Федеральное агентство по образованию
Федеральное государственное образовательное учреждение
высшего профессионального образования
"Сибирский федеральный университет"

Бронов Сергей Александрович

ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ

Учебное пособие

Красноярск 2007

УДК 303.094.7

Бронов, С. А. Имитационное моделирование : учеб. пособие / С. А. Бронов; ФГОУ ВПО "Сибирский федеральный университет", кафедра "Системы автоматизированного проектирования". — Красноярск: СФУ, 2007. — 82 с.

Рассмотрена методология имитационного моделирования с применением программы GPSS World. Описаны основные приёмы работы с программой, включая программирование и проведение имитационных экспериментов. Пособие является часть комплекта, включающего также пособие для самостоятельной работы, в котором собраны основные практические приёмы работы в GPSS World

Пособие предназначено для направления подготовки баклавров 230100 — "Информатика и вычислительная техника", но может применяться также для других направлений подготовки при изучении курсов, связанных с теорией операций, системами поддержки принятия решений и т. п.

.

Содержание

Введение

При исследовании реальных объектов приходится иметь дело не с самими этими объектами, а с их моделями, которые получаются после упрощения их свойств и представления в той или иной форме. При моделировании технических объектов используют математический аппарат алгебраических и дифференциальных уравнений, алгебру логики и др. При изучении организационно-технических объектов эти подходы невозможны, так как в таких объектах существунную роль играют неопределённость и возможность изменения характеристики различным образом, не всегда предсказуемым заранее. В этом случае применяют имитационное моделирование с помощью специальных программ. Первая программа имитационного моделирования появилась в 1961 году и называлась GPSS. Она оказалась столь удачной, что продолжает существовать в настоящее время и используется под современными операционными системами на персональных компьютерах. Она получила обновлённое название — GPSS World. Изучению этой программы и с её помощью — основ имитационного моделирования — посвящён курс данной учебной дисциплины.

Главным в данном курсе является развитие навыков практического применения программы GPSS World, что возможно, главным образом, при самостоятельных занятиях. Поэтому основным учебным пособием в данном курсе является пособие для самостоятельной работы, в котором на конкретных примерах показаны приёмы и возможности программирования в GPSS World.

Теоретический курс даёт некоторый дополнительный фон для рассматриваемых практических материалов, является в некотором смысле обзорным и напоминающим. В имитационном моделировании широко используются понятия из курсов учебных дисциплин "Теория вероятностей" и "Программирование". В данном курсе эти понятия не повторяются, вследствие чего студенты должны в случае необходимости воспользоваться соответствующими учебными пособиями по этим учебным дисциплинам.

1 Основные понятия и принципы имитационного моделирования

Люди всегда хотели знать заранее, как будет вести себя по отношению к ним окружающий мир. Для этого они внимательно наблюдали за объектами этого мира, пытаясь выяснить природу и характерные особенности их поведения. В дальнейшем человеку захотелось самому воздействовать на окружающий мир, чтобы обеспечить себе комфортное существование. Так родилась техника — т. е. искусственно созданная часть мира. Создавать объекты можно только, зная хорошо их будущие свойства.

Чтобы проанализировать поведение объектов, необходимо иметь для этого соответствующие условия. В частности, достаточное количество времени, хорошие инструменты наблюдения, возможность "заглянуть внутрь" объекта, затормозить в нём слишком процессы или, напротив, ускорить слишком медленные, и внимательно изучать их в таком состоянии и т. д. В реальности большинство этих требований невыполнимы. Поэтому люди научились исследовать объекты, наблюдая не за ними, а за их моделями.

Моделью называют некое подобие реального объекта, обладающее существенными для изучения свойствами (интересующими исследователя или влияющими на интересующие свойства). Самая простая модель — его развёрнутое название. Например, автомобиль — то, что само движется. Ветряная мельница — устройство, перемалывающее зерно под действием ветра. Но на самом деле, такие "квазимодели" дают слишком мало информации о свойствах объекта, а потому, кроме названия, задаются ещё разнообразные характеристики объектов. Эти характеристики можно заавать различным образом: словами, цифрами, формулами, математическими выражениями, алгоритмами. Словесное (вербальное) выражение легко формулируется, но весьма неточно и не отражает всего, связанного с объектом. Цифровое отражение характеристик объекта более конкретно и несёт существенно больше информации, но, как правило, отражает лишь конкретный вариант состояния объекта. Гораздо более информативными являются математические выражения (например, уравнения) и алгоритмы: они позволяют представить не только мгновенное состояние объекта, но и процессы в нём, т. е. характер изменения характеристик. Таким образом обеспечивается возможность предугадывания поведения объекта при выполнении некоторого множества условий.

При исследование систем можно решать *задачу анализа* или *задачу синтеза*.

Анализ предполагает исследование функционирования существующего объекта, например, технического устройства, завода или магазина. Цели такого исследования могут быть разными, например, определение возможностей производства готовой продукции, величины прибыли, уровня загруженности персонала и оборудования, отчислений по налогам и сборам, работы в кооперации с другими организационными объектами и др. Соответственно

этому могут анализироваться материальные и финансовые потоки, использование людских ресурсов и т. д.

Синтез предполагает, что исследуемый объект не существует, а создаётся заново. При этом требуется определить его структуру (состав элементов и связей между ними) и параметры. Элементами структуры могут быть станки, приборы, сотрудники, автоматизированные рабочие места, склады, транспорты и др. Параметрами является количество сотрудников, площадь складских помещений, размеры финансовых средств и др. При синтезе ставится задача выбрать такую структуру и такие параметры, чтобы гарантировать достижения каких-то показателей при функционировании объекта (обычно это те же показатели, что и при анализе).

Таким образом, анализ и синтез — задачи взаимно противоположные:

– в случае анализа заданы структура и параметры, а ищутся показатели функционирования;

– в случае синтеза заданы желаемые показатели функционирования, а определяются будущие структура и параметры.

Задачи анализа и синтеза решаются с помощью математических моделей, так как использовать для экспериментов сам реальный объект обычно невозможно, особенно, если это — организационный объект (технические объекты чаще подвергаются экспериментальным исследованиям, как в процессе создания, так и в процессе функционирования).

Математическое описание содержит представление процессов взаимодействия между отдельными элементами организационного объекта с помощью формул, уравнений, алгоритмов, табличных зависимостей. Уравнения могут быть алгебраическими, дифференциальными, логическими. В целом может использоваться весь арсенал математического представления реальных процессов.

Математическое описание затем преобразуется в математические модели. Различие между математическим описанием и математической моделью следующее. Математическое описание содержит *неупорядоченный* набор математических выражений различного вида, через которые связываются параметры процесса. А математическая модель содержит *упорядоченный* набор тех же математических выражений, в которых выделены входные величины, выходные величины, функциональные зависимости между ними и параметры этих зависимостей. Из одного и того е математического описания можно получить несколько математических моделей — в зависимости от того, что считать входом, а что — выходом.

Пример математического описания:

$$z_1 = f_1(z_2),\ z_2 = f_2(z_3),\ \ z_3 = f_3(z_4).$$

Математические модели:

Вариант 1: $z_1 = f_1\{f_2[f_3(z_4)]\}$. Здесь входная величина — z_4, выходная — z_1.

Вариант 2: $z_4 = f_3^{-1}\left\{f_2^{-1}\left[f_1^{-1}(z_1)\right]\right\}$. Здесь входная величина z_1, выходная — z_4.

Вариант 3: $z_1 = f_1[f_2(z_3)]$. Здесь входная величина z_3, выходная — z_1, а z_2 вообще отсутствует.

Возможны и другие варианты математических моделей на основе того же самого математического описания.

Конкретные функции:

$$z_1 = a_1 z_2,\ z_2 = a_2 z_3,\ z_3 = a_3 z_4.$$

Здесь переменные — z, параметры — a. Соответственно, можно записать несколько вариантов моделей:

Вариант 1: $z_1 = a_1\{a_2[a_3 z_4]\} = a_1 a_2 a_3 z_4$. Здесь входная величина z_4, выходная — z_1.

Вариант 2: $z_4 = \dfrac{1}{a_3}\left\{\dfrac{1}{a_2}\left[\dfrac{1}{a_1}z_1\right]\right\} = \dfrac{1}{a_3}\dfrac{1}{a_2}\dfrac{1}{a_1}z_1$. Здесь входная величина z_1, выходная — z_4.

Вариант 3: $z_1 = \dfrac{1}{a_1}\left[\dfrac{1}{a_2}z_3\right] = \dfrac{1}{a_1}\dfrac{1}{a_2}z_3$. Здесь входная величина z_3, выходная — z_1.

Но возможен и такой вариант, когда выходной величиной считают, например, a_1, а входной — z_3. Тогда модель имеет вид:

Вариант 4: $a_1 = \dfrac{z_1}{z_2}\left[\dfrac{1}{a_2}z_3\right] = \dfrac{z_1}{z_2}\dfrac{1}{a_2}z_3$. Здесь входная величина z_3, выходная — a_1, параметры — a_2, a_3, z_1.

Различают два вида задач — прямую и обратную.

Прямая задача заключается в том, чтобы при заданных входных переменных определить значения выходных:

$$y = f(x),$$

заданы изменения x, например, во времени, и требуется определить y.

Обратная задача заключается в том, чтобы определить, какими должны быть входные значения x, чтобы обеспечить желаемые значения y:

$$x = f^{-1}(y).$$

В простейшем случае это просто, если уравнения разрешимы относительно своих входных переменных. Но это бывает не всегда. Тогда для решения обратной задачи приходится многократно решать прямую и выбирать те решения, которые оказываются наиболее подходящими — близкими к желаемым.

Таким образом, прямая задача является основной и наиболее часто решаемой. Она относится к *анализу* систем. Именно эта прямая задача и называется *моделированием*. Обратная задача является задачей *синтеза*. Она не всегда может решаться. Формально невозможность решения обратной задачи связана с тем, что не всегда исходные уравнения математического описания могут преобразовываться так, чтобы выделить входную величину в виде $x = f^{-1}(y)$, т. е. определить *обратную функцию*. Например, из уравнения $y = a^{x\cos(x)} + \sin(x^2)$ невозможно выразить x в функции от y.

Для решения обратной задачи часто приходится упрощать исходные выражения с учётом реального соотношения параметров. Например, если в выше приведённом выражении параметр мал: $a = 0{,}00001$, то выражение можно упростить: $y \cong \sin(x^2)$, и уже из него получить обратную функцию: $x \cong \sqrt{\sin(y)}$. Поэтому часто задача синтеза может дать лишь приближённое решение. Для его проверки используется анализ по полному исходному выражению. При этом оказывается возможным учитывать те факторы, которые при синтезе не учитывались. Поэтому в реальности после решения задачи синтеза всегда решается задача анализа полученного синтезированного решения.

Рассмотренный пример связан с попытками *аналитического* решения задач анализа и синтеза. Но не всегда даже задачу анализа можно решить аналитически (выводом формул): например, не назходятся аналитически корни уарвнения больше 4-й степени и т.д. Тогд приходится использовать численные методы и ЭВМ.

Примерами *организационных объектов* являются: организации различных организационно-правовых форм (предприятия, банки, общественные объединения, государственные учреждения и т. п.), органы власти (исполнительной и законодательной, федеральной и муниципальной), конкретные мероприятия (конгрессы, встречи, аукционы, митинги, шествия, спортивные соревнования и т. п.), суды, адвокатские конторы, средства массовой информации и др. Они функционируют, производя промышленную продукцию, услуги, законодательные акты, управленческие решения и т. д. Каждый такой организационный объект имеет свою структуру, т. е. взаимосвязанные и взаимодействующие элементы. Через эти элементы внутри организации протекают различные потоки — материальные, финансовые, информационные и др. Каждый организационный объект для выполнения своих функций использует различные ресурсы — материальные, финансовые, человеческие,

интеллектуальные, информационные и др. В большинстве случаев организационные объекты взаимодействуют с внешней средой и получают от неё воздействия различного рода, способствующие или препятствующие выполнению их функций. В обществе организационные объекты взаимодействуют между собой и таким образом образуют организационные системы.

Рассмотрим пример организационного объекта.

Промышленное предприятие занимается производством определённого вида продукции (например, хлеба). Хлеб развозят по магазинам. При этом известны средние цифры себестоимости одного килограмма хлеба и его стоимость в магазине. Можно посчитать доход и прибыль.

Всю цепочку производства-сбыта можно разделить на несколько этапов:

- производство;
- хранение готовой продукции;
- транспортировка её в магазины;
- продажа;
- получение дохода и прибыли.

Каждый этап можно детализировать.

Например, **на этапе производства** продукцию получают путём последовательной переработки исходного сырья в нескольких цехах через некие технологические цепочки. При этом работает оборудование и персонал, выбирающий режимы работы. Всё это занимает определённое время и требует определённого объёма сырья,

Готовая продукция поступает на склад и там хранится до отправки в магазины. Продукция продаётся через сеть магазинов в городе и вывозится со склада специальным транспортом по расписанию. Продукция вырабатывается в определённое время суток, а развозится в другое время суток. На всех работах задействован персонал. Существуют расходы на сырьё, электроэнергию, тепло, аренду помещений, оплату работников, на транспортировку, ремонт оборудования. Необходимо учесть ряд факторов: наличие, объём и поступление исходного сырья, время работы оборудования в различных режимах, возможность его поломки, необходимость покупки запасных частей и время работы (с использованием соответствующего ремонтного первонала), скорость транспортировки на склад и со склада в магазины, последовательность переезда от магазина к магазину, затрачиваемое время погрузки и разгрузки, заработную плату всех работников, налоговые отчисления и др. Интерес могут представлять доходы фирмы, ритмичность её работы, достаточность задействованных ресурсов (оборудования, исходного сырья и т. д.), износ оборудования и др.

В настоящее время можно говорить о проектировании организационных систем так же, как проектируют технические системы. Для технических систем существует хорошо развитый аппарат теории автоматического управления, который предоставляет не только набор формальных методов проектирования, но также методологическую основу и общесистемные подходы к

проектированию систем различной природы. Поэтому целесообразно рассмотреть особенности организационных систем в сравнении с техническими системами.

В процессе проектирования организационных систем, также как и технических, выбирают структуру, параметры, переменные состояния (из которых может формироваться набор выходных переменных) и внешние воздействия (из которых выделяют управляющие воздействия). В процессе проектирования решаются две задачи — анализа и синтеза. Под анализом понимают определение поведения переменных состояния (и связанных с ними выходных переменных) при заданных управляющих воздействиях. Под синтезом понимают определение вида управляющих воздействий, позволяющих реализовать желаемое поведение переменных состояния.

Конечной целью проектирования обычно является синтез, но выполнить его сложно, так как задача синтеза с точки зрения математики связана с решением той или иной системы уравнений. В технических системах обычно используются алгебраические, дифференциальные или логические уравнения. При определённом их виде (например, в случае линейности) имеются соответствующие методы решения, т. е. синтез может быть выполнен. Но большинство технических систем описываются нелинейными уравнениями, что не даёт возможности выполнить синтез.

Анализировать систему проще, так как анализ с точки зрения математики часто сводится к более простым математическим операциям, например, к табулированию функций или к их численному решению известными методами. При этом математические модели могут быть существенно более сложными и включать, например, алгоритмические фрагменты, таблично заданные значения и др.

Поэтому при проектировании сложных в математическом описании систем обычно, как правило, исходное математическое описание упрощают и по нему осуществляют приближённый синтез. А затем по исходному детальному математическому описанию выполняют моделирование и проверку корректности выполненного синтеза. Обычно приходится подстраивать моделируемую систему, подбирая синтезированные параметры (или управляющие воздействия — в зависимости от целей синтеза).

Кроме использования на завершающем этапе синтеза, этап анализа может быть самостоятельным, когда необходимо исследовать поведение уже существующей системы при различных внешних воздействиях и различных параметрах.

Таким образом, анализ является неотъемлемой частью синтеза (на его завершающем этапе) и также отдельным этапом исследований. В данной работе рассматриваются вопросы анализа.

При анализе объекта главным является вопрос управления им через воздействие. Управление организационными объектами существенно отличается от управления техническими системами. В частности, для технических

систем возможно построение опытных образцов и отладка принятых технических решений на макетах, а эксперименты над организационными объектами, как правило, не возможны. В то же время, неправильные решения, принятые при управлении организационными системами, могут иметь весьма тяжёлые последствия, в том числе для участвующих в них людей. Поэтому единственной возможностью проверки правильности принимаемых решений является использование моделей в той или иной форме.

Особенностью организационных систем как объекта моделирования является неопределённость, связанная как с некоторыми внутренними параметрами, так и с внешними воздействиями. Эта неопределённость приводит к тому, что оказывается невозможным использовать классические математические модели и следует искать другие формы. Одной из таких форм является представление организационных систем в терминах теории массового обслуживания (систем с очередями). В таких системах используется дискретно-событийный характер представления процессов, когда любой процесс состоит из последовательности событий. Оказывается, что в таких моделях появляется возможность использовать параметры, характеризующие, например, *сложность* исследуемых явлений, их *значимость* и т. д. Эти и другие аналогичные свойства (характеристики) существенны прежде всего для организационных систем.

Характерной чертой организационной системы является то, что её деятельность многообразна и соответствующие процессы могут протекать по различным сценариям, зависящим от многочисленных случайных факторов. Это многообразие отражается алгоритмически (через соответствующие условные операторы) и мешает аналитическому исследованию таких систем, которые становятся, таким образом, системами с переменной структурой и переменными параметрами — т. е. самыми сложными с точки зрения анализа.

В настоящее время многие свойства организационных систем являются трудно формализуемыми, т. е. не сводятся к однозначно понимаемым математическим выражениям. Например, качество работы организационной системы не может определяться как минимизация или максимизация некоторого формального критерия, например, степени удовлетворения клиентов этой организации. Тем не менее, для принятия управленческих решений требуется в какой-то мере формализовать эти критерии, заменяя их другими (возможно, несколькими). Например, вместо удовлетворённости может использоваться критерий отсутствия ожидания обслуживания и др.

Таким образом, имитационное моделирование является важнейшим инструментом исследования (анализа и синтеза) сложных организационно-технических систем. Владение этим инструментом позволяет создавать системы поддержки принятия решений, в которых синтез осуществляется по некоторым алгоритмам с учётом упрощающих предположений, а затем выполняется анализ с учётом большинства реально водействующих факторов.

1.1 Имитационное моделирование как метод исследования организационно-технических объектов

Имитационное моделирование характеризуется прежде всего сочетанием двух факторов — неопределённости и возможности ветвления процессов в зависимости от конкретных реализаций этой неопределённости.

При имитационном моделировании система представляется потоком событий, происходящих в отдельные моменты времени, т. е. дискретно. Такое представление и такое моделирование называют дискретно-временны́м. Его отличие от функционального моделирования (например, на основе решения систем алгебро-дифференциальных уравнений) в том, что события происходят в соответствии с заданными интервалами времени и, возможно, в зависимости от уже произошедших событий. Интервалы времени (моменты совершения событий) вычисляются, как правило, с помощью генераторов случайных чисел с тем или иным законом распределения вероятностей.

При имитационном моделировании все события в системе происходят в соответствии с продвижением некоего индикатора времени, который называют *транзактом*. Транзактов может быть много, но в каждый момент времени активным является только один, который и вызывает то или иное событие. При этом движение транзакта может сопровождаться и не сопровождаться изменением времени. Физический смысл каждого транзакта может быть различным. Это могут быть люди, внешние воздействия, различного рода сигналы и т. п. Имитационная модель может проследить траектории прохождения транзактов при различных условиях, собрать статистическую информацию о них (среднее время нахождения в том или ином блоке, количество прошедших через блок транзактах и др.). Современные системы имитационного моделирования могут включать в себя также блоки функционального моделирования (например, язык имитационного моделирования GPSS World содержит специальное расширение — язык PLUS, во многом напоминающий упрощённый язык Паскаль).

Таким образом, имитационное моделирование может обеспечить решение различных исследовательских задач:

- определение реального алгоритма работы той или иной системы с учётом вероятностных характеристик отдельных элементов и сигналов и различных условий;
- вычисление статистических характеристик (средние, максимальные и минимальные значения, коэффициент использования);
- оптимизация структуры или параметров исследуемой системы;
- поиск сбоев и неисправностей в реальной системе и причин их возникновения;
- создание компьютерных деловых игр как компонентов систем поддержки принятия решений.

Имитационное моделирование может применяться для любых типов систем, но именно организационно-технические объекты наиболее удобно исследовать методами имитационного моделирования.

1.2 Вероятностные характеристики в имитационном моделировании

В теории вероятностей различают следующие случайные явления:

– случайные события;

– случайные величины;

– случайные процессы.

Случайное событие может произойти или не произойти в рассматриваемый момент времени или при рассматриваемых обстоятельствах.

Случайная величина принимает какое-то значение, которое невозможно рассчитать или заранее предвидеть.

Случайный процесс связан с последовательностью случайных событий или с последовательным изменением значений случайной величины во времени.

Для имитационного моделирования характерно рассмотрение именно случайных процессов, которые состоят из последовательности случайных событий и (или) значений случайных величин. Это может быть: возникновение или отсутствие события; выбор одного события из нескольких возможных; задание промежутка времени между событиями; задание значения какого-либо параметра и т. д. В зависимости от возникновения события или от значения какой-либо величины выбирается направление развития моделируемого процесса. Именно это сочетание *случайности* и *зависящего от неё выбора* создаёт в сложных объектах большое число возможных траекторий процесса (реализаций), что затрудняет его аналитическое исследование и приводит к необходимости имитационного моделирования.

