

В.М. Черненко́ый, М.В. Черненко́ый

## **ПРОЦЕССНО-АГРЕГАТИВНЫЕ СИСТЕМЫ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ**

Рекомендовано федеральным учебно-методическим объединением в системе высшего образования по укрупненной группе специальностей и направлений подготовки 09.00.00 «Информатика и вычислительная техника» в качестве учебного пособия для студентов, обучающихся по основным образовательным программам высшего образования по направлению подготовки бакалавров 09.03.01 «Информатика и вычислительная техника»

Москва  
2018

УДК 004.94  
ББК

Рецензенты:

Карпов Валерий Иванович д.т.н., профессор,  
Девятков Владимир Васильевич д.т.н., профессор

**Черненко В.М, Черненко М.В.**

Процессно-агрегативные системы имитационного моделирования. – М.: 2018. 160 с. ил.

Учебное пособие рассматривается, как методический материал, предназначенный для поддержки ряда учебных курсов цикла “Моделирование” в рамках подготовки бакалавров по направлению 09.03.01. “Информатика и вычислительная техника”. В предлагаемом учебном пособии упор делается на изучение и создание средств описания процессов функционирования систем, отображения их на квазипараллельный процесс и создание программных имитационных моделей. Пособие будет полезным для специалистов, работающих в области проектирования информационных систем, как содержащее компактное изложение методики создания имитационных программ с использованием алгоритмической модели описания процессов функционирования сложных систем.

ISBN

## СОДЕРЖАНИЕ

<b>ПРЕДИСЛОВИЕ</b> .....	<b>6</b>
<b>ВВЕДЕНИЕ</b> .....	<b>8</b>
<b>1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ АЛГОРИТМИЧЕСКОГО МЕТОДА ОПИСАНИЯ ПАРАЛЛЕЛЬНО-ПОСЛЕДОВАТЕЛЬНОЙ СОВОКУПНОСТИ ВЗАИМОДЕЙСТВУЮЩИХ ПРОЦЕССОВ</b> .....	<b>12</b>
1.1. ОПРЕДЕЛЕНИЕ ПРОЦЕССА.....	12
1.2. ОБЪЕКТ, ХАРАКТЕРИСТИКИ ОБЪЕКТА .....	15
1.3. АЛГОРИТМИЧЕСКАЯ МОДЕЛЬ ПРОЦЕССА (АМП) .....	17
1.4. СТРУКТУРА ТРЕКА .....	19
1.5. ПОДОБНЫЕ ПРОЦЕССЫ .....	21
1.6. РЕСУРСЫ, КОНФЛИКТЫ НА РЕСУРСАХ .....	23
1.7. БЛОКИ, ТИПЫ БЛОКОВ.....	24
1.8. СХЕМЫ ОПИСАНИЙ ФУНКЦИОНИРОВАНИЯ СИСТЕМ .....	26
1.9. ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ .....	29
<b>ГЛАВА 2. ПСЕВДОЯЗЫК ОПИСАНИЯ СЦЕПЛЕННЫХ ПРОЦЕССОВ</b> .....	<b>31</b>
2.1. ВВЕДЕНИЕ .....	31
2.2. ОСНОВНЫЕ ЭЛЕМЕНТЫ ЯЗЫКА .....	31
2.3. ОПИСАНИЕ ТИПОВ.....	33
2.4. ОПЕРАТОРЫ ПОСП .....	35
2.5. ПРОГРАММА БЛОКА ГЕНЕРАТОР .....	37
2.6. ОПИСАНИЕ БЛОЧНЫХ СТРУКТУР.....	38
2.7. ПРИМЕРЫ ПРОГРАММ БЛОКОВ.....	42
2.8. МАКРОСЫ И МАКРОРАСШИРЕНИЯ.....	46
2.9. МОДЕЛЬ ВЫЧИСЛИТЕЛЬНОГО КОМПЛЕКСА АСУТП.....	47
2.10. ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ ПРОРАБОТКИ .....	51

