



**«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

РАСЧЁТНО - ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

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)

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. Техничко-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.

КАЛЕНДАРНЫЙ ПЛАН 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	6
1 Аналитический раздел	8
1.1 Краткий обзор GPSS	8
1.2 Объекты языка GPSS	8
1.3 Управления процессом моделирования в GPSS	10
1.4 Выбор подмножества реализуемых блоков	11
1.5 Описание выбранных блоков	15
1.6 Аналитическая модель системы	17
1.6.1 Моделирование отказов и восстановлений	17
1.6.2 Укрупнение модели	19
1.6.3 Окончательная расчетная схема	22
1.7 Выводы	22
2 Конструкторский раздел	23
2.1	23
2.2 Монады	23
2.3 Нотация do	25
2.4 Монада State	26
2.5 Описание модели как вычислене с состоянием	27
2.6 Функции формирования блоков	28
2.7 Перегрузка функций	28
3 Технологический раздел	29
4 Организационно-экономический раздел	30
4.1 Формирование состава выполняемых работ и группировка их по стадиям разработки	30
4.2 Расчет трудоемкость выполнения работ	31
4.3 Расчет количества исполнителей	34
4.4 Календарный план-график	36
Список использованных источников	37

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 попрежнему остается имитационное моделирование.

Язык имитационного моделирования GPSS 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 был выбран 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;
- разработать и реализовать транслятор моделей GPSS в формат разработанной системы моделирования;
- провести тестирование разработанного программного обеспечения;
- провести моделирование некоторой эталонной системы массового обслуживания в разработанной системе, GPSS 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 блоков, которые следует реализовать в разрабатываемой системе моделирования.

1.1 Краткий обзор GPSS

GPSS стал одним из первых языков моделирования, облегчающих процесс написания имитационных программ. Он был создан в виде конечного продукта Джеффри 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.

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:

- параметры — набор значений связанных с транзактом. Каждый транзакт может иметь произвольное число параметров. Каждый параметр имеет уникальный номер, по которому на него можно сослаться;
- приоритет — определяет порядок продвижения транзактов при конкурировании за общий ресурс;
- текущий блок — номер блока, в котором транзакт находится в данный момент;

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

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

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

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

Диаграмма состояний транзакта показана на Рисунке 1.1.

