Введение

Компьютерное моделирование нашло практическое применение во всех сферах деятельности человека, начиная от моделей технических, технологических и организационных систем и заканчивая проблемами развития человечества и вселенной. Еще с детства человек через игрушки и игры узнает мир и таким образом моделирует действительность. Вместо того, чтобы учиться на своих ошибках или на ошибках других людей, целесообразно закреплять и проверять познание реальной действительности полученными результатами на компьютерной модели. В этом случае есть возможность «проигрывать» на модели любые ситуации, включая те, при которых реальная система вышла бы из строя. Это позволяет моделировать катастрофы, редкие события и т. п. Одно из преимуществ компьютерного моделирования – это также моделирование того, что не существует на самом деле, то есть моделирование виртуальной реальности.

Когда же необходимо использовать компьютерное моделирование? Всегда, когда можно поставить вопрос, «что будет, если...?». Следовательно, компьютерное моделирование используют, прежде всего, для принятия решений. Модель позволяет проигрывать любые ситуации и получать наиболее эффективные решения проблемы.

Имитационное моделирование анализируемой или проектируемой системы состоит из следующих этапов:

- содержательная постановка задачи;
- разработка концептуальной модели;
- разработка и программная реализация имитационной модели;
- проверка правильности, достоверности модели и оценка точности результатов моделирования;
- планирование и проведение экспериментов;
- принятие решений.

Это позволяет использовать имитационное моделирование как универсальный подход для принятия решений в условиях неопределенности с учетом в моделях трудно формализуемых факторов, а также применять основные принципы системного подхода для решения практических задач.

Одним из первых языков моделирования, облегчающих процесс написания имитационных программ, был язык GPSS, созданный в виде конечного продукта Джеффри Гордоном в фирме IBM в 1962 г. В настоящее время есть трансляторы для операционных систем DOS – GPSS/PC, для OS/2 и DOS – GPSS/H и для Windows – GPSS World.

GPSS (General Purpose Simulation System – система моделирования общего назначения) – язык моделирования, который используется для построения событийных дискретных имитационных моделей и проведения экспериментов.

Система GPSS представляет собой язык и транслятор. Как и любой другой язык она содержит словарь и грамматику. Транслятор языка работает в две фазы. На первой фазе проверяется синтаксис и семантика написания строк GPSS-программы, а на второй (интерпретирующей) осуществляется продвижение транзактов по модели от блока к блоку.

Системы массового обслуживания и их характеристики

Системы массового обслуживания можно описать, если задать:

1) входящий поток требований или заявок, которые поступают на обслуживание;
2) дисциплину постановки в очередь и выбор из нее;
3) правило, по которому осуществляется обслуживание;
4) выходящий поток требований;
5) режимы работы.

Входящий поток. Для задания входящего потока требований необходимо описать моменты времени их поступления в систему (закон поступления) и количество требований, которое поступило одновременно. Закон поступления может быть детерминированный (например, одно требование поступает каждые 5 мин) или вероятностный (например, требования появляются с равной вероятностью в интервале 5±2 мин). В общем случае входящий поток требований описывается распределением вероятностей интервалов времени между соседними требованиями. Часто предполагают, что эти интервалы времени независимые и имеют одинаковое распределение случайных величин, которые образуют стационарный входящий поток требований. Классическая теория массового обслуживания рассматривает так называемый *пуассоновский* (простейший) поток тре-

бований. Для этого потока число требований k для любого интервала времени распределено по закону Пуассона:

$$P_k(t) = \frac{(\lambda t)^k}{k!} e^{-\lambda t},\ k \geq 0, t \geq 0.$$

где λ - интенсивность потока требований (число требований за единицу времени).

На практике обоснованием того, что входящий поток требований имеет распределение Пуассона, является то, что требования поступают от большого числа независимых источников за определенный интервал времени. Примерами могут быть вызовы абонентов в телефонной сети, запросы к распределенной базе данных от абонентов сети за некоторое время и другие.

Дисциплины постановки в очередь и выбора из неё могут быть следующих типов:

- FIFO – раньше поступил, раньше обслужился;
- LIFO – последний поступил, первый обслужился;
- случайный выбор из очереди;
- выбор из очереди по параметрам (например, мужчины в очереди пропускают женщин вперёд).