При имитационном моделировании задают случайным образом какое-либо событие или значение какой-либо величины (т. е. причину), а затем прослеживают всю цепочку связанных с этой причиной следствий. Таким образом, имитационная модель в целом отражает причинно-следственные связи, но хотя при этом одна и та же причина вызывает одни и те же следствия, сами эти причины появляются случайным образом. Кроме того, на отдельных участках траектории процесса могут появляться дополнительные случайные воздействия и непредвиденным заранее образом менять направление развития процесса. Это повторяется большое число раз и таким образом создаётся *выборка результатов моделирования* при различных входных случайных явлениях. Характерно то, что при имитационном моделировании всегда имеют дело не с отдельными случайными явлениями, а с их совокупностью — потоком. Фактически, в процессе имитационного моделирования находится *выборка выходных случайных явлений* (событий, величин) в зависимости от *заданной выборки входных случайных явлений*.

Потоки случайных явлений (величи́н и событий) наиболее полно представляются самой выборкой, т. е. совокупностью всех состоявшихся событий и принятых значений. Понимание одинаковости и различия случайных процессов отличается от такового для детерминированных процессов. Детерминированные процессы тогда являются одинаковыми, когда все реализации двух сравниваемых выборок попарно одинаковы (фактически, это означает, что две функции должны быть равны, если равны их значения при любом одном и том же для обеих функций аргументе). Но это возможно только для произошедших явлений. Вообще, то, что произошло, уже не может в полном смысле слова считаться случайным, так как в момент реализации случайного явления неопределённость его появления снимается.

Чтобы можно было как-то охарактеризовать будущие процессы, используют специальные вероятностные характеристики (параметры), позволяющие различать их основные статистические свойства.

Но в некотором смысле выборки могут быть одинаковыми даже если их частные реализации не совпадают попарно.

В имитационной модели должны быть источники случайных величин, а их вероятностные характеристики должны соответствовать реальным неопределённостям в объекте моделирования.

Во многих случаях возможен переход от случайных величин к случайным событиям и наоборот. Например, если какая-то величина принимает случайное значение, то рассматривается случайная величина. Если рассматривается принятие величиной того или иного значения, то это — случайное событие.

Если рассматривается последовательность случайных событий, то может оказаться, что все события происходят одинаковое число раз или же некоторые события происходят чаще, а некоторые — реже. То же самое может происходить со случайными величинами: случайная величина принимает любое из возможных значений одинаковое число раз или некоторые значения — чаще, а некоторые — реже.

Это свойство случайных событий (и случайных величин) можно отобразить с помощью специального графика — *гистограммы*. По оси абсцисс откладываются виды событий (или значения случайной величины), а по оси ординат — число этих событий (или число принятия случайной величиной соответствующего значения). Такой график даёт представление о *распределении вероятностей* и его называют *плотностью распределения вероятностей*. Чем выше график в какой-то точке абсциссы, тем чаще происходит соответствующее событие или встречается соответствующее значение случайной величины. По оси ординат можно откладывать абсолютные значения или относительные, рассчитанные относительно общего числа событий или общего числа возможных значений случайной величины.

Можно проанализировать предельные случаи гистограмм.

Невероятные события представляют собой горизонтальную прямую, совпадающую с осью абсцисс (вероятность равна 0).

Однозначно определённые события представляют собой вертикальную прямую в точке, соответствующей этому определённому событию (вероятность равна 1).

Равновозможным событиям, которые происходят одинаково часто, соответствует горизонтальная кривая, параллельная оси абсцисс и соответствующая значению $1/N$ по оси ординат, где N — число событий.

Плотность распределения вероятностей называют также *дифференциальной функцией распределения*. Если её проинтегрировать, то будет получена *интегральная функция распределения* (или — просто *функция распределения*). Смысл интегральной функции распределения в том, что она показывает, как часто случаются все событий *левее* рассматриваемой точки (или принимаются соответствующие значения случайной величиной).

Обе функции распределения (интегральная и дифференциальная) связаны между собой операциями дифференцирования и интегрирования, а потому несут обычно одну и ту же информацию. Но эта информация представляется различным образом и разных случаях бывает нагляднее та или иная функция.

В GPSS World реализованы следующие типовые распределения вероятностей: бета (Beta), биномиальное (Binomial), Вейбулла (Weibull), дискретно-равномерное (Discrete Uniform), гамма (Gamma), геометрическое (Geometric), Лапласа (Laplace), логистическое (Logistic), логлапласово (LogLaplace), лог-логистическое (LogLogistic), логнормальное (LogNormal), нормальное (Normal), обратное Вейбулла (Inverse Weibull), обратное Гаусса (Inverse Gaussian), отрицательное биномиальное (Negative Binomial), Парето (Pareto), Пирсона типа V (Pearson Type V), Пирсона типа VI (Pearson Type VI), Пуассона (Poisson), равномерное (Uniform), треугольное (Triangular), экспоненциальное (Exponential), экстремального значения A (Extreme Value A), экстремального значения B (Extreme Value B).

Это позволяет использовать существующий аппарат теории вероятностей при имитационном моделировании, но остаётся не полностью решённым вопрос — в каких случаях следует использовать то или иное распределение вероятностей.

1.3 Общие принципы имитационного моделирования

1.3.1 Метод Монте-Карло как основа имитационного моделирования

Метод Монте-Карло (другое название — *метод статистических испытаний*) предназначен для изучения процессов с учётом случайного фактора. Случайным фактором может быть случайное значение какой-либо величины (например, время прихода клиента, количество свободных мест в

подъехавшем к остановке автобусе, рост покупателя в магазине готового платья и т. п.) или случайное событие (например, факт болезни сотрудника, наличие нужного товара в магазине, прибытие в аэропорт самолёта и т. п.).

В теории вероятностей и математической статистике разработаны *аналитические* методы исследования такого рода процессов с помощью формул, если известны законы распределения вероятностей, т. е. если они могут быть заданы приближёнными аналитическими выражениями. Но это не всегда возможно. Например, если в зависимости от случайного события изменяется направление процесса, то такую системы проанализировать аналитически (с помощью формул) может оказаться затруднительно. В этом случае следовало бы рассчитать все возможные варианты протекания процесса, а их может оказаться много сотен и тысяч. Поэтому даже если такие расчёты в принципе возможны, полученные результаты могут стать необозримыми и практически бесполезными. Именно для этого случая целесообразно применять *метод Монте-Карло*.

Суть *метода Монте-Карло* заключается в том, что в модель — туда, где проявляет себя неопределённость, вводят датчики случайных чисел. В результате с их помощью в нескольких местах модели производится розыгрыш *случайной величины* или *случайного события*, в зависимости от которых меняется ход или параметры исследуемого *случайного процесса*. В этом случае можно провести серию испытаний при различных ситуациях и затем попробовать выявить закономерности процесса. Это может оказаться существенно проще, чем аналитическая аппроксимация элементов исследуемого процесса, так как часто такая аппроксимация требует специальных исследований и не всегда возможна.

Случайности в разных частях модели причудливо сочетаются друг с другом, поэтому получаемая модель обычно имеет большое число возможных ветвлений. Это приводит к сравнительно простой модели сравнительно сложного процесса, когда появляется возможность учесть самые разнообразные факторы весьма простыми средствами. Кроме случайностей, в модели могут присутствовать и детерминированные фрагменты процесса с жёстко заданными причинно-следственными связями.

При сложности исследуемого процесса простота его модели связана с тем, что нет необходимости вручную специально просчитывать все возможные варианты — эту работу выполняет программа. В процессе счёта фиксируются все события, величины, параметры, которые затем подвергаются статистической обработке и отображаются тем или иным образом.

1.3.2 Дискретно-событийное моделирование

При имитационном моделировании процессы развиваются в функции от некоторой независимой переменной. Вообще говоря, она может иметь различный физический смысл, но очень часто независимой переменной является время. Во-первых, в жизни именно время является тем, что ни от чего не за-

висит, но на всё влияет. Во-вторых, многие факторы можно отразить с помощью времени: например, сложность выполнения работы можно оценить затрачиваемым на неё временем.

Процессы протекают во времени по-разному и цели их моделирования также могут быть разными. Например, если автобус едет по маршруту, то может казаться интересно, как меняется скорость его движения, когда он отъезжает от остановки, подъезжает к перекрёстку, тормозит перед остановкой и т. д. При этом можно использовать функциональное моделирование на основе законов механики. Это можно сделать на основе решения системы дифференциальных уравнений, описывающих динамику механических тел (законы Ньютона и другие физические законы), и может быть необходимо, например, для исследования расхода горючего, износа шин и тому подобных явлений. Если же исследуется объём перевозок, потоки пассажиров, получаемые доходы и т. п., то достаточно лишь самого факта и времени появления автобуса в нужном месте, т. е. интерес представляет *событие*. Между соседними во времени событиями имеются временны́е промежутки, в которых никаких событий не происходит. Следовательно, в программной модели все последовательные события могут протекать непосредственно друг за другом, а время между ними определяется расчётным путём. Это можно проиллюстрировать следующим образом.

При функциональном моделировании (например, при численном интегрировании системы дифференциальных уравнений) вначале определяется время (начальное время t_0, шаг интегрирования Δt, новое время $t_0 + \Delta t$), а затем вычисляются все переменные. При этом ось времени — равномерная (по крайней мере, равномерным является шаг вывода).

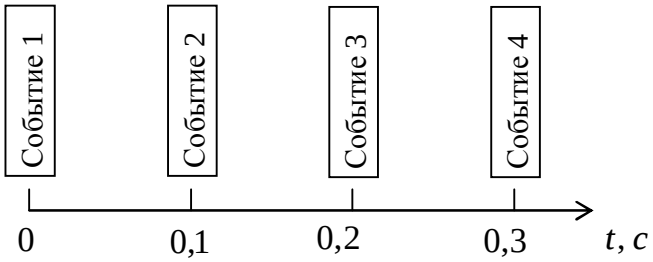


Рисунок 1.1 — Ось времени при функциональном моделировании (промежутки времени между соседними событиями одинаковы)

При имитационном моделировании рассматриваемого вида процесс привязки ко времени осуществляется иначе (Рисунок 1.2):

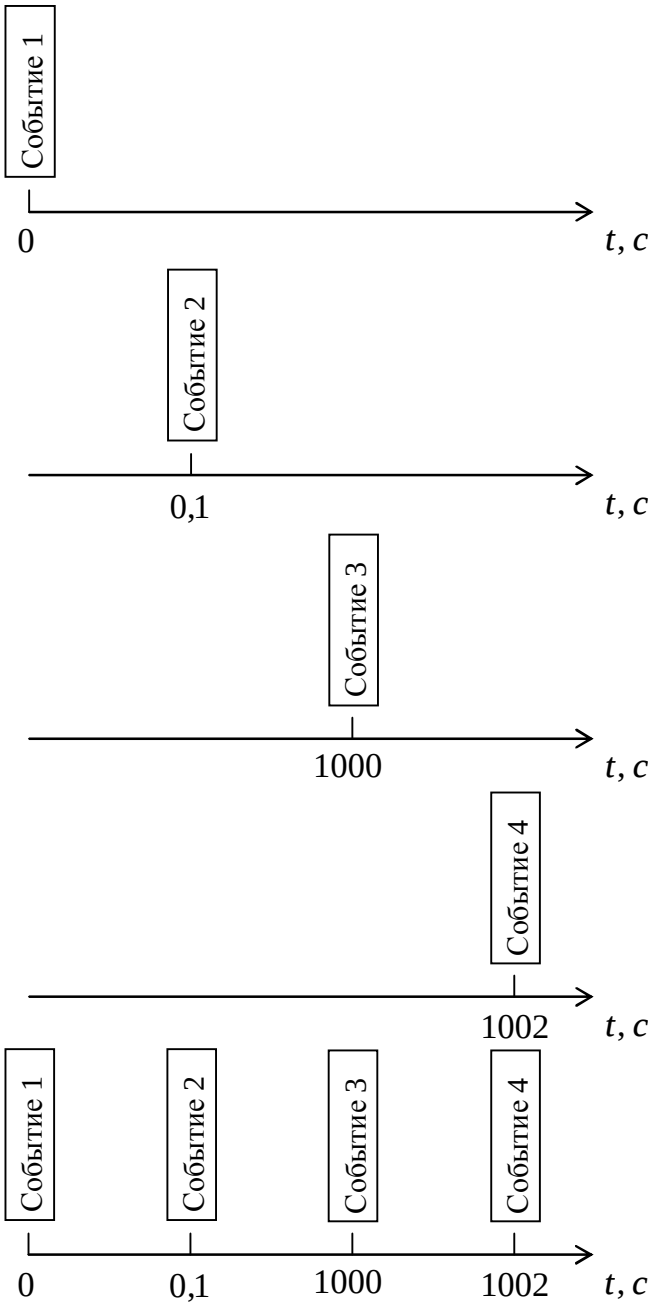


Рисунок 1.2 — Ось времени при дискретно-событийном моделировании (промежутки времени между соседними событиями разные)

Время или задано точно, или разыгрывается случайным образом в рамках существующих условий с помощью генератора случайных чисел. Этому времени соответствует какое-то событие. Если оно детерминированное (т. е. происходит обязательно), то оно просто фиксируется для заданного времени и программа переходит к следующему моменту времени, известному для следующего события. Если событие случайное, то возможность его появления разыгрывается в этот момент времени и процесс получает новое направление развития в зависимости от результата розыгрыша.

Таким образом, при имитационном моделировании приращение времени Δt производится сразу — ещё до момента наступления очередного события без промежуточных значений. Поэтому ось времени оказывается неравномерной: интервал времени между первым и вторым событиями может составлять, например, 0,1 *с*, а между вторым и третьим, например, 1000 *с*.

Если в модели заранее заданы времена наступления всех событий, то эти времена вычисляются сразу, а затем накладываются одно на другое и получается диаграмма событий во времени.

Если возможно ветвление процесса, т. е. в зависимости от условий может происходить одно или другое событие, то программа рассчитывает времена событий до точки ветвления, затем выбирает направление развития процесса и рассчитывает времена событий на этом направлении до следующей точки возможного ветвления.

Таким образом, всегда рассчитываются только те события, которые происходят, и тем самым экономится время счёта.

Такое моделирование называют дискретно-событийным: *дискретным* по отношению ко времени (используются времена только в точках событий) и *событийным* — по отношению к характеру моделируемых элементов (ими являются именно события). При имитационном моделировании организационных объектов используется именно дискретно-событийная форма.

Дискретно-событийное моделирование:

во-первых, отвечает сути изучаемых процессов, главным в которых является последовательность событий (которые, разумеется, могут иметь свои параметры);

во-вторых, является весьма экономичным, так как программа считает только в точках изменения состояния модели — в моменты событий (вообще говоря, состояние модели может и не измениться, если ожидаемое событие не произошло из-за невыполнения каких-то условий);

в-третьих, весьма точное в отношении начала и завершения событий, так как времена всегда рассчитываются без погрешностей.

Сами события могут иметь различную физическую природу и различное представление в модели. Событием может быть появление в модели какого-то объекта, например, приход посетителя, формирование сигнала времени, совпадение каких-то условий и др.

2 Программная среда имитационного моделирования GPSS World

2.1 Общие сведения о программе GPSS World

Программа GPSS (*General Purpose Systems Simulator*) является классической программой имитационного моделирования.

GPSS предназначена для моделирования систем массового обслуживания (систем с очередями) а также другх аналогичных систем, и имеет для этих целей специальные операторы, синтаксис, вспомогательные инструменты (статистическая обработка результатов, их накопление, графическое отображение).

GPSS была создана в 1961 году в фирме IBM *Джеффри Гордоном* (*Geoffrey Gordon*) почти в то же время, что и языки FORTRAN и ALGOL. Наряду с ними GPSS имелась на всех компьютерах того времени. Фирма *Minuteman Software* в 1982 году адаптировала программу к персональным компьютерам для работы в DOS (версия GPSS/PC), в 1993 году создала первую версию GPSS World для OS/2, а в 2000 году — усовершенствованную версию программы GPSS World для Windows (может применяться с Windows 98, 2000 и XP).

В настоящее время разработана версия GPSS World 5.0.3, которая поставляется в трёх вариантах:

GPSS World Commercial Version 5.0.3 — неограниченное количество блоков (стоимость — по договорённости с фирмой);

GPSS World Personal Version 5.0.3 — 2000 блоков ($695);

GPSS World Student Version 5.0.3 — 180 блоков ($9,95).

В России с декабря 2004 года распространяется версия GPSS World 5.0.2 также в трёх вариантах:

Однопользовательская лицензия студенческой версии GPSS World 5.0.2 — 500 блоков (1 500 руб.);

Однопользовательская лицензия персональной версии GPSS World 5.0.2 — 5000 блоков (23 000 руб.);

Однопользовательская лицензия академической версии GPSS World 5.0.2 — количество блоков не ограничено, поставляется для вузов (35 000 руб.);

Однопользовательская лицензия коммерческой версии GPSS World 5.0.2 — количество блоков не ограничено, поставляется для коммерческих фирм (120 000 руб.).

Эксплуатационные документы можно купить отдельно за 700 руб. плюс почтовые расходы.

Фирма *Minuteman Software* предлагает также бесплатный студенческий вариант программы GPSS World 4.3.5, который обладает всеми функциями соответствующей полной версии, но имеет ограничение на максимальное количество блоков (не более 150). Эта версия практически не отличается от по-

следней продаваемой в России версии GPSS World 5.0.2 (исправлены некоторые мелкие недочёты). Таким образом, студенческая бесплатная версия GPSS World 4.3.5 позволяет изучить все особенности программы, но не может быть использована для коммерческих целей при разработке больших имитационных моделей. Эксплуатационные документы также не очень нужны, так как вместе с версией GPSS World 4.3.5 предлагается бесплатный электронный вариант книг ***Reference Manual*** (справочник по системе с перечислением всех операторов, правил синтаксиса и встроенных инструментов), а также ***Tutorial Manual*** (примеры использования программы с соответствующими файлами, включенными в инсталляционный файл GPSS World 4.3.5).

Материалы о программе GPSS World размещены на сайтах:

http://www.minutemansoftware.com/ — оригинальный сайт фирмы-разработчика;

http://www.gpss.ru/index-h.html — российский сайт по имитационному моделированию.

Значение программы GPSS не только в том, что её саму можно использовать для моделирования разнообразных технологических, организационных и экономических процессов. При разработке GPSS впервые был применён объектно-ориентированный подход и она сама является первым успешно реализованным объектно-ориентированным программным продуктом. Именно в ней впервые было реализовано понятие *транзакт* (по-русски — *заявка*, *требование*) и другие понятия теории массового обслуживания. Программирование в GPSS похоже на таковое в других языках (имеются операторы и операнды, ссылки, ветвления и др.), но каждый оператор GPSS представляет собой целую систему (реально — подпрограмму), выполняющую внутри себя несколько функций, которые напрямую внешне не проявляются. Это — сбор статистики об обрабатываемых транзактах, её обработка, изменение различных параметров транзакта и др. Программа включает специфический набор операторов, которые стали основой для разработки других систем имитационного моделирования. В последнее время появились системы имитационного моделирования более высокого уровня (например, AnyLogic, ARENA), обеспечивающие визуальное моделирование (с помощью графического интерфейса в виде специальных блоков и связей между ними). Но в них также используются базовые понятия, введённые ранее в GPSS. Поэтому изучение GPSS позволяет легко осваивать затем и другие программы имитационного моделирования.

Имитационное моделирование близко к моделированию бизнес-процессов. Существуют программы, которые обеспечивают связь моделей бизнес-процессов с имитационными моделями. Но область использования имитационного моделирования гораздо шире. С его помощью можно проводить предпроектные исследования для технических объектов, выявляя параметры режимов их функционирования — например, в случае решения вопроса о количестве и плане размещения станков в цехе, электрооборудования в

организациях обслуживания и т. д. Уже на основе результатов предпроектных исследований можено выполнять техническое проектирование.

2.2 Визуализация результатов имитационного моделирования

Для вывода результатов имитационного моделирования в различных формах и представлениях используются следующие инструменты:

- *Журнал* ***Journal***;
- *Стандартный отчёт* ***Standard Report***;
- *Окна моделирования* ***Simulation Windows***;
- *Снимки моделирования* ***Simulation Snapshot***.

Журнал отражает все операции, совершаемые пользователем, и операции, выполняемые по его указаниям программой. *Журнал* можно сохранить в файле и затем проанализировать работу программы. При непрерывном моделировании в *Журнале* сохраняется информация о начале и окончании каждого процесса, а также возникшие ошибки. Наиболее эффективно использование *Журнала* в пошаговом режиме моделирования. На каждом шаге указывается, какой именно транзакт в данный момент является активным, в каком блоке он находится и в какой блок будет переходить, а также печатается комментарий к блоку. Если делать тщательно продуманные комментарии к блокам, то можно получить связный рассказ о работе программы, что помогает как при отладке, так и при исследовании сложных ветвящихся процессов.

Стандартный отчёт содержит основную информацию о программе (её текст, наличие блоков различных типов и др.), а также результаты моделирования в виде наиболее часто встречающихся статистических показателей. Содержимое *Стандартного отчёта* можно настраивать.

Окна моделирования предназначены для вывода информации в числовой или графической формах о переменных, имеющихся в программе. Существуют универсальные окна, например, для вывода графиков, а также специализированные, например, для построения гистограммы или для анимации движения транзактов через блоки в модели.

Снимки моделирования предназначены для фиксации в заданные моменты времени состояния программы с целью последующего анализа или её отладки. Фиксируются состояния *Цепи́ текущих событий*, *Цепи́ будущих событий*, *Цепи́ пользователя*, групп однородных транзактов, положение конкретного транзакта.