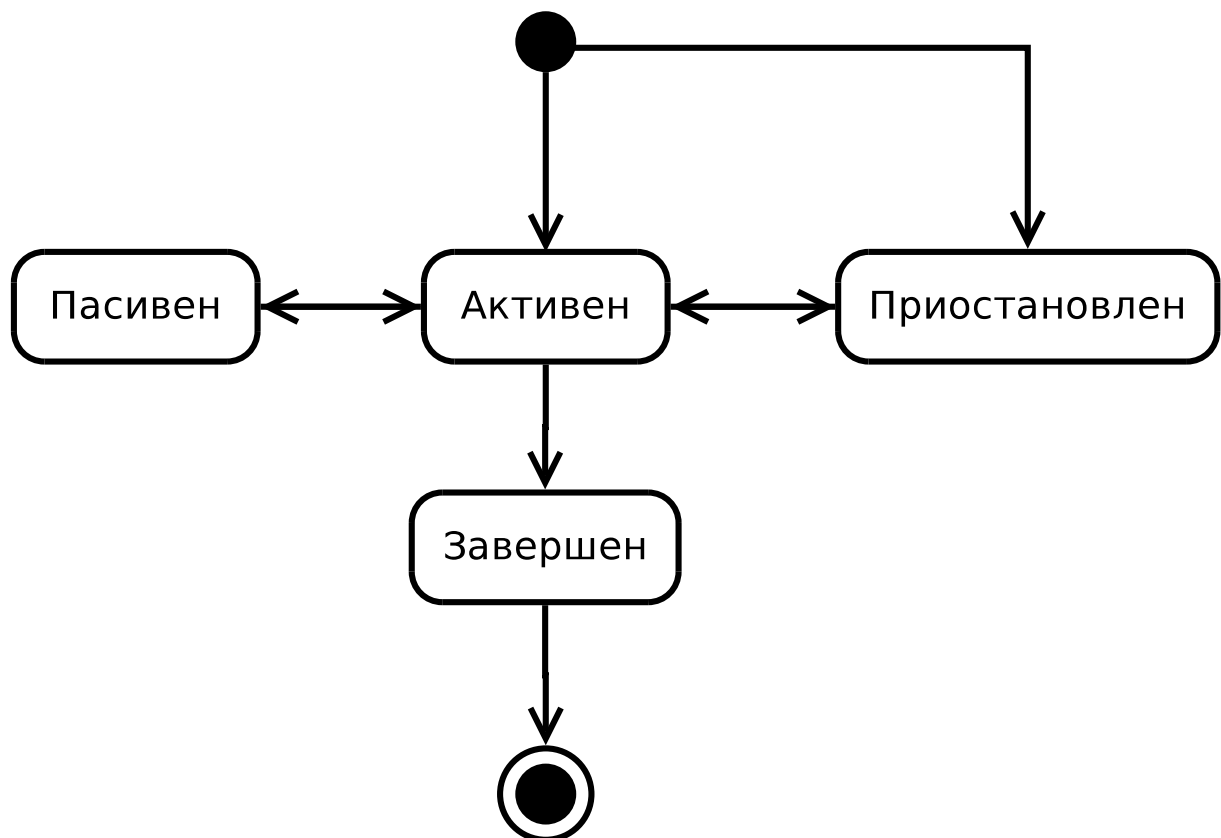


Рисунок 1.1 — Состояния транзакта

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.

1.3 Управление 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:

— 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;