Также на очередь могут накладываться ограничения по длине очереди или по времени пребывания в ней.

Правила обслуживания характеризуются длительностью обслуживания (распределением времени обслуживание), количеством требований, которые обслуживаются одновременно, и дисциплиной обслуживания. Время обслуживания бывает детерминированным или заданным вероятностным законом распределения.

Обслуживание может организовываться с помощью одного устройства (это так называемые ***одноканальные системы***) или с несколькими идентичными устройствами обслуживания, например, если установлено несколько кабин с телефонами-автоматами. Системы с идентичными устройствами обслуживания называют ***многоканальными системами***. Устройства обслуживания могут быть объединены в последовательную цепочку (***многофазные системы обслуживания***). В качестве примера многофазной системы об-служивания можно рассмотреть сборочный конвейер.

Дисциплины обслуживания определяют:

- при каких условиях прекращается обслуживание требований;
- как выбирается для обслуживания следующее требование;
- что делать с частично обслуженным требованием.

Различают дисциплины обслуживания ***бесприоритетные*** и ***приоритетные***. При бесприоритетном обслуживании порядок обслуживания определяется дисциплиной выбора из очереди, например, FIFO.

При приоритетном обслуживании требованию задаётся некоторый параметр, который определяет его приоритет. Этот параметр может задаваться в числовом виде (***статический приоритет***) или в виде функции, которая зависит от времени пребывания в системе (***динамический приоритет***). Из требований с одинаковыми приоритетами могут организовываться очереди.

Дисциплины обслуживания могут быть с ***относительными*** или ***абсолютными*** приоритетами. Относительный приоритет предусматривает, что поступление требования с более высоким приоритетом не перерывает обслуживания менее приоритетного требования (*обслуживание без прерывания*).

При использовании абсолютного приоритета появление требования с более высоким приоритетом перерывает обслуживание менее приоритетного требования (*обслуживание с прерыванием*). В таких системах могут происходить вложенные прерывания, если требование, которое вытеснило из обслуживания менее приоритетное требование, само будет прервано более приоритетным требованием и т.д. Прерванные требования могут или оставлять систему обслуживания, или снова становиться в очередь для дообслуживания.

Выходящий поток – это поток требований, которые покидают систему, причем требования в нём могут быть как обслуженные, так и не обслуженные. Структура выходящего потока может иметь большее значение для многофазных систем, где этот поток становится входящим для следующей фазы обслуживания.

Режимы работы СМО подразумевают, что устройства обслуживания время от времени могут выходить из строя (*режим отказа*) либо прерывать или сильно замедлять процесс обслуживания (*режим блокирования обслуживания*).

Основы дискретно-событийного моделирования

Определим основные понятия и термины, используемые в моделировании.

Система – множество объектов (например, людей и машин), которые взаимодействуют одновременно для достижения одной или большего количества целей.

Модель – абстрактное представление системы, обычно содержит структурные, логические или математические отношения, которые описывают систему в терминах состояний, объектов и их свойств, множеств, процессов, событий, действий и задержек.

Состояние системы – множество переменных, которые содержат всю информацию, необходимую для описания свойств системы в любое время.

Объект – любой элемент или компонент в системе, который должен быть представлен в модели в явном виде (например, обслуживающее устройство, клиент, машина).

Свойство или атрибут – свойства данного объекта (например, приоритет ожидающего клиента, маршрут процесса выполнения работ в цеху).

Список – множество связанных объектов, упорядоченное некоторым логическим способом (например, все клиенты, находящиеся в настоящее время в очереди ожидания, упорядочены по принципу «раньше прибыл, раньше обслужился» или по приоритету).

Событие – мгновенно возникающее изменение состояние системы (например, прибытие нового требования).

Список событий – список намеченных будущих событий, упорядоченных по времени возникновения, известный также как ***список будущих событий*** (СБС).

Действие – продолжительность времени указанного промежутка (например, время обслуживания или время между поступлениями заявок), для которого известно, когда оно начинается и заканчивается (хотя оно может быть определено в терминах статистического распределения).

Задержка – продолжительность времени неопределенного промежутка, для которого неизвестно заранее, когда он заканчивается (например, задержка клиента в очереди по правилу «последний пришел – первый обслужился», так как начало обслуживания зависит от будущих поступлений).