<b>ГЛАВА 3. ПОСТРОЕНИЕ ИМИТАЦИОННОГО ПРОЦЕССА.....</b>	<b>54</b>
3.1. ПОНЯТИЕ КВАЗИПАРАЛЛЕЛЬНОГО ПРОЦЕССА.....	54
3.2. АНАЛИЗ ОТНОШЕНИЯ СЦЕПЛЕННОСТИ ОПЕРАТОРОВ.....	55
3.3. КЛАССЫ ОДНОВРЕМЕННЫХ СОБЫТИЙ.....	59
3.4. МОДЕЛИРУЮЩИЙ АЛГОРИТМ СКАНИРУЮЩЕГО ТИПА .....	63
3.5. ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ .....	66
<b>ГЛАВА 4. СИСТЕМА ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ GPSS .....</b>	<b>67</b>
4.1. ОСНОВНЫЕ ПРИНЦИПЫ .....	67
4.2. ОБЩЕЕ ОПИСАНИЕ.....	68
4.3. СТАНДАРТНЫЕ ЧИСЛОВЫЕ И ЛОГИЧЕСКИЕ АТТРИБУТЫ.....	70
4.4. АРИФМЕТИЧЕСКИЕ И ЛОГИЧЕСКИЕ ВЫРАЖЕНИЯ.....	71
4.5. ТРАНЗАКТЫ .....	72
4.6. БЛОКИ ГЕНЕРАЦИИ И ЗАДЕРЖКИ ТРАНЗАКТОВ .....	73
4.7. РЕСУРСЫ.....	76
4.8. ЛОГИЧЕСКИЕ КЛЮЧИ.....	82
4.9. БЛОКИ И ОПЕРАТОРЫ ОРГАНИЗАЦИИ ВЫЧИСЛЕНИЙ.....	83
4.10. БЛОКИ УПРАВЛЕНИЯ ДВИЖЕНИЕМ ТРАНЗАКТОВ .....	88
4.11. БЛОКИ И ОПЕРАТОРЫ СБОРА СТАТИСТИКИ .....	91
4.12. БЛОКИ РАБОТЫ С СЕМЕЙСТВОМ ТРАНЗАКТОВ.....	94
4.13. ОСТАНОВ ПРОЦЕССА МОДЕЛИРОВАНИЯ.....	95
4.14. СВОДНЫЙ СПИСОК СЧА ОБЪЕКТОВ .....	96
4.15. ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНЫХ РАБОТ ПО СОЗДАНИЮ ИМИТАЦИОННЫХ МОДЕЛЕЙ .....	99
<b>ГЛАВА 5. СРЕДА МОДЕЛИРОВАНИЯ SIMIO .....</b>	<b>101</b>
5.1. ОБЪЕКТЫ.....	101
5.2. ПРОЦЕССЫ.....	106
5.3. ОЧЕРЕДИ В SIMIO.....	109
5.4. ТРАНЗАКТЫ И ТОКЕНЫ .....	111
5.5. СТАНДАРТНАЯ БИБЛИОТЕКА ОБЪЕКТОВ .....	115
5.6. ПРИМЕР МОДЕЛИ.....	120
5.7. ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ .....	126

<b>ПРИЛОЖЕНИЕ 1. ЛАБОРАТОРНЫЙ ПРАКТИКУМ ПО МОДЕЛИРОВАНИЮ НА GPSS.....</b>	<b>127</b>
<b>ПРИЛОЖЕНИЕ 2. ОПИСАНИЕ ОПЕРАТОРОВ И БЛОКОВ GPSS.....</b>	<b>145</b>
<b>СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ .....</b>	<b>158</b>

## ПРЕДИСЛОВИЕ

Учебное пособие рассматривается, как методический материал, предназначенный для поддержки ряда учебных курсов цикла "Моделирование" в рамках подготовки бакалавров по направлению 09.03.01. Материал книги ориентирован на изложение аспектов моделирования, связанных с проблемами создания программных имитационных моделей. Надо отметить, что в отличие от большинства технических моделей, которые, как правило, классифицированы в соответствии с научными приложениями, имитационное моделирование применимо практически в любой отрасли науки.

