения. Имена функций в Haskell обязаны начинаться со строчной буквы. Параметры функций обычно разделяются пробелами и не берутся в скобки (каррированные функции) либо заключаются в скобки и разделяются запятыми (не каррированные функции)[1]. Оба варианта описания параметров одинаково близки к синтаксису GPSS и можно выбрать любой из них, однако для второго варианта существенно легче реализовать перегрузку функций.

[1]Строго говоря все функции в Haskell принимают ровно один параметр. Функции от N параметров на самом деле принимают один параметр и возвращают функцию от N-1 параметра (каррированные функции) либо принимают параметр-кортеж (некаррированные).

2.7 Состояние транзакта

Основным объектом в процессе моделирования является транзакт. Моделирование представляет собой передвижение транзактов от блока к блоку, в процессе которого могут изменяться состояния тех или иных объектов системы (обслуживающих аппаратов, очередей, хранилищ).

Параметры, определяющие состояние транзакта приведены в таблице 2.1

Таблица 2.1 — Параметры состояния транзакта

Имя параметра	Тип	Описание
currentBlock	Int	Номер блока модели, в котором в данный момент находится транзакт.
nextBlock	Int	Номер блока в который транзакт попытается перейти.
priority	Int	Приоритет транзакта.
params	IntMap Double	Массив параметров транзакта.
ownership	String	Имя устройства, на котором в данный момент обрабатывается транзакт.

Также как часть состояния транзакта моет рассматриваться информация о том, в каком из глобальных или локальных списков событий он находится в данный момент. Так как в каждый момент транзакт должен находиться только в одном из списков, хранить ту информацию в отдельном поле не целесообразно.

2.8 Состояния объектов системы

В процессе перемещения по блокам транзакты изменяют состояния других объектов системы. Эти состояния во-первых, в свою очередь, оказывает влияние на движение транзактов, а во-вторых предназначено для сбора статических данных в процессе моделирования.

Параметры состояний обслуживающих аппаратов, хранилищ и очередей приведены в таблицах 2.2, 2.3 и 2.4 соответственно.

Таблица 2.2 — Параметры состояния обслуживающего аппарата

Имя параметра	Тип	Описание
idAvailable	Bool	Флаг, показывающий доступно или занято в данный момент обслуживающее устройство
toInterupted	Bool	Флаг, показывающий, захватил ли, обрабатывающийся в данный момент транзакт, устройство обычным образом или путем вытеснения обрабатывавшегося до этого транзакта.
captureCount	Int	Счетчик, показывающий сколько раз было захвачено данной устройство.
captureTime	Double	Суммарное время, в течение которого устройство было занято.
lastCaptureTime	Double	Момент времени, когда устройство было захвачено в последний раз.
utilization	Double	Процент времени, в течении которого устройство было занято.
ownerPriority	Int	Приоритет транзакта, обслуживающегося на устройстве в данный момент.
dc	[Transaction]	Список заявок, ожидающих освобождения устройства.
ic	[(Maybe Double, Transaction)]	Список заявок, вытесненных с устройства и ожидающих его освобождения для продолжения обслуживания.
pc	[Transaction]	Список заявок, не сумевших вытеснить, обрабатываемый в данный момент транзакт, и ожидающих освобождения устройства.

Таблица 2.3 — Параметры состояния хранилища

Имя параметра	Тип	Описание
capacity	Int	Емкость хранилища.
unused	Int	Число доступных единиц ресурса в хранилище.
avgInUse	Double	Среднее число занятых единиц ресурса.
useCount	Int	Число захватов ресурса за время моделирования.
lastMod	Double	Момент последнего захвата или освобождения ресурса.
utilization	Double	Средний процент захваченных единиц ресурса.
maxInUse	Int	Максимальное число одновременно захваченных единиц ресурса.
dc	[Transaction]	Список заявок, ожидающих освобождения достаточного числа ресурсов хранилища.

Таблица 2.4 — Параметры состояния очереди

Имя параметра	Тип	Описание
currentContent	Int	Число заявок в очереди в данный момент модельного времени.
maximumContent	Int	Максимальное число заявок в очереди за все время моделирования.
averageContent	Double	Среднее число заявок в очереди.
lastChangeTime	Double	Момент времени, когда в последний раз заявка встала в очередь или покинула ее.

2.9 Состояние системы в целом

В процессе имитационного моделирования система последовательно переходит из одного состояния в другое, до тех пор, пока не будет достигнуто некоторое условие остановки моделирования. В данном случае моделирование происходит до тех пор, пока в блоках TERMINATE не будет завершено заданное число транзактов.

Состояние моделируемой системы может быть описано параметрами, указанными в таблице 2.5. Моделирование продолжается до тех пор, пока значение параметра **toTerminate** не достигнет нуля.

Отношения перечисленных сущностей показаны на рисунке 2.2.

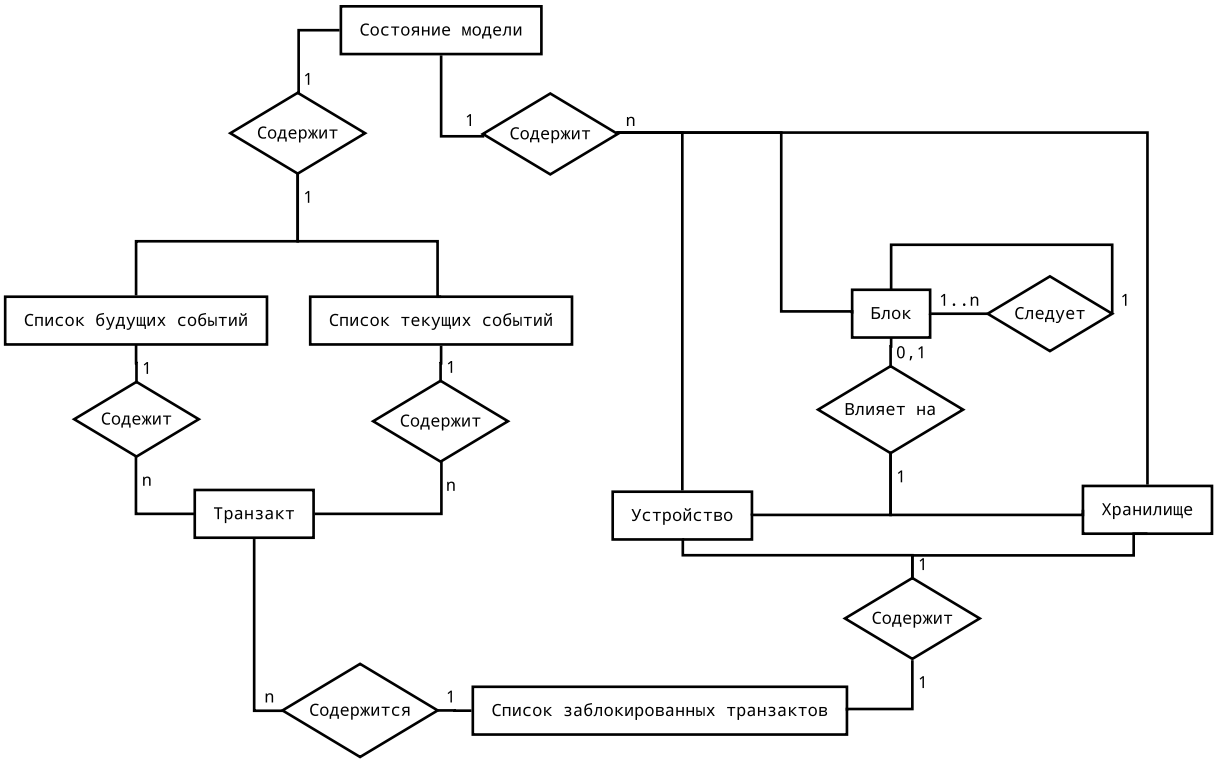

Рисунок 2.2 — Отношения между сущностями системы

2.10 Алгоритм имитационного моделирования

Процесс моделирования запускается при вызове функции, одним параметром которой является сформированная модель, а вторым — количество транзактов, которое необходимо завершить для окончания моделирования.

Таблица 2.5 — Параметры состояния системы

Имя параметра	Тип	Описание
currentTime	Double	Текущий момент модельного времени.
toTerminate	Int	Число транзактов, которое необходимо завершить для окончания моделирования.
blocks	Array Int SBlock	Список блоков, составляющих модель.
facilities	Map String Facility	Список состояний обслуживающих аппаратов.
storages	Map String Storage	Список состояний хранилищ.
queues	Map String Queue	Список состояний очередей.
cec	[Transaction]	Список будущих событий — список транзактов, продвижение которых требует наступления некоторого момента модельного времени. Упорядочен по возрастанию ожидаемого момента времени.
fec	[(Double,Transaction)]	Список текущих событий — список транзактов, продвижение которых возможно в данный момент модельного времени. Упорядочен по убыванию приоритета транзактов.

На первом шаге алгоритма происходит активация всех блоков GENERATE. Для каждого из них вычисляется время создания ближайшего транзакта и эти транзакты помещаются в список будущих событий.

На втором шаге из списка будущих событий извлекаются события, наступающие в ближайший момент модельного времени. Модельное время передвигается на момент наступления тих событий, а сами события помещаются в список текущих событий.

На третьем шаге, до тех пор пока список текущих событий не опустеет, происходит продвижение транзакта из того списка с наибольшим приоритетом. Продвижение каждого транзакта происходит до тех пор, пока транзакт тем или иным образом не покинет текущий список (например, войдет в блок ADVANCE и будет помещен в список будущих событий или попытается войти в блок занятого устройства и попадет в список транзактов, ожидающих освобождения того устройства).

Шаги два и три повторяются до тех пор, пока в процессе моделирования не будет завершено заданное число транзактов. Схема алгоритма показана на рисунке 2.3.

2.11 Обработка захода транзакта в блок

Каждый раз, когда заявка пытается зайти в очередной блок, вызывается функция обработчик, которая определяет, может ли транзакт это сделать и какие дополнительные действия при этом должны быть выполнены. Исключением является блок GENERATE, для которого обработчик (определяющий время создания нового транзакта) вызывается при выходе из него транзакта.

Обработчики индивидуальны для каждого типа блоков. В качестве примера на рисунках 2.4 и 2.5 показаны алгоритмы обработчиков блоков ENTER и LEAVE соответственно.

При входе транзакта в блок ENTER проверяется количество свободных единиц ресурса в соответствующем хранилище. Если ресурса достаточно, то транзакт успешно входит в блок, количество доступных ресурсов уменьшается и обновляется статистика использования хранилища. Затем транзакт продолжает движение по блокам. Если же транзакту требуется больше ресурсов, чем в данный момент есть в хранилище, транзакт попадает в список транзактов, ожидающих освобождения ресурсов.

При входе транзакта в блок LEAVE освобождается указанное количество ресурсов соответствующего хранилища и транзакт продолжает движение по блокам. Если есть транзакты, ожидающие освобождения ресурса, среди них выбирается транзакт с наибольшим приоритетом и делается попытка выделить ему необходимое количество ресурса. Если это удается,

то транзакт помещается в список текущих событий, в противном случае он возвращается в список ожидания.

2.12 Структура библиотеки

На рисунке 2.6 показана общая структура спроектированной библиотеки.

На диаграмме можно выделить:

— Модуль с описанием различных типов блоков и их параметров.

— Группу модулей, ответственных за описание состояния модели (в частности модули со структурами заявок, хранилищ, очередей и т.п.).

— Модуль содержащий непосредственно алгоритм имитационного моделирования.

— Группу модулей с обработчиками захода транзактов в те или иные блоки.

— Модуль формирующий результаты моделирования на основе заключительного состояния системы.

— Группу модулей предназначенных для формирования моделируемой системы и содержащие функции добавляющие в систему те или иные блоки.

— Интерфейсный модуль, реэкспортирующий все необходимые сущности библиотеки и предназначенный для непосредственного подключения к прикладной программе.

2.13 Демонстрационная программа

Для демонстрации возможностей спроектированной библиотеки, а также с целью удостовериться в адекватности выбранных алгоритмов моделирования и верности их реализации, целесообразно разработать демонстрационную программу.

Проектируемая программа должна проводить решение приведенной в предыдущем разделе задачи аналитически и при помощи разработанной библиотеки при различных входных параметрах и выводить результаты в удобной для сравнения форме. Целесообразно также предусмотреть воз-

можность автоматического варьирования выбранного параметра модели и построения графика зависимости результата от этого параметра при фиксированных прочих для аналитической и имитационной модели.

Предполагаемая структура такой программы показана на рисунке 2.7.

2.14 Аналитическая модель системы

Ниже представлен способ аналитического вычисления характеристик системы приведенной в предыдущем разделе.При выводе формул использовались методы укрупнения модели и укрупнения состояний описанные в [6].

2.14.1 Моделирование отказов и восстановлений

Состояние системы можно описать вектором $\xi(t) = (\xi_1(t), \xi_2(t))$, где $\xi_1(t)$ — число неисправных процессоров в момент времени t, $\xi_2(t)$ — число неисправных каналов в момент времени t.

На рисунке 2.8 показана структура фрагмента графа состояний системы, где $\beta_{ij} = \beta \frac{i}{i+j} min\{i+j, L\}$, $\delta_{ij} = \delta \frac{j}{i+j} min\{i+j, L\}$.

Проведем укрупнение состояний системы. Объединим в одно макросостояние все вершины графа, у которых одинаковым является первый компонент $\xi_1(t)$ — число неисправных процессоров. Полученный граф представлен на рисунке 2.9, где $\beta_i = \beta \sum_{j=0}^{N} \pi_j \frac{i}{i+j} min\{i+j, L\}$, π_j — вероятность того, что отказали ровно j каналов.

Тогда выражения для определения вероятностей стационарных состояний примут вид:

$$\begin{cases} p_0 = \left(1 + \dfrac{M\alpha}{\beta_1} + ... + \dfrac{M!\alpha^M}{\prod\limits_{i=1}^{M} \beta_i} \right)^{-1} \\ p_i = p_0 \dfrac{\alpha^i \prod\limits_{j=1}^{i} (M - j + 1)}{\prod\limits_{j=1}^{i} \beta_j}, \quad i = \overline{1,M} \\ \beta_i = \beta \sum\limits_{j=0}^{N} \pi_j \frac{i}{i+j} min\left\{i + j, L\right\} \end{cases} \tag{2.1}$$

Аналогичным образом объединим в одно макросостояние все вершины графа, у которых одинаковым является второй компонент $\xi_2(t)$ — число неисправных каналов. Полученный граф представлен на рисунке 2.10, а выражения для определения вероятностей стационарных состояний примут вид:

$$\begin{cases} \pi_0 = \left(1 + \dfrac{N\gamma}{\delta_1} + ... + \dfrac{N!\gamma^N}{\prod\limits_{i=1}^{N} \delta_i} \right)^{-1} \\ \pi_i = \pi_0 \dfrac{\gamma^i \prod\limits_{j=1}^{i} (N - j + 1)}{\prod\limits_{j=1}^{i} \delta_j}, \quad i = \overline{1,N} \\ \delta_j = \delta \sum\limits_{i=0}^{M} p_i \frac{j}{i+j} min\left\{i + j, L\right\} \end{cases} \tag{2.2}$$

Применяя формулы 2.1 и 2.2 итеративно получим вероятности отказов процессоров и каналов в системе. В качестве начального приближения можно взять $\pi_i = \frac{1}{M}$

2.14.2 Укрупнение модели

Заменим исходную модель агрегированной однофазной моделью АМ1 (см. рисунок 2.11). В агрегированный узел объединена подсистема, включающая в себя процессоры и каналы. Интенсивность обслуживания в этом узле зависит от числа находящихся в нем заявок.