Модельное время – неотрицательная возрастающая величина, отражающая течение времени в имитационной модели.

Часы – переменная, отражающая протекание времени моделирования.

Дискретно-событийное моделирование – моделирование системы в дискретные моменты времени, когда происходят события, отражающие последовательность изменения состояний системы во времени. В дальнейшем такое моделирование будем называть ***имитационным***.

Система моделирования GPSS

Язык GPSS – это язык декларативного типа, построенный по принципу объектно-ориентированного языка. Основными элементами этого языка являются транзакты и блоки, которые отображают соответственно динамические и статические объекты моделируемой системы. Стоит заметить, что содержательное значение транзактов определяет разработчик модели. Именно он устанавливает аналогию между транзактами и реальными динамическими элементами моделируемой системы, равно как и аналогию между блоками и статическими элементами реальной моделируемой системы.

С точки зрения программы, транзакт – это структура данных, которая содержит следующие поля:

- номер транзакта;
- время появления транзакта;
- номер блока, в котором находится транзакт;
- номер блока, куда он продвигается;
- момент времени начала продвижения;
- приоритет транзакта;
- параметры транзакта и др.

В языке GPSS все транзакты нумеруются по мере их появления в модели. Параметры транзактов отображают свойства моделируемого динамического объекта. Например, если моделируется движение автомобилей на участке дороги, то параметрами транзакта (автомобиля) в зависимости от целей моделирования могут быть скорость, тормозной путь, габариты и др.

В начале моделирования в GPSS-модели не существует ни одного транзакта.

Текст программы на языке GPSS представляет собой набор строк, в каждой из которых записывается 1 блок или 1 команда. В общем случае, любая непустая строка GPSS-программы выглядит следующим образом:

```
[Метка] Блок_или_команда [Операнды] [Комментарий]
```

Меткой может быть любой допустимый идентификатор, т.е. начинающаяся с буквы строка букв, цифр и символов '_', которая не совпадает ни с одним ключевым словом. Такие же идентификаторы могут быть именами различных сущностей языка GPSS: приборов, очередей, матриц и т.п. При этом стоит заметить, что различные сущности могут иметь одно и то же имя, например, `Q$Barber` и `F$Barber` ссылаются на разные сущности. Язык GPSS не чувствителен к регистру.

В процессе синтаксического разбора программы интерпретатор каждому идентификатору ставит в соответствие уникальное целое число, начиная с 10000. Но есть и исключение: если идентификатор является меткой блока модели, то этому идентификатору будет поставлен в соответствие порядковый номер блока в модели, который будет меньше 10000. Чтобы изменить назначенное по умолчанию число, можно воспользоваться командой `Equ`.

Операндами могут быть идентификаторы, целые и вещественные числа, строки, стандартные числовые атрибуты и выражения. В GPSS для хранения целых чисел выделяется 4 байта, для вещественных – 8. Если при вычислениях произойдёт переполнение, моделирование будет прервано.

Для построения выражений используются следующие операции:

Приоритет	Операция	Действие
8	^	возведение в степень
7	#	умножение
7	/	деление
7	\	целочисленное деление
6	@	остаток от деления
5	–	вычитание
5	+	сложение
4	>= или 'GE'	больше или равно
4	<= или 'LE'	меньше или равно
4	> или 'G'	больше
4	< или 'L'	меньше
3	= или 'E'	равно
3	!= или 'NE'	не равно
2	& или 'AND'	конъюнкция
1	| или 'OR'	дизъюнкция

В этой таблице 8 соответствует максимальному приоритету.

Выражение, встречающееся на месте любого операнда, должно быть взято в круглые скобки, за исключением нескольких команд:

```
(5+12@5 'e' 10-35\11)        ;
```
(значение этого выражения равно 1)

Комментарии в GPSS могут задаваться двумя способами:

- если строка начинается с символа **';'** или с **'*'**, то вся строка считается комментарием;
- необязательные поля комментария следуют за полями операндов любого блока или команды (кроме тех полей, где может быть произвольная комбинация операндов) и начинаются с символа ';'.