В предлагаемом пособии упор делается на изучение и создание средств и подходов к описанию моделируемых процессов, в результате использования которых создается собственно имитационная модель. В пособии использованы материалы учебных дисциплин, читаемых на кафедре "Системы обработки информации и управления" (ИУ-5) МГТУ им. Н.Э. Баумана.

Пособие содержит учебный материал, либо указания на него, который должен знать студент в процессе обучения по бакалаврской программе. В каждом разделе пособия содержатся примеры использования рассматриваемых методов или подходов, а также предлагаются задачи для самостоятельной проработки. Кроме этого, в приложении приведены описания лабораторных работ по конструированию имитационных программ. Методическая схема выполнения лабораторных работ предполагает первоначальное изучение студентом готовых программ, размещенных в библиотеке моделей. Затем в заключительной части лабораторного занятия студенту предлагается задание для самостоятельной работы по составлению имитационной программы.

Таким образом, в пособии изложена методика, которая позволяет наряду со знаниями обучать студента умению применять предлагаемые методы, а также прививать ему навыки решения конкретных задач проектирования.

Для освоения учебного материала необходимы знания в рамках учебных дисциплин "Программирование", "Математическая статистика", "ЭВМ", "Вычислительные сети", "Базы данных", "Операционные системы".

Учебное пособие включает разделы:

- теоретические основы алгоритмического метода описания параллельно-последовательной совокупности взаимодействующих процессов
- псевдоязык описания сцепленных процессов

- построение имитационного процесса
- система имитационного моделирования GPSS
- среда моделирования SIMIO

Указанные разделы пособия могут использоваться, как основа создания одной или нескольких учебных дисциплин в зависимости от условий реализации конкретного учебного плана.

Пособие соответствует требованиям Федерального государственного образовательного стандарта уровня бакалавров по направлению подготовки 09.03.01 "Информатика и вычислительная техника".

Виды профессиональной деятельности, к которым готовит выпускников цикл учебных разделов, изложенных в пособии:

- научно-исследовательская;
- проектно-конструкторская;
- проектно-технологическая.

Студент, освоивший знания, изложенные в пособии, должен быть готов решать следующие профессиональные задачи:

- проектирование программных и аппаратных средств (систем, устройств, деталей, программ, баз данных) в соответствии с техническим заданием с использованием средств автоматизации проектирования (проектно-конструкторская деятельность),
- применение современных инструментальных средств при разработке программного обеспечения (проектно-технологическая деятельность),
- математическое моделирование процессов и объектов на базе стандартных пакетов автоматизированного проектирования и исследований (научно-исследовательская деятельность).

Пособие ориентировано на студентов, обучающихся по программе бакалавриата по направлению подготовки 09.03.01 "Информатика и вычислительная техника", преподавателей дисциплин, ведущих занятия по указанной бакалаврской программе. Пособие будет полезным для специалистов, работающих в области проектирования информационных систем, как содержащее компактное изложение методики создания имитационных программ с использованием алгоритмической модели описания процессов функционирования сложных систем.

## ВВЕДЕНИЕ

Общеизвестно, что одним из главных элементов, необходимых для эффективного решения сложных задач, является создание и использование моделей. В технических приложениях, как правило, используются математические модели. Конечно, проектировщику всегда желательно иметь в своем распоряжении аналитическую модель, поскольку она строится для конкретных приложений с учетом особенностей решаемой задачи, имеет явно выраженную функциональную зависимость искомых характеристики от входных параметров. Однако, далеко не всегда возможно создание аналитических моделей. В этом случае используются методы имитационного моделирования. Построение имитационной модели основывается на изучении элементов системы, описании их функционирования и выявлении разнообразных связей и отношений между элементами. Здесь используется тот факт, что описание функционирования отдельного элемента значительно проще описания функционирования всей системы в целом.