Граф состояний полученной системы представлен на рисунке 2.12. Производительность системы может быть вычислена по формулам:

$$\begin{cases} \hat{\pi}_0 = \left(1 + \dfrac{K\lambda}{\xi_1} + ... + \dfrac{K!\lambda^K}{\prod\limits_{i=1}^{K} \xi_i} \right)^{-1} \\ \hat{\pi}_i = \hat{\pi}_0 \dfrac{\lambda^i \prod\limits_{j=1}^{i} (K - j + 1)}{\prod\limits_{j=1}^{i} \xi_j}, \quad i = \overline{1,K} \\ \xi^* = \sum\limits_{i=1}^{K} \xi_i \hat{\pi}_i \end{cases} \tag{2.3}$$

Однако, чтобы воспользоваться приведенными формулами, необходимо знать параметры связи μ_i. Чтобы их найти, рассмотрим укрупненную модель АМ2, структура и граф состояний которой показаны на рисунках 2.13 и 2.14. За состояние системы примем количество заявок на процессорной фазе, а интенсивности переходов могут быть выражены по формулам:

$$\mu_i = \mu \sum_{j=0}^{M} p_j min\{i, M - j\} \tag{2.4}$$

$$\nu_i = \nu \sum_{j=0}^{N} \pi_j min\{n - i + 1, N - j\} \tag{2.5}$$

Параметр связи может вычислен по следующим формулам:

$$\begin{cases} \hat{p}_0 = \left(1 + \dfrac{\nu_1}{\mu_1} + ... + \dfrac{\prod\limits_{i=1}^{n} \nu_i}{\prod\limits_{i=1}^{n} \mu_i} \right)^{-1} \\ \hat{p}_i = \hat{p}_0 \dfrac{\prod\limits_{j=1}^{i} (\nu_j)}{\prod\limits_{j=1}^{i} \mu_j}, \quad i = \overline{1,n} \\ \xi_n = \sum\limits_{i=1}^{n} \hat{p}_i \mu_i \end{cases} \tag{2.6}$$

2.14.3 Окончательная расчетная схема

Последовательность расчета производительности системы должна быть следующей (см. рисунок 2.15):

а) По формулам 2.1 и 2.2 вычислить $\pi_i, i = \overline{0,N}$ и $p_i, i = \overline{0,M}$.

б) По формулам 2.4, 2.5 и 2.6 вычислить $\xi_n, n = \overline{1,K}$.

в) Вычислить ξ^* по формулам 2.3.

2.15 Имитационная модель

Ниже представлена имитационная модель демонстрационной системы, реализованная при помощи разработанной системы моделирования.

2.15.1 Моделирование многоканальных обслуживающих устройств

Так как в исследуемой системе используются многоканальные обслуживающие устройства, каждый из каналов которых может выйти из строя независимо от других, не представляется возможным промоделировать их при помощи хранилищ (хранилища могут быть отключены только целиком и не поддерживают вытеснение транзактов). Поэтому придется каждый канал обслуживающего устройства моделировать отдельным многоканальным устройством, а выбор транзактом свободного канала осуществлять при помощи блока TRANSFER, работающего в режиме ALL.

Ниже приведен код функции, формирующий часть модели, ответственную за обслуживание заявки на одном из процессоров. В качестве параметров функция принимает номер процессора, интенсивность обработки и метку блока с которого начинается следующая фаза обслуживания.

```
proc i mu l =
    do seize ("proc" ++ show i)
       advance (1/mu, xpdis)
       release ("proc" ++ show i)
       transfer ((),l)
```

2.15.2 Моделирование отказов и восстановлений

Для моделирования отказов и восстановления каждого из каналов используется отдельный транзакт, который сперва захватывает соответствующее устройство при помощи блока PREEMPT в режиме прерывания (после этого устройство становится недоступно для транзактов, моделирующих задачи), после этого пытается захватить ресурс хранилища, моделирующего ремонтные бригады, и после того через некоторое время освобождает устройство и ресурс (устройство восстановлено и снова доступно для обработки задач). Время, остававшееся до окончания обработки транзакту, вытесненному в момент поломки, сохраняется в параметре транзакта, а сам транзакт перенаправляется на фазу дообслуживания.

Ниже приведен код функции, формирующей часть модели ответственную за отказы и восстановления одного из процессоров. В качестве параметров функция принимает номер процессора, интенсивности его отказов и восстановлений и метку блока, с которого начинается фаза дообслуживания.

```
breakReairProc i alpha beta l' =
    do generate (0,0,0,1)
       l <- advance (1/alpha, xpdis)
       preempt ("proc" ++ show i, (), l', 1, RE)
       enter "repairers"
       advance (1/beta,xpdis)
```

```
        return' ("proc" ++ show i)
        leave "repairers"
        transfer ((),1)
```

Функция формирующая фазу дообслуживания во всем подобна функции **proc** за исключением того, что время обслуживания берется из параметра транзакта, а не определяется случайным образом.

```
procFinish i l =
    do seize ("proc" ++ show i)
       advance (Pr 1)
       release ("proc" ++ show i)
       transfer ((),l)
```

2.15.3 Общая модель системы.

Функция формирующая при помощи вышеописанных общую модель системы показана ниже. В качестве параметров она принимает все параметры системы (количество задач, процессоров, каналов и ремонтных бригад и интенсивности обработки на различных фазах и отказов и восстановлений).

```
model m n k l lambda mu nu alpha beta gamma delta =
    do storage ("repairers", l)
       generate (0,0,0,1)
       advance 100
       terminate 1

       generate (0,0,0,k)
       userPhase <- advance (1/lambda, xpdis)
       seize "counter"
       release "counter"
       procStart <- transfer (All,userPhase +5,
                              userPhase + 5 + (m-1)*4,4)
       chanStart <- transfer (All,userPhase +5+4*m,
                              userPhase + 5 + m*4 + (n-1)*4,4)
```

```
mapM_ (\i -> proc i mu chanStart) [1..m]
ls <- mapM (\i -> chan i nu userPhase) [1..n]

let l = fromIntegral $ last ls
procFinishL <- transfer (All,l + 3,
                         l + 3 + (m-1)*4,4)
chanFinishL <- transfer (All,l + 3 + 4*m,
                         l + 3 + 4*m + (n-1)*4,4)
mapM_ (\i -> procFinish i chanStart) [1..m]
mapM_ (\i -> chanFinish i userPhase) [1..n]

when (alpha > 0) $ mapM_ (\i -> breakReairProc
                                  i alpha beta procFinishL
                         ) [1..m]
when (gamma > 0) $ mapM_ (\i -> breakReairChan
                                  i gamma delta chanFinishL
                         ) [1..n]
```

Первым делом функции объявляется хранилище емкостью равной числу ремонтных бригад. Затем создается группа блоков ответственная за ограничение времени моделирования.

Последующие блоки отвечают за создание транзактов, моделирующих задачи пользователей и время их обдумывания пользователем. Далее следуют блоки TRANSFER предназначенные для передачи заявки на свободный в данный момент процессор или канал. Далее путем многократного вызова функций **proc** и **chan** формируется часть модели ответственная за обработку задач на процессорной и канальной фазах.

Далее аналогичным образом, при помощи блоков TRANSFER и многократного вызова функций **procFinish** и **chanFinish** формируются фазы дообслуживания.

И в заключение при помощи функций **breakReairProc** и **breakReairChan** формируются блоки моделирующие отказы и восстановления.

2.16 Выводы

В результате проектирования был разработан синтаксис и алгоритм описания моделей, а также алгоритмы и структуры данных необходимые для имитационного моделирования. Были спроектированы структуры библиотеки имитационного моделирования и демонстрационной программы, призванной проиллюстрировать ее работу. Были разработаны аналитическая и имитационная модели демонстрационной системы, описанной в предыдущем разделе.