Основным элементом каждой строки GPSS-программы является блок или команда, от которых зависит смысл строки программы. Между командой и блоком есть принципиальное отличие, которое заключается в следующем. Когда интерпретатор производит разбор программы, он «вычленяет» все команды из текста программы в отдельный список. Т.о. в модели остаются только блоки. В случае успешной проверки синтаксиса программы интерпретатор начинает выполнение программы, заключающееся в последовательном исполнении команд из созданного списка. Большинство команд устанавливают различные свойства сущностей модели или параметры моделирования. В отличие от них команда `start` запускает непосредственно процесс моделирования, в течение которого в модели появляются транзакты, перемещающиеся по блокам модели до тех пор, пока не покинут модель. Моделирование завершается при выполнении условий завершения моделирования. После этого интерпретатор берёт и выполняет следующую команду из списка команд. И так до тех пор, пока он не дойдёт до конца списка команд.

Приведём пример. Пусть дана GPSS-программа:

```
kanal1  storage   3
        generate  2
        enter     kanal1
        advance   4
        leave     kanal1
        terminate 1

kanal2  storage   4
        generate  1.2,0.2
        enter     kanal2
summa   equ       7
        advance   5
        leave     kanal2
        terminate 1

        start     1000
        clear
kanal2  storage   5
summa   equ       8
        start     100
```

Поскольку в программе нет ошибок, то интерпретатор построит следующий список команд:

```
kanal1  storage   3
kanal2  storage   4
summa   equ       7
        start     1000
        clear
kanal2  storage   5
summa   equ       8
        start     100
```

А модель примет вид:

```
generate  2
enter     kanal1
advance   4
leave     kanal1
terminate 1

generate  1.2,0.2
enter     kanal2
advance   5
leave     kanal2
terminate 1
```

Когда начнётся выполнение программы интерпретатор сначала создаст многоканальное устройство с именем `kanal1` и 3 каналами обслуживания, затем многоканальное устройство с именем `kanal2` и 4 каналами обслуживания. Затем свяжет идентификатор `summa` со значением 7. И только при исполнении команды `start 1000` начнётся моделирование, которое будет заключаться в порождении двумя блоками `generate` транзактов, и продвижении этих транзактов по блокам `enter`, `advance`, `leave` и `terminate`, в последнем из которых транзакты будут покидать модель. Таким образом, прежде, чем моделирование завершится, будут промоделированы 1000 транзактов, полностью прошедших через модель.

Далее, когда моделирование завершилось (т.е. завершилось исполнение команды `start 1000`), интерпретатор начнёт исполнять следующую команду – `clear`, которая очистит результаты предыдущего моделирования. После неё будет выполнена команда `storage`, которая изменит ёмкость ранее созданного многоканального устройства `kanal2` до 5 каналов. Затем следующая команда `equ` изменит связанное с идентификатором `summa` значение на 8. После чего при выполнении команды `start 100` снова запустится процесс моделирования: снова будут порождаться транзакты, они вновь начнут продвигаться по модели, пока сотый транзакт не войдёт в один из блоков `terminate`. Только после этого завершится выполнение всей GPSS-программы.

Когда интерпретатор выполняет команду `start`, устанавливается начальное значение счётчика завершения равным операнду A команды `start`. Счётчик завершения – это внутренняя системная переменная, которая управляет прекращением моделирования: когда значение счётчика завершения становится меньше либо равным 0, моделирование останавливается. Изменять значение счётчика завершения можно с помощью блока `terminate`: при входе очередного транзакта в этот блок значение счётчика завершения уменьшается на значение операнда A блока `terminate`. Если операнд A опущен, то от счётчика завершения ничего не отнимается. Чтобы в ходе моделирования узнать текущее значение счётчика завершения, нужно воспользоваться стандартным числовым атрибутом `TG1`. О стандартных числовых атрибутах мы поговорим позднее.

Как уже говорилось, команда `start` начинает моделирование. Она имеет формат:

```
start A, B, C, D
```

Здесь операнд A – это начальное значение, которое будет присвоено счётчику завершения перед началом моделирования. Это обязательный операнд, т.е. он обязательно должен быть задан. Операнд B определяет, нужно ли после завершения моделирования создавать отчёт; это необязательный операнд. Если он задан равным `NP`, то отчёт не будет создан, иначе – будет. Операнд C в современной версии GPSS не используется. Если необязательный операнд D равен 1, то в стандартный отчёт о моделировании будет добавлена информация о транзактах, запланированных на вход в модель в будущем.