Все окна можно располагать рядом и одновременно наблюдать проходящие процессы с разных точек зрения и в различной форме. Но открытие окна влечёт за собой начало специальных вычислительных процессов, связанных с этим окном, поэтому время счёта с каждым открытым окном увеличивается.

В целом эти инструменты визуализации результатов гармонично сочетаются между собой и создают основу *технологии имитационного моделирования* с помощью программы GPSS World.

2.3 Операторы, блоки, команды и транзакты

В GPSS World различают два класса элементов — активные (транзакты) и пассивные (блоки и команды).

Программа на GPSS World, как и на любом другом языке программирования, представляет собой последовательность операторов — *блоков* и *команд*, которые являются *пассивными* сущностями, готовыми к выполнению своих функций.

В программах на обычных языках программирования управление операторами передаётся последовательно сверху вниз или в соответствии с действием условных операторов типа **`IF`** и безусловных типа **`GO TO`**.

В GPSS World каждый блок работает только тогда, когда он специальным образом активирован. В целом последовательность активации блоков такая же, как и в обычных программах — сверху вниз и в соответствии с операторами безусловного и условного перехода (оба типа есть в GPSS World). Но есть некоторые отличия. Среда выполнения программы на GPSS World снабжена внешним циклом (*Планировщиком*), который обеспечивает просмотр программы сверху вниз, выявление того блока, который должен быть активирован, и его выполнение. Это соответствует тем объектам, которые моделируются. Например, если моделируется движение автомобильного транспорта по дороге, то в программе будут присутствовать различные блоки, отражающие процесс поворота автомобиля, его остановку или движение, работу светофора и др. Планировщик должен выбрать, какой именно блок в данный момент должен сработать. Для передачи активности от блока блоку используется универсальное понятие *транзакт*.

Блоки имеют имена — литеральные названия или номера. Каждый способ задания имени блока имеет свои достоинства. Имена в виде названий позволяют легче ориентироваться в модели. Имена в виде номеров позволяют вычислять эти номера с помощью арифметических операций, что в некоторых случаях обеспечивает элегантность реализации алгоритмов, например, в циклах. В одной и той же модели для одних и тех же блоков можно использовать как номера, так и литеральные названия (синонимы). Транслятор всегда присваивает всем блокам номера, даже если для них были заданы литеральные названия, что отражается в *Стандартном отчёте*.

Транзакты — это виртуальные объекты, представляющие собой сигналы передачи активности от блока к блоку. Отсюда и их название на английском языке — *transaction* (передача). Транзакты — активные элементы GPSS World.

В программах на обычных языках программирования в первый момент её выполнения тоже *как бы* создаётся транзакт, который затем "проходит" по

всей программе, вызывая работу соответствующих операторов. Но этот "транзакт" полностью виртуальный.

В GPSS World транзакты специально генерируются и их может быть много. В GPSS World существует внешний контур управления перемещением транзактов (планировщик), который образует внешний бесконечный цикл проверок — какой транзакт в данный момент активен и в какой блок программы он может войти, чтобы активировать выполнение соответствующего оператора. Транзакты могут располагаться в различных местах программы, хотя их не видно. Но внешний контур управления всегда знает, где и какие транзакты находятся, какие из них пассивны, а какой (один!) готов стать активным. При создании каждый транзакт получает номер (фактически — имя) и массу параметров, т. е. с каждым транзактом связывается некоторая ячейка памяти (точнее, массив ячеек). При изменении состояния транзакта некоторые его параметры могут автоматически изменяться, а некоторые параметры можно изменять принудительно. Транзакты создаются специальным блоком **GENERATE** и уничтожаются блоком **TERMINATE**. Поэтому создание транзактов имеет физический смысл, так как приводит к выделению для них ячеек памяти, а уничтожение транзактов означает стирание этих ячеек и удаление их из памяти, связанной с данной программой (ячейки для транзактов выделяются в GPSS World динамически).

Транзакты могут иметь различный *приоритет* с точки зрения первоочерёдности обработки. В GPSS World приоритет задаётся числом от 0 до 127, т. е. всего может быть 128 градаций. Приоритеты присущи многим реальным явлениям. Например, в любой организации имеется определённая иерархия должностей (директор, замдиректора, начальник отдела и т. д.), в армии существуют воинские звания (генерал, полковник, майор и т. д.), в спорте существуют спортивные звания (разряды, кандидат в мастера спорта и мастер спорта, чемпион определённого уровня и т. д.) и др. Изначально приоритет транзакта задаётся при его создании оператором **GENERATE** (по умолчанию приоритет равен 0). Затем приоритет можно принудительно изменить в блоке **PRIORITY**, в том числе и с учётом наступивших условий. С учётом приоритета строятся *Очереди* — в первую очередь пропускаются транзакты с бóльшим приоритетом.

Пассивные элементы в GPSS World делятся на *блоки* и *команды*. Внешне в тексте программы они выглядят похоже и, как правило, состоят из оператора и операнда. Но назначение их разное.

Блоки — это те пассивные элементы программы, которые непосредственно взаимодействуют с транзактами. Говорят также, что транзакты *входят* в блоки.

Команды управляют работой всей программы в целом (запуск, остановка и т. д.) или описывают блоки, нуждающиеся в предварительном описании. Т. е. *команды* работают без непосредственного контакта с транзактами.

Список блоков GPSS World: **ADOPT**, **ADVANCE**, **ALTER**, **ASSEMBLE**, **ASSIGN**, **BUFFER**, **CLOSE**, **COUNT**, **DEPART**, **DISPLACE**, **ENTER**, **EXAMINE**, **EXECUTE**, **FAVAIL**, **FUNAVAIL**, **GATE**, **GATHER**, **GENERATE**, **INDEX**, **INTEGRATION**, **JOIN**, **LEAVE**, **LINK**, **LOGIC**, **LOOP**, **MARK**, **MATCH**, **MSAVEVALUE**, **OPEN**, **PLUS**, **PREEMPT**, **PRIORITY**, **QUEUE**, **READ**, **RELEASE**, **REMOVE**, **RETURN**, **SAVAIL**, **SAVEVALUE**, **SCAN**, **SEEK**, **SEIZE**, **SELECT**, **SPLIT**, **SUNAVAIL**, **TABULATE**, **TERMINATE**, **TEST**, **TRACE**, **TRANSFER**, **UNLINK**, **UNTRACE**, **WRITE**. Блоки выполняют конкретные операции с транзактами (в соответствии со своим назначением) и активируются при входе в них активного транзакта. Некоторые блоки являются обычными операторами, аналогичными операторам универсальных языков программирования, например блок **WRITE** (записать) выполняет обычную операцию записи данных в файл. Но многие блоки представляют собой подпрограммы, выполняющие ряд действий при "вхождении" в них транзакта. Например, блок **QUEUE** (очередь) при вхождении в него транзакта запоминает его номер, время входа, а при выходе — время выхода, одновременно рассчитывает среднее время нахождения транзактов в очереди (учитывая все прошедшие через него транзакты) и др. статистику.

Список команд GPSS World: **BVARIABLE** (описание двоичной переменной); **CLEAR** (возвращает исходное состояние модели для нового сеанса моделирования); **CONDUCT** (начало имитационного эксперимента); **CONTINUE** (продолжить моделирование после его остановки командой **HALT**); **EQU** (задание значения переменной); **FUNCTION** (описание функции); **FVARIABLE** (описание действительной переменной); **HALT** (остановка моделирования); **INCLUDE** (включить файл); **INITIAL** (задание начальных значений переменным); **INTEGRATE** (интегрирование непрерывной переменной); **MATRIX** (описание матрицы); **QTABLE** (описание таблицы для сбора данных с целью построения гистограммы *Очереди*); **REPORT** (управляет формированием и выводом на экран *Стандартного отчёта*); **RESET** (помечает начало нового эксперимента в их серии); **RMULT** (задаёт параметры генераторов случайных чисел); **SHOW** (задаёт команду извне); **START** (начало сеанса моделирования, запуск программы на счёт); **STEP** (шаговое моделирование); **STOP** (останов работы программы); **STORAGE** (описание многоканального устройства); **TABLE** (описание таблицы гистограмм); **VARIABLE** (описание переменной). Команды предназначены для описания некоторых элементов GPSS World, которые требуют предварительного описания (таблицы, матрицы, многоканальные устройства) и управления ходом моделирования.

Таким образом, различия между командами и блоками следующие:

– блок начинает выполнять свою функцию, только если в него вошёл активный транзакт;

– команда помещается транслятором в специальный список команд в оперативной памяти отдельно от блоков и начинает работать, когда для неё выполняются необходимые условия (нажата соответствующая клавиша, обнулён *Счётчик завершения* и др.).

Некоторые команды и блоки взаимодействуют друг с другом.

Команды-описания блоков связаны с соответствующими блоками непосредственно. Например, с помощью команды **STORAGE** описывается каждое конкретное *Многоканальное устройство* (имя, число каналов), что позволяет нормально работать с ним соответствующим операторам **ENTER** (войти в *Многоканальное устройство*), **LEAVE** (покинуть *Многоканальное устройство*) и др.

Некоторые команды и блоки взаимодействуют друг с другом опосредованно. Например, команда **START** задаёт число в *Счётчик завершения*, а блок **TERMINATE** может уменьшать его содержимое. Когда оно становится меньше или равно нулю, программа останавливается, т. е. фактически срабатывает команда **HALT**.

2.4 Основные принципы работы имитационных моделей

Термин *имитационное моделирование* применяется ко всем моделям, где имитируется поведение реальных систем с учётом случайных явлений. Но в данном случае речь идёт о имитационном моделировании узкого класса организационных объектов, которые принято называть *системами с очередями* или *системами массового обслуживания*. Для них характерным является наличие следующих особенностей:

- появление активного элемента (транзакта) — клиента в банке, автомобиля на перекрёстке, заявки на обслуживание со стороны клиентской части программы и т. п., причём время появления и само появление может быть случайным (как частный случай может быть и *детерминированным*, т. е. точно известным);
- попадание транзакта в какое-либо устройство для обслуживания (*Обслуживающее устройство*) и выполнение с ним какой-то операции — обслуживание клиента в банке, проезд автомобилем перекрёстка, работа серверной части программы с клиентской частью и т. п., причём обслуживание может занять случайный интервал времени;
- возможное появление нового транзакта ещё до того, как будет обслужен предыдущий элемент, т. е. возникновение очереди и нахождение элемента в очереди до того момента, когда *Обслуживающее устройство* освободится и его можно будет занять.

Эти три особенности — транзакт, обслуживание и очередь (с протеканием процесса во времени) являются характерными для систем массового обслуживания. Задачей имитационной модели в таких системах является сбор статистики об исследуемых процессах: средние, максимальные и минималь-

ные значения очередей и времён (обслуживания, нахождения в очереди), число обслуженных транзактов и т. п. Но имитационная модель, вообще говоря, может отражать гораздо более сложные явления. В ней может быть несколько видов транзактов различной физической природы (например: клиенты и служащие банка; автомобили, их пассажиры и сигналы светофора; запросы компьютера и время), причём каждый транзакт может иметь какие-то параметры, которые могут изменяться *Обслуживающим устройством*, могут вычисляться значения каких-то переменных, заполняться таблицы, выполняться проверки условий, принудительно включаться и выключаться блоки и т. д. В целом это позволяет моделировать не только очереди, но также все явления в исследуемом объекте.

В процессе моделирования каждый новый транзакт становится активным и продвигается от блока к блоку, пока не возникают условия, препятствующие его продвижению (например, он начинает обслуживаться в каком-то устройстве и задерживается на время обслуживания). В этот момент времени может появиться следующий транзакт, затем ещё один и т. д. Таким образом, в модели накапливаются транзакты, которые двигаются от своего начала (блоки **GENERATE**) к своему окончанию (блоки **TERMINATE**). Некоторые транзакты могут оказаться в тупиковых ситуациях и никогда уже не выберутся из тех блоков, в которых "застряли". Другие ждут момента, когда они станут активными. Транзакт может приобрести активность, когда выполняются все необходимые для этого условия. В простейшем случае эти условия сводятся к наступлению времени активизации (например, в блоках **GENERATE**). Но могут быть другие условия. Например, транзакт-автомобиль может проехать через перекрёсток только если другой такой же транзакт отсутствует и транзакт-свет светофора имеет зелёный цвет. При анализе процессов в сборочном цехе имеет место более сложный процесс, когда несколько транзактов-сборочных единиц собираются вместе на конвейере и тогда образуется новый "обобщённый" транзакт, например, компьютер. Напротив, в некоторых случаях некий общий транзакт, например, автомобиль, пройдёт диагностику только в том случае, если связанные с ним транзакты-параметры будут проверены и все примут нужные значения.

В рамках имитационной модели какого-либо объекта может возникнуть необходимость использовать наряду с принципом дискретно-событийного моделирования также функциональное непрерывное моделирование на основе численного решения систем дифференциальных уравнений. Например, при моделировании работы перекрёстка желательно моделировать также динамику разгона и торможения автомобилей наряду с событиями переключении светофора, решением повернуть в ту или иную сторону и т. д. Поэтому имитационные модели должны позволять также это. В GPSS World реализованы такие возможности с помощью встроенного языка PLUS, аналогичного языкам паскаль и др.

Время работы программы задаётся с помощью внутреннего *Счётчка завершения*. Для этого извне с помощью команды **START** задают некоторое число, а внутри программы помещают блок **TERMINATE** с операндом, значение которого вычитается из значения *Счётчика завершения* каждый раз, когда в **TERMINATE** войдёт очередной транзакт. Остановить программу можно и принудительно с помощью горячей клавиши или кнопки **HALT** в одном из окон.

Моделирование может состоять из одного или нескольких сеансов. При этом существует несколько возможностей их организации:

– непрерывный один сеанс;

– несколько последовательных сеансов, прерываемых командой **HALT** и продолжаемых командой **CONTINUE** (или соответствующими горячими клавишами);

– несколько последовательных сеансов, в каждом из которых меняются некоторые параметры модели;

– несколько сеансов при разных параметрах модели, начинаемых с исходного состояния (просмотр нескольких вариантов).

При прогоне нескольких вариантов с разными параметрами можно задать диапазон изменения этих параметров и шаг их изменений и организовать прогонку с использованием метода планирования эксперимента — тогда появляется возможность также выявить характер зависимостей между какими-то параметрами и результатами модели. Планирование эксперимента организуется в GPSS World автоматически с помощью специальных команд.

Если в ходе имитационного моделирования отдельные подпроцессы или явления требуют представления с помощью классических приёмов программирования, то для этого используется встроенный язык PLUS, напоминающий упрощённый Pascal. Работа фрагментов программы на языке PLUS может протекать во времени и тогда это время становится модельным временем (учитывается при определении моментов наступления других событий). Если время не учитывается, то фрагмент программы запускается активным транзактом, отрабатывается, а полученные результаты используются затем в других частях программы блоками GPSS World.

2.5 Цепи транзактов

Транзакты создаются оператором **GENERATE** и продвигаются в модели от блока к блоку и в конечном счёте уничтожаются оператором **TERMINATE** — их может быть много и они могут находиться в разных местах модели. В каждый момент времени, когда должно наступить какое-либо событие, необходимо выбрать транзакт для передмещания. Поскольку транзактов может быть много и они могут обладать различными параметрами, влияющими на перемещение, процесс просмотра каждый раз *всех* транзактов может оказаться весьма трудоёмким, а время моделирования — слишком

большим. Поэтому транзакты предварительно классифицируются различным образом и помещаются в различного рода списки, называемые *Цепями транзактов* (***Transaction Chains***). Это позволяет *Планировщику* (программе управления моделированием) каждый раз не просматривать те транзакты, которые заведомо не обладают нужными свойствами для перемещения в данный момент. Каждый транзакт обязательно находится в каком-либо списке, т. е. в какой-либо *Цепи*. В GPSS World имеются следующие списки (*Цепи*):

Цепь будущих событий (***Future Events Chain***);

Цепь текущих событий (***Current Events Chain***);

Цепь задержки для *Обслуживающих устройств* и *Многоканальных устройств* (Facility or Storage ***Delay Chain***);

Цепь ожидания для *Обслуживающих устройств* (Facility ***Pending Chain***);

Цепь прерываний (***Interrupt Chains***);

Цепь повторений (***Retry Chain***);

Цепь пользователя (***User Chain***).

Транзакты могут находиться лишь в одной из *Цепей* (*Цепь будущих событий*, *Цепь текущих событий*, *Цепь задержки*, *Цепь пользователя*) или одновременно в нескольких. Некоторые *Цепи* связаны с вполне определёнными типами блоков.

Цепь текущих событий (Current Events Chain)

Цепь текущих событий (***Current Events Chain*** — CEC) представляет собой список транзактов в порядке их предстоящего превращения в активный транзакт. Активный транзакт продвигается по программе, в то время как остальные ждут своей очереди. *Цепь текущих событий* взаимодействует с *Цепью будущих событий*. Транзакт, находящийся в *Цепи текущих событий*, становится активным и продвигается по программе от блока к блоку, пока условия в программе позволяют ему оставаться активным. Когда по каким-то причинам активный транзакт задерживается в каком-то месте программы, он исключается из *Цепи текущих событий* и записывается в другую *Цепь*, соответствующую его новому положению, в частности, перемещается в *Цепь будущих событий*. Если транзакт покидает *Цепь текущих событий* через оператор **`TERMINATE`**, то он уже больше никуда не записывается.

Цепь будущих событий (Future Events Chain)

Цепь будущих событий (***Future Events Chain*** — FEC) содержит упорядоченные по времени транзакты, которые ждут своей очереди стать активными и перейти в *Цепь текущих событий*.

Блоки **`GENERATE`** и **`ADVANCE`** помещают активный транзакт в *Цепь будущих событий*. Эти блоки прибавляют время, указанное в их операнде, к текущему времени и вычисляют значение будущего момента времени, когда данный транзакт должен будет стать активным. У всех помещённых в *Цепь будущих событий* транзактов имеется указание на момент времени, когда

они должны стать активными. Системные часы переходят последовательно от одного такого момента к другому, активируя соответствующие транзакты. При этом определяется, в какой блок должен войти активизируемый транзакт, и этот блок начинает работать, выполняя возложенную на него функцию.

Блоки **PREEMPT** и **DISPLACE** удаляют транзакты из *Цепи будущих событий*. Такие транзакты могут быть проверены блоком **ADVANCE**. Когда *Планировщик* обнаруживает, что в *Цепи текущих событий* нет активного транзакта, он берёт его из *Цепи будущих событий*. Это приводит к продвижению модельного системного времени до момента, соответствующего этому событию.

Цепь задержки (Delay Chain)

Цепь задержки (***Delay Chain***) представляется собой список транзактов, которые были помещены в неё из-за того, что операторы **SEIZE** или **PREEMPT** не приняли их, например, из-за низкого приоритета (в это время был взят на обслуживание транзакт с более высоким приоритетом). Помещённые в *Цепь задержки* транзакты будут ожидать своей очереди на обслуживание. Возможность их обслуживания проверяется после освобождения *Обслуживающего устройства* от обслуженного транзакта и из них выбирается тот, который обладает на момент выбора наибольшим приоритетом. *Цепь задержки* — наиболее часто используемая программой, так как она присуща каждому *Обслуживающему устройству*.

Цепь повторений (Retry Chains)

Существуют блоки, которые проверяют различные условия, прежде чем пропустить транзакт (**GATE**, **TEST**, **TRANSFER ALL**, **TRANSFER BOTH**). Если условия не выполняются, то соответствующий транзакт блокируется и помещается в *Цепь повторений*. Там он ждёт, когда условия изменятся и он будет пропущен соответствующим блоком. Таким образом, *Цепь повторений* (***Retry Chain***) содержит список транзактов, которые были приостановлены в продвижении операторами **GATE** (Переместить в зависимости от состояния), **TEST** (Переместить в зависимости от сравнения), **TRANSFER ALL** (переместить в новое место ВСЁ), **TRANSFER BOTH** (переместить в новое место ОБА). Т. е. в *Цепи повторений* сохраняются транзакты, перемещение которых обусловлено сравнением с каким-то условием, которое пока не выполнено. При последующих просмотрах последовательности операторов программа каждый раз проверяет выполнение этих условий и выбирает тот транзакт, для которого условие выполнилось, чтобы переместить его по назначению (в соответствии с оператором).

Цепь прерываний (***Interrupt Chain***) содержит транзакты, которые выгружаются при входе в блок **PREEMPT** (перехват: "Отстранить транзакт от обслуживания") транзакта с бóльшим приоритетом, чем активный транзакт, на-

ходящийся уже в *Обслуживающем устройстве*. Новый транзакт "перехватывает" это *Обслуживающее устройство*, а старый (недообслуженный) транзакт помещается в *Цепь прерываний*. В последующем при возникновении благоприятных условий обслуживание этого транзакта может быть возобновлено. *Цепь прерываний* и *Цепь повторений* различаются тем, что в *Цепи прерываний* содержатся невпущенные транзакты, а в *Цепи прерываний* — принудительно выгруженные.

Цепь Многоканальных устройств

Каждое *Многоканальное устройство* имеет по две *Цепи*:

Цепь задержки (DELAY CHAIN) — содержит транзакты, которые ожидают входа в *Многоканальное устройство* в соответствии с приоритетом и очередью;

Цепь повторений (RETRY CHAIN) — содержит транзакты, которые ожидают входа в *Многоканальное устройство*; их вход был невозможен по каким-то причинам, например, из-за выключения *Многоканального устройства*.