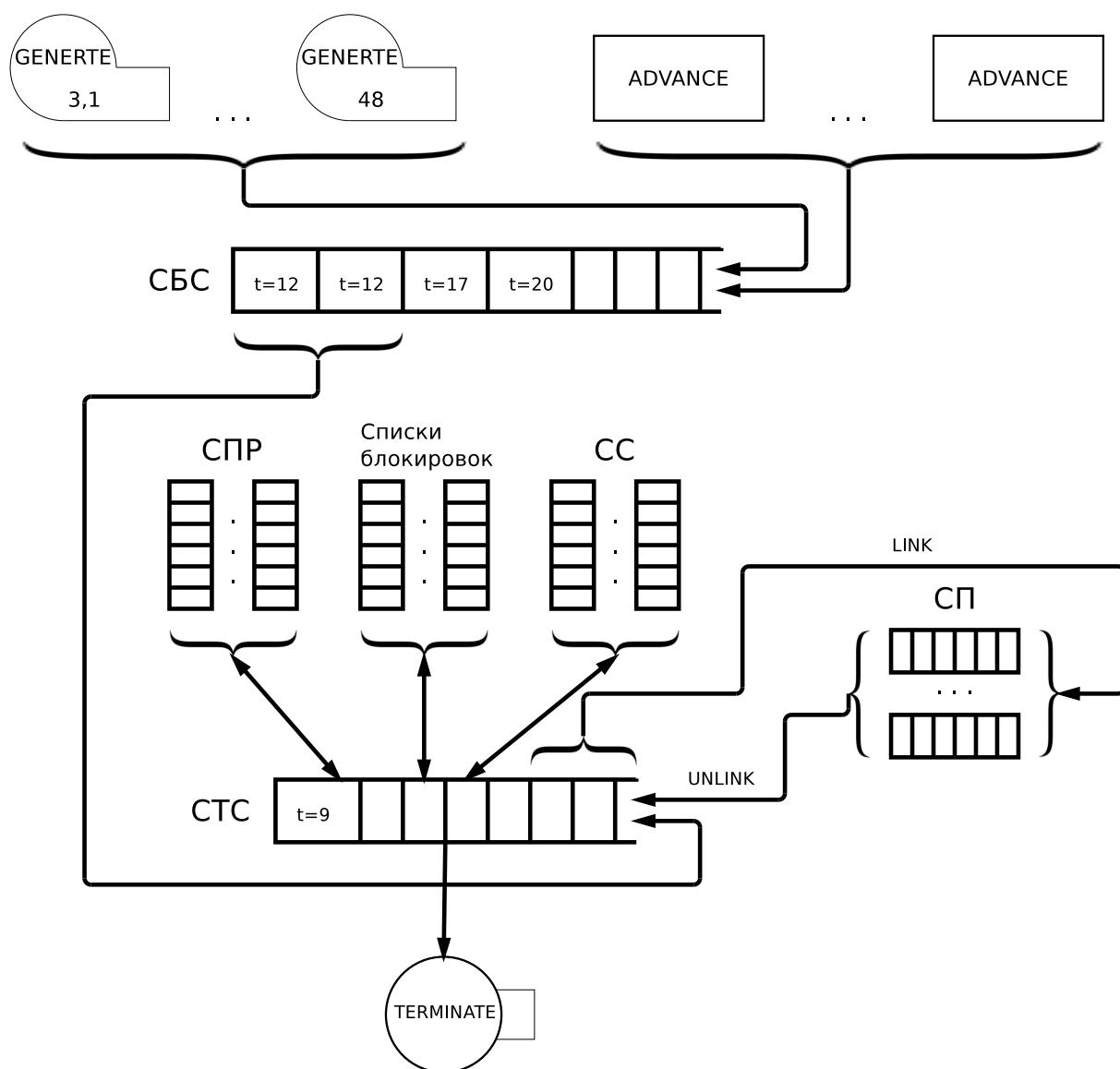


Рисунок 1.2 — Списки GPSS

— 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 с 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 сле-

дует 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 беспriorитетной дисциплиной. Интенсивность отказов, восстановлений, средние времена обработки сообщения и среднее время обдумывания также известны.

Схема модели данной системы показана на Рисунке 1.3

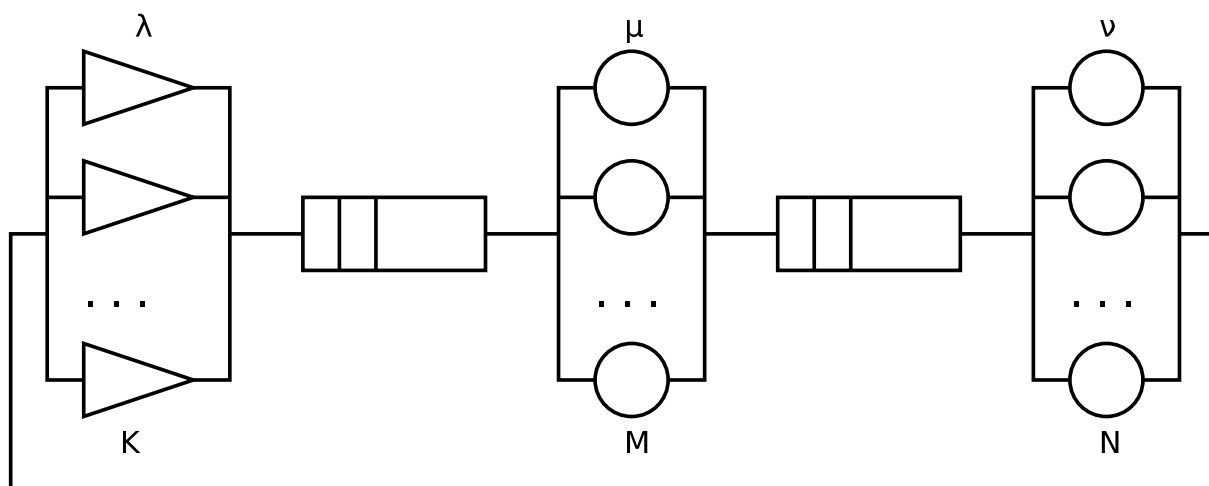


Рисунок 1.3 — Схема моделируемой системы

Как и подающее 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 SAVAIL и SUNAVAIL 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
4 ;Общее время моделирования
5 GENERATE , , , 1
6 ADVANCE 1000
7 TERMINATE 1
8
9 ;Основная часть модели
10 GENERATE , , , 10
11
12 ;фаза обдумывания
13 LUSER ADVANCE 4,1
14     TRANSFER ALL, LCPU1, LCPUN, 4
15
16 ;Первая фаза обработки
17 LCPU1 SEIZE CPU1
18     ADVANCE 2,1
19     RELEASE CPU1
20     TRANSFER , LPHASE2
21
22 ...
23
24 LCPUN SEIZE CPUN
25     ADVANCE 2,1
26     RELEASE CPUN
27     TRANSFER , LPHASE2
28
29 LPHASE2 TRANSFER ALL, LCHAN1, LCHANM, 4
30
31 ;Вторая фаза обработки
32 LCHAN1 SEIZE CHAN1

```