После того, как с помощью команды `start` было задано начальное значение счётчика завершения, интерпретатор просматривает модель и обрабатывает все блоки `generate`.

Блок generate порождает транзакты через определённые промежутки времени и осуществляет их вход в модель. Этот блок имеет формат:

```
generate A, B, C, D, E
```

Операнд A задаёт среднее время между входящими в модель транзактами, не является обязательным. Может быть любым положительным числовым значением.

Операнд B задаёт максимальное отклонение от среднего времени между входящими в модель транзактами. Не является обязательным и должен быть положительным числовым значением.

Операнд C задаёт момент времени, начиная с которого будут порождаться транзакты. Необязательный, положительный.

Операнд D задаёт максимальное количество транзактов, которые может породить данных блок generate. Необязательный, положительный, целочисленный.

Операнд E задаёт приоритет порождаемых транзактов. Необязательный, положительный, целочисленный.

Примеры:

```
generate 4,1        ; (транзакты поступают в модель в соответствии с равномерным распределением из интервала [3; 5])
generate 0.1        ; (транзакты поступают в модель каждые 0.1 единицы времени)
generate 4,1,15     ; (транзакты поступают в модель в соответствии с равномерным распределением из интервала [3; 5], начиная с момента времени 15)
generate 4,,,8      ; (транзакты поступают в модель каждые 4 единицы времени, но всего порождается только 8 транзактов, после чего транзакты перестают создаваться в этом блоке)
generate ,,,10,2    ; (10 транзактов порождается в момент времени 0 с приоритетом 2, после чего транзакты перестают создаваться в этом блоке)
```

Общий механизм работы блока generate такой. Сразу после начала моделирования интерпретатор просматривает модель и обрабатывает все блоки generate: в каждом таком блоке порождается по 1 транзакту. В соответствии с заданными операндами вычисляются моменты времени, когда эти транзакты должны будут войти в модель. Когда наступает один из таких моментов, соответствующий транзакт входит в модель, а блок generate «вместо» него планирует на вход модель новый транзакт, вычисляя момент его будущего входа в модель, исходя из значений операндов данного блока.

В модели может быть несколько блоков generate, как на ранее приведённом примере. Но при прохождении по модели никакой транзакт не должен пытаться войти в блок generate. Такая попытка приведёт к ошибке и останову моделирования. Поэтому нельзя располагать 2 блока generate подряд один за другим.

Рассмотрим способы завершения моделирования. Моделирование можно остановить, когда будет обработано заданное количество транзактов. Общий вид такой программы:

```
generate    10,2
; ...
; блоки, описывающие моделируемую систему
; ...
terminate   1
start       120
```

Данная программа завершится, когда будет обработано 120 транзактов.

Также моделирование можно остановить по истечении заданного времени. Для этого отдельными блоками generate и terminate моделируют работу таймера:

```
generate    10,2
; ...
; блоки, описывающие моделируемую систему
; ...
terminate

generate    1500      ; (здесь моделируется работа таймера)
terminate   1
start       1
```

Эта программа завершится в момент модельного времени 1500.

Ещё один способ останова моделирования – ожидание выполнения какого-либо условия. В следующем примере моделирование завершается, когда длина очереди `queueName` превысит 2 транзакта:

```
generate    10,2
; ...
; блоки, описывающие моделируемую систему
; ...
terminate

generate    ,,,1
test g      q$queueName,2      ; (здесь моделируется проверка условия)
terminate   1
start       1
```

После того, как транзакт вошёл в модель через блок `generate`, он последовательно перемещается по блокам модели до тех, пока не встретит блок terminate. В языке GPSS существуют блоки, которые изменяют обычную последовательность блоков, которые проходит транзакт. Это, например, блоки `transfer` и `test`. С их помощью можно организовывать ветвления и циклы.

Блок `test` имеет формат:

```
test O A, B, C
```

Здесь O – операция сравнения между собой операндов A и B. Является обязательной, может принимать значения: `'E'`, `'G'`, `'GE'`, `'L'`, `'LE'` или `'NE'`.

Операнды A и B – сравниваемые значения. Обязательные операнды, могут быть любыми положительными числовыми значениями.