Цепь задержки содержит транзакты, ожидающие входа в *Многоканальное устройство*. Когда транзакт готовится войти в оператор **ENTER**, чтобы попасть в *Многоканальное устройство*, запрашивается число его доступных каналов. Максимальное число каналов задаётся командой **STORAGE** при описании соответствующего *Многоканального устройства*. В свою очередь, транзакт может запрашивать для занятия не один, а несколько каналов. Если запрос транзакта не удовлетворён, то транзакту запрещается входить в блок **ENTER**, он помещается *Цепь задержки* данного *Многоканального устройства* и он перестаёт быть активным. Программа ищет новый активный транзакт. Когда какой-либо транзакт покидает *Многоканальное устройство*, программа просматривает *Цепь задержки* и отыскивает транзакт с наибольшим приоритетом, чтобы попробовать вновь ввести его в *Многоканальное устройство*. При этом используется дисциплина обработки транзактов "первый годный с пропуском" ("first fit with skip"), при которой каждый транзакт проверяется по его запросу на требуемое число каналов и если запрос удовлетворяется, то соответствующий транзакт впускается в блок **ENTER** и помещается в *Цепь текущих событий*. Если запрос не удовлетворяется, то транзакт остаётся в *Цепи задержки*.

Многоканальное устройство может находиться в различных состояниях, в частности, в выключенном. Если в таком состоянии оно не может принять транзакт, то он помещается в *Цепь повторений*. После изменения состояния *Многоканального устройства* (например, его включения) транзакты из этой *Цепи* начинают поступать в блок **ENTER**.

2.6 Фунционирование Цепей транзактов

Основными являются *Цепь текущих событий* и *Цепь будущих событий*.

В *Цепь текущих событий* включены все события, которые запланированы на текущий момент модельного времени, не зависимо от того, являются они условными или безусловными. *Планировщик* просматривает в первую очередь *Цепь текущих событий* и пытается выявить те транзакты, которые готовы для перемещения. Если в *Цепи текущих событий* таких транзактов нет, то *Планировщик* обращается к *Цепи будущих событий* и отыскивает в ней события ближайшего большего момента времени, переносит их в *Цепь текущих событий* и вновь проверяет её.

В начальный момент моделирования после команды **START** *Планировщик* обращается ко всем блокам **GENERATE**, в которых заданы свои генераторы случайных чисел и поэтому можно определить время появления очередных транзактов в каждом из них — эти моменты заносятся в *Цепь будущих событий*. Затем *Планировщик* обращается к *Цепи текущих событий* с целью поиска транзактов, готовых к перемещению. Поскольку таких транзактов в начальный момент нет, то *Планировщик* просматривает *Цепь будущих событий* и выбирает из неё все события, которые должны состояться в ближайший момент модельного времени. Поскольку все события при имитационном моделировании связаны с перемещением транзактов, которых пока нет, то в начальный момент времени именно появление транзактов является ближайшим событием. Это может быть один транзакт, а может быть и несколько: блок **GENERATE** может быть запрограммирован на появление нескольких транзактов одновременно или могут совпасть времена появления транзактов в нескольких блоках **GENERATE**. *Планировщик* в действительности обрабатывает их последовательно, но модельное время для всех — одно и то же.

Если в первый рабочий момент модельного времени появляется один транзакт, то он помещается в *Цепь текущих событий*, становится активным и *Планировщик* прытается продвинуть его по модели. Если появляется одновременно несколько транзактов, то проверяяются их приоритеты и устанавливается очередь: транзакты с бóльшим приоритетом оказываются ближе к началу очереди. Если приоритеты одинаковые, то очерёдность транзактов соответствует последовательности их появления в модели.

Если транзакт вошёл в блок **ADVANCE**, то он в нём задерживается для обработки, причём заранее можно вычислить с помощью параметров этого блока момент освобождения этого транзакта: в результате этот транзакт переносится в *Цепь будущих событий*. В последующем *Планировщик* будет регулярно просматривать *Цепь будущих событий* и в определённый момент времени извлечёт этот транзакт и вновь переместит его в *Цепь текущих событий*.

В процессе моделирования могут возникнуть ситуации, что перемещение каких-либо транзактов заблокировано в связи с невыполнением некоторых условий. Если продолжать каждый раз просматривать эти транзакты, то это может занять много процессорного времени. Поэтому целесообразно

контролировать указанные условия (они связаны с конкретными блоками модели) и в случае их прежнего состояния не анализировать задержанные ими транзакты (число условий обычно намного меньше числа задержанных из-за них транзактов). Таким образом вводится понятие *Цепей задержки*, куда и перемещаются такие транзакты из *Цепи текущих событий*. Если в некоторй момент времени условия изменились, то в первую очередь проверяются транзакты, размещённые в *Цепи задержки*, связанной с этим условием.

2.7 Системные числовые атрибуты (System Numerical Attributes)

В процессе работы программы транзакты перемещаются по модели от блока к блоку, в некоторых блоках они задерживаются, в некоторых изменяются их параметры. Одновременно с этим сама программа собирает информацию о процессах в блоках. Например, для всех *Очередей* собирается следующая информация: число вошедших и вышедших транзактов, а также время нахождения транзакта в очереди. Одновременно производится анализ этой информации и выполняются некоторые простейшие статистические вычисления, определяются: максимальное, минимальное и среднее число транзактов в очереди; минимальное, максимальное и среднее время нахождения транзакта в очереди; всё это — с учётом и без учёта транзактов, которые вошли в очередь, но прошли её без задержки по любым причинам (например, в очереди никого не было или они имели приоритет).

Вся эта информация может быть использована в процессе моделирования для выбора действий в качестве условий формирования тех или иных событий. Для получения доступа к этим статистическим параметрам используются специальные средства — *Системные числовые атрибуты* (СЧА) (System Numerical Attributes). Они представляют собой особую форму записи подпрограмм-функций, извлекающих необходимую информацию из внутренних массивов данных (Приложение Б).

Каждый тип устройств (Очереди, Обслуживающие устройства, Многоканальные устройства, Матрицы и др.) имеют свой набор СЧА. Общий вид записи СЧА: ***X$Y***, где знак доллара является условным обозначением СЧА, слева от которого записывается условное обозначение типа СЧА (в виде одной буквы), а справа — имя конкретного блока, информацию о котором требуется получить. Например, выражение Q$Ochered означает, что требуется узнать текущий размер очереди (Q) с именем Ochered. В этом смысле СЧА можно считать также упорядоченным видом переменных, так как результатом их работы является извлечение и предоставление значения соответствующего параметра. С ними можно работать, как с обычными переменными, т. е. умножать, использовать для определения номера блока (блоки можно задавать именами или номерами или и тем и другим способом одновременно) и т. д. В современных языках программирования СЧА могли бы записываться, например, в виде функции, тогда это могло бы выглядеть так: Q(Ochered).

3 Разработка имитационных моделей

Наиболее наглядно видны возможности имитационного моделирования применительно к организационно-техническим объектам, где важную роль играют не только технических средства, но и взаимодействие между ними и человеком.

Примерами *организационных объектов* являются: организации различных организационно-правовых форм (предприятия, банки, общественные объединения, государственные учреждения и т. п.), органы власти (исполнительной и законодательной, федеральной и муниципальной), конкретные мероприятия (конгрессы, встречи, аукционы, митинги, шествия, спортивные соревнования и т. п.), суды, адвокатские конторы, средства массовой информации и др. Они функционируют, производя промышленную продукцию, услуги, законодательные акты, управленческие решения и т. д. Каждый такой организационный объект имеет свою структуру, т. е. взаимосвязанные и взаимодействующие элементы. Через эти элементы внутри организации протекают различные потоки — материальные, финансовые, информационные и др. Каждый организационный объект для выполнения своих функций использует различные ресурсы — материальные, финансовые, человеческие, интеллектуальные, информационные и др. В большинстве случаев организационные объекты взаимодействуют с внешней средой и получают от неё воздействия различного рода, способствующие или препятствующие выполнению их функций. В обществе организационные объекты взаимодействуют между собой и таким образом образуют организационные системы.

Рассмотрим пример организационного объекта.

Промышленное предприятие занимается производством определённого вида продукции (например, хлеба). Хлеб развозят по магазинам. При этом известны средние цифры себестоимости одного килограмма хлеба и его стоимость в магазине. Можно посчитать доход и прибыль.

Всю цепочку производства-сбыта можно разделить на несколько этапов:

- производство;
- хранение готовой продукции;
- транспортировка её в магазины;
- продажа;
- получение дохода и прибыли.

Каждый этап можно детализировать.

Например, **на этапе производства** продукцию получают путём последовательной переработки исходного сырья в нескольких цехах через некие технологические цепочки. При этом работает оборудование и персонал, выбирающий режимы работы. Всё это занимает определённое время и требует определённого объёма сырья,

Готовая продукция поступает на склад и там хранится до отправки в магазины. Продукция продаётся через сеть магазинов в городе и вывозится со

склада специальным транспортом по расписанию. Продукция вырабатывается в определённое время суток, а развозится в другое время суток. На всех работах задействован персонал. Существуют расходы на сырьё, электроэнергию, тепло, аренду помещений, оплату работников, на транспортировку, ремонт оборудования. Необходимо учесть ряд факторов: наличие, объём и поступление исходного сырья, время работы оборудования в различных режимах, возможность его поломки , необходимость покупки запасных частей и время работы (с использованием соответствующего ремонтного первонала), скорость транспортировки на склад и со склада в магазины, последовательность переезда от магазина к магазину, затрачиваемое время погрузки и разгрузки, заработную плату всех работников, налоговые отчисления и др. Интерес могут представлять доходы фирмы, ритмичность её работы, достаточность задействованных ресурсов (оборудования, исходного сырья,)

Исследование систем можно проводить различным образом. Наиболее плодотворным является имитационное моделирование.

Имитационное моделирование характеризуется прежде всего сочетанием двух факторов — неопределённости и возможности ветвления процессов в зависимости от конкретных реализаций этой неопределённости.

При имитационном моделировании система представляется потоком событий, происходящих в отдельные моменты времени, т. е. дискретно. Такое представление и такое моделирование называют дискретно-временны́м. Его отличие от функционального моделирования (например, на основе решения систем алгебро-дифференциальных уравнений) в том, что события происходят в соответствии с заданными интервалами времени и в зависимости от уже произошедших событий. Интервалы времени (моменты совершения событий) вычисляются, как правило, с помощью генераторов случайных чисел с тем или иным законом распределения вероятностей.

При имитационном моделировании все события в системе происходят в соответствии с продвижением некоего индикатора времени, который называют *транзактом*. Транзактов может быть много, но в каждый момент времени активным является только один, который и вызывает то или иное событие. при этом движение транзакта может сопровождаться и не сопровождаться изменением времени. Физический смысл каждого транзакта может быть различным. Это могут быть люди, внешние воздействия, различного рода сигналы и т. п. Имитационная модель может проследить траектории прохождения транзактов при различных условиях, собрать статистическую информацию о них (среднее время нахождения в том или ином блоке, количество прошедших через блок транзактах и др.). Современные системы имитационного моделирования могут включать в себя также блоки функционального моделирования (например, язык имитационного моделирования GPSS World содержит специальное расширение — язык PLUS, во многом напоминающий упрощённый язык Паскаль).

Таким образом, имитационное моделирование может обеспечить решение различных исследовательских задач:

- определение реального алгоритма работы той или иной системы с учётом вероятностных характеристик отдельных элементов и сигналов и различных условий;
- вычисление статистических характеристик (средние, максимальные и минимальные значения, коэффициент использования);
- оптимизация структуры или параметров исследуемой системы;
- поиск сбоев и неисправностей в реальной системе и причин их возникновения;
- создание компьютерных деловых игр как компонентов систем поддержки принятия решений.

Имитационное моделирование может применяться для любых типов систем. Среди таких систем выделяются организационные. Именно их удобно исследовать методами имитационного моделирования.

3.1 Основные этапы моделирования в программной среде GPSS World

Методология и технология моделирования в программной среде GPSS World в целом аналогичны таковым в других программных средах, но имеется также ряд отличий. Они связаны со спецификой объектов моделирования, самого языка GPSS World, целей имитационных экспериментов, предоставляемого инструментария по созданию программных моделей и собственно моделирования.

В общем виде моделирование включает следующие этапы:

– анализ объекта моделирования;

– разработка функциональной (математической) модели;

– разработка обобщённого алгоритма программной модели применительно к языку GPSS World;

– разработка программной модели на языке GPSS World;

– отладка программной модели на языке GPSS World;

– подготовка модельного эксперимента в GPSS World;

– проведение модельного эксперимента в GPSS World.

Анализ объекта моделирования имеет целью выявить протекающие в нём процессы и влияющие на них особенности объекта, т. е. его структуру (элементы и связи между ними) и параметры (элементов и связей). Первоначально в распоряжении разработчика модели имеется лишь словесное описание объекта моделирования. На его основе разработчик должен выявить, что́ необходимо моделировать (обобщённая цель моделирования), какими являются реальные процессы в объекте, какие характеристики этих процессов существенны, а какие — нет. Несущественные характеристики (параметры) объекта отбрасываются и не учитываются при моделировании. При исследо-

ваниях объекта методами имитационного моделирования все физические процессы представляются информационными процессами. Для каждого физического процесса выявляется его смысл (что происходит в реальном объекте), его входные и выходные величины, функциональные связи между входами и выходами. Для них ищутся эквивалентные информационные соответствия (например, сложность процесса может заменяться временем его протекания и т. д.). Здесь же определяется назначение модели, т. е. выделяются из прочих все процессы, которые будут моделироваться, а также определяется, какие их характеристики необходимо получать в результате моделирования. Результатом этого этапа является структурная схема объекта моделирования, отображающая моделируемые процессы как последовательность преобразований входных переменных в выходные.

Разработка функциональной (математической) модели основывается на полученной ранее структурной схеме объекта моделирования. В структурной схеме объекта моделирования показываются потоки между различными объектами (элементами), а в функциональной модели математическими средствами отражаются функциональные преобразования, которые происходят с этими объектами. При этом используются как математические функции, так и алгоритмические операции. Результатом этого этапа является функциональная модель в виде схемы с указанием функций и операций.

Функциональная (математическая) модель объекта представляет собой представление объекта на том или ином математическом языке: языке алгебраических выражений и уравнений, языке дифференциального или интегрального исчисления, языке алгебры логики, алгоритмов и т. п. При этом осуществляется расчленение объекта на составные элементы, определение функций каждого элемента и представление их в виде математических зависимостей, увязка элементов между собой. Такая модель может быть представлена в виде структурной или функциональной схемы, которая наглядно отобразит поведение потока транзактов, операции их обработки и т. п. Структурная (функциональная) схема может строиться с использованием различных условных графических обозначений — как стандартизованных, так и не стандартизованных.

Разработка обобщённого алгоритма программной модели применительно к языку GPSS выполняется на основе функциональной модели с учётом базовых понятий языка GPSS, формально отражённых в его операторах, синтаксисе и технологии программирования. При этом происходит преобразование структуры математической модели в структуру программной модели. Если моделируемый процесс не имеет ветвлений, то программа имеет простую последовательную структуру и её структурирование сводится к выделению фрагментов программы, отвечающих за конкретные последовательности операций, для лучшего понимания прохождения процесса. Если имеются ветвления в зависимости от условий или циклы, то эти структурные особенности должны тщательно анализироваться и отображаться в структуре

алгоритма, что упрощает в последующем написание текста программы. На основе структурной (или функциональной) модели разрабатывается программная модель применительно к операторам языка GPSS. При этом используются условные графические обозначения в соответствии с требованиями Единой системы программной документации (ЕСПД). Необходимо учитывать, что в GPSS выполнение программы осуществляется не в только в порядке следования операторов, но прежде всего — в порядке происхождения событий, связанных с прохождением активных транзактов. Фактически любая программа на GPSS представляет собой цикл относительно Планировщика GPSS, который просматривает всю программу и выявляет события, которые готовы произойти (дискретно-событийное моделирование). Этот внешний цикл не отражается в обобщённом алгоритме, но подразумевается. Результатом этого этапа является структура алгоритма, изображённая с помощью стандартных *условных графических обозначений*.

Разработка программной модели на языке GPSS сводится к поэлементному уточнению алгоритма и преобразованию его в текст программы. На этой стадии выбираются решения по программированию всех операций (обычно существует несколько вариантов). Выбирается та или иная технология программирования. Применительно к GPSS технологии программирования несколько отличаются от таковых для алгоритмических языков общего назначения, так как GPSS отражает идеологию событийно-временного моделирования. Кроме того, в GPSS сочетаются концепции объектно-ориентированного и структурного программирования, что накладывает отпечаток на технологию разработки программ. С позиций структурного программирования можно использовать нисходящее (от общего к частному) или восходящее (от частного к общему) программирование, а также методы постепенного усложнения программы или другие. Результатом этого этапа является текст работоспособной программы на GPSS.

Программная модель на языке GPSS записывается с обязательными комментариями, которые можно располагать как перед операторами, так и справа от них. Комментарии, расположенные перед операторами, никак не связаны с процессом работы программы. Комментарии справа от операторов могут быть использованы для отладки программы в шаговом режиме. Это необходимо учитывать, придавая этим комментариям соответствующий смысл, который может помочь затем отлаживать программу.

Обычно программа на GPSS состоит из нескольких частей — модулей. Расположение этих модулей часто не имеет значения, так как они срабатывают не в порядке их расположения, а в порядке следования событий, связанных с модулями. Для этого следует выделить в программе эти части и классифицировать их. Такими частями могут быть:

– описание *Матриц*, *Многоканальных устройств* и т. п.;

– задание начальных значений *Сохраняемых величин* и *Матриц*;

– задание функций пользователя;

– описание *Включаемых файлов* (команда **INCLUDE**);

– таймеры различного рода (начало и окончание работы программы, выработка прерываний и т. п.);

– обработка массивов переменных (обычно *Матриц*) в цикле;

– операции ввода исходной информации из фалов;

– вывод результатов в файлы и др.

Располагать эти модели в программе, можно, как правило, произвольным образом, так как благодаря предварительной трансляции программы, её исполняемый вариант (который останется в оперативной памяти) всегда будет сформирован нужным образом. Поэтому расположение модулей в программе определяется главным образом *стилем программирования* на GPSS. Поэтому предварительно желательно определиться со стилем написания программы. Например, можно все описания (матриц, *Многоканальных устройств*, начальных значений *Сохраняемых величин* и др.) размещать в начале программы. Это позволит видеть, какие структурные элементы заданы. Но можно помещать их и в конец программы, чтобы они не мешали рассмотрению работающих операторов. Можно размещать задания и описания в начале того модуля, в котором они работают (если они работают только в одном модуле). Таким образом, существует возможность выработать и затем использовать свой стиль программирования на GPSS.

Модули программы GPSS можно разделить, в зависимости от функций, на два больших класса: исполняемые и описательные. Описательные модели не содержат транзактов, а исполняемые обязательно их содержат.

В некоторых модулях осуществляется генерирование собственных транзактов, которые в этом же модуле и уничтожаются. Поэтому такие модули обрамляются операторами **GENERATE** и **TERMINATE**. Имеются также модули, в которых используются сторонние транзакты, попадающие в них из других модулей. Тогда в них отсутствуют операторы **GENERATE**, но могут присутствовать операторы **TERMINATE**. Есть модули, которые порождают транзакты, а затем направляют их в другие модули, поэтому в них имеются операторы **GENERATE**, но отсутствуют операторы **TEMINATE**. В целом должен выполняться общий принцип построения программы: в оператор **GENERATE** не должны входить посторонние транзакты — они должны или обходить операторы **GENERATE**, или уничтожаться на подходе к ним.

Отдельный вопрос — о выборе типа оператора **TERMINATE**, который может не только уничтожать транзакт, но и влиять на содержимое *Счётчика завершения* (**TERMINATE 0** или **TERMINATE A**. где **A** — число, не равное нулю). Необходимо чётко определиться, по какой физической величине (по каким событиям, которым соответствуют определённые транзакты в модели) будет определяться момент окончания моделирования. Число этих событий задаётся командой **START**, задающей начальное максимальное значение *Счётчика завершения*, а затем содержимое *Счётчика завершения* уменьша-

ется при каждом выполнении операции **TERMINATE** с операндом, отличным от нуля.

Внутри модулей выполнение операторов соответствует их расположению и логике программы. Последовательность выполнения программы соответствует траектории перемещения активного транзакта. Поэтому для выполнения того или иного модуля, того или иного оператора требуется активизировать соответствующий транзакт.

Отладка программной модели на языке GPSS заключается в том, чтобы выявить возможные ошибки программы, ликвидировать их и добиться корректной работы программы во всех режимах при всех возможных сочетаниях входных параметров (на предыдущем этапе работоспособность программы выражалась лишь в том, что транслятор не выдавал ошибки, а процессы в целом соответствовали ожидаемым). В процессе отладки разрабатываются: стратегия отладки, тестовые примеры, процедуры тестирования. Результатом этого этапа является корректно работающая программа, готовая для проведения модельных экспериментов.

Отладка программной модели, с одной стороны, проводится в соответствии с общими принципами верификации программ (поиск ошибок программирования), а с другой — учитывает особенности дискретно-событийного характера моделирования с помощью GPSS. Для отладки используется специальный инструментарий, предусмотренный оболочкой GPSS:

– шаговый режим;
– комментарии справа от операндов;
– журнал (***Journal***);
– окна (***Simulation Window***);
– снимки (***Simulation Snapshot***).

Каждый из этих инструментов выполняет свою функцию, но наибольший эффект достигается при их совместном использовании.

Основой отладки программы является шаговый режим, который позволяет обеспечить перемещение транзакта к следующему оператору. При этом в журнале появляется запись об очередном шаге и комментарий, соответствующий каждому оператору (расположенный справа от оператора и в одной с ним строке). Поэтому целесообразно создавать комментарии, однозначно указывающие на произошедшее событие. В GPSS имеются также так называемые *Окна* (***Simulation Window***) — специальные окна, в которых выводятся значения различных переменных, графики, гистограммы, анимированные события.