В результате, имитационное моделирование есть процесс конструирования модели реальной системы и постановки экспериментов на этой модели с целью либо понять поведение системы, либо оценить различные стратегии, обеспечивающие функционирование данной системы. Как отмечено в [8], в отличие от большинства технических моделей, которые могут быть классифицированы в соответствии с научными дисциплинами, в которые они уходят своими корнями (например, с физикой или химией), имитационное моделирование применимо в любой отрасли науки. Его применяют в технике, в системах обработки информации, коммерческой деятельности, экономике, на транспорте, в исследовании проблем городов и глобальных систем, а также во многих других областях.

Имитационная модель выполняет еще одну важнейшую функцию: она выступает как средство общения. Естественные языки, в основе которых лежит слово, в той или иной мере оказываются неточными, когда дело доходит до сложных понятий и описаний. В этом случае имитационные модели, описанные по соответствующим правилам, могут помочь нам устранить эти неточности, предоставляя в наше распоряжение более действенные, более успешные способы общения. Модель делает более понятной общую структуру исследуемого объекта и вскрывает важные причинно-следственные связи.



В предлагаемом учебном пособии упор делается на изучение и создание средств описания процессов функционирования систем, в результате использования которых создается собственно имитационная модель.

Под функционированием системы понимается процесс изменения ее состояния во времени. В данном контексте рассматривается способ описания такого процесса с учетом того, что система имеет высокую размерность, разделяется на множество объектов, различным способом связанных между собой, руководствуется сложными алгоритмами, описывающими переход из одного состояния в другое.

Первая глава пособия посвящена изложению теоретической схемы описания взаимодействующих последовательно-параллельных процессов. Начальные теоретические положения процессных описаний были заложены в работах Дейкстры, В. Бусленко, Н. Моисеева и др. Теоретические положения процессных описаний активно использовались Грэди Бучем при создании системы UML и ее приложений (например, ARIS).

В качестве определения процесса принято определение, данное в работе «Лекции по теории сложных систем» (авторы Н. Бусленко, Н. Коваленко, В. Калашников). Далее определяются понятия объекта, подпроцесса, операций над процессами, анализируется операторный способ задания процесса, особенности взаимодействия процессов в объектах, вводятся понятия отношения сцепленности процессов и объектов. На основании проведенного анализа предлагается в качестве основного способа описания процесса принять алгоритмическую модель (АМП). В основе АМП лежат понятия элементарного оператора, трека и инициатора. Трек представляет собой линейный граф операторов, элементарный оператор – совокупность операторов состояния и условия продвижения инициатора, инициатор – динамический объект, обладающий свойством иницирования выполнения элементарного оператора.

Предложенная модель позволила выполнить ряд формальных преобразований над каждым из компонент АМП. Так, преобразования трека, как графа, позволили сформировать понятие структуры, классифицировать виды структур, определить способы развертки структуры в трек. Анализ элементарного оператора позволил сформулировать понятия эквивалентности, определить обобщенные и объединенные операторы, уровни описания, преобразования описаний, классифицировать условные элементарные операторы. Дальнейшее развитие понятия инициатора позволило определить понятие локальной среды процесса. В результате удастся задать способы компактного описания совокупности связанных процессов, определить понятие ресурса, конфликта процессов на ресурсе и дать способы его разрешения. Компактное задание процесса или группы однородных процессов сводится к блочному описанию взаимодействующих процессов. Предлагаются 3 типа блоков: агрегаты, процессоры и контроллеры. Блоки отличаются друг от друга типами операторов, структурами

описаний, отношением к инициаторам, взаимодействием с локальной средой процессов. Предложенная формализованная схема позволяет выполнять графовые преобразования с целью декомпозиции всей схемы на ресурсные узлы (описываемые, как СМО, в терминологии теории массового обслуживания), сети узлов (сети массового обслуживания), независимые участки процессов. В результате можно дать рекомендации об использовании того или иного математического аппарата (прямые и вложенные описания, применения ТМО, имитационного моделирования и пр.).