Операнд C задаёт номер блока, в который помещается транзакт, если результатом операции сравнения является ложь.

Этот блок работает в режимах «отказа» (если операнд C указан) или «альтернативного перехода» (если операнд C не указан). В каждом из режимов происходит сравнение операндов A и B с использованием операции O. В случае успешного сравнения транзакт переходит в блок, следующий за `test`, иначе в режиме «альтернативного перехода» пытается войти в указанный операндом C блок или в режиме «отказа» – задерживается на входе в данный блок `test`. Приведём примеры:

```
test G Q$Line1,Q$Line2,Block1   ; (блок работает в режиме «альтернативного
                                  перехода»)
test GE C1,70000                ; (блок работает в режиме «отказа»)
```

Блок `transfer` имеет формат:

```
transfer A, B, C, D
```

Операнд A задаёт режим работы блока `transfer`. Необязательный операнд.

Необязательные операнды B, C и D имеют различное назначение в зависимости от режима.

Если операнд A опущен, то блок работает в «безусловном режиме», при котором транзакт всегда переходит в блок, указанный в операнде B. Например:

```
transfer , New_Place
```

Если операнд A не является ключевым словом, блок работает в «дробном режиме», при котором транзакт переходит в блок, указанный операндом C, с вероятностью, заданной операндом A, и с вероятностью 1–A в блок, указанный операндом B (или в следующий за `transfer` блок, если B опущен). Вероятность может быть задана вещественным числом от 0 до 1 или целым числом от 0 до 1000. В последнем случае вероятность определяется как отношение $P = \frac{операнд\ A}{1000}$. Следующие примеры эквивалентны:

```
transfer 0.75,,New_Place
transfer 750,,New_Place
```

Если операнд A равен `both`, то блок работает в режиме «both». В этом режиме транзакт пытается войти в блок, заданный операндом B. Если транзакту было отказано во входе, то он пытается войти в блок, заданный операндом C. Если транзакт смог войти в один из этих блоков, то он покидает блок `transfer`, перемещаясь в соответствующий блок. Если транзакт не смог войти ни в один из этих блоков, то он задерживается на входе в данный блок `transfer`. Пример:

```
transfer both,First,Second
```

Если операнд A равен `all`, то блок работает в режиме «all». В этом режиме транзакт последовательно пытается войти во все блоки, начиная от блока, заданного операндом B, до блока, заданного операндом C, с шагом, равным значению операнда D. В результате транзакт либо войдёт в один из тестируемых блоков, либо будет задержан на входе в данный блок `transfer`. Пример:

```
transfer all,First,Last,2
```

Если операнд A равен `pick`, то блок работает в режиме «pick». В этом режиме в последовательности блоков, начинающейся с блока, заданного операндом B, и заканчивающейся блоком, заданным операндом C, транзакт случайным образом выбирает блок, на вход которому он планируется. Пример:

```
      transfer  pick,bl1,bl3
bl1   terminate 1
bl2   terminate 1
bl3   terminate 1
```

Если операнд A равен `'fn'`, то блок работает в режиме «function». В этом режиме номер блока, в который транзакт будет пытаться войти, определяется как сумма значения функции, заданной операндом B, и значения необязательного операнда C. Аналогично блок `transfer` работает в режиме «parameter», когда операнд A равен `'p'`, а операнд B задаёт имя параметра транзакта, используемого для вычисления суммы. Примеры:

```
transfer fn,Func1,5
transfer p,Placemarker,1
```

Если операнд A равен `'sbr'`, то блок работает в режиме «subroutine». В этом режиме транзакт всегда переходит в блок, заданный операндом B, при этом в параметре транзакта, заданном операндом C, сохраняется месторасположение данного блока `transfer`. Т.е. имитируется вызов подпрограммы.

```
transfer sbr,New_Place,Placemarker
```

Чтобы транзакту вернуться из «подпрограммы», можно использовать блок `transfer` в режиме «parameter»:

```
transfer p,Placemarker,1          ;  (см. выше)
```

Существует ещё один режим работы блока `transfer` – «Simultaneous», но он в настоящее время редко используется.