Общим принципом отладки программ в GPSS является установка необходимых окон, мозаичное группирование их на экране удобным образом, запуск программы в шаговом режиме и проверка значений переменных, вида выводимых графиков и т. п. после каждого шага.

В GPSS имеются следующие окна:

Blocks Window *— Окно блоков.*
Expression Window *— Окно выражений.*
Facilities Window *— Окно обслуживающих устройств.*
Logicswitches Window *— Окно логических ключей.*
Matrix Window *— Окно матриц.*
Plot Window *— Окно графиков.*
Queues Window *— Окно очередей.*
Savevalues Window *— Окно сохраняемых величин.*
Storages Window *— Окно многоканальных устройств.*
Table Window *— Окно гистограмм.*

Для работы с этими окнами необходимо после трансляции программы (***Command → Create Simulation***) использовать команды меню ***Window → Simulation Window***, чтобы попасть на вкладку с выше перечисленными названиями окон и вызвать необходимое окно. Без оттранслированной программы соответствующие команды меню неактивны.

Некоторые из окон вызываются всегда (*Окно блоков* ***Blocks Window***, *Окно выражений* ***Expression Window***, *Окно обслуживающих устройств* ***Facilities Window***), так как информация для них имеется в любой программе. Некоторые окна вызываются только, если соответствующая им информация имеется в программе (*Окно очередей* ***Queues Window*** — если есть очереди, *Окно матриц* ***Matrix Window*** — если есть матрицы, *Окно гистограмм* ***Table Window*** — если для гистограмм собирается соответствующая информация с помощью команды **TABULATE**, *Окно логических ключей* ***Logicswitches Window***, *Окно сохраняемых величин* ***Savevalues Window***, *Окно многоканальных устройств* ***Storages Window***).

Blocks Window (*Окно блоков*) визуально показывает перемещение транзактов через блоки (операторы и команды программы) с отражением мгновенной информации. Оно в компактной форме представляет собой список операторов программы в той последовательности, в какой они присутствуют в программе. Операторы помечены условными графическими обозначениями, номерами (по порядку размещения в программе) и сокращёнными наименованиями. При запуске программы слева от блоков появляются условные значки, означающие появление транзактов, их перемещение, появление очередей, переполнение очередей и т. п.

Facilities Window *— Окно обслуживающих устройств*, в которых выполняется обслуживание (обработка) транзактов. Все транзакты обслуживаются в этих устройствах в течение некоторого времени. Можно получить информацию о мгновенных, максимальных, средних временах обслуживания транзактов.

Logicswitches Window *— Окно логических ключей*, являющихся специфическими объектами GPSS, задаваемых специальным оператором **LOGIC**. Они бывают необходимы, чтобы выполнять некоторые условные операции. Транзакты могут переключать *Логические ключи* в одно из двух состояний (0

или 1). В любом другом месте программы оператор может проанализировать состояние того или иного *Логического ключа* и в зависимости от него выполнить то или иное действие. Например, может фиксироваться проход транзакта через какое-то *Обслуживающее устройство* в середине программы и на основании этого приниматься решение в конце программы.

Plot Window — *Окно графиков*, предназначенное для визуализации результатов моделирования с помощью СЧА. Графики имеют стандартный вид и имеют несколько простейших настроек (масштаб по осям и др.).

Queues Window — *Окно очередей*. *Очереди* — разновидность устройств в GPSS. Очереди возникают, если *Обслуживающее устройство* ***Facilities*** не готово принять транзакт по каким-то причинам (занято, у транзакта не тот приоритет и др.). Очереди в программе могут быть или не быть заданы — не зависимо от того, имеются ли они в действительности. Задание имени очереди означает, что в процессе моделирования для неё начинает накапливаться информация, которую и можно посмотреть в *Окне очередей*.

Storages Window — *Окно многоканальных устройств* обслуживания, у которых все каналы идентичные. Вместо них можно использовать соответствующее количество одноканальных устройств, но, во-первых, при большом количестве каналов (десятки, сотни, тысячи) получится необозримая программа; во-вторых, чтобы собирать и обрабатывать информацию в этих каналах, придётся писать дополнительные фрагменты программы большого размера. Поэтому для *Многоканальных устройств* используется специальный оператор **STORAGE**. Если в программе такие операторы отсутствуют, то *Окно многоканальных устройств* будет пусто. Если они есть, то в *Окне многоканальных устройств* появится таблица. В первой колонке содержится список *Многоканальных устройств* из программы, а рядом с ним — некоторая информация.

Table Window — *Окно гистограмм*. В нём выводятся статистические результаты, накопленные в процессе работы программы. Гистограмма бывает полезной как в конце модельного эксперимента, так и в ходе отладки программы. В последнем случае она динамически пересчитывается и изменяется после каждого шага. Полезно наблюдать за ней и в процессе непрерывного модельного эксперимента, так как она в динамике показывает изменение распределения статистических величин.

Report — *Стандартный отчёт* является тем документом, в котором накапливаются основные результаты моделирования. Он позволяет получить полное представление о том, чем завершилось моделирование, содержит максимальные и средние значения различных переменных и др. Рассмотренные выше *Окна* позволяют детализировать информацию, имеющуюся в *Стандартном отчёте* и визуализировать её, но этой информации обычно самой по себе достаточно, чтобы делать выводы о результатах моделирования.

Для некоторых конкретных типов блоков (*Очередей*, *Обслуживающих устройств*, *Логических ключей*, *Матриц*, *Многоканальных устройств*, *Сохраняемых величин*) существуют специальные окна (соответственно, ***Queue Window***, ***Facilities Window***, ***Logicswitches Window***, ***Matrix Window***, ***Storages Window***, ***Savevalues Window***), в которые по умолчанию выводится большое число параметров, характерных для этих блоков. Но может потребоваться вывод и других параметров, тогда следует воспользоваться окном ***Expression Window***. Для других блоков специальных окон нет и тогда окно ***Expression Window*** является единственным, где можно посмотреть их состояние. Кроме простого вывода СЧА, можно выполнять арифметические выражения с ними и получать таким образом более гибкую систему контроля параметров протекающих процессов.

Expression Window — *Окно выражений* показывает динамику изменения параметров модели в процессе моделирования, которые можно задать с помощью выражений. В качестве выражений используются *Стандартные числовые атрибуты* (СЧА) совместно с переменными, имеющими имя: например, **Q$***Имя* — означает длину очереди с именем этой очереди, определённым в программе. Например, **Q$*Barber*** означает, что в программе имеется очередь, названная ***Barber***. Часть выражения — **Q$** (с долларом справа) является *Стандартным числовым атрибутом* с определённой функцией (в данном случае функцией определения значения переменной ***Barber***, являющейся очередью).

Для повышения результативности отладки программы в неё могут включаться дополнительные операторы. Часто для контроля изменения значений внутренних переменных программы (которые не должны выводиться среди конечных результатах) оказывается целесообразно создавать вспомогательные *Сохраняемые величины* (***Savevalues***), так как для их вывода существует уже окно ***Savevalues Window***. Если требуется контролировать различные параметры *Очередей*, *Обслуживающих устройств* или *Многоканальных устройств*, то можно использовать универсальное *Оно выражений* ***Expression Window***. В нём можно выполнять также математические операции над значениями переменных. В некоторых случаях это помогает быстрее проверить корректность программы. При анализе содержимого *Окон* необходимо ясно представлять, какие именно данные выводятся — для активного транзакта или для уже прошедшего ранее события. Это определяется на этапе формирования в программе информации для соответствующих *Окон*.

Подготовка модельного эксперимента в GPSS заключается в разработке его стратегии и тактики. Стратегия модельного эксперимента определяет, что именно и для чего будет рассчитывать программа, что считать входными параметрами эксперимента, а что — выходными. Здесь же выявляются критерии оценки результатов эксперимента (например, минимум какой-то величины соответствует оптимальной настройки, и т. п.). Результатом

этого этапа является обобщённая постановка задачи эксперимента (стратегия), а также план эксперимента (тактика).

Цель модельного эксперимента при имитационном моделировании зависит от объекта моделирования и протекающих в нём процессов, а потому может быть различной. Но в целом цель моделирования может быть отнесена к одному из следующих классов:

– определение результирующих статистических характеристик процесса (коэффициента использования устройств, гистограммы распределения событий, максимальных, минимальных и средних значений случайных величин и т. п.);

– определение траекторий движения процессов (по связанным с ними транзактам) при различных условиях работы (при нормальных условиях, аварийных ситуациях);

– проверка работоспособности тех или иных механизмов реализации процессов в объекте;

– поиск параметров, влияющих на результирующие показатели процесса, и определение степени этого влияния (под результирующими показателями можно понимать статистические показатели, работоспособность и др.);

– поиск оптимальной структуры объекта;

– поиск оптимальных параметров объекта.

При подготовке модельного эксперимента решают несколько взаимосвязанных проблем:

– выбираются величины, которые необходимо рассчитать в результате эксперимента;

– выбирают величины, которые необходимо задать постоянными (параметры);

– выбирают величины, которые необходимо варьировать в процессе эксперимента.

При построении плана эксперимента можно воспользоваться методами планирования эксперимента, но это не всегда возможно, так как цели экспериментов могут быть разными (методы планирования эксперимента обеспечивают минимизацию числа варьируемых значений входных переменных для изучения характера зависимости между входными и выходными величинами).

Проведение модельного эксперимента в GPSS включает ряд операций:

- трансляция программы на GPSS;
- выбор окон GPSS для отражения текущей информации;
- задание исходных параметров (и, возможно, перетрансляция программы);
- запуск программы оператором **START** (или иным способом — в зависимости от сути эксперимента);

• получение стандартного отчёта и результатов в окнах GPSS;

• сохранение полученных результатов данного модельного эксперимента (с возможным перенесением их в таблицы результатов);

• задание новых исходных данных;

• перетрансляция программы и запуск на счёт при новых исходных данных и т. д.:

• обработка результатов эксперимента при всех прогонах программы;

• анализ результатов и формулирование выводов.

Приведённая краткая характеристика этапов моделирования раскрывается далее более подробно и иллюстрируется конкретными примерами.

Проведение модельного эксперимента выполняется с применением инструментов GPSS, которые и определяют технологию моделирования.

Инструменты включают в себя:

– *Окна GPSS*;

– возможность управления процессом извне (без вмешательства в программу);

– возможность шагового режима моделирования;

– возможность вмешательства в программу с перетрансляцией и продолжением моделирования без потери информации (можно сохранить *Стандартный отчёт* и *Журнал*).

Последовательность действий зависит от моделируемого объекта и целей моделирования, но в целом включает те же этапы, что и отладка модели:

– на экране монитора устанавливаются и удобным образом размещаются все необходимые окна (они могут сохраняться при перетранслировании программы после вмешательства в неё);

– готовятся, если необходимо, файлы исходных данных (для включения оператором **INCLUEDE** или для считывания операторами работы с файлами);

– выбираются режимы работы (автоматический или шаговый);

– делается необходимое количество шагов (или задаётся необходимое значение счётчика завершения оператором **START**);

– анализируются промежуточные результаты;

– выполняются все прогоны программы, предусмотренные планом модельного эксперимента.

Для анализа процессов используется визуализация результатов с помощью инструментов программы GPSS или результаты выводятся в файл на диск, а затем представляются с помощью других программ (MS Word, Excel и т. д.).

Для последующего анализа результатов сохраняются записи в *Журнале* и *Стандартном отчёте*, которые затем могут быть перенесены (при необходимости) в текстовые отчёты по результатам модельного эксперимента.

3.2 Некоторые приёмы программирования в GPSS World

3.2.1 Создание и уничтожение транзактов

В рамках концепции *дискретно-событийного моделирования* все события в имитационной модели GPSSW происходят только в связи с транзактами. Поэтому создание транзактов является непременным условием функционирования модели. Для этого используется блок **GENERATE**. В процессе моделирования возможно появление большого и даже неограниченного числа транзактов, каждый из которых физически представляет собой массив ячеек памяти, в которые записываются его параметры (номер, приоритет и др.). В GPSSW предусмотрено динамическое распределение памяти, поэтому генерирование транзактов очень быстро может привести к заполнению всей памяти компьютера. В GPSSW для размещения транзактов, кроме оперативной памяти, используется также дисковая память, что несколько улучшает ситуацию, но, во-первых, дисковая память также не безгранична, а во-вторых, её использование существенно замедляет процесс моделирования. Поэтому все отработавшие транзакты должны быть удалены, для чего используется блок **TERMINATE**. Хотя бы один блок **GENERATE** является обязательным в любой модели, а блок **TERMINATE** — желательным.

Количество блоков **GENERATE** и **TERMINATE** не ограничено, но при этом следует учитывать следующие обстоятельства.

Как правило, в реальной модели каждый транзакт имеет какой-то физический смысл. Это, например, клиент, момент времени, деталь, самолёт и т. д. Транзакты одного типа генерируются своим блоком **GENERATE**, а блок **TERMINATE** может использоваться для всех типов транзактов.

Модель должна быть построена таким образом, чтобы транзакты никогда не входили в блок **TERMINATE**. Вообще говоря, на этапе трансляции модели программа не проверяет такую возможность, поэтому не выдаёт ошибку даже при явном нарушении этого правла. Но если в процессе работы модели какйо-то транзакт попытается войти в блок **GENERATE**, возникнет ошибка. С точки зрения программирования этого можно избежать различным образом: можно каждому блоку generate поставить в соответствие свой блок **TERMINATE** или с помощью операторов изменения траектории движения транзактов **TRANSFER** "обходить" встречающиеся блоки **GENERATE**.

Оператор **GENERATE** выполняет единственную функцию (хотя и сразличными дополнительными возможностями) — создание транзакта, в то время как оператор **TERMINATE** — две функции: уничтожение транзакта (с освобождением ячейки памяти) и уменьшение содержимого *Счётчика завершения*. При этом уничтожение транзакта осуществляется всегда, а взаимодействие со *Счётчиком завершения* — только по специальному указанию.

Блок GENERATE

Блок **GENERATE** предназначен для создания потока транзактов и может содержать до 5 операндов:

```
GENERATE A,B,C,D,E
```

A — временной интервал появления транзакта; *B* — половина интервала возможного отклонения времени появления *C* — начальная временна́я задержка; *D* — максимальное число транзактов, которое должно быть сгенерировано; *E* — приоритет генерируемых транзактов.

Примеры:

```
GENERATE 14
```

означает, что каждый новый транзакт появляется непрерывно через каждые ровно 14 единиц времени и имеет приоритет 0.

```
GENERATE 14,5
```

означает, что каждый новый транзакт появляется непрерывно через каждые 14±5 единиц времени (с равномерным распределением времени появления в интервале от 9 до 19 единиц времени) и имеет приритет 0.

```
GENERATE 14,5,100
```

означает, что первый транзакт появится через 100 единиц времени, а каждый последующий новый транзакт будет появляться непрерывно через каждые 14±5 единиц времени (с равномерным распределением времени появления в интервале от 9 до 19 единиц времени) с приритетом 0.

```
GENERATE 14,5,100,86
```

означает, что первый транзакт появится через 100 единиц времени, а каждый последующий новый транзакт будет появляться непрерывно через каждые 14±5 единиц времени (с равномерным распределением времени появления в интервале от 9 до 19 единиц времени), всего будет сгенерировано 86 транзактов, каждый — с приоритетом 0

```
GENERATE 14,5,100,86,47
```

означает, что первый транзакт появится через 100 единиц времени, а каждый последующий новый транзакт будет появляться непрерывно через каждые 14±5 единиц времени (с равномерным распределением времени появления в интервале от 9 до 19 единиц времени), всего будет сгенерировано 86 транзактов, каждый — с приоритетом 47.

Любой из указанных операндов может быть опущен: если опускаются все последние операнды, то они просто не записываются; если опускается промежуточный операнд, то вместо него ставится запятая, например:

```
GENERATE 14,,,7
```

означает, что каждый новый транзакт появляется непрерывно через каждые ровно 14 единиц времени и имеет приоритет 7.

Операндами могут быть не только числа, но и переменые, в качестве которых могут выступать СЧА, функции **FUNCTON** и переменные **VARIABLE**.

Если необходимо генерировать

Блок TERMINATE

Блок **TERMINATE** предназначен для вывода транзакта из модели (с освобождением занимамых им ячеек памяти) и уменьшения значения *Счётчика завершения* может содержать до 1 операнда или ни одного:

```
TERMINATE A
```

A — число, на которое уменьшается значение *Счётчика завершения*, задаваемое перед началом моделирования командой **START**, при вхождении транзакта в блок **TERMINATE**.

Сложные модели могут содержать несколько фрагментов со своими блоками **GENERATE**, создающими отдельные независимые потоки транзактов, возможно, различной физической природы. Они должны двигаться так, чтобы не входить в другие блоки **GENERATE**. Поэтому, чтобы отделить один фрагмент от другого, каждый из них может завершаться своим блоком **TERMINATE**. Но можно иметь в программе единственный блок **TERMINATE**, на который перенаправлять транзакты перед очередным блоком **GENERATE** с помощью специальных блоков, например, блока **TRANSFER**.

В модели может быть много блоков **TERMINATE**, при этом всегда нужно хорошо представлять себе, необходимо ли, чтобы конкретный блок не только уничтожал вошедший в него транзакт, но и взаимодействовал со *Счётчиком завершения*. В сложных моделях, как правило, большинство блоков **TERMINATE** не имеют операнда, а потому только уничтожают транзакты. Если не будет ни одного блока **TERMINATE** с ненулевым операндом, то программа сможет работать бесконечно с возможностью прерывания её командой **HALT** (путём нажатия соответствующей кнопки на панели любого из окон). Возможна также работа в пошаговом режиме **STEP**. Но в этом случае не появляется автоматически *Стандартный отчёт*. Для автоматического окончания работы хотя бы один блок **TERMINATE** должен иметь ненулевой операнд с воздействием на уменьшение значения *Счётчика завершения*.

3.2.2 Движение транзактов и его изменение

3.2.2.1 Возможности изменения движения транзактов

Работа модели представляет собой непрерывное перемещение транзактов, так как все блоки выполяют действие только в том случае, если в них вошёл транзакт. Нормальным образом транзакты движутся сверху вниз, переходя последовательно от одного блока к другому. Существует большое количество блоков, которые позволяют изменить направление движения транзактов в зависимости от различных обстоятельств: **ALTER**, **DISPLACE**, **EXAMINE**, **FUNAVAIL**, **GATE**, **LINK**, **LOOP**, **OPEN**, **PREEMPT**, **READ**, **REMOVE**, **SCAN**, **SELECT**, **SPLIT**, **TRANSFER**, **TEST**, **UNLINK**, **WRITE**. Из них специально для изменения направления движения транзактов предназначен блок **TRANSFER**, остальные — решают другие задачи и попутно могут задавать новое место для перемещения транзактов, если не выполняются учитываемые ими условия. Ниже рассматриваются блоки **TRANSFER**, **TEST**. Остальные блоки рассматриваются в других разделах.

Блок **TRANSFER** предназначен для измененения направления движения транзактов, может работать в 9 режимах и имеет следующий синтаксис:

TRANSFER *A,B,C,D*

Операнды:

A — указатель режима, который может иметь значения (ключевые слова):

пробел — безусловный переход;

число с точкой — статистический режим;

BOTH — выбор незанятого устройства из двух;

ALL — выбор незанятого устройства из нескльких;

PICK — случайный выбор направления из нескольких;

P — переход по значению параметра;

FN — переход по значению функции;

SIM — переход по индикатору задержки;

SBR — переход на подпрограмму с возможностью возврата;

B,*C*,*D* — параметры режимов (для каждого режима — свои).

Режим безусловного перехода

В этом режиме входящий в блок **TRANSFER** транзакт направляется на указанную метку:

TRANSFER *,met*

Здесь список операндов начинается с запятой; это означает, что вместо операнда *A* записан пробел как указатель режима безусловного перехода. На

месте операнда *B* размещается метка ***met***, на которую перенапрвляется транзакт. Примеры использования блока **TRANSFER** в режиме безусловного перехода приведены ниже.

Обход участков программы (Рисунок 3.1) предусматривает, что первоначально транзакт обрабатывается более удалённым *Обслуживающим устройством* 2 (для этого он направляется через фрагмент программы, связанный с *Обслуживающим устройством* 1, на метку ***t20***), а затем возвращается для обработки на *Обслуживающее устройство* 1 (метка ***t10***).

```
          GENERATE  6               ; Генерирование очередного транзакта
          TRANSFER  ,t20            ; Переход на метку t20
;--------- Обработка транзактов на устройстве 1
t10       SEIZE     1
          ADVANCE   12,2
          RELEASE   1
          TERMINATE 1               ; Уход транзакта
;--------- Обработка транзактов на устройстве 2
t20       SEIZE     2
          ADVANCE   12,4
          RELEASE   2
          TRANSFER  ,t10            ; Переход на метку t10
```

Рисунок 3.1 — Блок **TRANSFER** в режиме безусловного перехода. Обход участков программы

```
;--------- Поток транзактов 1
          GENERATE  6,2
          QUEUE     1
          SEIZE     1
          DEPART    1
          ADVANCE   12,2
          RELEASE   1
          TRANSFER  ,tFin           ; Переход на метку tFin
;--------- Поток транзактов 2
          GENERATE  6,3
          QUEUE     2
          SEIZE     2
          DEPART    2
          ADVANCE   12,4
          RELEASE   2
          TRANSFER  ,tFin           ; Переход на метку tFin
;--------- Завершение обработки транзакта и его уход
tFin      QUEUE     QFin
          SEIZE     Fin
          DEPART    QFin
          ADVANCE   12,2
          RELEASE   Fin
          SAVEVALUE XFin+,1         ; Подсчёт всех транзактов
          TERMINATE 1               ; Уход транзакта
```

Рисунок 3.2 — Блок **TRANSFER** в режиме безусловного перехода. Направление всех транзактов на общую часть модели

Иногда оказывается удобным направлять все транзакты, обрабатываемые в разных местах программы, на один конечный блок обработки (Рисунок 3.2).