Во второй главе изложена лингвистическая система, позволяющая описать в языковой форме взаимодействие объектов в составе сложной системы в ходе ее функционирования. Как было указано выше, имитационная модель выполняет еще одну важнейшую функцию: она выступает как средство общения. Именно поэтому лингвистическая система предлагается в форме псевдоязыка, поскольку она ориентирована на создание аппарата описания, а не на машинную реализацию.

Концепция описания должна опираться на некоторую инвариантную к предметной области концепцию. В качестве такой концепции используются результаты исследований, проведенные в главе 1 настоящего пособия. Разработанная лингвистическая система носит название "Псевдоязык Описания Сцепленных Процессов" (ПОСП). Основное назначение этой системы состоит в разработке и использовании достаточно простых, но универсальных, средств описания совокупности процессов в их взаимосвязи. Эти описания должны служить средством общения между специалистами, выполняющими моделирование либо планирование процессов функционирования сложных систем. Поэтому язык описания носит префикс "псевдо-", что подчеркивает его направленность на объяснение, прежде всего собеседнику, идей организации процессов и механизмов их взаимодействия. Требования к синтаксической точности и деталям второго плана в языке значительно ослаблены. Язык предполагает включение макрорасширений, что расширяет возможности описания типовых конструкций. Поскольку ПОСП опирается на языковые средства, полученные в результате исследований алгоритмической модели описания процессов, имеющей очень широкий диапазон приложений, то программа, написанная на ПОСП, достаточно просто отображается на широкий класс специализированных языков моделирования.

В третьей главе рассмотрен алгоритм построения имитационного процесса. В основе построения имитационного алгоритма лежат правила построения квазипараллельного процесса, опирающиеся на алгоритмическую модель описания процессов. Эти правила включают способ управления системным временем, понятия активных и пассивных событий в треках процессов, построение класса одновременных событий (КОС), алгоритм генерации квазипараллельного процесса. Показано, что первым событием в КОС является активное событие, все остальные события - пассивны. Приводится схема и алгоритм универсального моделирующего алгоритма

сканирующего типа, содержащего такие программы, как календарь и алгоритм проверки условий, структуры их поддержки в виде таблицы будущих времен и таблицы условий. Алгоритм следует из правил, сформулированных для отображения системы параллельных описаний на один квазипараллельный процесс.

В четвертой главе излагается система имитационного моделирования GPSS. Система GPSS реализует классическую концепцию процессных описаний. Поскольку система ПОСП опирается на процессный подход при описании процессов функционирования систем, то отображение описаний ПОСП на язык GPSS производится совершенно естественно. Кроме того, система моделирования имеет огромную популярность, трансляторы для системы GPSS реализованы на всех ЭВМ во всех версиях операционных систем.

В GPSS транзакты выступают в роли инициаторов, локальная среда - в виде параметров транзакта, блоки GPSS в большинстве своем реализуют функции операторов ПОСП. GPSS содержит и блоки агрегатного типа, например, GENERATE, развиты условные операторы продвижения инициатора. Блоки выполнены в форме библиотечных процедур. Как правило, функционально большая часть блоков GPSS может быть представлена, как макросы ПОСП. При описании синтаксиса блоков GPSS в главе проводится параллель с описанием функций этого блока в виде фрагмента программы на ПОСП. В главе приведен синтаксис основных операторов языка GPSS, знание которых необходимо для составления типовых имитационных программных конструкций. Расширенное описание ряда блоков приведено в приложении. В приложении также приведен перечень лабораторных работ, призванных привить студентам навыки практической разработки имитационных программ на GPSS. В процессе выполнения лабораторных работ студенты индивидуально выполняют поставленные задачи. Методическая схема выполнения лабораторных работ предполагает первоначальное изучение студентом готовых программ, размещенных в библиотеке моделей. Затем в заключительной части лабораторного занятия студенту предлагается задание для самостоятельной работы по составлению имитационной программы.