```

33      ADVANCE 2,1
34      RELEASE CHAN1
35      TRANSFER ,LUSER
36
37      ...
38
39 LCHANM SEIZE CHANM
40      ADVANCE 2,1
41      RELEASE CHANM
42      TRANSFER ,LUSER
43
44 ;Моделирование отказов и восстановлений.
45      GENERATE ,,,1,10
46 CPUBROKE1 ADVANCE 20,4
47      PREEMPT CPU1
48      ENTER REPAIRERS
49      ADVANCE 10,4
50      LEAVE REPAIRERS
51      RETURN CPU1
52      TRANSFER ,CPUBROKE1
53
54      ...
55
56      GENERATE ,,,1,10
57 CPUBROKEN ADVANCE 20,4
58      PREEMPT CPUN
59      ENTER REPAIRERS
60      ADVANCE 10,4
61      LEAVE REPAIRERS
62      RETURN CPUN
63      TRANSFER ,CPUBROKEN
64
65      GENERATE ,,,1,10
66 CHANBROKE1 ADVANCE 20,4
67      PREEMPT CHAN1
68      ENTER REPAIRERS
69      ADVANCE 10,4
70      LEAVE REPAIRERS
71      RETURN CHAN1
72      TRANSFER ,CHANBROKE1
73
74      ...
75
76      GENERATE ,,,1,10
77 CHANBROKEN ADVANCE 20,4
78      PREEMPT CHANM

```

79	ENTER REPAIRERS
80	ADVANCE 10,4
81	LEAVE REPAIRERS
82	RETURN CHANM
83	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.

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

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

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

Д — Ограничение на максимальное допустимое число созданных транзактов. Не обязательный параметр. По умолчанию ограничение отсутствует.

Е — 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.

А — Имя или номер многоканального устройства. Обязательный параметр.

В — 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, в случае если устройство занято, а приоритет поступившего транзакта выше, чем у обслуживающегося в данный момент, обслуживающийся транзакт вытесняется с устройства. Его дальнейшее поведение определяется параметрами блока PREEMPT.

RELEASE A

Блок RELEASE освобождает одноканальное устройство.