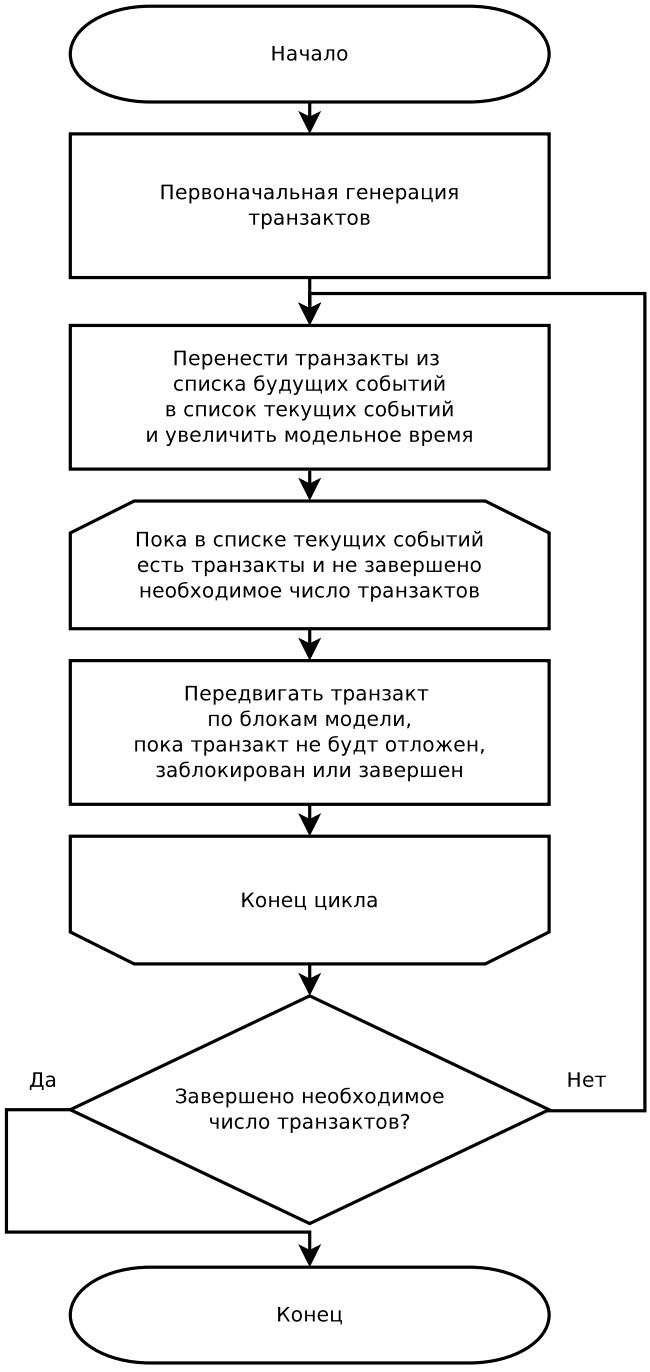

Рисунок 2.3 — Алгоритм имитационного моделирования

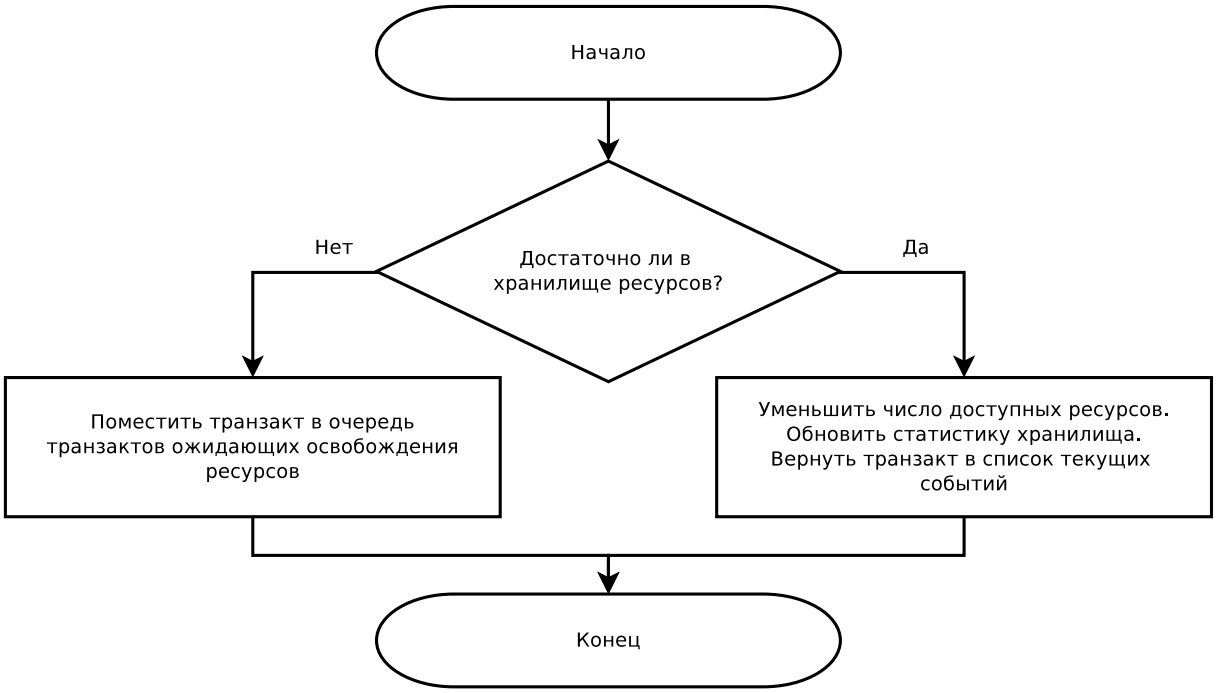

Рисунок 2.4 — Алгоритм обработки блока ENTER

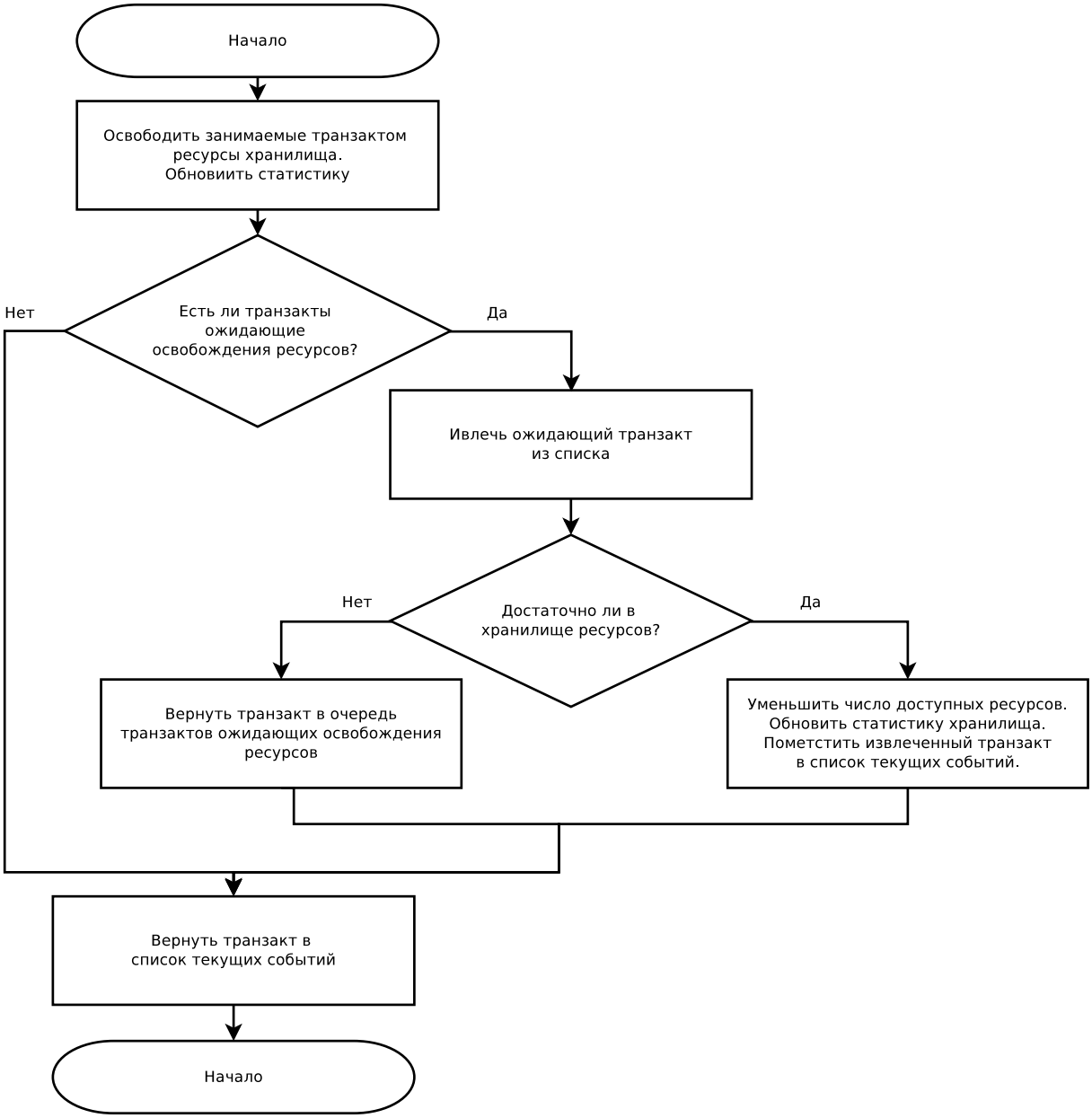

Рисунок 2.5 — Алгоритм обработки блока LEAVE

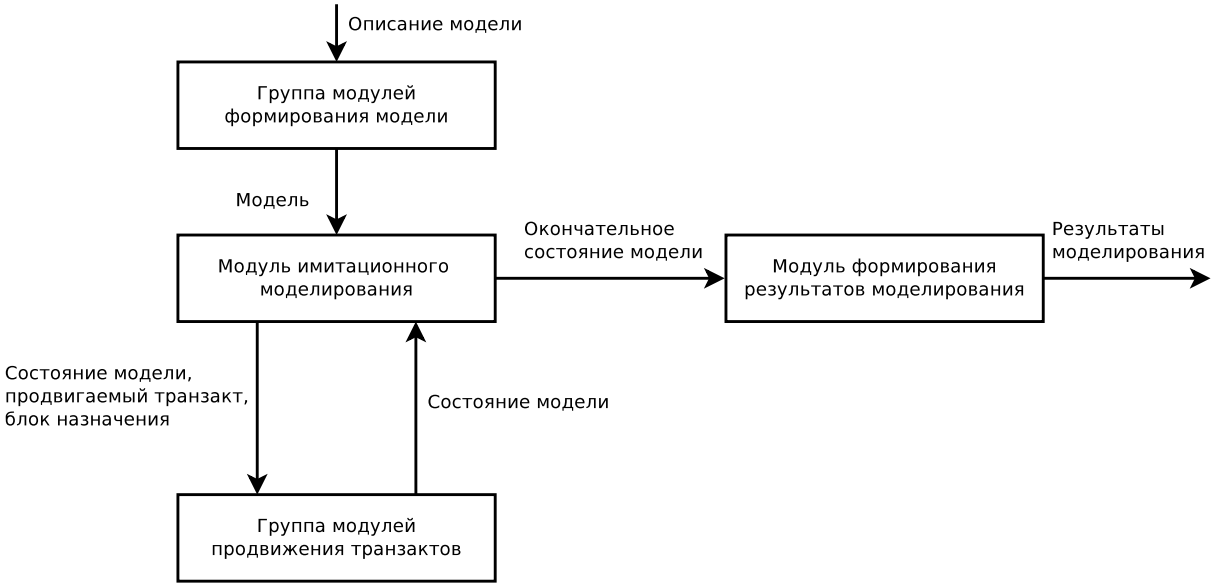

Рисунок 2.6 — Структура разработанной библиотеки

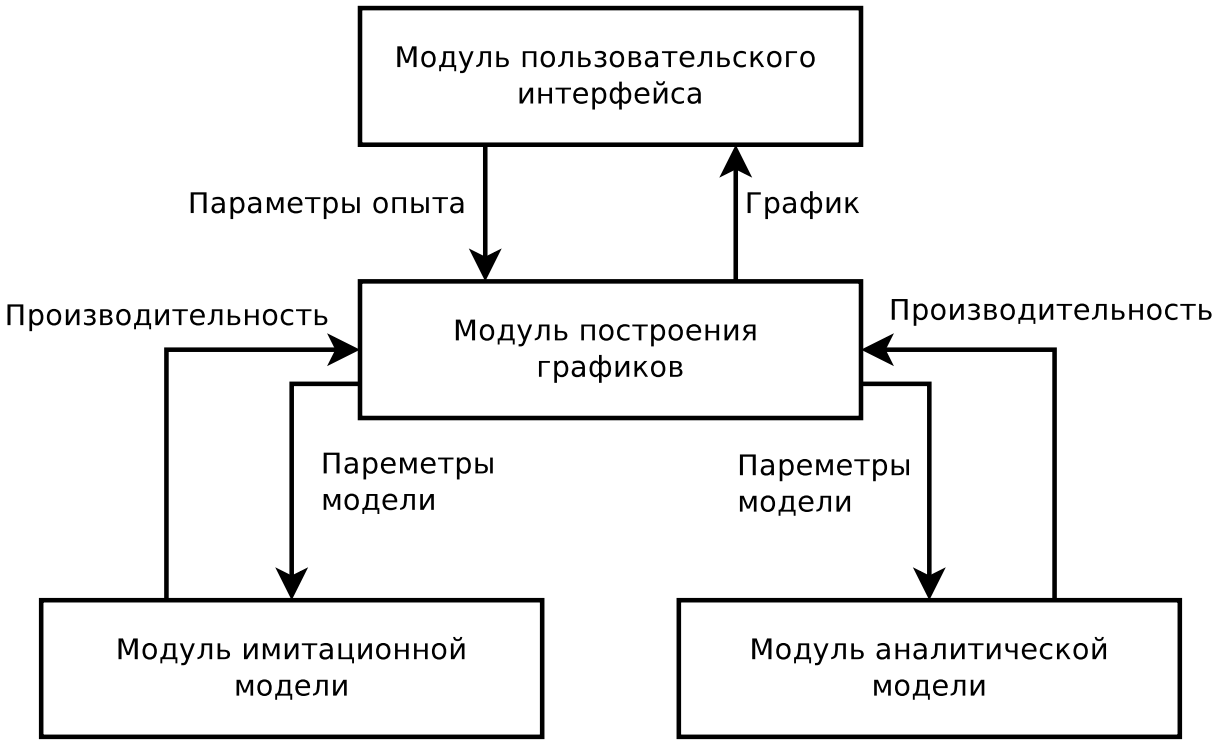

Рисунок 2.7 — Структура демонстрационной программы

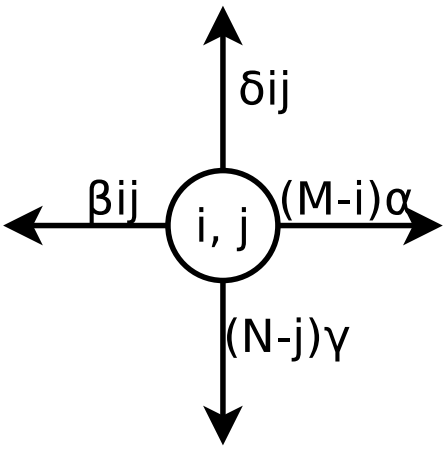

Рисунок 2.8 — Структура фрагмента графа состояний системы

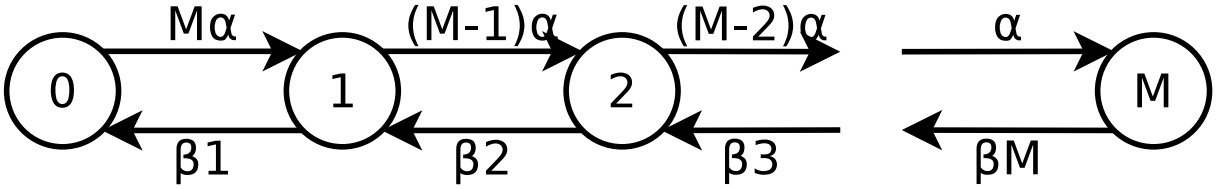

Рисунок 2.9 — Граф состояний системы

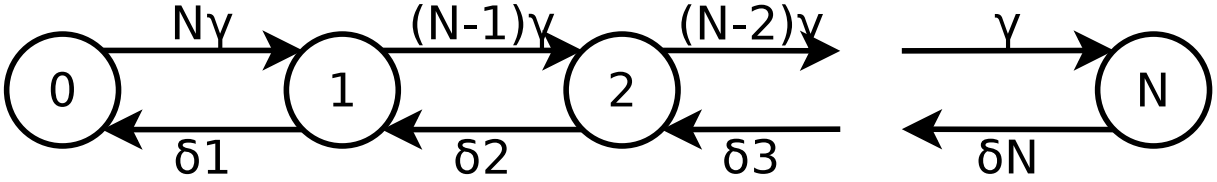

Рисунок 2.10 — Граф состояний системы

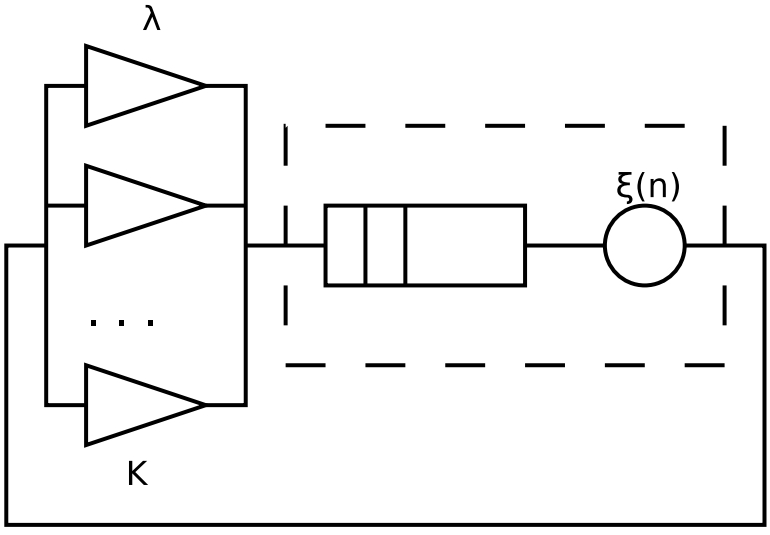

Рисунок 2.11 — Укрупненная модель АМ1

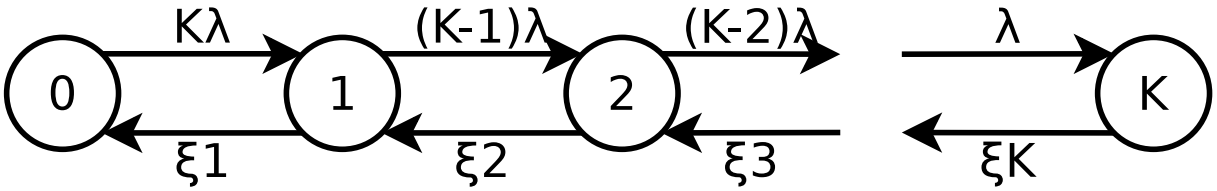

Рисунок 2.12 — Граф состояний модели АМ1

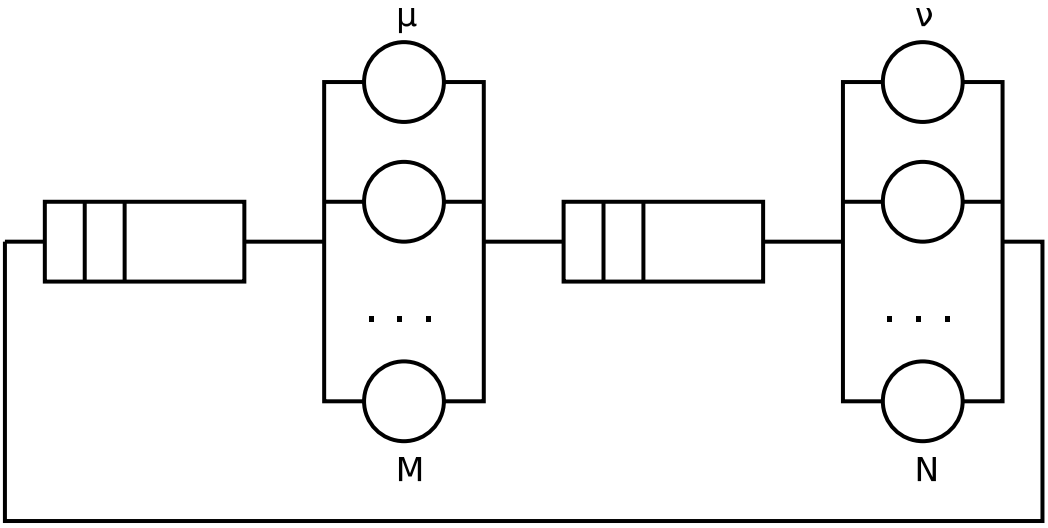

Рисунок 2.13 — Укрупненная модель АМ2

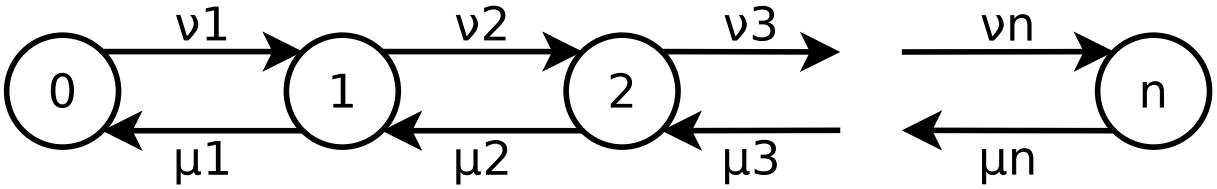

Рисунок 2.14 — Граф состояний модели АМ2

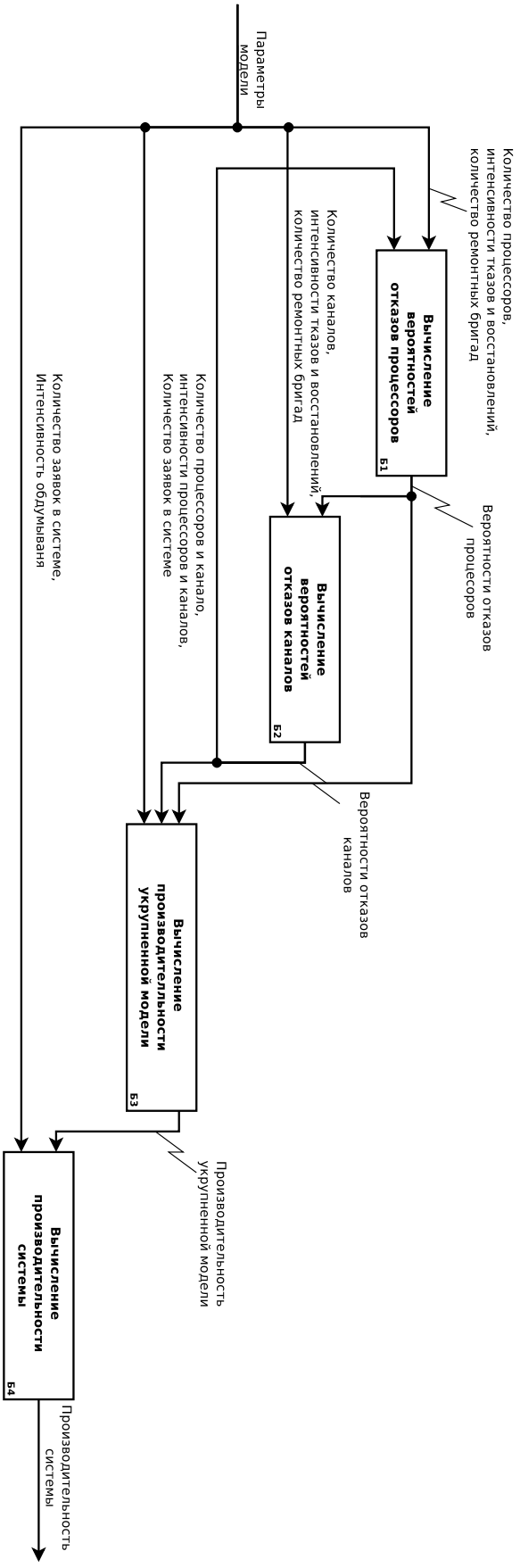

Рисунок 2.15 — Последовательность расчета производительности системы

3 Технологический раздел

В данном разделе описывается реализация разработанной системы и демонстрационной программы, средства разработки, сборки и развертывания, а также тестирование.