В рассматриваемой модели существуют три разных потока транзактов, генерируемых тремя блоками **GENERATE**, каждый из них обрабатывается своим фрагментом программы, а затем направляется на общий фрагмент конечной обработки (метка ***tFin***), в том числе для подсчёта числа полностью обработанных транзактов с помощью блока **SAVEVALUE**, и на один для всех блок вывода транзакта из модели **TERMINATE**.

Статистический режим

В этом режиме входящий в блок **TRANSFER** транзакт направляется на одну из двух меток с заданной вероятностью:

TRANSFER *A*,***met1***,***met2***

Здесь:

A — десятичная дробь (число с точкой), показывающее, какую долю транзактов необходимо направить на метку ***met2***; на метку ***met1*** направляется (1-*A*) доля транзактов;

met1 — метка, на которую направляется (1-*A*) транзактов;

met2 — метка, на которую направляется *A* транзактов.

В примере (Рисунок 3.3) представлен типичный вариант, когда часть транзактов должна обрабатываться одним устройством, а другая часть — вторым устройством.

```
          GENERATE  6                ; Поток транзактов
          TRANSFER  .43,t01,t02      ; 43% на метку t02, 57% на метку t01
;--------- Обработка транзактов 1
t01       QUEUE     1
          SEIZE     1
          DEPART    1
          ADVANCE   10,1
          RELEASE   1
          SAVEVALUE Num01+,1         ; Подсчёт транзактов метки t01
          TRANSFER  ,tFin            ; Переход на метку tFin
;--------- Обработка транзактов 2
t02       QUEUE     3
          SEIZE     3
          DEPART    3
          ADVANCE   9,2
          RELEASE   3
          SAVEVALUE Num02+,1         ; Подсчёт транзактов метки t02
          TRANSFER  ,tFin            ; Переход на метку tFin
;--------- Уход транзакта
tFin      SAVEVALUE XFin+,1          ; Подсчёт всех транзактов
          TERMINATE 1                ; Уход транзакта
```

Рисунок 3.3 — Блок **TRANSFER** в статистическом режиме

Через каждые 6±2 единицы времени генерируется очередной транзакт, который блоком **TRANSFER** направляется на одну из меток: 43% от общего числа транзактов идут на метку ***t02*** и (100–43=57)% — на метку ***t01***. Для каждого вошедшего в блок **TRANSFER** транзакта выбор метки осуществляется случайным образом и при малом числе вошедших транзактов указанные пропорции не соблюдаются. Но при увеличении числа транзактов можно обнаружить, что реальные пропорции распределения их по меткам приближаются к заданным: при 5 транзактах пропорции составляют 1/4; при 10 — 4/6; при 100 — 43/57 (полное совпадение); при 1000 — 423/577 и т. д.

В статистическом режиме всегда делается выбор между двумя направлениями движения транзактов и нельзя использовать большее количество направлений.

Режим BOTH — выбор незанятого устройства из двух

В этом режиме входящий в блок **TRANSFER** транзакт направляется на один из двух блоков модели, готовых принять транзакт:

```
TRANSFER BOTH,met1,met2
```

Здесь:

met1 — первая метка;

met2 — вторая метка.

В режиме **BOTH** при входе транзакта в блок **TRANSFER** начинается просмотр меток (вначале первая, затем вторая) и проверяется, может ли транзакт войти в блок, помеченный проверяемой меткой. Эти метки могут соответствовать различным типам блоков. Некоторые блоки (**SEIZE**, **ENTER**) могут находиться в режиме "свободно" или "занято". Именно в этом случае целесообразно применение блока **TRANSFER** в режиме **BOTH**. Если блок с проверяемой меткой свободен (в блоке **SEIZE** нет транзактов, в блоке **ENTER** транзактами заняты не все каналы), то активный транзакт направляется в этот блок. Если блок с проверяемой меткой занят транзактами (в блоке **SEIZE** присутствует 1 транзакт, в блоке **ENTER** транзактами заняты все каналы), то транзакт остаётся в блоке **TRANSFER** и начинает проверяться блок, помеченный второй меткой. Если и второй блок занят, то транзакт остаётся в блоке **TRANSFER** до следующей проверки, пока не освободится какой-либо блок из помеченных метками. Блок **TRANSFER** может удерживать любое количество транзактов, которые внутри него выстраиваются в очередь с учётом приоритетов.

В примере (Рисунок 3.4) рассматривается простейший вариант использования режима **BOTH**.

В данном случае в качестве объектов, на которые напрвляются транзакты, использованы одноканальные *Обслуживающие устройства* — блоки **SEIZE**, помеченные метками ***t01*** и ***t02***.

```
          GENERATE  5                ; Поток транзактов
          TRANSFER  BOTH,t01,t02     ; Выбор незанятого устройства
;--------- Обработка транзактов 1
t01       SEIZE     1
          ADVANCE   10,1
          RELEASE   1
          SAVEVALUE Num01+,1         ; Подсчёт транзактов метки t01
          TRANSFER  ,tFin            ; Переход на метку tFin
;--------- Обработка транзактов 2
t02       SEIZE     2
          ADVANCE   9,2
          RELEASE   2
          SAVEVALUE Num02+,1         ; Подсчёт транзактов метки t02
          TRANSFER  ,tFin            ; Переход на метку tFin
;--------- Уход транзакта
tFin      SAVEVALUE XFin+,1          ; Подсчёт всех транзактов
          TERMINATE 1                ; Уход транзакта
```

Рисунок 3.4 — Блок **TRANSFER** в режиме **BOTH**.
Выбор незанятого устройства из двух *Обслуживающих устройств*.

Каждый из них обрабатывает поступающие транзакты. Внутри соответствующих фрагментов программы выполняются задержки (блоки **ADVANCE**) на заданное время, а также подсчёт обработанных транзактов (блоки **SAVEVALUE**). Каждый фрагмент программы направляет обработанные транзакты на общий выход (метка ***tFin***), где посчитывается общее число обработанных транзактов.

```
          GENERATE  5                ; Поток транзактов
          TRANSFER  BOTH,t01,t02     ; Выбор незанятого устройства
;--------- Обработка транзактов 1
t01       QUEUE     1
          SEIZE     1
          DEPART    1
          ADVANCE   10,1
          RELEASE   1
          SAVEVALUE Num01+,1         ; Подсчёт транзактов метки t01
          TRANSFER  ,tFin            ; Переход на метку tFin
;--------- Обработка транзактов 2
t02       QUEUE     2
          SEIZE     2
          DEPART    2
          ADVANCE   9,2
          RELEASE   2
          SAVEVALUE Num02+,1         ; Подсчёт транзактов метки t02
          TRANSFER  ,tFin            ; Переход на метку tFin
;--------- Уход транзакта
tFin      SAVEVALUE XFin+,1          ; Подсчёт всех транзактов
          TERMINATE 1                ; Уход транзакта
```

Рисунок 3.5 — Блок **TRANSFER** в режиме **BOTH**.
Выбор незанятого устройства из двух *Очередей*

Если в качестве целевого блока использовать, например, *Очередь* (блок **QUEUE)**, которая способна принимать любое количество транзактов

(Рисунок 3.5), то все транзакты будут направляться на *Очередь* с меткой ***t01***, так как *Очередь* не выдаёт сигнала о занятости.

Точно также будет вести себя программа, если использовать другие типы блоков, которые способны задерживать транзакты: они не будут информировать блок **TRANSFER**, направивший на них транзакт, о невозможности его принятия, а просто задержат его в себе. Это означает, что в этом случае все транзакты будут входить в первый из помеченных блоков. Другое дело, если первым указан блок с возможностью отказа, а вторым — без возможности отказа.

Если в качестве целевого блока используется *Многокальное устройство* (блок **ENTER** с описанием командой **STORAGE**), то его занятость связана не с числом вошедших в него транзактов, а с занятостью всех его каналов (Рисунок 3.6). Если блок **ENTER** содержит указание при входе транзакта занимать 1 канал (например, по умолчанию при отсутствии операнда *B*), то число занятых каналов равно числу вошедших транзактов. Но если при входе в *Многоканальное устройство* транзакт занимает более 1 канала (задаётся операндом *B*), то при анализе занятости будет учитываться этот параметр: например, если свободен 1 канал, а *B*=2, то *Многоканальное устройство* будет считаться занятым.

```
Mu1       STORAGE   5
;---------
          GENERATE  2                ; Поток транзактов
          TRANSFER  BOTH,t01,t02     ; Выбор незанятого устройства
;--------- Обработка транзактов 1
t01       SEIZE     Ustr1
          ADVANCE   10,1
          RELEASE   Ustr1
          SAVEVALUE Num01+,1         ; Подсчёт транзактов метки t01
          TRANSFER  ,tFin            ; Переход на метку tFin
;--------- Обработка транзактов 2
t02       ENTER     Mu1,2
          ADVANCE   9,2
          LEAVE     Mu1,2
          SAVEVALUE Num02+,1         ; Подсчёт транзактов метки t02
          TRANSFER  ,tFin            ; Переход на метку tFin
;--------- Уход транзакта
tFin      SAVEVALUE XFin+,1          ; Подсчёт всех транзактов
          TERMINATE 1                ; Уход транзакта
```

Рисунок 3.6 — Блок **TRANSFER** в режиме **BOTH**.
Выбор незанятого устройства из двух *Очередей*

В приведённом примере (Рисунок 3.6) используются разнотипные целевые блоки: одноканальное *Обслуживающее устройство* **SEIZE** (метка ***t01***) и *Многоканальное устройство* **ENTER** (метка ***t02***) с числом занимаемых при входе транзакта каналов 2. При указанных параметрах в **ENTER** никогда

не войдёт больше 2 транзактов, так как общее число его каналов 5, а каждый транзакт при входе занимает 2 канала.

Режим ALL — выбор незанятого устройства из нескольких

В этом режиме входящий в блок **TRANSFER** транзакт направляется на один из нескольких блоков модели, готовых принять транзакт:

```
TRANSFER ALL,met1,met2,N
```

Здесь:

met1 — метка первого целевого блока в диапазоне;

met2 — метка последнего целевого блока в диапазоне;

N — приращение (в строках).

В примере (Рисунок 3.7) блок **GENERATE** создаёт новый транзакт через каждые 4 единицы времени.

```
          GENERATE  4                ; Генерирование очередного транзакта
          TRANSFER  ALL,t01,t02,4    ; Распределение по блокам
;---------
t01       SEIZE     1
          ADVANCE   12,4
          RELEASE   1
          TERMINATE 1
;---------
          SEIZE     2
          ADVANCE   12,6
          RELEASE   2
          TERMINATE 1
;---------
t02       SEIZE     3
          ADVANCE   12,8
          RELEASE   3
          TERMINATE 1
```

Рисунок 3.7 — Блок **TRANSFER** в режиме **ALL**.
Выбор первого незанятого блока **SEIZE** из заданных
(одноканальные *Обслуживающие устройства*)

Этот транзакт входит в блок **TRANSFER**, который работет в режиме **ALL**. Он просматривая строки с метки ***t01*** по метку ***t01*** через 4 строки и пытается направить вошедший транзакт в первый встетившийся свободный блок. В данном случае в этих строках находятся блоки **SEIZE** для *Обслуживающих устройств* с номерами соотвественно 1, 2 и 3. Просмотр начинается с первого блока с меткой ***t01***. Если этот блок не занят, то транзакт переходит в него и далее выполняются действия в соответствии с моделью. Если блок с меткой ***t01*** занят, то анализируется блок, отстоящий от метки ***t01*** на 4 строки — в данном слчае это блок с меткой ***t02***: если он свободен, то транзакт направляется в него, а если занят — в блок, отстоящий от метки ***t02*** на 4 строки —

в данном слчае это блок с меткой ***t03***: : если он свободен, то транзакт направляется в него. Если все блоки заняты, то транзакт остаётся в блоке **TRANSFER** до следующей попытки.

Эта попытка будет произведена после тог, как произойдёт новое событие: после порождения блоком **GENERATE** нового транзакта (он попадёт в очередь после всех) или после окончания задержки какого-либо ранее прошедшего транзакта в блоках **ADVANCE** и прохождения им соответствующего блока **RELEASE**, особождающего соответствующее *Обслуживающее устройство*.

Если операнд *D* не задан, то по умолчанию принимается шаг 1 строка.

В рассмотренном примере (Рисунок 3.7) в качестве принимающих используются блоки *Обслуживающих устройств*, но вместо них могут быть любые другие блоки, так как в параметрах указываются метки программы, которыми можно пометить любые строки.

При этом необходимо, чтобы проверяемые блоки располагались с фиксированным интервалом строк (в рассмотренном примере интервал равен 4 строкам).

Разумеется, этот приём является некоторым усложнением программирования, так как при модификации модели число строк между соседними целевыми блоками может измениться и шаг в операнде **TRANSFER** придётся корректировать. Поэтому режим **ALL** не очень удобен для использования, его можно заменить другими блоками, например, блоком **TEST** с проверкой занятости нужных блоков в цикле. Но если модель хорошо отлажена, то его вполне можно применять, так как он простейшим образом обеспечивает решение задачи выбора свободного блока для перемещения транзакта.

Особенностью рассмотренного примера (Рисунок 3.7) является то, что в качестве целевых блоков были заданы блоки **SEIZE**, которые способны принять только один транзакт, т. е. блоки, которые могут быть полностью заняты. При использовании других типов целевых блоков будут другие результаты.

Если будут проверяться, например, блоки **QUEUE** (Рисунок 3.8), которые могут принимать неограниченно количество транзактов, то каждая проверка будет показывать незанятость первого блока **QUEUE** (строка 3), вследствие чего все транзакты будут направляться только в него.

Если будут проверяться, например, блоки **ENTER** (Рисунок 3.9) для занятия *Многоканальных устройств*, которые могут принимать ограниченное количество транзактов, то транзакты будут направляться в более близкие блоки **ENTER** до тех пор, пока все их каналы не будут заняты, а лишь затем — в более дальние.

Естественно, что поскольку в режиме **ALL** оперируют с метками, то проверяемые целевые блоки могут быть разного типа, например, **SEIZE** и **ENTER**.

```
          GENERATE  4                ; Генерирование очередного транзакта
          TRANSFER  ALL,3,15,6       ; Распределение по блокам
;---------
          QUEUE     1                ; строка 3 (комментарии не учитываются)
          SEIZE     1
          DEPART    1
          ADVANCE   12,4
          RELEASE   1
          TERMINATE 1
;---------
          QUEUE     2                ; строка 9
          SEIZE     2
          DEPART    2
          ADVANCE   12,6
          RELEASE   2
          TERMINATE 1
;---------
          QUEUE     3                ; строка 15
          SEIZE     3
          DEPART    3
          ADVANCE   12,8
          RELEASE   3
          TERMINATE 1
```

Рисунок 3.8 — Блок **TRANSFER** в режиме **ALL** .
Выбор первого незанятого блока **QUEUE** из заданных (*Очереди*)

```
Ustr1     STORAGE   3                ; Многоканальное устройство 1; 3 канала
Ustr2     STORAGE   5                ; Многоканальное устройство 2; 5 каналов
Ustr3     STORAGE   7                ; Многоканальное устройство 3; 7 каналов
;---------
          GENERATE  1                ; Генерирование очередного транзакта
          TRANSFER  ALL,3,11,4       ; Распределение по блокам
;---------
          ENTER     Ustr1            ; строка 3 (комментарии не учитываются)
          ADVANCE   12,4
          LEAVE     Ustr1
          TERMINATE 1
;---------
          ENTER     Ustr2            ; строка 7
          ADVANCE   11,6
          LEAVE     Ustr2
          TERMINATE 1
;---------
          ENTER     Ustr3            ; строка 11
          ADVANCE   10,8
          LEAVE     Ustr3
          TERMINATE 1
```

Рисунок 3.9 — Блок **TRANSFER** в режиме **ALL**.
Выбор первого неполностью занятого блока **ENTER** из заданных
(*Многоканальные устройства*)

Более сложная ситуация возникает при использовании блоков, которые сами могут задерживать транзакты, например, **TEST**, **GATE** и др. Эти блоки не сообщают блоку **TRANSFER** о задержанных транзактах, так как способны

накапливать их в любом количестве, а поэтому возможна ошибка исполнения программы — если у *Обслуживающих устройств*, *Очередей* и *Многоканальных устройств* есть индикатор "занято" или "свободно", то у **TEST** и подобных ему такого индикатора нет, как нет и соответствующих СЧА.

Режим PICK — случайный выбор направления из нескольких

В этом режиме входящий в блок **TRANSFER** транзакт направляется на один из блоков модели, выбранный случайным образом:

TRANSFER PICK,*met1,met2*

Операнды:

met1 — метка первого целевого блока в диапазоне;

met2 — метка последнего целевого блока в диапазоне;

```
          GENERATE  2               ; Генерирование очередного транзакта
          TRANSFER  PICK,t01,t02    ; Распределение по блокам
;--------- Перенаправление транзактов
t01       TRANSFER  ,t11
          TRANSFER  ,t12
          TRANSFER  ,t13
          TRANSFER  ,t14
t02       TRANSFER  ,t15
;--------- Обработка 1
t11       SEIZE     1
          ADVANCE   12,1
          RELEASE   1
          TERMINATE 1
;--------- Обработка 2
t12       SEIZE     2
          ADVANCE   12,2
          RELEASE   2
          TERMINATE 2
;--------- Обработка 3
t13       SEIZE     3
          ADVANCE   12,3
          RELEASE   3
          TERMINATE 3
;--------- Обработка 4
t14       SEIZE     4
          ADVANCE   12,4
          RELEASE   4
          TERMINATE 4
;--------- Обработка 5
t15       SEIZE     5
          ADVANCE   12,5
          RELEASE   5
          TERMINATE 5
```

Рисунок 3.10 — Блок **TRANSFER** в режиме **PICK**.
Случайный выбор направления перемещения из заданных

В режиме **PICK** операнд *D* отсутствует и шаг всегда равен 1 строке.
Все выбираемые блоки (строки) выбираются с равной вероятностью.

В модели (Рисунок 3.10) существует 5 фрагментов программы для обработки транзакта.

Случайный выбор фрагмента осуществляется блоком **TRANSFER** в режиме **PICK** с перенаправлением транзактов с помощью вспомогатльных блоков **TRANSFER** в режиме безусловного перехода. В результате для каждого нового трназакта случайным образом выбирается фрагмент модели **Обработка** 1 ... **Обработка** 5. При этом не проверяется занятость блоков и транзакты накапливаются в соответствующих блоках **TRANSFER** фрагмента **Перенаправление транзактов**. Выбор всех направлений — равновероятный. Чтобы изменить тип вероятностного распределения, необходимо использовать другие способы.

Режим P — переход по значению параметра

В этом режиме входящий в блок **TRANSFER** транзакт направляется на один из блоков модели, выбранный по параметру:

TRANSFER P,*Param*,*met*

Операнды:

Param — номер или имя параметра активного транзакта, содержащего целые число;

met — метка: к номеру строки с этой меткой прибавляется значение ***Param***, чтобы получить номер строки, на которую направляется транзакт.

```
          GENERATE  ,,,3              ; Создаются 3 транзакта
          ASSIGN    Param,((XN1-1)#5); Номер вошедшего транзакта
          TRANSFER  P,Param,t01       ; Выбор направления для транзактов
;--------- Обработка транзактов 1
t01       SEIZE     Ustr1
          ADVANCE   10,1
          RELEASE   Ustr1
          SAVEVALUE Num01+,1          ; Подсчёт транзактов
          TRANSFER  ,tFin             ; Переход на метку tFin
;--------- Обработка транзактов 2
          SEIZE     Ustr2
          ADVANCE   9,2
          RELEASE   Ustr2
          SAVEVALUE Num02+,1          ; Подсчёт транзактов
          TRANSFER  ,tFin             ; Переход на метку tFin
;--------- Обработка транзактов 3
          SEIZE     Ustr3
          ADVANCE   8,3
          RELEASE   Ustr3
          SAVEVALUE Num03+,1          ; Подсчёт транзактов
          TRANSFER  ,tFin             ; Переход на метку tFin
;--------- Уход транзакта
tFin      SAVEVALUE XFin+,1           ; Подсчёт всех транзактов
          TERMINATE 1                 ; Уход транзакта
```

Рисунок 3.11 — Блок **TRANSFER** в режиме **P**.
Переход по значению параметра к последовательным строкам

Возможным вариантом использования рассматриваемого режима является переход транзакта последовательно на несколько строк в программе (Рисунок 3.11).

В данном случае (Рисунок 3.11) генерируется ограниченное количество транзактов — 3 шт. Каждому из них с помощью блока **ASSIGN** создаётся параметр ***Param***, которому присваивается значение, вычисляемое с помощью *Выражения в скобках* **((XN1-1)#5)**, которое имеет следующий смысл: *Системный числовой атрибут* **XN1** даёт номер активного транзакта, из которого вычитается 1, а полученный результат умножается на 5 (в GPSS **#** — знак умножения по умолчанию). Полученное значение записывается в параметр ***Param***, который в блоке **TRANSFER** складывается с номером строки, соответствующим метке ***t01***.