А — 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.

1.6 Аналитическая модель системы

Для того, чтобы протестировать разработанную систему моделирования, целесообразно (помимо прочего) разработать аналитическую модель, позволяющую вычислить искомые характеристики системы, с целью сравнить их с величинами, полученными при помощи имитационного моделирования. Для этого были использованы приемы и методы описанные в [6].

1.6.1 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 1.4 показана структура фрагмента графа состояний системы, где $\beta_{ij} = \beta_{i+j} \min\{i + j, L\}$, $\delta_{ij} = \delta_{i+j} \min\{i + j, L\}$.

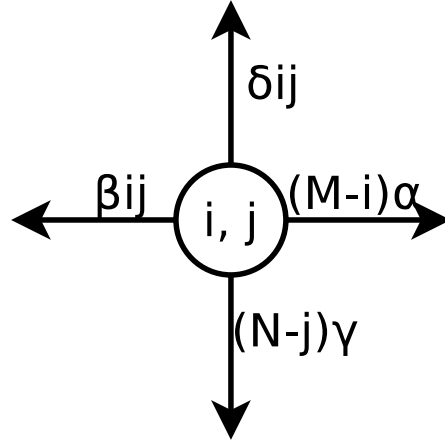


Рисунок 1.4 — Структура фрагмента графа состояний системы

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 1.5, 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.

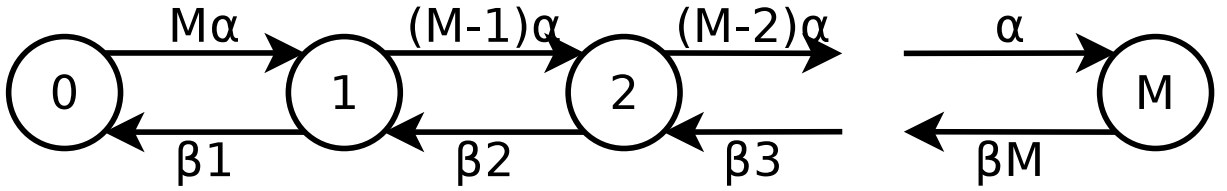


Рисунок 1.5 — Граф состояний системы

Тогда выражения для определения вероятностей стационарных состояний примут вид:

$$\left\{ \begin{array}{l} p_0 = \left(1 + \frac{M\alpha}{\beta_1} + \dots + \frac{M!\alpha^M}{\prod_{i=1}^M \beta_i} \right)^{-1} \\ p_i = p_0 \frac{\alpha^i \prod_{j=1}^i (M-j+1)}{\prod_{j=1}^i \beta_j}, \quad i = \overline{1, M} \\ \beta_i = \beta \sum_{j=0}^N \pi_j \frac{i}{i+j} \min\{i+j, L\} \end{array} \right. \quad (1.1)$$

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 1.6, а выражения для определения вероятностей стационарных состояний примут вид:

$$\left\{ \begin{array}{l} \pi_0 = \left(1 + \frac{N\gamma}{\delta_1} + \dots + \frac{N!\gamma^N}{\prod_{i=1}^N \delta_i} \right)^{-1} \\ \pi_i = \pi_0 \frac{\gamma^i \prod_{j=1}^i (N-j+1)}{\prod_{j=1}^i \delta_j}, \quad i = \overline{1, N} \\ \delta_j = \delta \sum_{i=0}^M p_i \frac{j}{i+j} \min\{i+j, L\} \end{array} \right. \quad (1.2)$$