3.1 Выбор средств программной реализации

Так как целью работы является разработка библиотеки для языка Haskell, целесообразно вести разработку на этом же языке. Язык Haskell относится к функциональным языкам общего назначения и обладает следующими особенностями:

— Чистые функции. Большинство функций в языке Haskell являются чистыми, то есть детерминированными и не имеющими побочных эффектов. При использовании таких функций программист может быть уверен, что при вызове такой функции не будет произведено каких-либо неявных действий (запись данных в файл, изменение значения переданных параметров, изменение состояния некоторого объекта и т.п.) и на одних и тех же входных параметрах функция всегда вернет одинаковый результат. Это существенно упрощает разработку и тестирование программ.

— Ленивые вычисления. Все вычисления в Haskell по умолчанию являются ленивыми, то есть ни одно значение не будет вычислено до тех пор, пока его значение действительно не понадобится. Этот механизм позволяет экономить вычислительные мощности и работать со структурами вроде бесконечных списков. Однако при неаккуратном использовании это может привести к неэффективному расходованию памяти.

— Строгая статическая типизация. Строгая типизация позволяет писать более надежные программы, так как несоответствие типов приведет к сообщению об ошибке, а не к некорректному поведению программы и будет быстрее обнаружено и исправлено. Статическая типизация позволяет выявить такие ошибки еще на этапе компиляции программы.

— Автоматическое управление памятью. Как и большинство современных языков программирования Haskell берет на себя выделение и освобождение памяти. Это гарантировать отсутствие в разрабатываемой про-

грамме таких ошибок как переполнение буфера, не инициализированные переменные и т.п.

Помимо этого для Haskell существует обширный централизованный архив библиотек Hackage[7], и поисковая система Hoogle[8], позволяющая найти описание функции не только по ее названию но и по сигнатуре.

3.1.1 Построение графиков

Для построения графиков была выбрана система Gnuplot[9]. Это свободно распространяемая, кросплатформенная программа предназначенная для построения двух- и трехмерных графиков функций, заданных как аналитически, так и в виде наборов данных. Gnuplot поддерживает вывод результатов в различных форматах: растровых (PNG, JPEG), векторных (SVG, PDF), в виде кода LaTeX, в интерактивном режиме и др. Система используется для построения графиков в таких математических пакетах как GNU Octave, Maxima и других.

Для использования возможностей Gnuplot в программах на Haskell существует несколько библиотек, опубликованных на Hackage. В данной работе была использована библиотека EasyPlot.

3.1.2 Построение пользовательского интерфейса

Для построения пользовательского интерфейса демонстрационной программы была использована библиотека wxHaskell[10], которая в свою очередь является оберткой вокруг библиотеки для построения пользовательского интерфейса на C++ wxWidgets.

Особенности wxWidgets:

— Созданные с помощью данной библиотеки приложения переносимы на большинство современных ОС.

— В разработанном интерфейсе используются элементы управления привычные для пользователей целевой ОС. То есть стиль интерфейса программы будет отличаться на различных ОС и будет соответствовать рекомендуемому стилю для конкретной системы.

wxHaskell является надстройкой над wxWidgets, позволяющей создавать графический интерфейс к программам на языке Haskell. Она поддерживает большую часть функционала wxWidgets, позволяет описывать интерфейс в «декларативном» стиле с использованием функциональных связок и абстракций высокого уровня. Библиотека особенно удобна для создания демонстрационных версий программ, так как во многом берет на себя решение задачи корректного расположения элементов управления на экране.

3.1.3 Сборка и развертывание библиотеки

Для сборки разработанной библиотеки и развертывания ее на целевой машине была использована система Cabal. Данная система предоставляет единый интерфейс для создания и установки инсталяционных пакетов с программами и библиотеками на Haskell. Система связана с архивом библиотек Hackage и позволяет устанавливать хранящиеся там пакеты и оформить собственную программу в виде пакета для Hackage.

Информация о создаваемом пакете указывается в файле **.cabal** в директории проекта. В нем указываются:

— имя пакета;
— текущая версия;
— информация об авторе и лицензии;
— допустимые версии компилятора;
— используемые расширения компилятора;
— входящие в состав пакета библиотеки и исполняемые программы;
— пакеты Hackage, необходимые для работы пакета;
— и др.

Фрагмент файла **.cabal** для разработанной библиотеки приведен ниже.

Листинг 3.1 — Фрагмент описания пакета

```
1 name:                hsGPSS
2 version:             0.1.0.0
3 author:              Minix
4 maintainer:          migorec@gmail.com
```

```
5  build-type:           Simple
6  cabal-version:        >=1.8
7
8  -- Описание библиотеки
9  library
10   -- Модули экспортируемые библиотекой
11   exposed-modules:  Simulation.HSGPSS
12                     Simulation.HSGPSS.Prelude
13                     Simulation.HSGPSS.Random.Functions
14
15   -- Модули включаемые в библиотеку, но не экспортируемые
16   other-modules:  Simulation.HSGPSS.Blocks,
17                   Simulation.HSGPSS.Blocks.Generate,
18                   ...
19                   Simulation.HSGPSS.Random
20
21   -- Пакеты от которых зависит библиотека
22   build-depends:        base >=4.5 && <5, array ==0.4.*, mtl ==2.1.*,
         containers ==0.5.*, random ==1.0.*
23
24 -- Исполняемая программа с тестами
25 executable tests
26     -- Головной модуль
27     main-is: Simulation/HSGPSS/Tests/Main.hs
28     -- Прочие модули
29     other-modules:  Simulation.HSGPSS.Tests.Blocks.Generate,
30                     ...
31                     Simulation.HSGPSS.Tests.Chains
32     -- Зависимости
33     build-depends:        base >=4.5 && <5, array ==0.4.*, mtl ==2.1.*,
          HUnit ==1.2.*,  containers ==0.5.*, random ==1.0.*, statistics
          ==0.10.*, vector ==0.10.*
```

Сборка и установка пакета выполняется следующими командами:

— **cabal configure** — подготовка к сборке программы: определение целевой платформы, зависимостей и др.;

— **cabal build** — запуск процесса компиляции;

— **cabal install** — установка пакета в систему. Включает в себя первые две команды;

— **cabal clean** — удаляет все временные файлы созданные предыдущими командами;

3.2 Тестирование

Для модульного тестирования разработанной библиотеки использовалась библиотека HUnit[11]. Это фреймворк основанный на идеях JUnit, но предназначенный для тестирования программ на Haskell.

Типичный базовый тест состоит из текстового описания, вычисляемого выражения и ожидаемого результата. Базовые тесты объединяются в группы, те, в свою очередь, могут объединяться в большие группы и т.д. В итоге образуется единая древовидная структура тестов. Ниже показан пример описания теста.

```
emptyFEC = TestCase (assertEqual "for (addFE [] (1,defTransact)),"
                                 ([(1,defTransact)])
                                 (addFE [] (1,defTransact))
                    )
```

По результатам выполнения тестов HUnit выводит следующую статистику: общее число тестов, число проведенных тестов, число тестов вызвавших непредвиденное исключение (что говорит об ошибке в самом тесте) и число тестов, закончившихся неудачей (что обычно говорит об ошибке в тестируемой программе).

В таблице 3.1 приведен протокол тестирования функции **addFE**, добавляющей новое событие в список будущих событий.

В общей сложности было проведено 148 тестов. Степень покрытия кода тестами показана в таблице 3.2. Вывод тестовой программы представлен ниже.

```
Cases: 148  Tried: 148  Errors: 0  Failures: 0
```

Тестирование проводилось на машинах с различными операционными системами: Ubuntu 12.04, Ubuntu 13.04, MS Windows 7. На всех библиотека скомпилировалась без ошибок и все тесты были пройдены успешно.

3.2.1 Тестирование стохастических функций

При организации имитационного моделирования важную роль играют функции генерации случайных величин с заданным законом распреде-

Таблица 3.1

Название теста	emptyFEC
Описание теста	Добавление события в пустой список
Ожидаемый результат	Список из одного события
Результат	Тест пройден

Название теста	nearestAddFE
Описание теста	Добавление в список события с наименьшим временем наступления
Ожидаемый результат	Добавленное событие становится первым в списке
Результат	Тест пройден

Название теста	lastAddFE
Описание теста	Добавление в список события с наибольшим временем наступления
Ожидаемый результат	Добавленное событие становится последним в списке
Результат	Тест пройден

Название теста	middleAddFE
Описание теста	Добавление в список события с промежуточным временем наступления
Ожидаемый результат	Добавленное событие занимает место в списке в соответствии со своим временем наступления
Результат	Тест пройден

Название теста	multyAddFE
Описание теста	Добавление в список события с тем же временем наступления, что и у одного из событий в списке
Ожидаемый результат	Добавленное событие занимает место в списке сразу за событием с тем же временем наступления
Результат	Тест пройден

ления. При тестировании таких функций приходится отказаться от описанного выше принципа «выражение — ожидаемое значение», поскольку результат вычисления функции различается от запуска к запуску.

Таблица 3.2 — Степень покрытия кода тестами

Подсистема	Покрытие функций	Покрытие условий	Покрытие выражений
Формирование моделей	82%	100%	83%
Имитационное моделирование	72%	96%	93%

Для тестирования таких функций следует многократно вычислить значение тестируемой функции и сравнить собранную статистику с ожидаемым распределением. В данной работе в качестве критерия соответствия полученных значений ожидаемому распределению использовался критерий согласия Пирсона.

Для вычисления статистики по этому критерию и вычисления квантилей распределения χ^2 использовался пакет statistics из архива библиотек Hackage.

3.2.2 Сравнение аналитической и имитационной модели

В качестве дополнительной проверки на корректность разработанных алгоритмов и безошибочность их реализации был проведен ряд опытов с демонстрационной программой. Были построены графики зависимости производительности моделируемой системы от различных параметров при прочих фиксированных параметрах.

Опыт 1

Варьируется число задач в системе. Количество процессоров и каналов равно единицы. Интенсивность обдумывания равна единице. Отказов и восстановлений нет. Интенсивности обработки на канальной и процессорной фазе равны 5. Результаты показаны на рисунке 3.1.

Опыт2

Опыт проводится при тех же параметрах, но число каналов и интенсивность обслуживания на процессорной фазе увеличены в двое. Результаты показаны на 3.2.

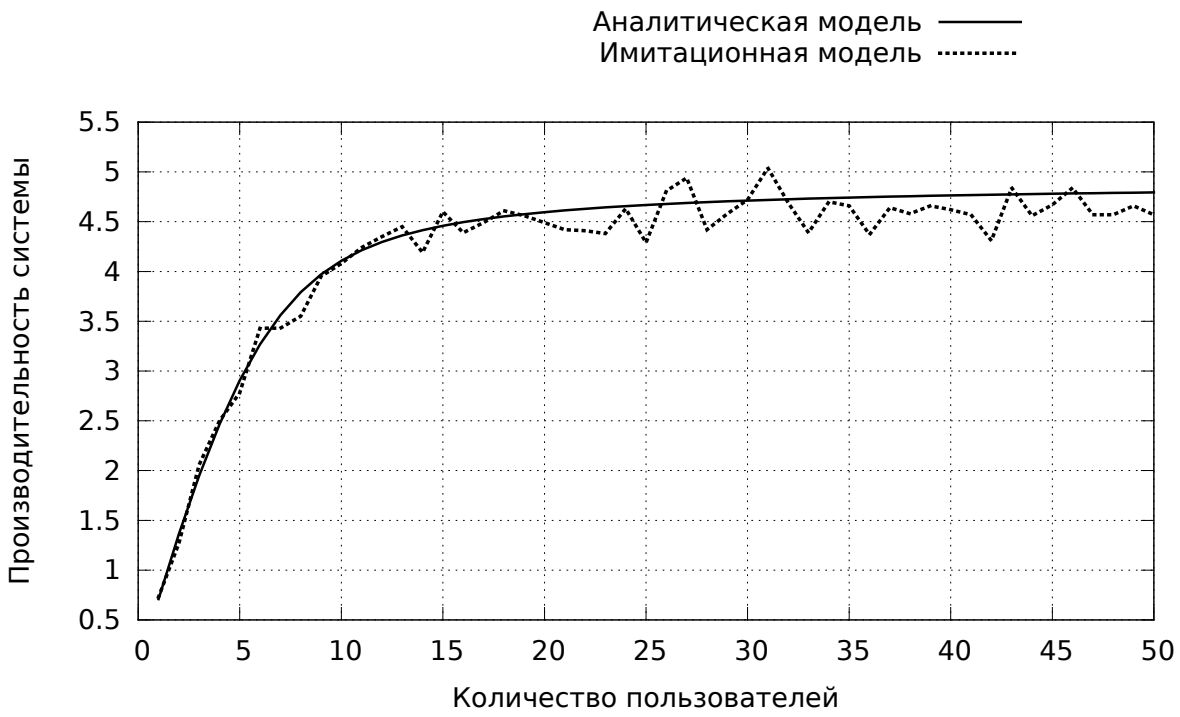

Рисунок 3.1 — Опыт 1

Опыт3

Варьируется число процессоров. Число задач и каналов равняется 10. Интенсивность обдумывания равна единице. Интенсивности обработки задач на процессорной и канальной фазах равны 5. Результаты опыта показаны на рисунке 3.3.

Опыт4

Опыт проводится при тех же параметрах, что и предыдущий, но количество каналов уменьшено вдвое. Результаты опыта показаны на рисунке 3.4.

Опыт5

Варьируется интенсивность обработки на канальной фазе. количество задач равно 10. Количество процессоров и каналов равно 4. Интенсивность обдумывания — 5, интенсивность обработки на процессорной фазе — 20 Результаты опыта показаны на рисунке 3.5.