При моделировании перемещения транзактов по блокам модели в языке GPSS используется понятие модельного времени. Следует отличать физическое время моделирования, т.е. время, в течение которого интерпретатор GPSS моделирует GPSS-программу, и модельное время, т.е. время, которое существует в моделируемой системе и воспроизводится в модели. Различают также *абсолютное модельное время* – это время, прошедшее с начала моделирования (т.е. с момента запуска на выполнения первой команды `start` из модели), и *относительное модельное время* – время, прошедшее с момент выполнения последней команды `reset`.

Приведём пример небольшой GPSS-программы и опишем, как она работает:

```
      generate  3
      seize     pribor
      advance   0.5
      release   pribor
end   terminate 1

      generate  2
      advance   3
      transfer  ,end

      start     3
```

После синтаксического разбора, интерпретатор GPSS начинает выполнять указанные в тексте команды. В данном случае первая выполняемая команда – это команда `start`. Она устанавливает значение счётчика завершения равным 3 и начинает моделирование. Сначала интерпретатор просматривает все блоки `generate` и планирует на вход в модель по одному транзакту из каждого такого блока. В нашем случае планируется 2 транзакта: на момент времени 3 из первого блока `generate` и на момент времени 2 из второго. Поскольку в начале моделирования в модели ещё нет ни одного транзакта, то часы модельного времени становятся равными 2, т.е. наиболее раннему моменту входа очередного транзакта в модель. При этом транзакт входит в модель через второй блок `generate`, сразу после чего, используя параметры блока, интерпретатор планирует следующий транзакт, порождаемый данным блоком, на вход в модель в момент времени 2+2. Т.к. в блоке `generate` задержка никогда не происходит, то вошедший в модель транзакт пытается войти в следующий блок, т.е. в `advance`. Войдя в этот блок, транзакт задерживается в нём на 3 единицы времени, поэтому следующий раз он попытается продвинуться по модели в момент времени 2+3.

Т.к. теперь в момент времени 2 не осталось транзактов, которые могут продвинуться по модели, то интерпретатор выбирает следующий активный транзакт – в данном примере тот, который должен войти в модель через первый блок `generate` в момент времени 3, – и делает часы модельного времени равными 3. Когда в момент времени 3 этот транзакт входит в модель, в первом блоке `generate` «вместо него» порождается следующий транзакт, планируемый на вход в модель в момент времени 3+3. Вошедший же в модель транзакт пытается продвинуться по модели как можно дальше: входит в блок `seize`, занимая `pribor`, затем в `advance`, после чего планируется на момент времени 3+0.5.

И вновь интерпретатор выбирает из списка будущих событий тот транзакт, который будет пытаться продвинуться по модели в наиболее ранний момент времени. В данном случае будет выбран транзакт, который только что вошёл в блок `advance 0.5`, а часы модельного времени будут установлены в 3.5. Далее этот транзакт войдёт в следующий блок, освободив `pribor`, а затем покинет модель через блок `terminate`, уменьшив счётчик завершения на 1.

И т.д.

Практически все сущности языка GPSS обладают свойствами, характеризующими состояние конкретного экземпляра сущности. Для доступа к их значениям в языке GPSS используются *системные (или стандартные) числовые атрибуты*. Каждая сущность языка имеет свой набор системных числовых атрибутов. СЧА задаются следующим образом:

`[класс][сущность]`

где `[класс]` – это вид СЧА, определяющий значение какого свойства той или иной сущности языка GPSS требуется вернуть в качестве результата вычисления СЧА;

`[сущность]` – указание на конкретный экземпляр сущности, для которого в соответствии с классом СЧА будут производиться вычисления значения СЧА.

`[сущность]` может принимать следующие значения:

j, где *j* – целое число, задающее номер сущности;

$Name, где *Name* – имя сущности;

j, где *j* – целое число, задающее номер параметра активного транзакта, значение которого задаёт номер сущности;

Name или *$Name*, где *Name* – имя параметра, значение которого задаёт номер сущности.

Среди СЧА есть исключения из указанного формата их задания. Это так называемые *атомарные*, или *элементарные*, СЧА. К ним относятся a1, ac1, c1, m1, pr, tg1, xn1. Они не требуют указания на конкретный экземпляр сущности. Другим исключением является СЧА класса mx. Оно имеет формат:

```
mx[сущность] (m, n)
```

где m – номер строки матрицы, n – номер столбца.