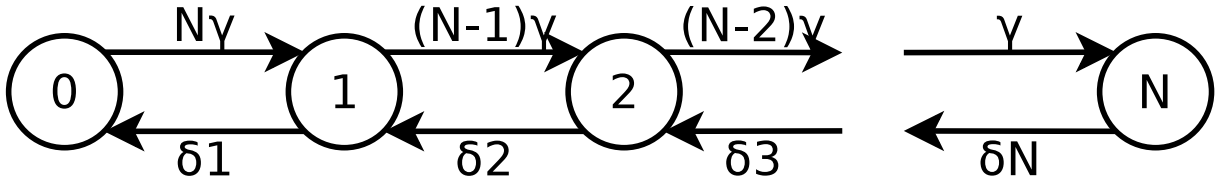


Рисунок 1.6 — Граф состояний системы

Применяя формулы 1.1 и 1.2 итеративно получим вероятности отказов процессоров и каналов в системе. В качестве начального приближения можно взять $\pi_i = \frac{1}{M}$

1.6.2 Укрупнение модели

Заменяем исходную модель агрегированной однофазной моделью АМ1 (см. рисунок 1.7). В агрегированный узел объединена подсистема, включающая в себя процессоры и каналы. Интенсивность обслуживания в этом узле зависит от числа находящихся в нем заявок.

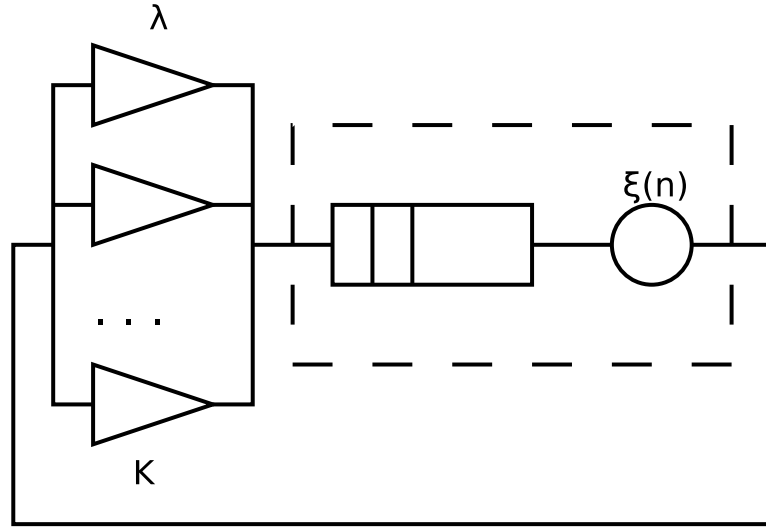


Рисунок 1.7 — Укрупненная модель AM1

Граф состояний полученной системы представлен на рисунке 1.8.
Производительность системы может быть вычислена по формулам:

$$\left\{ \begin{array}{l} \hat{\pi}_0 = \left(1 + \frac{K\lambda}{\xi_1} + \dots + \frac{K!\lambda^K}{\prod_{i=1}^K \xi_i} \right)^{-1} \\ \hat{\pi}_i = \hat{\pi}_0 \frac{\lambda^i \prod_{j=1}^i (K-j+1)}{\prod_{j=1}^i \xi_j}, \quad i = \overline{1, K} \\ \xi^* = \sum_{i=1}^K \xi_i \hat{\pi}_i \end{array} \right. \quad (1.3)$$

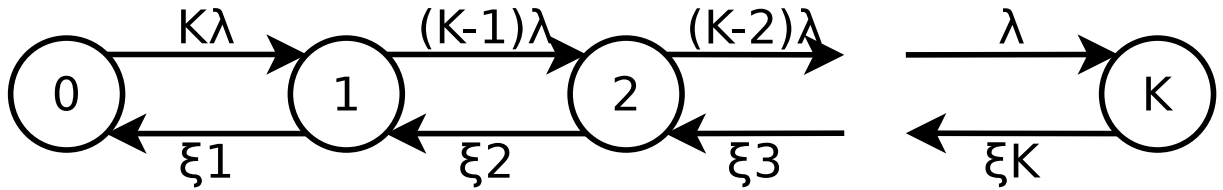


Рисунок 1.8 — Граф состояний модели AM1

Однако, чтобы воспользоваться приведенными формулами, необходимо знать параметры связи μ_i . Чтобы их найти, рассмотрим укрупнен-

ную модель AM2, структура и граф состояний которой показаны на рисунках 1.9 и 1.10. За состояние системы примем количество заявок на процессорной фазе, а интенсивности переходов могут быть выражены по формулам:

$$\mu_i = \mu \sum_{j=0}^M p_j \min \{i, M - j\} \quad (1.4)$$

$$\nu_i = \nu \sum_{j=0}^N \pi_j \min \{n - i + 1, N - j\} \quad (1.5)$$

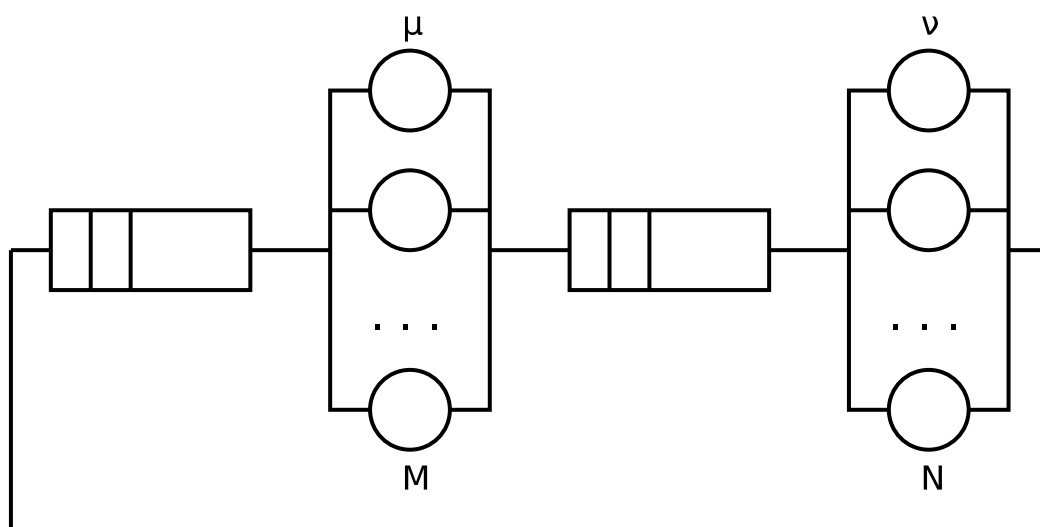


Рисунок 1.9 — Укрупненная модель AM2

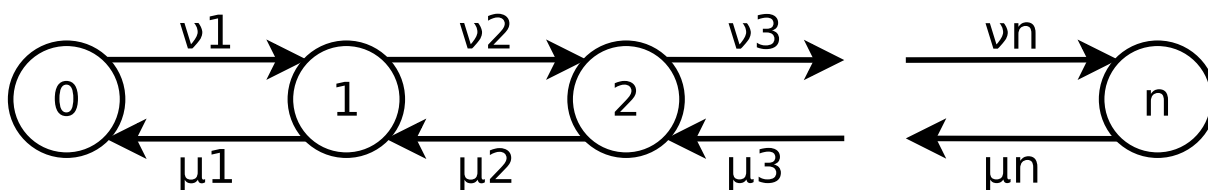


Рисунок 1.10 — Граф состояний модели AM2

Параметр связи может вычислен по следующим формулам:

$$\left\{ \begin{array}{l} \hat{p}_0 = \left(1 + \frac{\nu_1}{\mu_1} + \dots + \frac{\prod_{i=1}^n \nu_i}{\prod_{i=1}^n \mu_i} \right)^{-1} \\ \hat{p}_i = \hat{p}_0 \frac{\prod_{j=1}^i (\nu_j)}{\prod_{j=1}^i \mu_j}, \quad i = \overline{1, n} \\ \xi_n = \sum_{i=1}^n \hat{p}_i \mu_i \end{array} \right. \quad (1.6)$$

1.6.3 Окончательная расчетная схема

Последовательность расчета производительности системы должна быть следующей:

- а) По формулам 1.1 и 1.2 вычислить $\pi_i, i = \overline{0, N}$ и $p_i, i = \overline{0, M}$.
- б) По формулам 1.4, 1.5 и 1.6 вычислить $\xi_n, n = \overline{1, K}$.
- в) Вычислить ξ^* по формулам 1.3.

1.7 Выводы

Был проведен анализ устройства системы GPSS и осуществлен выбор подмножества блоков, необходимых для моделирования не сложных систем массового обслуживания. Также была построена аналитическая модель такой системы, с целью использовать ее как эталон при тестировании разрабатываемой системы моделирования.

2 Конструкторский раздел

В данном разделе проводится выбор синтаксиса описания моделей в разрабатываемой системе, а также описываются алгоритмы и структуры данных, используемые при формировании моделей и непосредственно при моделировании.

2.1

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 с 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 некоторый контекст по умолчанию, в зависимости от выбранной монады. Для списка это будет список из одного элемента, для монады IO — дей-

¹Следует 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.

ствие 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 преобразования¹:

1. `do {e} → e`

¹В 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.

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

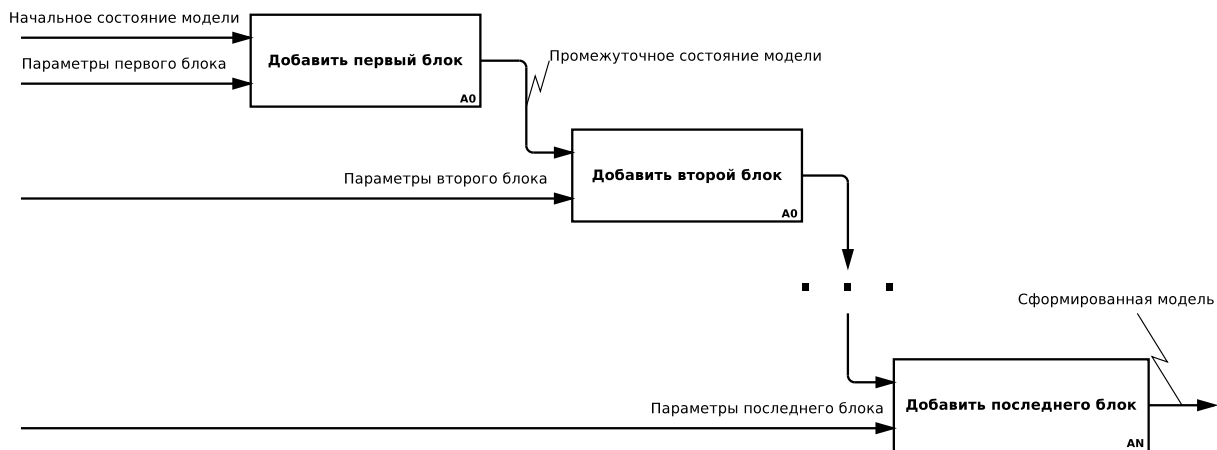


Рисунок 2.1 — Процесс формирования модели