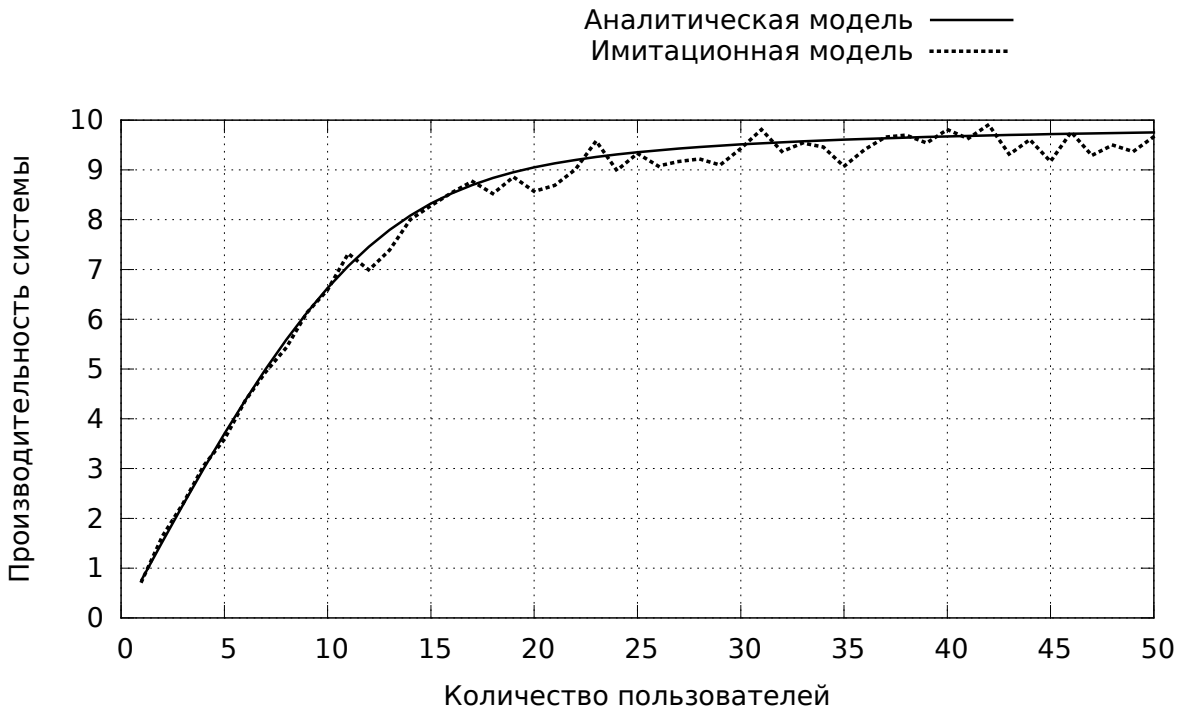

Рисунок 3.2 — Опыт 2

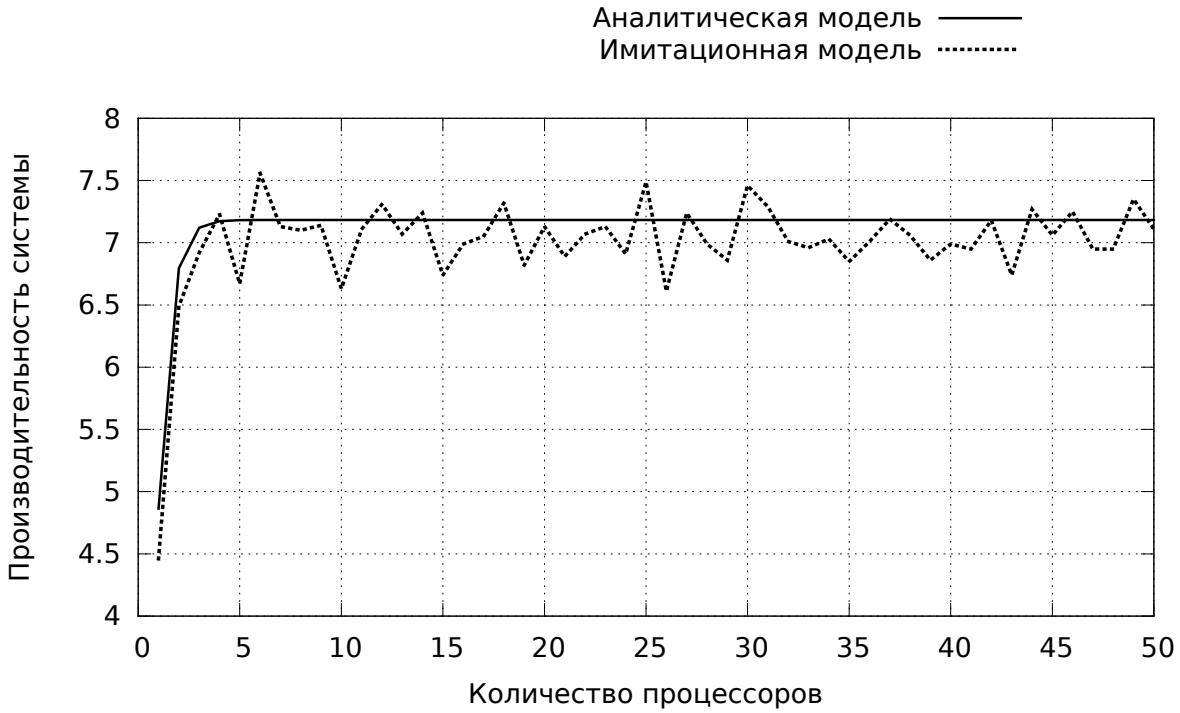

Рисунок 3.3 — Опыт 3

Опыт6

Опыт проводится при тех же параметрах, что и предыдущий, но количество задач уменьшено вдвое. Результаты опыта показаны на рисунке 3.6.

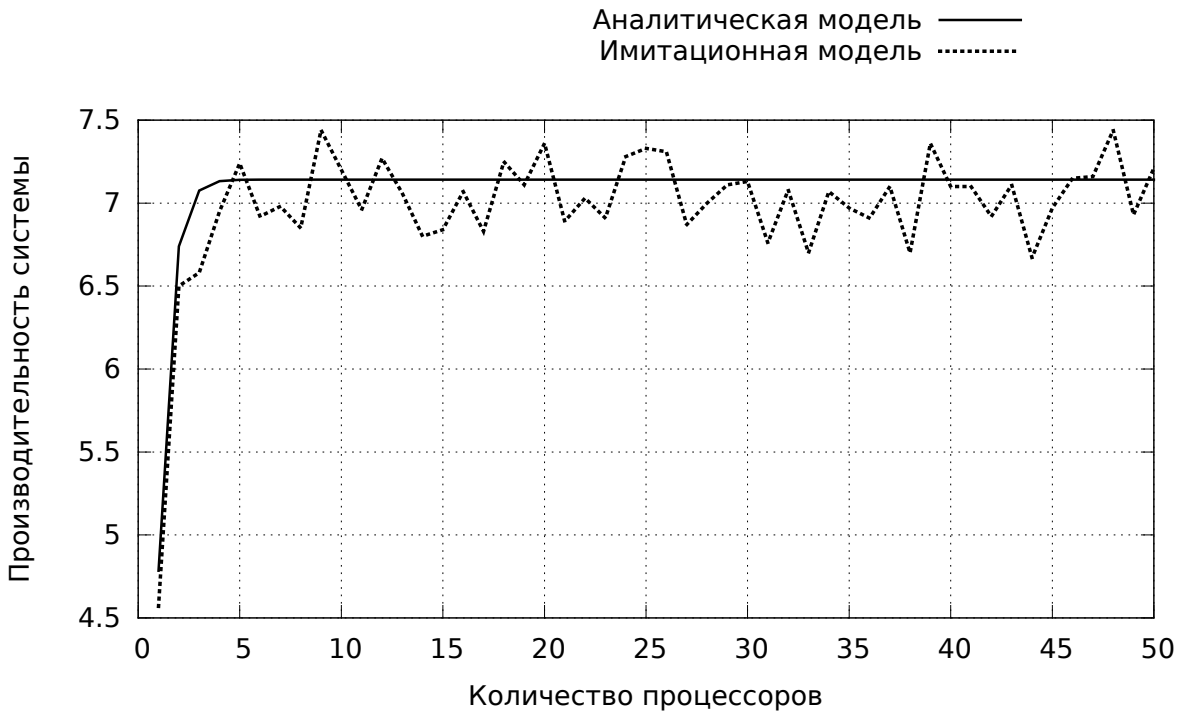

Рисунок 3.4 — Опыт 4

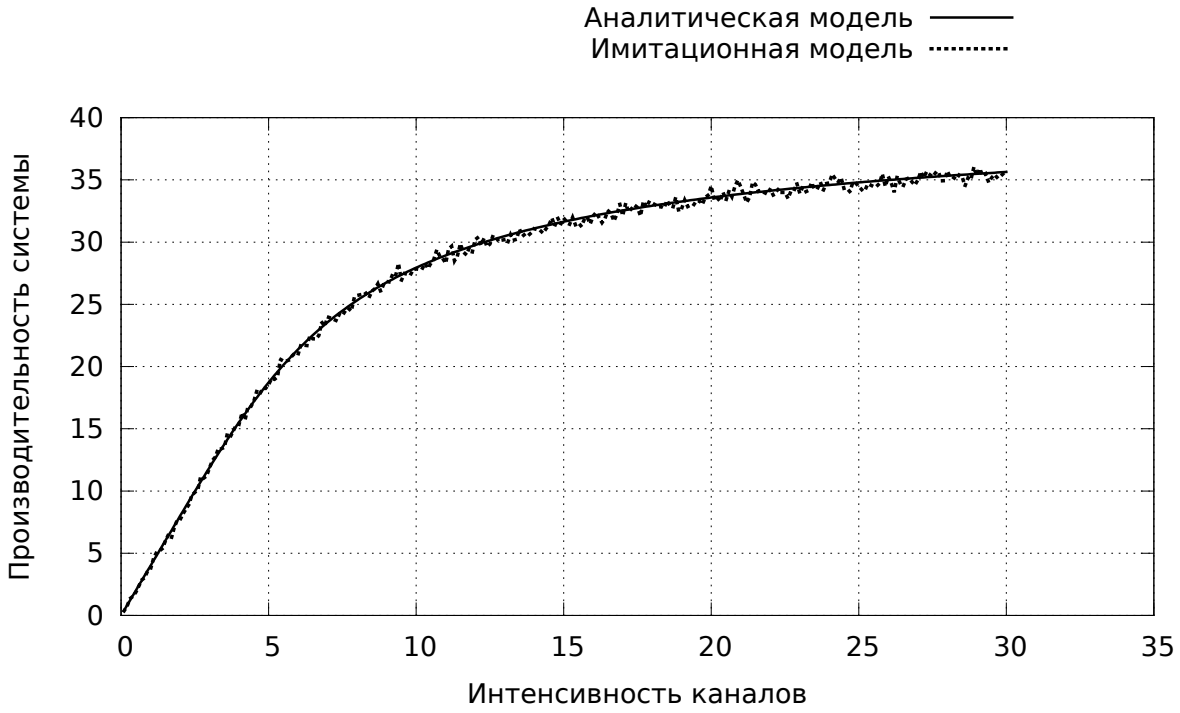

Рисунок 3.5 — Опыт 5

Опыт7

Варьируется интенсивность отказов процессоров. Количество задач равно 10. Количество процессоров и каналов равно 4. Интенсивность обду-

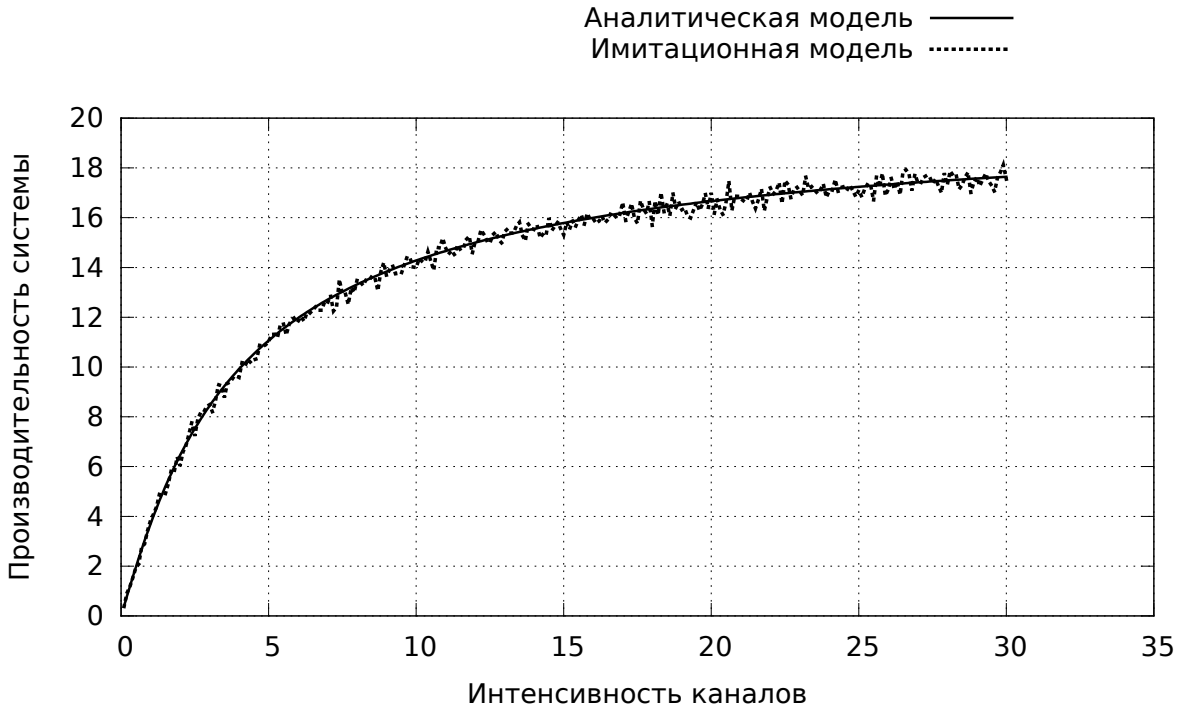

Рисунок 3.6 — Опыт 6

мывания — 1, интенсивность обработки на процессорной и канальной фазах — 3. Интенсивность восстановления процессоров равна 10. Количество ремонтных бригад — 5.Результаты опыта показаны на рисунке 3.7.

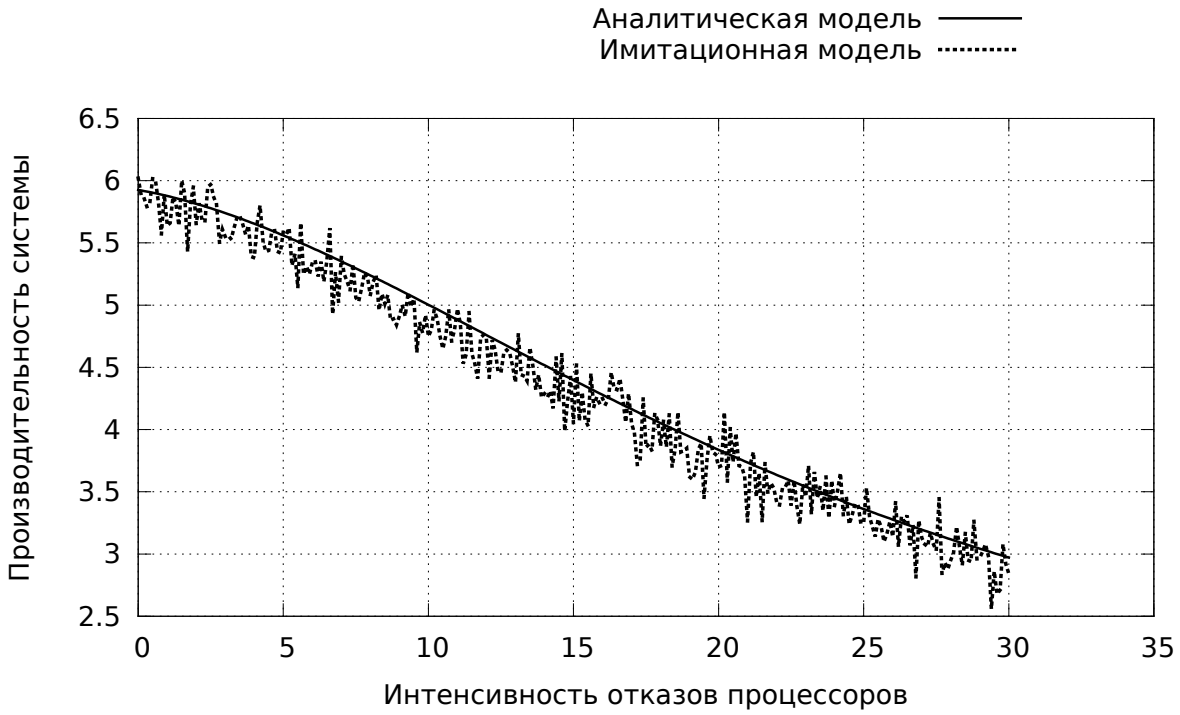

Рисунок 3.7 — Опыт 7

Опыт8

Опыт проводится при тех же параметрах, что и предыдущий, но интенсивность восстановления процессоров уменьшена вдвое. Результаты опыта показаны на рисунке 3.8.