Для первого транзакта при вычислении значения параметра ***Param*** выражение в скобках даёт смещение относительно метки ***t01***: (1-1)×5=0 и блок **TRANSFER** определяет соответствующую строку с блоком **SEIZE** ***Ustr1***, в который первый транзакт направится, обслужится и продвинется дальше с переходом в конечном счёте на метку ***tFin*** и уходом из модели через блок **TERMINATE**.

Для второго транзакта при вычислении значения параметра ***Param*** выражение в скобках даёт смещение относительно метки ***t01***: (2-1)×5=5 и блок **TRANSFER** определяет соответствующую строку с блоком **SEIZE** ***Ustr2***, в который второй транзакт направится, обслужится и продвинется дальше с переходом в конечном счёте на метку ***tFin*** и уходом из модели через блок **TERMINATE**.

Для третьего транзакта при вычислении значения параметра ***Param*** выражение в скобках даёт смещение относительно метки ***t01***: (3-1)×5=10 и блок **TRANSFER** определяет соответствующую строку с блоком **SEIZE** ***Ustr3***, в который первый транзакт направится, обслужится и продвинется дальше с переходом в конечном счёте на метку ***tFin*** и уходом из модели через блок **TERMINATE**.

Так естественным образом обеспечивается сдвиг в точке перехода транзакта с учётом его номера.

Интересно, что в данном случае параметр ***Param*** используется не как *Системный числовой атрибут* (т. е. не в виде P$***Param***), а в своём обычном виде, как и в блоке **ASSIGN**. Это возможно потому, что блок **TRANSFER** работает в режиме параметра, о чём говорит указатель **P** в качестве первого операнда.

Другой вариант использования рассматриваемого режима (Рисунок 3.12) релизует попеременное направление транзактов на одну из двух строк, т. е. на одно из двух *Облуживающих устройств* — блоки **SEIZE**.

```
          INITIAL   LS$Num,0
;---------
          GENERATE  10,2              ; Поток транзактов
          ASSIGN    Param,(LS$Num#5)
          LOGIC I   Num
          TRANSFER  P,Param,t01       ; Выбор устройства
;--------- Обработка транзактов 1
t01       SEIZE     Ustr1
          ADVANCE   10,1
          RELEASE   Ustr1
          SAVEVALUE Num01+,1          ; Подсчёт транзактов
          TRANSFER  ,tFin             ; Переход на метку tFin
;--------- Обработка транзактов 2
          SEIZE     Ustr2
          ADVANCE   9,2
          RELEASE   Ustr2
          SAVEVALUE Num02+,1          ; Подсчёт транзактов
          TRANSFER  ,tFin             ; Переход на метку tFin
;--------- Уход транзакта
tFin      SAVEVALUE XFin+,1           ; Подсчёт всех транзактов
          TERMINATE 1                 ; Уход транзакта
```

Рисунок 3.12 — Блок **TRANSFER** в режиме **P**.
Переход по значению параметра попеременно на одну или другую строку

Для этого, во-первых, с помощью блока **ASSIGN** параметру ***Param*** транзакта присваивается значение, вчиляемое с помощью выражения **(LS$Num#5)**, которое имеет следующий смысл: произведение значения *Логического ключа* с именем ***Num*** (для его получения используется *Системный числовой атрибут* **LS$*Num***) на число 5 (знак **#** в GPSS по умолчанию означает операцию умножения, так как знак ***** используется в качестве индикатора комментариев). Во-вторых, в качестве операнда *C* в блоке **TRANSFER** используется метка ***t01***, относительно которой происходит рассчитанное смещение. *Логический ключ* — это переменная, которая может принимать одно из двух значений: 0 или 1. В данном случае при проходе каждого очередного транзакта через *Логический ключ* **LOGIC** его значсение изменяется на противоположное (он работает в режиме инвертирования своего значения, о чём говорит вспомогательный указатель **I**). *Логических ключей*, вообще говоря, может быть много, и данный имеет имя ***Num***. Чтобы установить его начальное значение в 0, используется команда **INITIAL** в начале программы (впрочем, по умолчанию начальное значение *Логического ключа* ***Num*** всё равно было бы равно 0).

Работает модель следующим образом. Начальное состоятие *Логического ключа* ***Num*** равно 0.

При появлении первого транзакта с помощью блока **ASSIGN** создаётся его параметр ***Param*** и ему присваивается значение (0×5)=0. Далее транзакт проходит блок **LOGIC** и соответствующий ему *Логический ключ* ***Num*** переключается в противоположное состояние 1. В блоке **TRANSFER** определяется

смещение относительно строки с меткой ***t01***. — для первого транзакта оно равно 0, поэтому транзакт идёт на метку ***t01***.

При появлении второго транзакта с помощью блока **ASSIGN** создаётся его параметр ***Param*** и ему присваивается значение (1×5)=5. Далее транзакт проходит блок **LOGIC** и соответствующий ему *Логический ключ* ***Num*** переключается в противоположное состояние 0. В блоке **TRANSFER** определяется смещение относительно строки с меткой ***t01***. — для второгоо транзакта оно равно 5 строк, поэтому транзакт идёт на 5 строк ниже метки ***t01*** — на блок **SEIZE** ***Ustr2***.

При появлении третьего транзакта повторится процесс, рассмотренный для первого транзакта, припоявлении четвёртого — рассмотренный для второго и т. д. Все нечётные транзакты будут идти на блок **SEIZE** ***Ustr1***, а все чётные — на блок **SEIZE** ***Ustr2***.

Можно найди и другие применения данного режима **P**, но, как и для режимов **ALL** и **PICK**, в данном случае некоторым неудобством является задание шага в виде количества строк: при модификации программы оно может меняться и об этом необходимо помнить.

Режим FN — переход по значению функции

В этом режиме входящий в блок **TRANSFER** транзакт направляется на один из блоков модели, выбранный по значению функции:

```
TRANSFER FN,Fun,N
```

Операнды:

Fun — номер или имя функции, в которой заданы метки для перехода транзактов;

N — целое число, прибавляемое к строке с меткой, определённой с помощью ***Fun***, чтобы получить номер строки, на которую направляется транзакт.

Один из вариантов использования рассматриваемого режима (Рисунок 3.13) релизует направление транзактов на последовательные фрагменты программы, например, *Облуживающие устройства* — блоки **SEIZE**.

В рассматриваемом примере (Рисунок 3.13) в начале описывается *Функция* (команда **FUNCTION**), у которой ***Fun*** — имя (по которому к ней затем можно обращаться), **XN1** — аргумент (в данном случае аргументом является *Системный числовой атрибут* **XN1**, означающий номер активного транзакта), **L*6*** — указатель типа *Функции* и его размера, 6 — число её значений. В соответствии с указателем **L*6*** под описанием *Функции* приводятся её значения парами {аргумент,функция}, разделённые знаком **/**. В данном примере предусмотрено создание 6 транзактов и направление их в три различных места программы.

```
Fun1      FUNCTION  XN1,D6           ; Номер транзакта - строка перехода
1,t01/2,t02/3,t03/4,t02/5,t03/6,t01
;---------
          GENERATE  ,,,6             ; Создаются 6 транзактов
          TRANSFER  FN,Fun1,0        ; Выбор направления для транзактов
;--------- Обработка транзактов 1
t01       SEIZE     Ustr1
          ADVANCE   10,1
          RELEASE   Ustr1
          SAVEVALUE Num01+,1         ; Подсчёт транзактов
          TRANSFER  ,tFin            ; Переход на метку tFin
;--------- Обработка транзактов 2
t02       SEIZE     Ustr2
          ADVANCE   9,2
          RELEASE   Ustr2
          SAVEVALUE Num02+,1         ; Подсчёт транзактов
          TRANSFER  ,tFin            ; Переход на метку tFin
;--------- Обработка транзактов 3
t03       SEIZE     Ustr3
          ADVANCE   8,3
          RELEASE   Ustr3
          SAVEVALUE Num03+,1         ; Подсчёт транзактов
          TRANSFER  ,tFin            ; Переход на метку tFin
;--------- Уход транзакта
tFin      SAVEVALUE XFin+,1          ; Подсчёт всех транзактов
          TERMINATE 1                ; Уход транзакта
```

Рисунок 3.13 — Блок **TRANSFER** в режиме **FN**.
Переход по значению *Функции*

При появлении первого транзакта блок **TRANSFER** вызывает *Функцию* ***Fun1***. При этом в соответствии с описанием *Функции* (в команда **FUNCTION**) в качестве аргумента берётся номер активного транзакта **XN1** и по нему определяется первая пара {аргумент,функция}={1,***t01***}, т. е. значение *Функции* принимается равным метке ***t01***. Относительно этой метки определяется смещение, заданное операндом ***C*** блока **TRANSFER**, который в данном случае равен 0, поэтому транзакт переходит на метку ***t01***. Там он обрабатывается *Обслуживающим устройством* **SEIZE** ***Ustr1*** и последовательно — другими блоками. Второй транзакт вызывает обращение ко второй паре {аргумент,функция}={2,***t02***}, поскольку смещение для всех транзактов равно 0, второй транзакт будет направлен на строку с меткой ***t02*** (*Обслуживающее устройство* **SEIZE** ***Ustr2***), и т. д. В данном примере первый транзакт переходит на метку ***t01***, второй — на метку ***t02***, третий — на метку ***t03***, четвёртый — на метку ***t02***, пятый — на метку ***t03***, шестой — на метку ***t01***. Этим демонстрируется характерная возможность данного режима работать, во-первых, с различными точками передачи транзактов, не обязательно разделёнными одинаковым числом строк (как в режиме **ALL** и **PICK**), а во-вторых, свободное изменение последовательности точек. Но при этом не предусматривается ни автоматическая проверка занятости целевых блоков (как в режиме **ALL**), ни случайный характер выбора направления перемеще-

ния транзактов (как в режиме **PICK**). В случае необходимости эти свойства придётся обеспечивать дополнительными средствами.

Режим SIM — переход по значению индикатора задержки

В этом режиме входящий в блок **TRANSFER** транзакт направляется на один из блоков модели, выбранный в зависимости от состояния *Индикатора задержки*:

```
TRANSFER SIM,Loc1,Loc2
```

Операнды:

Loc1 — строка, на которую транзакт направляется, если *Индикатор задержки* выключен;

Loc2 — строка, на которую транзакт направляется, если *Индикатор задержки* включен.

Режим **SIM** (Simultaneous = sɪməl'teɪnjəs = одновременный) обеспечивает возможность передачи транзакта с учётом того, был ли он задержан.

Заключение

В настоящее время созданы и продолжают создаваться новые варианты программ имитационного моделирования. Можно выделить два основных отличия их от GPSS World: во-первых, они всё более ориентируются на визуальное программирование с помощью условных графических обозначений блоков и связей между ними, а во-вторых, базируются на универсальных языках программирования типа Pascal, Java и т. п. Это позволяет, с одной стороны, упростить создание программных моделей сравнительно несложных объектов, используя всего несколько графических обозначений и связи между ними, а с другой стороны, в случае необходимости — использовать всю мощь базовых языков. Но как показывает опыт, графическое представление сложных объектов не упрощает, а усложняет программирование и восприятие модели. И в этом случае оказывается лучше классическое программирование. А потому GPSS World всё ещё оказывается востребованной не только в учебном процессе, но и в профессиональной деятельности.

Библиографический список

1 Боев, В. Д. Моделирование систем. Инструментальные средства GPSS World: учеб. пособие / В. Д. Боев. — СПб.: БХВ-Петербург, 2004. — 368 с. — ISBN 5-94157-515-7.

2 Емельянов, А. А. Имитационное моделирование экономических процессов: учеб. пособие / А. А. Емельянов, Е. А. Власова, Р. В. Дума; Под ред. А. А. Емельянова. — М.: Финансы и статистика, 2002. — 368 с. — ISBN 5-279-02572-0.

3 Кельтон, В. Имитационное моделирование. Классика CS. — 3-е изд. / В. Кельтон, А. Лоу. — СПб.: Питер; Киев: Издательская группа BHV, 2004. — 847 с. — ISBN 5-94723-981-7; ISBN 966-552-118-7.

4 Кудрявцев, Е. М. GPSS World. Основы имитационного моделирования различных систем / Е. М. Кудрявцев. — М.: ДМК Пресс, 2004. — 320 с. — (Серия "Проектирование"). — ISBN 5-94074-219-X.

5 Нейлор, Т. Машинные имитационные эксперименты с моделями экономических систем / Т. Нейлор. — М.: Мир, 1975. — 392 с.

6 Томашевский, В. Н. Имитационное моделирование в среде GPSS / В. Н. Томашевский, Е. Г. Жданова. — М.: Бестселлер, 2003. — 416 с. — ISBN 5-98158-004-6.

7 Фомин, Г. П. Системы и модели массового обслуживания в коммерческой деятельности: учеб. пособие / Г. П. Фомин. — М.: Финансы и статистика, 2000. — 144 с. — ISBN 5-279-02307-8

8 Шеннон, Р. Е. Имитационное моделирование систем: наука и искусство / Р. Е. Шеннон. — М.: Мир, 1978. — 420 с.

9 Шмидт, Б. Искусство моделирования и имитации. Введение в имитационную систему Simpex 3 / Б. Шмидт; перевод на русский язык и научное редактирование д. т. н., проф. Ю. А. Ивашкина и д. т. н., проф. В. Л. Конюха. — Дельфт; Эрланген; Гент; Сан-Диего: Международное общество моделирования и имитации SCS. Европейское изд-во, 2003. — [463 с.]

10 Шрайбер, Т. Дж. Моделирование на GPSS / Т. Дж. Шрайбер. — М.: Машиностроение, 1979. — 592 с.

Приложение А.
Математические операции в GPSS World

Операторы математических операций используются для того, чтобы комбинировать элементы в математических выражениях. В GPSS при выполнении математических операций все значения переменных приводятся к численным, даже если они были заданы как литеральные.

A Setting is available in the Simulation Page of the Settings Notebook which switches the roles of the [*****] operator and the [**#**] operator.

Математические операции в GPSS World:

^*Выражение* — возведение в степень. Пример: **A^B** возвращает **A** в степени **B**.

(или *****) — умножение. Пример: **A#B** (или **A*B**) возвращает произведение **A** на **B**. По умолчанию символ **#** является знаком умножения, а символ ***** (в первой колонке программы) — индикатором строки примечаний. Но можно поменять значения этих символов на обратные через меню **Edit** → **Settings...** → **Switch # and ***. Для перехода в режим, когда умножение задаётся звездочкой *****, а комментарии решёточкой **#**, необходимо поставить галочку в окошке рядом с указателем **Switch # and ***.

/ — деление. Пример: **A/B** возвращает частное деления **A** на **B**.

**** — целочисленное деление. Пример: **A\B** возвращает целую часть результата деления **A** на **B**.

@ — целый остаток. **A@B** возвращает целвый остаток от деления **A** на **B**.

- — вычитание. Пример: **A-B** возвращает результат вычитания **B** из **A**.

+ — сложение. Пример: **A+B** возвращает сумму **A** и **B**.

>= 'GE' — "больше или равно". Пример: **A>=B** возвращает 1, если условие выполнено, и 0 — если не выполнено. Используется, в частности, с оператором **TEST**.

<= 'LE' — "меньше или равно". Пример: **A<=B** возвращает 1, если условие выполнено, и 0 — если не выполнено. Используется, в частности, с оператором **TEST**.

> 'G' — "больше". Пример: **A>B** возвращает 1, если условие выполнено, и 0 — если не выполнено. Используется, в частности, с оператором **TEST**.

< 'L' — "меньше". Пример: **A<B** возвращает 1, если условие выполнено, и 0 — если не выполнено. Используется, в частности, с оператором **TEST**.

= 'E' — "равно". Пример: **A=B** возвращает 1, если условие выполнено, и 0 — если не выполнено. Используется, в частности, с оператором **TEST**.

!= 'NE' — "не равно". Пример: **A!=B** возвращает 1, если условие выполнено, и 0 — если не выполнено.

& 'AND' — логическое "И". Пример: **A&B** возвращает 1, если оба операнда **A** и **B** не 0, и 0 — если хотя бы один из них 0.

| 'OR' — логическое "ИЛИ". Пример: **A'OR'B** возвращает 1, если хотя бы один из операндов **A** или **B** не 0, 0 — если оба они 0.

Приоритет математических операций в выражении:

^ возведение в степень;

(or *****) **/ ** умножение, деление, целочисленное деление;

@ целый остаток

- + вычитание и сложение;

>= <= > < операции сравнения;

= != равенство и неравенство;

& логическое "И";

| логическое "ИЛИ".

Приложение Б.
Системные числовые атрибуты в GPSS
(System Numerical Attributes)

Назначение Системных числовых атрибутов

Системные числовые атрибуты (СЧА), являются специфической формой подпрограмм-функций, которые предназначены для использования в процессе моделирования. При вызове они возвращают числовые или строковые значения указанных переменных и могут быть использованы в GPSS в качестве *операндов* (в операторах и командах) или в *выражениях* (например, в *Окнах* для просмотра результатов).

Их использование обусловлено тем, что в GPSS наряду с обычными переменными используются такие специфические объекты как *Обслуживающие устройства*, *Очереди*, *Многоканальные устройства*, *Цепи событий*. Они имеют по несколько характеризующих их атрибутов (параметров). Например, *Очередь* характеризуется числом вошедших в неё, находящихся в ней и вышедших из неё транзактов, некоторыми статистическими величинами и др. Эти параметры рассчитываются при моделировании и сохраняются внутри программы-интерпретатора. Чтобы получить доступ к ним, используются СЧА.

СЧА делятся на различные классы: СЧА *Очередей*, СЧА *Обслуживающих устройств*, СЧА общесистемные и др. Вид СЧА и порядок их использования показан ниже.

Системные Числовые Атрибуты — спецификаторы устройств (SNA Entity Specifiers)

Большинство СЧА могут быть представлены в одной из нескольких форм, начинающихся с наименования СЧА-класса. Например, выражение **W22** при вызове возвращает номер транзакта, находящегося в блоке № 22 (все блоки имеют наименование, например, очереди с различными именами, или номера). Этот номер присваивается транслятором при трансляции программы по порядку сверху вниз и виден в *Стандартном отчёте*. СЧА-класс задаётся первой буквой — в данном случае **W**.

СЧА могут быть записаны в следующих формах (для примера используется класс **W**):

W*j* — где j (положительное целое число) — номер блока в моделировании.

W$*Name* — где *Name* — название (обозначение) запрошенного блока, символ доллара **$** — знак присоединения к имени блока.

W*** *j* — где j (положительное целое число) — номер параметра активного транзакта, содержащего номер запрашиваемого блока (косвенная адресация); символ ***** — знак присоединения номера блока при косвенной адресации.

W* *Name* — где *Name* — название параметра активного транзакта, содержащего номер запрашиваемого блока (косвенная адресация); символ ***** — знак присоединения номера блока при косвенной адресации.

W*$*Name* — где *Name* — название параметра активного транзакта, содержащего номер запрашиваемого блока. Знак **$** не является необходимым, но используется как разделитель. По сути это выражение эквивалентно выражению **W*** *Name* (косвенная адресация).

W* *Parameter* — означает, что тот или иной вид записи **W*** *j*, **W*** *Name*, или **W*$***Name* может быть использован.

СЧА-класс **MX** для *Матриц* является особым случаем. Он содержит 3 косвенных адреса. Например, выражение

```
MX*Sales(*Partnumber,*January)
```

определяет *Матрицу*, чей номер находится в параметре с названием **Sales** (активного транзакта) и обращается к элементу *Матрицы*, имеющему номер строки **Partnumber** и номер столбца **January**. Обычно для инициализации параметров (в данном случае **Sales**, ***Partnumber** и ***January**) у активного в данный момент транзакта используется блок **ASSIGN**.

СЧА-классы **A1**, **AC1**, **C1**, **M1**, **MP**, **PR** и **TG1** отражают общесистемные переменные, не имеют параметров и используются сами по себе.

При программировании в GPSSW можно использовать подсказку о возможных формах использования СЧА в качестве операндов.

Доступные системные числовые атрибуты

Следующие СЧА могут быть использованы в качестве операндов в операторах (командах) и в выражениях. Во всех случаях *Entnum* заменяется спецификатором устройства, который может быть именем (располагается справа вплотную от разделителя **$**) или номером, или (для косвенной адресации) он может быть заменён на символ *****, за которым следует имя или номер.

Ниже приведён полный список СЧА в GPSS World:

A1 — Номер ансамбля активного транзакта (целое число). Ансамбль появляется, когда в системе находятся родственные транзакты, полученные, например, после создания копий одного из транзактов (родителя) оператором **SPLIT**.

AC1 — Значение абсолютного времени моделирования — начиная с использования последней команды **CLEAR** (действительное число), а при её отсутствии — с момента начала моделирования. Моделирование может прерываться (командами **HALT**, в связи с окончанием заданного времени и по другим причинам), а затем продолжаться. Абсолютное время показывает всё время моделирования.

BV*Entnum* — Значение булевой (логической) переменной с именем или номером *Entnum* (действительное число).

C1 — Значение относительного системного времени моделирования с момента последнего использования команды **RESET** (действительное число).

CA*Entnu* — Среднее число транзактов в *Цепи пользователя* с названием *Entnum* (действительное число).

CC*Entnum* — Общее число входов в *Цепь пользователя* с именем *Entnum* (целое число).

CH*Entnum* — Текущее значение числа транзактов в *Цепи пользователя* с именем *Entnum* (целое число).

CM*Entnum* — Максимальное число транзактов в *Цепи пользователя* с именем *Entnum* (целое число).

CT*Entnum* — Среднее время нахождения транзактов в *Цепи пользователя* с именем *Entnum* (действительное число).