```

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.

2.7 Перегрузка функций

¹Строго 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).

3 Технологический раздел

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

Продолжение на следующей странице

Продолжение таблицы 4.1

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

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

, где

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

Трудоемкость разработки технического задания рассчитывается по формуле 4.2

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

, где

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

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

$$T_{ЗРЗ} = t_3 \cdot K_{ЗРЗ} \quad (4.3)$$

$$T_{ЗРП} = t_3 \cdot K_{ЗРП} \quad (4.4)$$

, где

t_3 — норма времени на разработку ТЗ на ПП в зависимости от его функционального назначения и степени новизны, чел.-дни; $K_{ЗРЗ}$ — коэффициент, учитывающий удельный вес трудоемкости работ, выполняемых разработчиком ТЗ; $K_{ЗРП}$ — коэффициент, учитывающий удельный вес трудоемкости работ, выполняемых разработчиком ПО на стадии ТЗ.

$$t_3 = 24 \text{ чел.-дн. (управление НИР)}$$

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

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

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

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

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

Трудоемкость разработки технического проекта $\tau_{ТП}$ 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.

$$\tau_{ТП} = (t_{ТРЗ} + t_{ТРП}) \cdot K_B \cdot K_P \quad (4.5)$$

, где

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

Значение коэффициента K_B определяется из выражения:

$$K_B = \frac{K_{П} \cdot n_{П} + K_{НС} \cdot n_{НС} + K_B \cdot n_B}{n_{П} + n_{НС} + n_B} \quad (4.6)$$

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

$$K_{П} = 1, K_{НС} = 0.72, K_{B=2.08}$$

$$K_B = \frac{1 \cdot 3 + 0.72 \cdot 1 + 2.08 \cdot 0}{3 + 1 + 0} = 0.505$$

$$K_P = 1.26$$

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

Трудоемкость разработки рабочего проекта $\tau_{РП}$ 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:

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

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.

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

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

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

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:

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

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

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

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

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

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

Подставив полученные данные в формулу 4.1 получим:

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

4.3 Расчет количества исполнителей

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

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

$$N = \frac{Q_P}{F} \quad (4.10)$$

где Q_P — затраты труда на выполнение проекта (разработка и внедрение ПО); F — фонд рабочего времени.

Величина фонда рабочего времени определяется соотношением:

$$F = T \cdot F_M \quad (4.11)$$

где T — фвремя 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:

$$F_M = \frac{t_p \cdot (D_K - D_B - D_{\Pi})}{12} \quad (4.12)$$

где t_p — продолжительность рабочего дня; D_K — общее число дней в году; D_B — число выходных дней в году; D_{Π} — число праздничных дней в году.

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

$$F = 4 \cdot 168 = 672$$

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

4.4 Календарный план-график

Таблица 4.3 — Планирование разработки

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

Таблица 4.4 — Календарный ленточный график работ

Стадия разработки	Должность исполнителя	Трудоемкость			
		18			
Техническое задание	Ведущий программист	6			
	Программист 1				
Эскизный проект	Ведущий программист		26		
	Программист 1		27		
	Программист 2		7	7	
Технорабочий проект	Ведущий программист			36	
	Программист 1			32	
	Программист 2			23	
Внедрение	Ведущий программист				8
	Программист 1				11

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 Wworld 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.