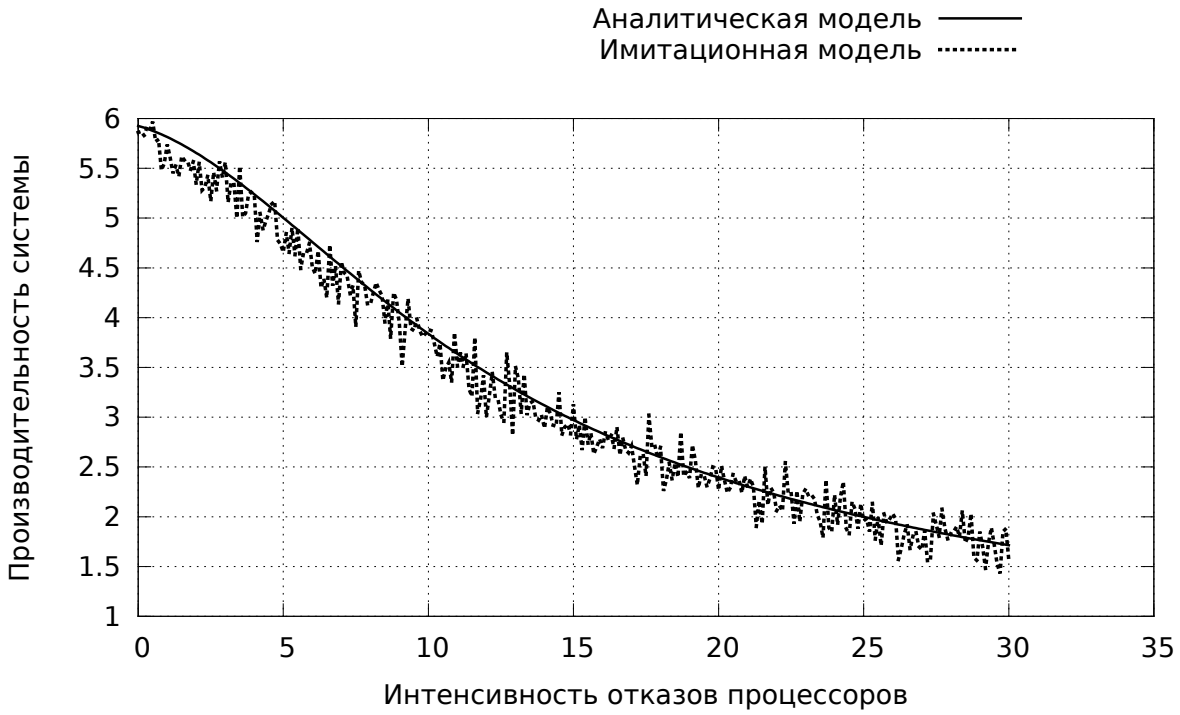

Рисунок 3.8 — Опыт 8

Опыт9

Варьируется количество ремонтных бригад. Количество задач и процессоров — 5. Количество каналов 15. Интенсивность обдумывания —1. Интенсивность обработки на процессорной и канальной фазах —5. Интенсивность отказов каналов —14, интенсивность восстановлений —7. Результаты опыта показаны на рисунке 3.9.

Опыт10

Опыт проводится при тех же параметрах, что и предыдущий, но интенсивность восстановления взята равной интенсивности отказов. Результаты опыта показаны на рисунке 3.10.

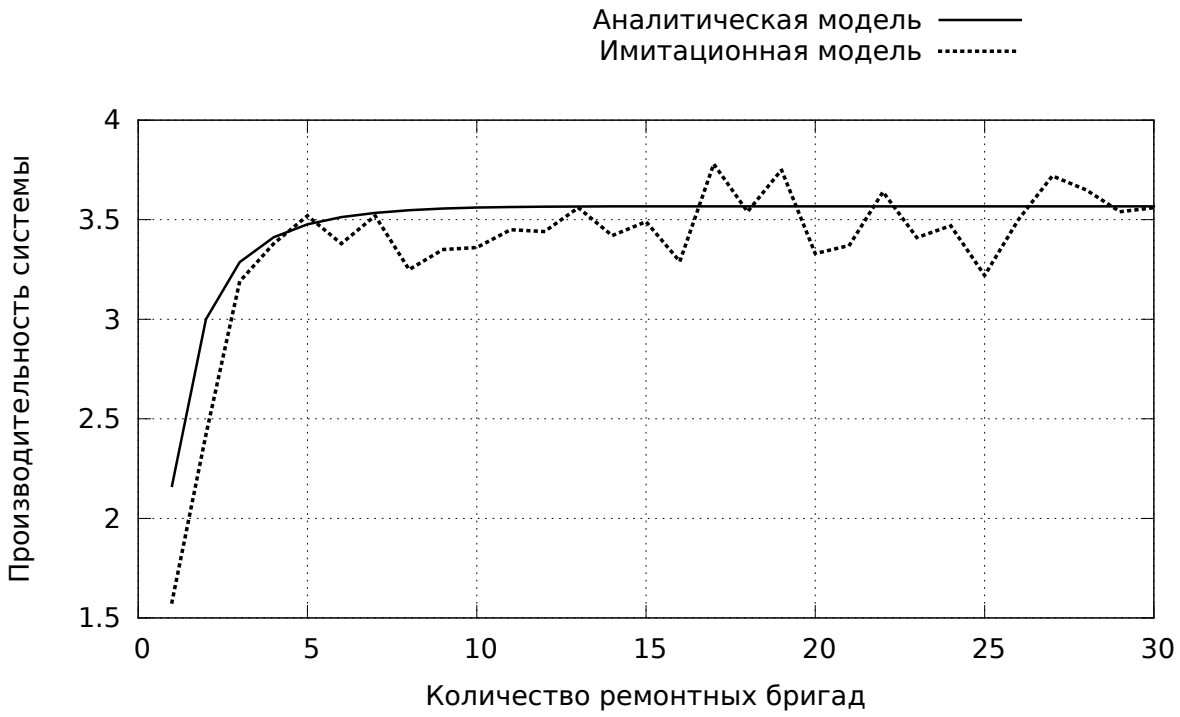

Рисунок 3.9 — Опыт 9

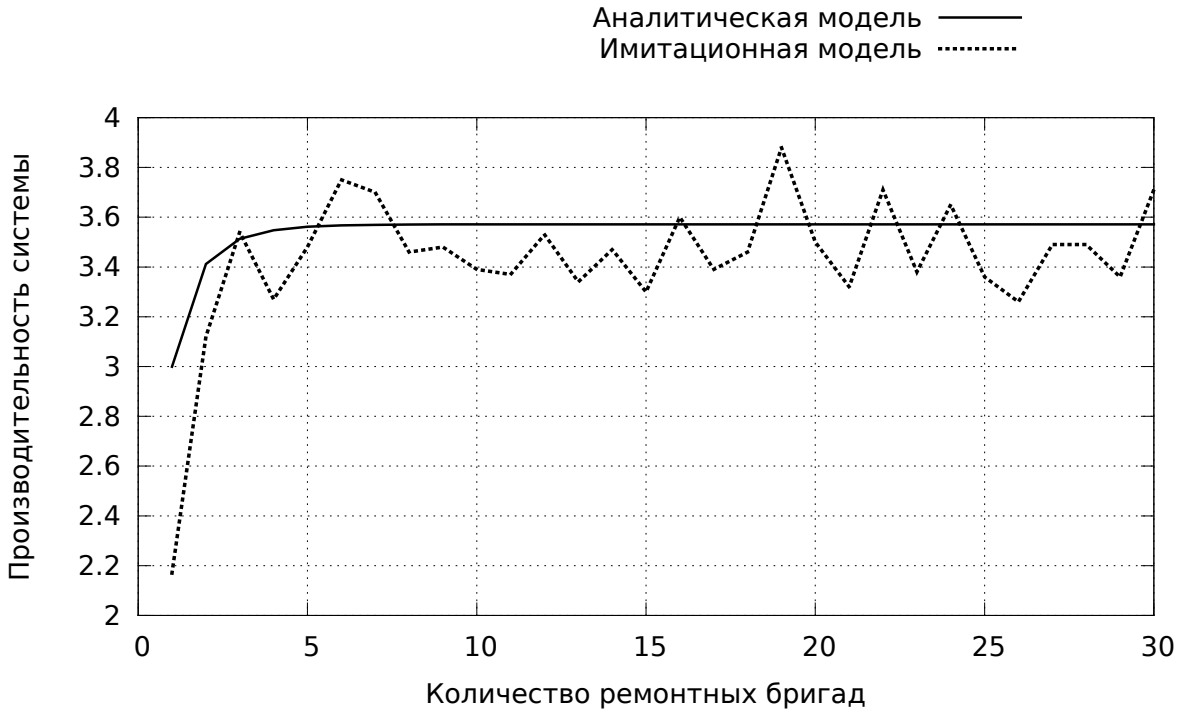

Рисунок 3.10 — Опыт 10

Из проведенных опытов видно, что данные полученные при помощи разработанной библиотеки и имитационной модели качественно соответствуют результатам аналитических исследований. Некоторый разброс значений объясняется стохастическим характером модели.

3.3 Выводы

Были реализованы разработанные в предыдущем разделе методы и алгоритмы. Разработанная библиотека была протестирована при помощи модульного тестирования. Также был проведен ряд опытов подтверждающих адекватность построенной имитационной модели и корректность реализованных алгоритмов.

4 Организационно-экономический раздел

Организационно-экономическая часть процесса разработки программного продукта предусматривает выполнение следующих работ:

— формирование состава выполняемых работ и группировка их по стадиям разработки;

— расчет трудоемкости выполнения работ;

— установление профессионального состава и расчет количества исполнителей;

— определение продолжительности выполнения отдельных этапов разработки;

— построение календарного графика выполнения разработки;

— контроль выполнения календарного графика.

4.1 Формирование состава выполняемых работ и группировка их по стадиям разработки

Разработку программного продукта можно разделить на следующие стадии:

— техническое задание;

— расчет трудоемкости выполнения работ;

— эскизный проект;

— технический проект;

— рабочий проект;

— внедрение.

Допускается объединение технического и рабочего проекта в технорабочий проект.

Планирование длительности этапов и содержания проекта осуществляется в соответствии с ЕСПД ГОСТ 34.603–92 и распределяет работы по этапам. как показано в таблице 4.1.

Таблица 4.1 — Распределение работ проекта по этапам

Основные стадии	№	Содержание работы
1. Техническое задание	1	Постановка задачи
	2	Выбор средств проектирования и разработки
2. Эскизный проект	3	Разработка структуры системы
	4	Разработка алгоритмов описания моделей и моделирования
	5	Разработка вспомогательных алгоритмов
	6	Разработка пользовательского интерфейса
2. Технорабочий проект	7	Реализация алгоритмов описания моделей и моделирования
	8	Реализация вспомогательных алгоритмов
	9	Реализация пользовательского интерфейса
	10	Отладка программного продукта
	11	Исправление ошибок и недочетов
	12	Разработка документации к системе
	13	Итоговое тестирование системы
4. Внедрение	14	Установка и настройка программного продукта

4.2 Расчет трудоемкость выполнения работ

Трудоемкость разработки программной продукции зависит от ряда факторов, основными из которых являются следующие:

— степень новизны разрабатываемого программного продукта;

— сложность алгоритма его функционирования;

— объем используемой информации, вид ее представления и способ обработки;

— уровень используемого алгоритмического языка программирования.

Разрабатываемый программный продукт можно отнести:

— по степени новизны — к категории В. Разработка программной продукции имеющей аналоги.

— по степени сложности алгоритма функционирования — к 1-ой группе (программная продукция реализующая моделирующие алгоритмы).

Трудоемкость разработки программного продукта $\tau_{\text{ПП}}$ может быть определена как сумма величин трудоемкости выполнения отдельных стадий разработки ПП из выражения 4.1.

$$\tau_{\text{ПП}} = \tau_{\text{ТЗ}} + \tau_{\text{ЭП}} + \tau_{\text{ТП}} + \tau_{\text{РП}} + \tau_{\text{В}} \quad (4.1)$$

, где

$\tau_{\text{ТЗ}}$ — трудоемкость разработки технического задания; $\tau_{\text{ЭП}}$ — трудоемкость разработки эскизного проекта; $\tau_{\text{ТП}}$ — трудоемкость разработки технического проекта; $\tau_{\text{РП}}$ — трудоемкость разработки рабочего проекта; $\tau_{\text{В}}$ — трудоемкость внедрения.

Трудоемкость разработки технического задания рассчитывается по формуле 4.2

$$\tau_{\text{ТЗ}} = T_{\text{ЗРЗ}} + T_{\text{ЗРП}} \quad (4.2)$$

, где

$T_{\text{ЗРЗ}}$ — затраты времени разработчика постановки задач на разработку ТЗ, чел.-дни; $T_{\text{ЗРП}}$ — затраты времени разработчика ПО на разработку ТЗ, чел.-дни.

Значения величин $T_{\text{ЗРЗ}}$ и $T_{\text{ЗРЗ}}$ рассчитываются по формулам 4.3 и 4.4.

$$T_{\text{ЗРЗ}} = t_{\text{З}} \cdot K_{\text{ЗРЗ}} \quad (4.3)$$

$$T_{\text{ЗРП}} = t_{\text{З}} \cdot K_{\text{ЗРП}} \quad (4.4)$$

, где

$t_{\text{З}}$ — норма времени на разработку ТЗ на ПП в зависимости от его функционального назначения и степени новизны, чел.-дни; $K_{\text{ЗРЗ}}$ — коэффициент, учитывающий удельный вес трудоемкости работ, выполняемых разработчиком ТЗ; $K_{\text{ЗРЗ}}$ — коэффициент, учитывающий удельный вес трудоемкости работ, выполняемых разработчиком ПО на стадии ТЗ.

$t_{\text{З}} = 24$ чел.-дн. (управление НИР)

$K_{\text{ЗРЗ}} = 0.65$ (совместная разработка)

$K_{\text{ЗРП}} = 0.35$ (совместная разработка)

$\tau_{\text{ТЗ}} = 24 \cdot 0.65 + 24 \cdot 0.35 = 24$ чел.-дн.

Аналогично рассчитывается трудоемкость эскизного проекта $\tau_{\text{ЭП}}$:

$\tau_{\text{ЭП}} = 70 \cdot 0.5 + 70 \cdot 0.5 = 70$ чел.-дн.

Трудоемкость разработки технического проекта $\tau_{\text{ТП}}$ зависит от функционального назначения ПП, количества разновидностей форм входной и выходной информации и определяется как сумма времени, затраченного разработчиком постановки задач и разработчиком программного обеспечения по формуле 4.5.

$$\tau_{\text{ТП}} = (t_{\text{ТРЗ}} + t_{\text{ТРП}}) \cdot K_{\text{В}} \cdot K_{\text{Р}} \tag{4.5}$$

, где

$t_{\text{ТРЗ}}$ и $t_{\text{ТРП}}$ — норма времени, затрачиваемого на разработку ТП разработчиком постановки задач и разработчиком программного обеспечения соответственно, чел.-дни; $K_{\text{В}}$ — коэффициент учета вида используемой информации, $K_{\text{Р}}$ — коэффициент учета режима обработки информации.