Ещё одним исключением является СЧА класса p. При обращении к нему можно опускать название класса СЧА. Например:

p1 эквивалентно *1, или p$namepar эквивалентно *namepar.

Всего в языке GPSS более 50 классов СЧА. Перечислим некоторые СЧА:

ac1 – возвращает значение абсолютного системного времени, т.е. время моделирования после последнего блока clear;

c1 – возвращает значение относительного системного времени, т.е. время моделирования после последнего блока reset;

m1 – возвращает разность между абсолютным системным временем и “Mark Time” активного транзакта, которое может быть установлено с помощью блока mark;

mp– возвращает разность между абсолютным системным временем и значением заданного параметра активного транзакта;

n – возвращает полное количество входов транзактов в заданный блок;

pr – возвращает значение приоритета активного транзакта;

tg1 – возвращает текущее значение счётчика завершения;

w – возвращает номер транзакта в заданном блоке;

xn1 – возвращает номер активного транзакта.

Примеры:

```
savevalue aaa,mp$par1
test le   n*par2,*5,p10
```

Опишем, как выполняется последний пример: если число входов транзактов в блок, номер которого равен значению, находящемуся в параметре транзакта par2, меньше либо равен значению, лежащему в параметре номер 5, то вошедший в данный блок транзакт переходит в следующий за test блок, иначе в блок, номер которого лежит в параметре номер 10.

Моделирование равномерно распределённых псевдослучайных чисел в языке GPSS реализовывается с помощью встроенных генераторов псевдослучайных чисел. Количество генераторов в языке не ограничено, но управлять их начальным значением можно только для генераторов с номерами от 1 до 7. Для остальных генераторов начальное значение всегда равно номеру генератора. Неявно генераторы используются в блоках generate, advance, transfer. Чтобы получить очередное псевдослучайное значение из генератора, существует СЧА rn:

```
rn[номер_генератора]
```

Например:

```
rn*par3
```

Это СЧА возвращает целое псевдослучайное число из диапазона от 0 до 999 из генератора, номер которого лежит в параметре транзакта par3. Для изменения начального значения первых семи генераторов существует команда rmult, имеющая следующий формат:

```
rmult A,B,C,D,E,F,G
```

где A, B, C, D, E, F, G – начальные значения для 1–7 генераторов, присутствие каждого из которых в списке операндов необязательно.

Для моделирования неравномерных псевдослучайных чисел можно либо вручную преобразовать равномерно распределённую величину, либо переложить эту работу на интерпрета-

тор, воспользовавшись встроенными функциями, такими как poisson, normal, exponential. Все эти функции имеют общую форму записи:

```
имя_функции(номер_генератора,параметры_распределения)
```

где параметры_распределения – это список разделённых через запятую параметров, таких как математическое ожидание, среднеквадратическое отклонение и т.п. Например,

```
exponential(1,10,2)
```

Здесь 10 – это параметр сдвига, а 2 – это параметр масштаба.

Функции в языке GPSS задаются с помощью команды function. Она имеет следующий формат:

```
name function  A, B
x1,y1/x2,y2/…/xn,yn
```

где name – имя функции, A – аргумент функции, B – описание типа функции и количества точек, по которым строится функция, xi,yi – пары точек и значений функции, с помощью которых задаётся график функции. Операнд B имеет формат:

```
[тип_функции][число_точек]
```

Если тип_функции = 'C', то создаётся непрерывная функция, график которой проходит через все перечисленные точки, соединяя их прямыми, т.е. осуществляется линейная интерполяция.

Если тип_функции = 'D', то создаётся дискретная функция, график которой представляет собой кусочно-постоянную функцию.

Если тип_функции = 'E', то создаётся дискретная функция, график которой представляет собой кусочно-постоянную функцию. Отличается о функции типа 'D' тем, что yi могут быть СЧА или выражениями.

Если тип_функции = 'L', то создаётся функция, определённая только в точках 1, 2, 3,… и возвращающая для таких аргументов (xi) соответствующее значение (yi), т.е.. это функция-список.

Если тип_функции = 'M', то создаётся функция, определённая только в точках 1, 2, 3,… и возвращающая для таких аргументов (xi) соответствующее значение (yi), в качестве которого может быть СЧА или выражения.