В пятой главе изложены правила построения имитационных программ с помощью системы моделирования Simio. Эта система выбрана, как наиболее последовательная реализация потоково - агрегативной концепции. Simio поддерживает систему понятий объектного моделирования во всех аспектах модельной среды. С точки зрения ПОСП объекты в Simio представляют собой агрегаты. Объект может моделировать машину, робота, самолет, клиента, доктора, танк, автобус, корабль или иные вещи, которые потребуются моделировать. Объекты создаются разработчиками, хранятся в библиотеках и могут быть использованы в разных моделях.

Агрегатный подход позволил разработчикам широко использовать визуальную концепцию работы с программой модели, начиная от момента создания программы, включая задание параметров объектов, верификацию модели и отображение результатов моделирования. Средства визуализации лежат в основе всего описательного механизма Simio. Модель строится путем комбинирования объектов, представляющих физическую трактовку системы. Модель Simio визуально может выглядеть как схема реальной системы. Логический алгоритм и анимация строятся с помощью интуитивно понятного интерфейса. На объект может быть наложена анимация для отображения изменяющегося состояния объекта. В отличие от других систем моделирования процесс создания объекта достаточно прост и является полностью визуальным. В Simio нет необходимости писать код или скрипт для создания новых объектов.

Таким образом, студенты по ходу реализации учебной программы, опирающейся на материал пособия, изучают достаточно широкий класс подходов к описанию процессов и построению имитационных программ.

# ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ АЛГОРИТМИЧЕСКОГО МЕТОДА ОПИСАНИЯ ПАРАЛЛЕЛЬНО-ПОСЛЕДОВАТЕЛЬНОЙ СОВОКУПНОСТИ ВЗАИМОДЕЙСТВУЮЩИХ ПРОЦЕССОВ

## 1.1. Определение процесса

Система отличается многочисленными характеристиками, определяемыми аспектами ее описания. Нас будет интересовать сейчас и в дальнейшем анализ процесса ее функционирования. Под функционированием системы понимается процесс изменения ее состояния во времени. В общем случае необходимо учитывать, что система имеет высокую размерность, разделяется на множество объектов, различным способом связанных между собой, руководствуется сложными алгоритмами, описывающими переход из одного состояния в другое.

*Всю совокупность параметров системы, определяющих процесс функционирования или участвующих в нем, назовем параметрическим множеством системы*  $Q = \{q_i\}_{i=1}^n$ , где  $q_i$  – некоторый параметр. Каждый параметр  $q_i$  принимает множество значений, обозначаемое в дальнейшем как  $\sigma(q_i)$ .

Например, выберем в качестве системы центральный процессор (ЦП) и жесткий диск (Д).

Для ЦП определим следующие параметры:

количество ядер процессоров (КП);

производительность одного ядра процессора (ПОП);

состояние процессора (СП);

стоимость процессора (СтП).

Для Д определим следующие параметры:

скорость записи и считывания (СЗС);

состояние диска (СД).

Тогда множества значений параметров можно определить следующим образом:

$\sigma(\text{КП}) = \{1, 2, 4\}$ ;

$\sigma(\text{ПОП}) = \{1, 5, 10, 50\}$ , эти значения могут соответствовать какой-либо системе оценки производительности, например, обозначать среднее количество миллионов операций за единицу времени;

$\sigma(\text{СП}) = \{\text{“работает”}, \text{“простаивает”}, \text{“тестируется”}\}$ ;

$\sigma(\text{СтП}) = \{2 - 20\}$ , эти значения стоимости могут быть выражены, например, в некоторых условных единицах;

$\sigma(\text{СЗС}) = \{10 - 50\}$ , эти показатели скорости операций на диске могут быть заданы аналогично параметру ПОП;

$\sigma(\text{СД}) = \{\text{“работает”}, \text{“простаивает”}, \text{“ремонтируется”}\}$ .