Значение коэффициента $K_{\text{В}}$ определяется из выражения:

$$K_{\text{В}} = \frac{K_{\text{П}} \cdot n_{\text{П}} + K_{\text{НС}} \cdot n_{\text{НС}+K_{\text{Б}} \cdot n_{\text{Б}}}}{n_{\text{П}} + n_{\text{НС}} + n_{\text{Б}}} \tag{4.6}$$

где $K_{\text{П}}$, $K_{\text{НС}}$, $K_{\text{Б}}$ — значения коэффициентов учета вида используемой информации для переменной, нормативно-справочной информации и баз данных соответственно; $n_{\text{П}}$, $n_{\text{НС}}$, $n_{\text{Б}}$ — количество наборов данных переменной, нормативно-справочной информации и баз данных соответственно.

$K_{\text{П}} = 1$, $K_{\text{НС}}$ - 0.72, $K_{\text{Б}=2.08}$

$K_{\text{В}} = \frac{1 \cdot 3 + 0.72 \cdot 1 + 2.08 \cdot 0}{3+1+0} = 0.505$

$K_{\text{Р}} = 1.26$

$\tau_{\text{ТП}} = (33 + 10) \cdot 0.505 \cdot 1.26 = 28$, чел.-дн.

Трудоемкость разработки рабочего проекта $\tau_{\text{РП}}$ зависит от функционального назначения ПП, количества разновидностей форм входной информации, сложности алгоритма функционирования, сложности контроля информации, степени использования готовых программных модулей, уровня алгоритмического языка программирования и определяется по формуле:

$$\tau_{\text{РП}} = K_{\text{К}} \cdot K_{\text{Р}} \cdot K_{\text{Я}} \cdot K_{\text{З}} \cdot K_{\text{ИА}} \cdot (t_{\text{РРЗ}} + t_{\text{РРП}}) \tag{4.7}$$

где $K_{\text{К}}$ — коэффициент учета сложности контроля информации; $K_{\text{Я}}$ — коэффициент учета уровня используемого алгоритмического языка программирования; $K_{\text{З}}$ — коэффициент учета степени использования готовых программных модулей; $K_{\text{ИА}}$ — коэффициент учета вида используемой информации и сложности алгоритма ПП.

$K_{\text{К}} = 1, K_{\text{Р}} = 1.44, K_{\text{Я}} = 1, K_{\text{З}} = 0.7, t_{\text{РРЗ}} = 9$ чел.-дн., $t_{\text{РРП}} = 54$ чел.-дн., $K_{\text{П}} = 1.2, K_{\text{НС}} = 0.65, K_{\text{Б}} = 0.54$

$K_{\text{ИА}} = \frac{1.2 \cdot 3 + 0.65 \cdot 1 + 0.54 \cdot 0}{3+1+0} = 1.06$

$\tau_{\text{РП}} = 1 \cdot 1.44 \cdot 1 \cdot 0.7 \cdot 1.06 \cdot (9 + 54) = 67$

Так как при разработке ПП стадии «Технический проект» и «Рабочий проект» объединены в стадию «Техно-рабочий проект», то трудоемкость ее выполнения $\tau_{\text{ТРП}}$ определяется по формуле:

$$\tau_{\text{ТРП}} = 0.85 \cdot (\tau_{\text{ТП}} + \tau_{\text{РП}}) \tag{4.8}$$

$\tau_{\text{ТРП}} = 0.85 \cdot (28 + 67) = 91$

Трудоемкость выполнения стадии внедрения $\tau_{\text{В}}$ может быть рассчитана по формуле:

$$\tau_{\text{В}=(t_{\text{ВРЗ}}+t_{\text{ВРП}})\cdot K_{\text{К}}\cdot K_{\text{Р}}\cdot K_{\text{З}}} \tag{4.9}$$

где $t_{\text{ВРЗ}}$, $t_{\text{ВРП}}$ — норма времени, затрачиваемого разработчиком постановки задач и разработчиком ПО соответственно на выполнение процедур внедрения ПП, чел.-дни.

$\tau_{\text{В}} = (10 + 11) \cdot 1 \cdot 1.26 \cdot 0.7 = 19$ чел.-дн.

Подставив полученные данные в формулу 4.1 получим:

$\tau_{\text{ПП}} = 24 + 70 + 91 + 19 = 204$ чел.-дн.

4.3 Расчет количества исполнителей

Средняя численность исполнителей при реализации проекта разработки и внедрения ПО определяется соотношением:

$$N = \frac{Q_{\text{Р}}}{F} \tag{4.10}$$

Таблица 4.2 — Распределение трудоемкости по стадиям разработки проекта

Этап	Трудоемкость этапа, чел.-дн.	№	Содержание работы	Трудоемкость, чел.-дн.
1	24	1	Постановка задачи, разработка ТЗ	20
		2	Выбор средств проектирования и разработки	4
2	70	3	Разработка структуры системы	15
		4	Разработка алгоритмов описания модели и моделирования	30
		5	Разработка вспомогательных алгоритмов	15
		6	Разработка пользовательского интерфейса	10
3	91	7	Реализация алгоритмов описания модели и моделирования	23
		8	Реализация вспомогательных алгоритмов	16
		9	Реализация пользовательского интерфейса	10
		10	Отладка программного продукта	12
		11	Исправление ошибок и недочетов	15
		12	Разработка документации к системе	8
		13	Итоговое тестирование системы	7
4	19	14	установка и настройка ПП	19
			Итого:	204

где $Q_{\text{Р}}$ — затраты труда на выполнение проекта (разработка и внедрение ПО); F — фонд рабочего времени.

Величина фонда рабочего времени определяется соотношением:

$$F = T \cdot F_M \tag{4.11}$$

где T — время выполнения проекта в месяцах, равное 4 месяцам; F_M — фонд времени в текущем месяце, который рассчитывается из учета числа дней в году, числа выходных и праздничных дней:

$$F_M = \frac{t_{\text{р}} \cdot (D_{\text{К}} - D_{\text{В}} - D_{\text{П}})}{12} \tag{4.12}$$

где $t_{\text{р}}$ — продолжительность рабочего дня; $D_{\text{К}}$ — общее число дней в году; $D_{\text{В}}$ — число выходных дней в году; $D_{\text{П}}$ — число праздничных дней в году.

$F_M = \frac{8 \cdot (365 - 103 - 10)}{12} = 168$

$F = 4 \cdot 168 = 672$

$N = \frac{204 \cdot 8}{672} = 3$ — число исполнителей проекта.

4.4 Календарный план-график

Таблица 4.3 — Планирование разработки

Стадия разработки	Трудоемкость	Должность исполнителя	Распределение трудоемкости	Численность
Техническое задание	24	Ведущий программист Программист 1	18(75%) 6(25%)	1 1
Эскизный проект	70	Ведущий программист Программист 1 Программист 2	26(37%) 27(39%) 17(24%)	1 1 1
Технорабочий проект	91	Ведущий программист Программист 1 Программист 2	36(40%) 32(35%) 23(25%)	1 1 1
Внедрение	19	Ведущий программист Программист 2	8(42%) 11(58%)	1 1

Таблица 4.4 — Календарный ленточный график работ

Стадия разработки	Должность исполнителя	Трудоемкость			
Техническое задание	Ведущий программист Программист 1	18 6			
Эскизный проект	Ведущий программист Программист 1 Программист 2		26 27 7 7		
Технорабочий проект	Ведущий программист Программист 1 Программист 2			36 32 23	
Внедрение	Ведущий программист Программист 1				9 10

Из таблицы 4.4 видно, что благодаря параллельной работе ведущего программиста и программистов можно добиться сокращения сроков разработки с 204 дней до $18 + 27 + 36 + 11 = 92$ дней, т.е. в 2.2 раза.

4.5 Расчет затрат на разработку ПП

Затраты на выполнение проекта состоят из затрат на заработную плату исполнителям, затрат на закупку или аренду оборудования, затрат на организацию рабочих мест и затрат на накладные расходы. В табли-

це 4.5 приведены затраты на заработную плату и страховые взносы во внебюджетные фонды. Суммарно эти отчисления для организаций, осуществляющих деятельность в области информационных технологий, составляют 14%: в Пенсионный фонд (ПФ) — 8%, в Фонд социального страхования (ФСС), Федеральный и территориальный фонды обязательного медицинского страхования (ФФОМС и ТФОМС) — по 2%. Для ведущего программиста предполагается ставка 2500 рублей за полный рабочий день, для первого программиста — 1500 рублей, для второго — 1000 рублей.

Таблица 4.5 — Затраты на зарплату и отчисления во внебюджетные фонды

	Февраль					
Исполнитель	Рабочих дней	Зарплата	ПФ	ФСС	ФФОМС	ТФОМС
Ведущий программист	20	50000	4000	1000	1000	1000
Программист 1	20	30000	2400	600	600	600
Программист 2	19	19000	1520	380	380	380
Итого:						112860
	Март					
Исполнитель	Рабочих дней	Зарплата	ПФ	ФСС	ФФОМС	ТФОМС
Ведущий программист	22	55000	4400	1100	1100	1100
Программист 1	22	33000	2640	660	660	660
Программист 2	6	6000	480	120	120	120
Итого:						107160
	Апрель					
Исполнитель	Рабочих дней	Зарплата	ПФ	ФСС	ФФОМС	ТФОМС
Ведущий программист	21	52500	4200	1050	1050	1050
Программист 1	21	31500	2520	630	630	630
Программист 2	10	10000	800	200	200	200
Итого:						107160
	Май					
Исполнитель	Рабочих дней	Зарплата	ПФ	ФСС	ФФОМС	ТФОМС
Ведущий программист	20	50000	4000	1000	1000	1000
Программист 1	20	30000	2400	600	600	600
Программист 2	19	19000	1520	380	380	380
Итого:						112860
	Июнь					
Исполнитель	Рабочих дней	Зарплата	ПФ	ФСС	ФФОМС	ТФОМС
Ведущий программист	16	40000	3200	800	800	800
Программист 1	15	22500	1800	450	450	450
Программист 2	3	3000	240	60	60	60
Итого:						74670
Общая сумма:						446310

Расходы на материалы, необходимые для разработки ПП указаны в таблице 4.6.

Таблица 4.6 — Материальные затраты

№	Наименование материала	Единица измерения	Кол-во	Цена за единицу, руб.	Сумма, руб.
1	Бумага А4	Пачка 500 л.	1	150	150
2	Картридж для лазерного принтера Brother HL-2030R	Шт.	2	1400	2800
				Итого:	2950

При разработке понадобится 3 компьютера (по одному на человека), один принтер, компьютерная мебель (3 комплекта), аксессуары для компьютеров (3 компьютера). Стоимость одной ПЭВМ составляет 35000 рублей. Месячная норма амортизации $K = 4\%$ (Срок службы — 25 месяцев). Срок службы составляет 3 года, месячная амортизация $K = 2.78\%$. Срок службы компьютерной мебели: столы — 5 лет, стулья — 2 года, месячные нормы амортизации — $K = 1.67\%$ и $K = 4.17\%$ соответственно. компьютерные аксессуары имеют срок службы 2 года, норма амортизации $K = 4.17\%$.

Затраты на амортизацию приведены в таблице 4.7

Таблица 4.7 — Амортизационные отчисления

Наименование оборудования	Балансовая цена, руб.	Кол-во, шт.	К, %	Время использования, мес.	Сумма отчислений, руб.
Компьютер	35000	3	4	4	16800
Принтер	2600	1	2.78	4	290
Стол	3000	3	1.67	4	600
Стул	1000	3	4.17	4	500
Клавиатура	500	3	4.17	4	350
Мышь	500	3	4.17	4	350
				Итого:	18790

Также необходимо учесть арендную плату за помещение, которая составляет 14000 рублей за месяц. За 4 месяца на аренду помещения уйдет 56000 рублей.

Общие затраты на разработку ПП составят $446310 + 2950 + 18790 + 56000 = 524050$ рублей.

4.6 Расчет экономической эффективности

Основными показателями экономической эффективности является чистый дисконтированный доход (ЧДД) и сок окупаемости вложенных средств.

Чистый дисконтированный доход определяется по формуле:

$$\text{ЧДД} = \sum_{t=0}^{T} R_t - \text{З}_t \cdot \frac{1}{(1+E)^t} \tag{4.13}$$

где T — горизонт расчета по месяцам; t — период расчета; R_t — результат достигнутый на шаге (стоимость)t; З_t — затраты; E — приемлемая для инвестора норма прибыли на вложенный капитал.

Коэффициент E установим равным ставке рефинансирования ЦБ РФ — 8% годовых (0.67% в месяц). В результате анализа рынка программной продукции, аналогичной разрабатываемой, планируется продажа 2 единиц в 1-й месяц и 3 единицы в остальные. Планируемая цена ПП составляет 60000 рублей. Предполагаемые накладные расходы, связанные с реализацией составят 2000 рублей в месяц.

В таблице 4.8 приведен расчет ЧДД по месяцам работы над проектом, а на Рисунке 4.1 изображен график ЧДД (начерчен до момента, когда ЧДД принимает положительные значения).

Таблица 4.8 — Расчет ЧДД

Месяц	Текущие затраты, руб.	Затраты с начала года, руб.	Текущий доход, руб.	Доход с начала года, руб.	ЧДД, руб.
1	66451	66451	0	0	-66451
2	126201	192652	0	0	-192652
3	126201	318853	0	0	-318853
4	131901	450754	0	0	-450754
5	73296	524050	60000	60000	-464050
6	2000	526050	180000	240000	-286050
7	2000	528050	180000	420000	-108050
7	2000	530050	180000	600000	69950

Срок окупаемости проекта — 8 месяцев.

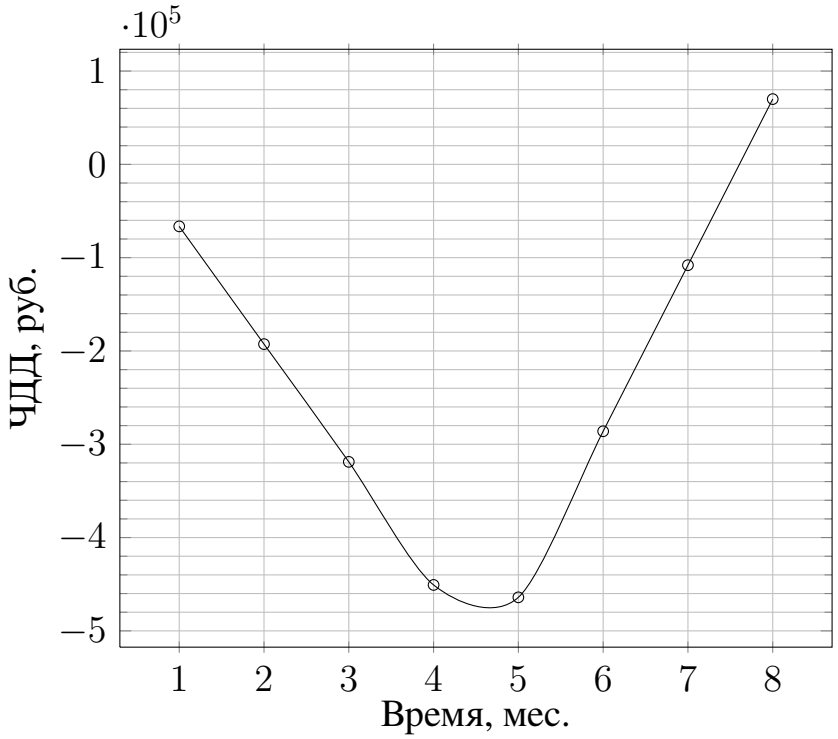

Рисунок 4.1 — График изменения ЧДД

4.7 Выводы

По результатам расчета трудоемкости выполнения работ, затрат на выполнение работ и прибыли от реализации программного продукта был найден чистый дисконтированный доход по месяцам разработки и реализации программного продукта. На основе полученных данных можно сделать вывод, что проект эффективен и срок его окупаемости составляет 8 месяцев с начала разработки.