F*Entnum* — Флаг занятости *Обслуживающего устройства*. Если *Обслуживающее устройство* с именем *Entnum* в настоящее время занято, **F***Entnum* возвращает 1, в противном случае возвращается 0 (целое число).

FC*Entnum* — Число захватов *Обслуживающего устройства* с именем *Entnum* с помощью команд **SEIZE** или **PREEMPT** (целое число).

FI*Entnum* — Флаг прерывания *Обслуживающего устройства*. Если *Обслуживающее устройство* с именем *Entnum* в настоящее время перехвачено (транзактом с более высоким приоритетом), **FI***Entnum* возвращает 1, иначе возвращается 0 (целое число).

FN*Entnum* — Функция. Возвращает результат выполнения функции с именем *Entnum* (действительное число).

FR*Entnum* — Коэффициент использования *Обслуживающего устройства*, равный отношению времени занятости транзактами *Обслуживающего устройства* с именем *Entnum*. к общему времени моделирования (действительное число).

FT*Entnum* — Среднее время удержания транзактом *Обслуживающего устройства* с именем *Entnum* (действительное число).

FV*Entnum* — Флаг готовности *Обслуживающего устройства*. Если *Обслуживающее устройство* с именем *Entnum* готово принять очередной транзакт, то **FV***Entnum* возвращает 1, если не готово — возвращается 0 (целое число).

GN*Entnum* — Счётчик *Цифровой группы*. **GN***Entnum* возвращает количество членов *Цифровой группы* с именем *Entnum* (целое число).

GT*Entnum* — Счётчик *Группы транзактов*. **GT***Entnum* возвращает количество членов *Группы транзактов* с именем *Entnum* (целое число).

LS*Entnum* — Значение *Логического ключа* с именем *Entnum*. **LS***Entnum* возвращает 1 или 0 (целое число). *Логические ключи* — это переменные, которые используются в программе для запоминания какого-то события и выстраивания логики работы программы. Например, можно с помощью *Логиче-*

ского ключа фиксировать включенность или выключенность какого-то устройства. *Логических ключей* может быть много и поэтому они имеют свои уникальные имена.

MB*Entnum* — Проверка принадлежности транзакта в блоке с именем *Entnum* и активного транзакта одному *Ансамблю*. **MB***Entnum* возвращает 1, если оба транзакта принадлежат одному ансамблю (целое число). *Ансамбли* нужны, например, при моделировании сборки сложного устройства из совокупности деталей. Может моделироваться сборка нескольких устройств и тогда будут существовать несколько *Ансамблей*. Чтобы не перепутать детали, их проверяют на предназначенность соответствующему устройству.

MP*Parameter* — Время перехода: текущее абсолютное системное время минус значение параметра с именем *Parameter* (действительное число).

MX*Entnum*(m,n) — Сохраняемое значение *Матрицы*. Возвращает значение элемента матрицы в строке *m* и столбце *n* матрицы *Entnum*.

M1 — Время перехода. **M1** возвращает абсолютное системное время минус "Временная Метка" транзакта (действительное число).

N*Entnum* — Счётчик входов в блок. Сообщает общее число входов транзактов в блок с именем *Entnum* (целое число).

P*Parameter* или ***Parameter* — Значение параметра с именем *Parameter* активного транзакта (целое или действительное числа, строка). При косвенной адресации используют форму SNA***Parameter*.

PR — Приоритет активного транзакта (целое число в диапазоне 0…127).

Q*Entnum* — Текущее число транзактов в *Очереди* с именем *Entnum* (целое число).

QA*Entnum* — Среднее число транзактов в *Очереди* с именем *Entnum* за время моделирования (действительное число).

QC*Entnum* — Общее число входов транзактов в *Очередь* с именем *Entnum* (целое число).

QM*Entnum* — Максимальное число транзактов в *Очереди* с именем *Entnum* за время моделирования (целое число).

QT*Entnum* — Среднее время нахождения транзактов в *Очереди* с именем *Entnum* (действительное число).

QX*Entnum* — Среднее время нахождения транзактов в *Очереди* с именем *Entnum*, исключая "нулевые входы", когда транзакт входит в *Очередь* и тут же её покидает, не теряя на это время (действительное число).

QZ*Entnum* — Число "нулевых входов" транзактов в *Очередь* с именем *Entnum*, когда транзакты проходят *Очередь* не задерживаясь (целое число).

R*Entnum* — Размер незанятой ёмкости *Многоканального Устройства* с именем *Entnum* (целое число).

RN*Entnum* — Случайное число. **RN***Entnum* возвращает числа в диапазоне 0…999 генератора случайных чисел с именем (номером) *Entnum* (целое число).

S*Entnum* — Количество транзактов, находящихся в данный момент в *Многоканальном устройстве* с именем *Entnum*. **S***Entnum* возвращает число транзактов, вошедших в *Многоканальное устройство* с именем *Entnum* (целое число).

SA*Entnum* — Среднее значение использованной ёмкости *Многоканального устройства* с именем *Entnum* (действительное число).

SC*Entnum* — Число использований транзактами *Многоканального устройства* с именем *Entnum* (целое число).

SE*Entnum* — Незанятость *Многоканального устройства*. **SE***Entnum* возвращает 1, если *Многоканальное устройство* с именем *Entnum* полностью доступен, возвращает 0 в ином случае (целое число).

SF*Entnum* — Заполненность *Многоканального устройства*. **SF***Entnum* возвращает 1, если *Многоканальное устройство* с именем *Entnum* полностью заполнено, возвращает 0 в ином случае (целое число).

SR*Entnum* — Коэффициент использования *Многоканального устройства*. Отношение среднего числа использований *Многоканального устройства Entnum* к общей его ёмкости (действительное число в диапазоне 0…1000).

SM*Entnum* — Максимальное значение использования *Многоканального устройства* с именем *Entnum* (целое число).

ST*Entnum* — Среднее время удержания на единицу *Многоканального устройства* с именем *Entnum* (действительное число).

SV*Entnum* — Готовность *Многоканального устройства*. **SV***Entnum* возвращает 1, если *Многоканальное устройство* с именем *Entnum* в состоянии готовности, возвращает 0 в ином случае (целое число).

TB*Entnum* — Среднее значение невзвешенных аргументов *Таблицы Entnum*. (действительное число).

TC*Entnum* — Число включений в *Таблицу Entnum*. (целое число).

TD*Entnum* — Среднеквадратическое отклонение для *Таблицы Entnum*. (действительное число).

TG1 — Число, записанное в *Счётчике завершеня*. **TG1** возвращает число, которое остаётся после работы блока **TERMINATE** с положительным операндом *A*. Это значение первоначально задаётся командой **START** и для продолжения моделирования должно быть больше 0 (целое число).

V*Entnum* — Результат вычисления арифметической или с плавающей точкой переменной с именем *Entnum* (действительное число).

W*Entnum* — Текущее (на данный момент) число транзактов в блоке с именем *Entnum* (целое число).

X*Entnum* — Значение *Сохраняемой величины* с именем *Entnum* (целое или действительное числа, строка).

XN1 — Номер активного транзакта (целое число).

Z1 — Размер свободной оперативной памяти компьютера (целое число).

Приложение В.
Термины и определения

блок (Block) — общее название пассивных объектов GPSS, а именно операторов и команд;

включённый файл (Include file) — файл с фрагментом программы на GPSS в формате имя.txt, который хранится отдельно и включается в программу специальной командой, а далее используется как неотъемлемая часть программного кода; позволяет создавать библиотеки фрагментов для многократного использования, визуально уменьшать листинг основной программы;

пассивный объект — оператор или команда GPSS; противопоставляется активным объектам — транзактам;

активный объект — транзакт; противопоставляется пассивным объектам — операторам и командам;

транзакт (Transaction) — виртуальный объект, передающий управление следующему блоку (по порядку следования блоков сверху вниз или в соответствии с логикой работы программы); физический смысл транзакта может быть различным — моменты времени, клиенты, телефонные звонки, прибывшие в аэропорт самолёты и т. д.

очередь (Queue) — очередь; предназначена для сбора статистики, связанной с ожиданием перед обслуживанием в том или ином устройстве;

устройство (Entity) — пассивный объект (*Обслуживающее устройство* ***Facility***, *Многоканальное устройство* ***Storage***, *Очередь* ***Queue***, *Логический ключ* ***Logicswitch***);

обслуживающее устройство (Facility) — одноканальное устройство, выполняющее какую-либо операцию с транзактом; вид операции не имеет значения, главным является фиксация прохождения транзакта через это устройство (сравнить с *многоканальным устройством*);

многоканальное устройство (Storage) — устройство с несколькими каналами обслуживания; применяется, когда параллельное использование нескольких одноканальных устройств оказывается слишком громоздким (например, при десятках, сотнях, тысячах каналов); пропускает через себя транзакты, если они требуют столько свободных каналов, сколько есть в наличии;

счётчик завершения (Termination Count) — счётчик, определяющий момент завершения сеанса моделирования; перед началом моделирования в него командой **`START`** записывают некоторое число, которое в процессе моделирования уменьшается с помощью операторов **`TERMINATE`** на заданное число единиц; при достижении значения 0 программа останавливается; работа счётчика завершений аналогична работе обычного оператора цикла в других языках программирования, но в данном случае этот цикл является внешним по отношению ко всей программе;

логический ключ (Logicswitch) — специальная переменная, которая имеет своё имя и может принимать только два значения: 1 или 0; она может служить для фиксации каких-то событий или использоваться в рамках логических выражений; вместо неё можно использовать обычные переменные ***Savevalue***, но *Логические ключи* занимают меньше памяти (когда-то это было весьма актуально);

окно моделирования (Simulation Window) — вспомогательный инструмент для визуализации процессов в программе; в окнах выводятся значения заданных в программе переменных, графики, таблицы (матрицы), анимация перемещения транзактов по блокам программы;

окно сохраняемых величин (Savevalues Window) — разновидность *Окна моделирования*, в котором показываются текущие значения всех *Сохраняемых величин* модели; испльзуется преимущественно для отладки модели или интерактивного моделирования;

снимок моделирования (Simulation Snapshot) — мгновенно полученная информация о текущем состояния *Цепей*, транзактов и их *Групп*; используется преимущественно для отладки модели;

дискретно-событийное моделирование — разновидность имитационного моделирования, при котором основными явлениями процесса являются события, которые происходят (начинаются и заканчиваются) в определённые дискретные моменты времени; программа перемещается от события к событию через дискретные интервалы времени, внутри которых состояние модели остаётся неизменным;

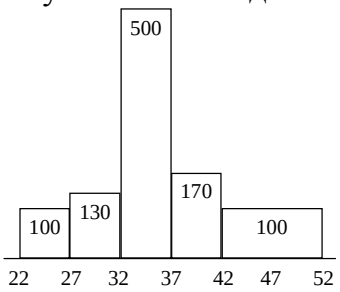
имитационное моделирование (Simulation) — моделирование с учётом вероятностного характера некоторых явлений и использованием метода Монте-Карло, когда в разных местах программной модели с помощью генераторов случайных чисел разыгрывается возможность появления того или иного события или значение того или иного параметра;

гистограмма (от греческих слов ἱστός — столбец и γράμμα — написание) — разновидность графического представления эмпирического распределения. Для построения гистограммы всё множество значений результатов наблюдений $X_1, \ldots, X_n$ делится на k интервалов группировки точками $x_0, \ldots, x_k$ (обычно интервалы выбирают равными), затем подсчитывается число m_i наблюдений, попавших в интервал $[x_{i-1}, x_i)$, и частота $h_i = m_i / n$. На оси абсцисс отмечаются точки $x_0, \ldots, x_k$ и строятся прямоугольники с основаниями, равными отрезкам $[x_{i-1}, x_i]$, $i = 1, \ldots, k$, и высотами, равными $h_i / (x_i - x_{i-1})$. В случае равных интервалов $[x_{i-1}, x_i)$ высоты прямоугольников иногда принимаются равными либо h_i, либо m_i. Пусть, например, измерение диаметров стволов елей дало следую-

щие результаты:

число стволов	100	130	500	170	100
диаметр, см	22 – 27	27 – 32	32 – 37	37 – 42	42 – 52

Гистограмма для этого случая имеет вид:



цепь транзактов (Transaction Chains) — список транзактов; каждый транзакт обязательно находится хотя бы в одном списке; в GPSS есть Цепь текущих событий (где записаны активные транзакты), Цепь будущих событий (где записаны транзакты, которые станут активными), Цепь пользователя (где записаны транзакты, выведенные из активного состояния пользователем) и др.; в списке, кроме номера транзакта, указываются некоторые его параметры, необходимые для выяснения перспектив его дальнейшего продвижения в программе;

сохраняемая величина (Savevalue) — аналог обычной переменной в других языках программирования; для *Сохраняемых величин* существует специальное *Окно сохраняемых величин*, в котором можно наблюдать изменение их значений в процессе моделирования без специальных дополнительных указаний;

поток транзактов () — последовательность транзактов, формируемая оператором **GENERATE**;

поток данных (Data Stream) — информация разного рода из файлов с диска или из оперативной памяти;

оператор (Operator) — один из блоков, выполняющий операцию над операндами; в GPSS оператор выполняет свои функции только если в него вошёл транзакт;

операнд (Operand) — параметры оператора или команды; записываются справа от них; в общем обозначаются буквами латинского алфавита *A, B, C* и т. д.; у каждого блока свой набор операндов; если какой-то операнд отсутствует, его пропускают, ставя запятую (например, если пропущен операнд *C*, то последовательность операндов записывают: *A,B,,D*);

команда (Command) — один из блоков, задающий поведение программы;

планировщик (Transaction Sheduler) — внешний по отношению к модели встроенный механизм, отслеживающий последовательность событий в соответствии с заданными и сформировавшимися условиями и выбирающий очередное активное событие, его время появления и соответствующий ему активный транзакт;

имя (Name) — буквенно-цифровое обозначение;

положительное целое (PosInteger) — положительное целое число, может служить номером строки, номером *Обслуживающего устройства*, *Очереди*, *Многоканального устройства*;

выражение в скобках (ParenthesizedExpression) — арифметическое выражение, заключаемое в скобки

СЧА (SNA) — (см. Системный числовой атрибут)

параметр (Parameter) — связанная с активным транзактом переменная, которая может иметь номер или любое имя; при ссылках на параметр с номером: P1, P23 (параметр № 1, параметры № 23); при ссылках на параметр с именем: P$Name, P$God_rojhdenija (имя, год рождения и т. п.);

функция (Function) — последовательность пар {аргумент,функция}, соответствующая табличному заданию функции

индикатор задержки [Delay Indicator] — флаг, устанавилваемый в состояние "включен" или "выключен" в зависимости от того,

объект моделирования — реальный объект (технический, организацонный, природный и т. д.), для которого создаётся модель

предмет моделирования — аспект объекта моделирования, который исследуется с помощью модели (процесс функционирования, энергетические характеристики, точностные характеристики, неисправности и т. д.)

Приложение Г.
Краткий англо-русский словарь терминов GPSS World

Существует значительное количество информации по GPSS World на английском языке. Сам язык GPSS World содержит много операторов, команд и понятий, которые следует правильно понимать и произносить. Поэтому ниже приводится краткий словарь базовых понятий GPSS World с переводом на русский язык и транскрипцией для правильного произношения. Ввиду того, что в таблицах символов MS Word не содержится всех знаков транскрипции, некоторые из них заменены на схожие и понятные без дополнительных пояснений.

General Purpose Systems Simulator ['dжənərəl 'pə:pəs 'sıstıms simju'leitə:] — Имитатор Систем Общего Назначения

CEC = Current Events Chain ['kʌrənt ı'vent ᵗʃeın] — *Цепь текущих событий* — содержит информацию о транзактах, помещённых в неё

FEC = Future Events Chain ['fju:ᵗʃə ı'vents ᵗʃeın] — *Цепь будущих событий* — содержит информацию о транзактах, помещённых в неё

XN — транзакт

acquisition [ækwı'zıʃn] — достижение, занятие

adopt [ədˈopt] — усыновить, признать потомком

advance [ədv'α:ns] — продвинуть

all [o:l] — все

alter ['o:ltə] — сменить

assemble [ə'sembl] — объединять

assign [ə'saın] — присвоить

attempt [ə'temt] — пробовать, предпринять попытку

block [blok] — блок

both [bouθ] — оба

buffer ['bafə] — буфер, место для промежуточного размещения

bvariable [bı-'vєəгıəbl] — двоичная переменная

close [klous] — закрыть (файл)

command [kə'mα:nd] — команда

count [kaunt] — счёт, считать

create [krı'eıt] — создать

customer ['kʌstəmə] — заказчик, клиент

delay [dı'leı] — задерживать, откладывать

depart [dı'pα:t] — покинуть (очередь)

depth [depθ] — глубина

develop [dı'veləp] — развить

digit ['dıdжıt] — цифра

displace [dıs'pleıs] — удалить

edit ['edıt] — редактировать

enter ['entə] — войти

entity ['entıtı] — элемент, объект, сущность

equ = equıvalent [ıkw'ıvələnt] — равный, эквивалентный, одинаковый

examine [ɪg'zæmɪn] — проверить, опросить
execute ['əksɪkju:t] — исполнить
expression [ɪks'preʃn] — выражение (математическое)
facility [fə'sɪlɪtɪ] — средство обработки (*Обслуживающее устройство*)
favail = f-avail***able*** [ef-ə'veɪleɪbl] — доступность *Обслуживающего устройства* (***Facility***)
file [faɪl] — файл
funavail = f-unavail***able*** [ef-'ʌnə'veɪleɪbl] — недоступность *Обслуживающего устройства* (***Facility***)
function ['fʌŋkʃən] — функция
fvariable = f-variable [ef-'vɛərɪəbl] — действительная переменная
gate [geɪt] — ворота
gather ['gæɪðə] — собирать
generate ['dженəreɪt] — создать, произвести
hold [hould] — содержать (в себе)
immediate [ɪ'mɪ:djət] — непосредственный, немедленного исполнения
index ['ɪndeks] — указатель
initial [ɪ'nɪʃəl] — начальный
integer ['ɪntɪdжə] — целое (число)
integration ['ɪntɪgreɪʃn] — объединить
join [dжoɪn] — соединить
journal [dжo:nl] — журнал (совокупность записей)
key [ki:] — ключ, клавиша
keyword [ki:wə:d] — зарезервированное (ключевое) слово
leave [li:v] — покинуть
line [laɪn] —линия
link [lɪŋk] — соединить, связать
logic ['lodжɪk] — логический
loop [lu:p] — петля
mark [mα:k] — пометка, отметка
match [mæ^{t}ʃ] — разбить, разложить, расщепить
mode [moud] — способ, режим
model [modl] — модель
msavevalue = m-save-value [mɪ-seɪv-'vælju] — матричная сохраняемая величина
new [nju:] — новый
open [oupn] — открыть (файл)
own [oun] — свой, собственный
owner ['ounə] — собственник, владелец
pick [pɪk] — выбирать
plus [plʌs] — плюс
poll [poul] — подсчёт (голосов)
preempt [pri:'emt] — занять по преимуществу

priority [praɪ'orɪtɪ] — приоритет, предпочтение, преимущество
qtable = q-table [kju-teibl] — таблица гистограмм для *Очереди* (***Queue***)
queue [kju:] — очередь (*Очередь*)
read [ri:d] — читать
ready [redɪ] — готовый, ужé
release [rɪ'li:s] — освободить *Обслуживающее устройство* (***Facility***)
remove [rɪ'mu:v] — удалить
report [rɪ'po:t] — отчёт
request [rɪ'kwest] — обращение, запрос
reset [rɪ'set] — переустановить
retranslate [rɪ:trɑ:ns'leɪt] — перетранслировать
retry ['ri:'traɪ] — пересматривать, рассматривать
return [ri:'to:n] — возврат
savail [es-ə'veɪleɪbl] — (s-available) доступность *Многоканального устройства* (***Storage***)
save [seɪv] — сохранить
savevalue = save-value [seɪv-'vælju] — сохраняемая величина
scan [skæn] — сканировать, просматривать
search [so:ᵗʃ] — искать
seed [sɪ:d] — установка для генератора случайных чисел
seek [sɪ:k] — искать, разыскивать
seize [sɪ:z] — занять, захватить
select [sɪ'lekt] — выбрать, избрать, отобрать
setting ['setɪŋ] — установка (параметра)
several ['sævrəl] — несколько
simulation [sɪmju:'leɪʃn] — моделирование (имитационное)
simultaneous [sɪməl'teɪnjəs] — одновременный
show [ʃou] — показать
split [splɪt] — расщепить, разделить на части
standard ['stændəd] — стандартный
start [stɑ:t] — старт, начать
stop [stop] — стоп, остановиться
storage ['sto:rədж] — хранилище, накопитель, *Многоканальное устройство*
sunavail = s-unavailable [es-ʌn' ə'veɪleɪbl] — недоступность Многоканального устройства (***Storage***)
schedule ['ʃedju:l] — расписание, планирование времени
table [teɪbl] — таблица (гистограмм)
tabulate ['tæbjuleɪt] — табулировать, записывать данные в таблицу
terminate ['tə:mɪneɪt] — завершить, уничтожить
test [test] — проверка, тестирование
tile [taɪl] — мозаика
time [taɪm] — время

trace [treɪs] — след, трасса, отслеживать, трассировать
transact [træn'zækt] — работа, активность
transaction [træn'zækʃn] — выполнение работы, проявление активности
transfer [træns'fə:] — переместить
translate [trɑ:ns'leɪt] — транслировать
unlink [ʌn'lɪnk] — отвязать, отсоединить
untrace [ʌn' treɪs] — не трассировать
variable ['vɛəɾɪəbl] — переменная
window ['wɪndou] — окно
write ['raɪt] — записать (файл)