Определим *пространство состояний системы* как Декартово произведение  $S = \prod_{\forall i} \sigma(q_i)$ . В этом пространстве каждый параметр выступает в роли координаты, а размерность пространства равна мощности множества  $Q$ . Элемент пространства  $S$  есть возможное *состояние системы*. В рассмотренном выше примере размерность пространства состояний равна 6, а само пространство состояний имеет вид:

$S = \sigma(\text{КП}) \times \sigma(\text{ПОП}) \times \sigma(\text{СП}) \times \sigma(\text{СтП}) \times \sigma(\text{СЗС}) \times \sigma(\text{СД})$

Возможным состоянием системы  $s \in S$  может быть следующий кортеж:

$s = \langle 2, 10, \text{“простаивает”}, 15, 40, \text{“ремонтируется”} \rangle$

В дальнейшем нас будет интересовать процесс изменения состояний системы во времени. Примем за основу определение процесса, предложенное в работе [3].

*Процесс*  $Z$  есть четверка:

$$Z = \langle S, T, F, \alpha \rangle$$

где:

$S$  - пространство состояний системы, определенное ранее;

$T$  - множество моментов времени *изменения* состояний системы;

$F$  - график процесса, определяемый как отображение  $T \rightarrow S$ , причем это отображение должно быть функциональным (однозначным);

$\alpha$  - отношение линейного порядка на  $T$ .

Если множество  $T$  задано как упорядоченное, то в определении процесса  $\alpha$  может быть опущено. В общем случае множества  $T$  и  $S$  могут быть как дискретными, так и непрерывными.

Интервал времени  $[t_n, t_k]$ , где  $t_n = \min\{T\}$ ,  $t_k = \max\{T\}$ , назовем *интервалом определения процесса*.

Поскольку пространство  $S$  координатного типа, то в случае необходимости подчеркнуть систему координат  $Q$ , на которой оно определено, будем обозначать его также  $S_Q$ .

В этих обозначениях, если множество  $T$  задано, как упорядоченное, а пространство  $S$  определено на множестве параметров  $Q$ , определить процесс можно как:  $Z = \langle S_Q, T, F \rangle$ .

Кортеж  $\langle \tilde{X}_1, \dots, \tilde{X}_i, \dots, \tilde{X}_n \rangle$ , где  $\tilde{X}_i$  - значения элементов множества  $X$ , будем обозначать как  $\langle \dots \tilde{X} \dots \rangle_X$ .

Определим фазовое пространство  $\Phi$  процесса  $Z$  как  $\Phi = T \times S$ . Тогда график  $F$  есть подмножество  $\Phi$ .

Если  $f \in \Phi$ , то  $f = \langle t, \langle \dots \tilde{q} \dots \rangle_Q \rangle$ , где  $t \in T$ .

Если  $Q_1 \subseteq Q$ , то определим понятие проекции  $f$  на пространство  $S_{Q_1}$  как:

$Pr_{S_{Q_1}} f = \langle t, \langle \dots \tilde{q} \dots \rangle \rangle$ . Проекцией  $f$  на  $T$  является  $t$ .

Введем понятие подпроцесса  $Z^i$  как плотное во времени подмножество процесса  $Z$  на интервале  $[t_i; t_j]$  при условии, что  $[t_i; t_j] \subseteq [t_n, t_k]$ . Плотность во времени означает, что на интервале  $[t_i; t_j]$  нет ни одной точки, принадлежащей  $T$  и не относящейся к подпроцессу  $Z^i$ . Этот интервал назовем *интервалом определения подпроцесса*. Понятие подпроцесса позволяет рассматривать процесс в виде последовательности подпроцессов и производить операции разделения и объединения фрагментов процесса.