5 Раздел по охране труда

5.1 Гигиенические требования к персональным ЭВМ и организации работы

Разработка ПО требует длительного взаимодействия с вычислительными системами. Работа с ПЭВМ связана с рядом вредных и опасных факторов, таких как статическое электричество, рентгеновское излучение, электромагнитные поля, блики отраженный свет, ультрафиолетовое излучение. При длительном воздействии на организм эти факторы негативно влияют на здоровье человека.

5.1.1 Микроклимат

Работа как программиста, так и пользователя относится к категории 1а, поскольку не предполагает больших физических усилий. Нормы, установленные СанПиН 2.2.2/2.4.1340-03 для категории работ 1а приведены в таблице 5.1.

Таблица 5.1 — Нормы микроклимата

Период год	Температура, °С		Влажность, %		Скорость воздуха, м/с	
	Оптим.	Допуст.	Оптим.	Допуст.	Оптим.	Допуст.
Холодный	22–24	21–25	40–60	75	0.1	0.1
Теплый	23–25	22–28	40–60	55 при 28°С	0.1	0.1

Вредным фактором при работе с ЭВМ является также запыленность помещения. Этот фактор усугубляется влиянием на частицы пыли электростатических полей персональных компьютеров.

Для устранения несоответствия параметров указанным нормам проектом предусмотрено использование системы кондиционирования как наиболее эффективного и автоматически функционирующего средства.

Нормы установленные содержания в воздухе положительных и отрицательной ионов, установленные СанПиН 2.2.4.1294–03, приведены в таблице 5.2.

Для обеспечения требуемых уровней предусмотрено использование системы ионизации Сапфир-4А.

Таблица 5.2 — Уровни ионизации воздуха при работе на ПЭВМ

Уровни	Число ионов в кубометре воздуха	
	n^+	n^-
Минимально необходимое	400	600
Оптимальное	1500–3000	3000–5000
Максимально допустимое	50000	50000

5.1.2 Шум и вибрации

Уровень шума на рабочем месте программиста не должен превышать 50 дБА, а уровень вибрации не должен превышать норм установленных СанПиН 2.2.2.542–96 (см. таблицу 5.3).

Таблица 5.3 — Допустимые нормы вибрации на рабочих местах с ПЭВМ

Среднегеометрические частоты октавных полос, Гц	Допустимые значения по виброскорости	
	м/с	дБ
2	4.5×10	79
4	2.2×10	73
8	1.1×10	67
16	1.1×10	67
31.5	1.1×10	67
63	1.1×10	67
Корректированные значения и их уровни в дБ	2.0×10	72

При разработке ПО внутренними источниками шума являются вентиляторы, а также принтеры и другие периферийные устройства ЭВМ. Внешние источники шума — прежде всего, шум с улицы и из соседних помещений. Постоянные внешние источники шума, превышающего нормы, отсутствуют.

Для устранения превышения нормы проектом предусмотрено применение звукопоглощающих материалов для облицовки стен и потолка помещения, в котором осуществляется работа с вычислительной техникой.

5.1.3 Освещение

Наиболее важным условием эффективной работы программистов и пользователей является соблюдение оптимальных параметров системы освещения в рабочих помещениях.

Естественное освещение осуществляется через светопроемы, ориентированные в основном на север и северо-восток (для исключения попадания прямых солнечных лучей на экраны компьютеров) и обеспечивает коэффициент естественной освещенности (КЕО) не ниже 1.5%.

В качестве искусственного освещения проектом предусмотрено использование системы общего освещения. в соответствии с СанПин 2.2.2/2.4.1340–03 освещенность на поверхности рабочего стола должна находиться в пределах 300–500 лк. Разрешается использование светильников местного освещения для работы в документами (при этом светильники не должны создавать блики на поверхности экрана).

Правильное расположение рабочих мест относительно источников освещения, отсутствие зеркальных поверхностей и использование матовых материалов ограничивает прямую (от источников освещения) и отраженную (от рабочих поверхностей) блескость. При этом яркость светящихся поверхностей не превышает $200\frac{\text{кд}}{\text{м}^2}$, яркость бликов на экране ПЭВМ не превышает $40\frac{\text{кд}}{\text{м}^2}$, и яркость потолка не превышает $200\frac{\text{кд}}{\text{м}^2}$.

В соответствии с СанПин 2.2.2/2.4.1340–03 проектом предусмотрено использование люминесцентных ламп типа ЛБ в качестве источников света при искусственном освещении. В светильниках допускается применение ламп накаливания. Применение газоразрядных ламп в светильниках общего и местного освещения обеспечивает коэффициент пульсации не более 5%.

Таким образом, проектом обеспечиваются оптимальные условия освещения рабочего помещения.

5.1.4 Рентгеновское излучение

В соответствии с СанПиН 2.2.2/2.4.1340-03 проектом предусмотрено использование ПЭВМ, конструкция которого обеспечивает мощность

экспозиционной дозы рентгеновского излучения в любой точке на расстоянии 0.5 м. от экрана и корпуса не более 0.1 мбэр/час (100 мкР/час). Результаты сравнения норм излучения приведены в таблице 5.4.

Таблица 5.4 — Сравнение норм рентгеновского излучения в различных стандартах

	Допустимое значение мкР/час, не более
СанПиН 2.2.2/2.4.1340-03	100
ТСО-99	500
MPR II	500

Как видно из таблицы, стандарты MPR II и ТСО–99 предъявляют менее жесткие требования к рентгеновскому излучению, чем СанПиН. Но при соблюдении оптимального расстояния между пользователем и монитором дозы рентгеновского излучения не опасны для большинства людей.

5.1.5 Неионизирующие электромагнитные излучения

Допустимые значения параметров неионизирующих излучений в соответствии с СанПин 2.2.2/2.4.1340-03 приведены в таблицах 5.5 и 5.6.

Таблица 5.5 — Предельно допустимые значения напряженности электрического поля

Диапазон частот	Допустимые значения
5 Гц – 2 кГц	25 В/м
2 – 400 кГц	2.5 В/м

Таблица 5.6 — Предельно допустимые значения плотности магнитного потока

Диапазон частот	Допустимые значения
5 Гц – 2 кГц	250 нТл
2 – 400 кГц	5 нТл

Величина поверхностного электрического потенциала не должна превышать 500 В.

Мониторы, используемые в настоящее время, удовлетворяют более жестким нормам MPR II, а значит и СанПиН.

5.1.6 Визуальные параметры

Неправильный выбор визуальных эргономических параметров приводит к ухудшению здоровья пользователей, быстрой утомляемости, раздражительности. В связи с этим, проектом предусмотрено, что конструкция вычислительной системы и ее эргономические параметры обеспечивают комфортное и надежное считывание информации. Требования к визуальным параметрам, их внешнему виду, дизайну, возможности настройки представлены в СанПиН 2.2.2/2.4.1340–03. Визуальные эргономические параметры монитора и пределы из изменений приведены в таблице 5.7.

Таблица 5.7 — Визуальные эргономические параметры ВДТ и пределы из изменений

Наименование параметров	Пределы значений параметров	
	не менее	не более
Яркость экрана (фона), $\frac{кд}{м^2}$ (измеренная в темноте)	35	120
Внешняя освещенность экрана, лк	100	250
Угловой размер экрана, угл.мин.	16	60

Для выполнения этих требований проектом предусмотрено использование современных мониторов, имеющих достаточно широкий набор регулируемых параметров. В частности, для удобного считывания информации реализована возможность настройки положения монитора по горизонтали и вертикали. Мониторы оснащены специальными устройствами и средствами настройки ширины, высоты, яркости, контраста и разрешения изображения. кроме того, в современных мониторах зерно изображения имеет размер в пределах 0.27 мм, что обеспечивает высокую четкость и непрерывность изображения. Наконец, на поверхность дисплея нанесено матовое покрытие, чтобы избавиться от солнечных бликов.

5.2 Расчет искусственного освещения

При расчете освещенности от светильников общего равномерного освещения наиболее часто применяют метод расчета по световому потоку. При расчете освещения по этому методу необходимое количество светильников для освещения рабочего места рассчитывается по формуле:

$$N = \frac{E_{min} \cdot S \cdot K}{F_{\text{Л}} \cdot \text{З} \cdot z \cdot h} \tag{5.1}$$

где E_{min} — нормируемая минимальная освещенность; S — площадь помещения, м^2; $F_{\text{Л}}$ — световой поток лампы, лк; K — коэффициент запаса; z — коэффициент неравномерности освещения (для люминесцентных ламп —1.1); h — коэффициент использования светового потока в долях единицы.

E_{min} определяется на основании нормативного документа СНиП23–05–95. В соответствии с произведенным выбором в предыдущем разделе, для работы программиста $E_{min} = 300$ лк (общее освещение).

Работы, производятся в помещении, требуют различения цветных объектов при невысоких требованиях к цветоразличению, поэтому в качестве источника освещения была выбрана лампа люминесцентная холодно-белая (ЛХБ), 1940 лк, 30 Вт. В помещениях общественных и жилых зданий с нормальными условиями среды: К=1.4.

Для люминесцентных ламп коэффициент неравномерности освещения Z=1.1.

Коэффициент использования h зависит от типа светильника, от коэффициентов отражения потолка $\rho_{\text{П}}$, стен $\rho_{\text{С}}$, расчетной поверхности $\rho_{\text{Р}}$ и индекса помещения.

Высота подвеса над рабочей поверхностью Нр=3 м. Размеры помещения А=3.5 м, В=3 м. Определим индекс помещения по формуле:

$$\phi = \frac{A \cdot B}{H_P \cdot (A + B)} = \frac{3.5 \cdot 3}{s \cdot (3.5 + 3)} = 0.54 \tag{5.2}$$

Для светлого фона примем:$\rho_{\text{П}} = 70$ $\rho_{\text{С}} = 50$ $\rho_{\text{Р}} = 10$. h = 59%.

Освещение проектируется при помощи светильников ОДОР с минимальной освещенностью $E_{min} = 300$ лк, P=40 Вт. Число ламп в ОДОР равно 2. Необходимое число светильников для данной комнаты:

$$N = \frac{300 \cdot 9 \cdot 1.4}{1940 \cdot 0.59 \cdot 1.1 \cdot 2} = 2\text{шт} \tag{5.3}$$

Общее количество ламп $n = 2 \times 2 = 4$ шт. Длина светильника ОДОР=1.26 м. Поскольку длина помещения 3 м, то светильники помещаются в два ряда. схема размещения светильников показана на Рисунке 5.1.

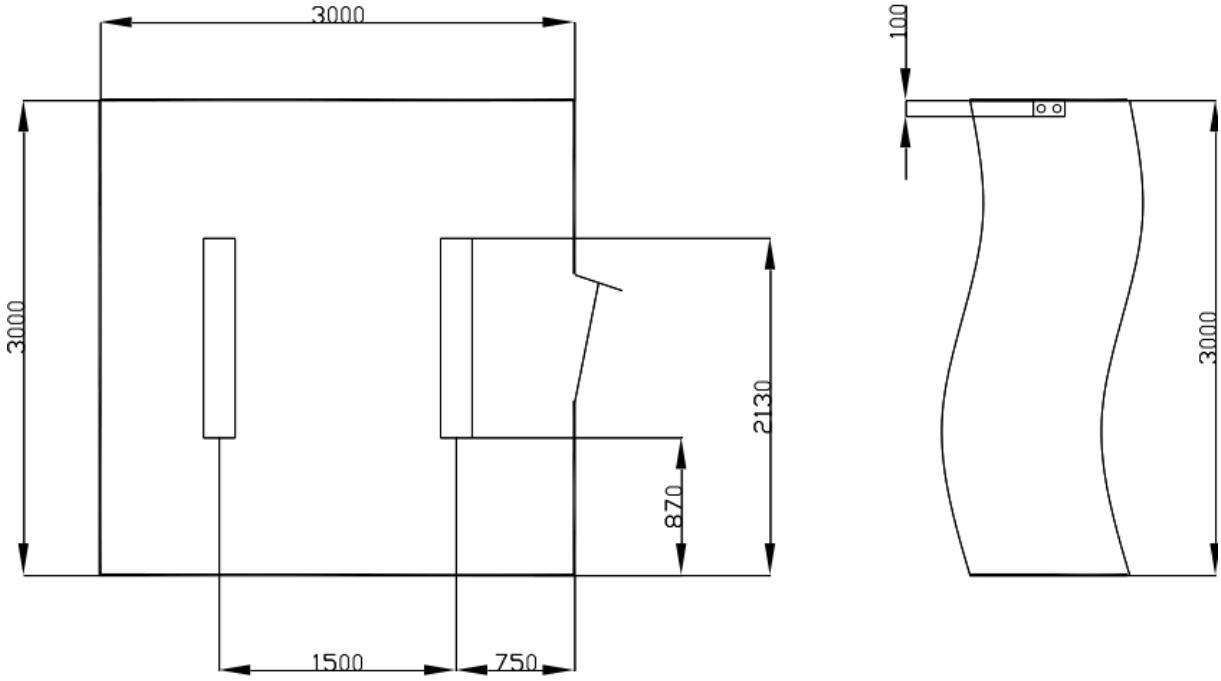

Рисунок 5.1 — План размещения светильников в машинном зале

Суммарная мощность светильников: $30 \cdot 4 = 160$ Вт. Суммарный световой поток: $1940 \cdot 4 = 7760$ лм.

Режим труда и отдыха должен зависеть от характера работы: при вводе данных, редактировании программ, чтении информации с экрана непрерывная продолжительность работы с монитором не должна превышать 4 часов. При 8 часовом рабочем дне, через каждый час работы необходимо проводить перерыв 5–10 минут, а каждые два часа перерыв в 15 мин.

Заключение

В результате проделанной работы был проведен обзор системы моделирования GPSS и решены следующие задачи:

— Обоснован выбор подмножества блоков GPSS, реализуемых в работе.

— Разработан и реализован алгоритм формирования моделей.

— Реализованы алгоритмы имитационного моделирования сформированных моделей.

— Разработаны аналитическая и имитационная модель типовой системы.

— Реализована библиотека имитационного моделирования.

— Реализована демонстрационная программа, позволяющая оценить возможности разработанной библиотеки и оценить точность выбранных методов.

— Проведено сравнение результатов аналитического и имитационного моделирования тестовой системы, что подтвердило корректность реализации разработанной библиотеки.

Список использованных источников

1. *Квитка М. Е. Сёмкин Ю. Ю., Томила С. О.* Разработка свободного аналога языка GPSS. — 2008.

2. *Р.В., Душкин*. Справочник по языку Haskell. / Душкин Р.В. — М.: ДМК Пресс, 2008.

3. *Р.В., Душкин*. Функциональное программирование на языке Haskell. / Душкин Р.В. — М.: ДМК Пресс, 2007.

4. *Томашевский В., Жданова Е.* Имитационное моделирование в среде GPSS. / Жданова Е. Томашевский В. — М.: Бестселлер, 2003.

5. GPSS Wrorld Reference Manual. — 2009.

6. *Куров А.В., Рудаков И.В.* Определение показателей производительности вычислительных систем методами теории массового обслуживания . / Рудаков И.В. Куров А.В. — М.: Изд-во МГТУ им. Н.Э. Баумана, 2005.

7. Архив библиотек языка Haskell. — http://hackage.haskell.org/packages/hackage.html.

8. Архив библиотек языка Haskell. — http://www.haskell.org/hoogle/.

9. Официальный сайт gnuplot. — http://www.gnuplot.info/.

10. Официальная страница wxHaskell. — http://www.haskell.org/haskellwiki/Wxhaskell.

11. HUnit 1.0 User's Guide. — 2002.