## 1.2. Объект, характеристики объекта

Будем рассматривать *объект*, как составную часть системы, характеризующуюся *параметрическим множеством объекта*  $O_1 \subseteq Q$ . В дальнейшем для краткости, когда это не вызывает двусмысленности, вместо понятий *параметрическое множество системы* и *параметрическое множество объекта* будем говорить *система* и *объект* соответственно.

Пространство состояний  $S_{O_1}$  объекта  $O_1$  определяется аналогично системе, как  $S_{O_1} = \prod_{q_i \in Q_1} \sigma(q_i)$ . Будем предполагать, что система всегда имеет полное разбиение на объекты. Таким образом:  $\bigcup_{i=1} O_i = Q$ . Разбиение является *непересекающимся*, если  $O_m \cap_{m \neq l} O_l = \emptyset$ . В противном случае разбиение произведено на *пересекающиеся* объекты.

Если задан процесс  $Z_Q$  в системе, то процесс в объекте  $O_1$  может быть определен, как:

$$Z_{O_1} = \underset{S_{O_1}}{Pr} Z_Q$$

### Построение отображения F.

Пусть имеем объект  $O_1$  в системе  $Q$ . Тогда генерация процесса  $Z_{O_1}$  в объекте  $O_1$  (построение отображения F) может быть выполнена путем задания оператора  $H^{O_1}$  [3]:

$$s_i^{O_1} = H^{O_1}(A_i^{O_1}, t_i, \omega) \quad (1)$$

где:

$$t_i \in T_{O_1};$$

$s_i^{O_1}$  - элемент множества  $S_{O_1}$ . Индекс  $t_i$  подчеркивает тот факт, что это значение множества  $S_{O_1}$  соответствует моменту времени  $t_i$ ;

$A_i^{O_l}$  - множество аргументов оператора  $H^{O_l}: A_i^{O_l} \subseteq Q$ ;

$\omega$  - случайное число.

Важно обратить внимание на то, что, если пространство состояния объекта определяется на параметрах  $O_l \subset Q$ , то множество аргументов является самостоятельным подмножеством параметров всей системы  $Q$ . При этом необходимо учитывать, что состав элементов множества аргументов *в общем случае зависит от времени*.

#### Отношение сцепленности и зависимости объектов.

Рассмотрим два объекта  $O_l$  и  $O_m$  в системе  $Q$ . Пусть  $O_l \cap O_m = \emptyset$ , а процессы в них заданы следующими операторами:

$$s_{t_i}^{O_l} = H^{O_l}(A_i^{O_l}, t_i, \omega); \quad s_{t_i}^{O_m} = H^{O_m}(A_i^{O_m}, t_i, \omega).$$

Если  $O_m \cap A_i^{O_l} = \emptyset$  и  $O_l \cap A_i^{O_m} = \emptyset$ , то такие процессы и объекты называются *не сцепленными* в момент времени  $t_i$ .

Если  $O_l \cap A_i^{O_m} \neq \emptyset$ , то объект  $O_m$  *сцеплен* с объектом  $O_l$  в момент времени  $t_i$ .

То же относится и к их процессам. Это означает, что для определения состояния объекта  $O_m$  в момент времени  $t_i$ , необходимо знание состояния объекта  $O_l$  в это же время. Обозначим отношение сцепления как  $O_l \rightarrow O_m$ . Если  $O_m \cap A_i^{O_l} \neq \emptyset$ , то объект  $O_l$  *сцеплен* с объектом  $O_m$  в момент времени  $t_i$ :  $O_m \rightarrow O_l$ . Если одновременно  $O_m \cap A_i^{O_l} \neq \emptyset$  и  $O_l \cap A_i^{O_m} \neq \emptyset$ , то объекты  $O_m$  и  $O_l$  *взаимно-сцеплены* в момент времени  $t_i$ :  $O_m \leftrightarrow O_l$ . При операторном способе описания процессов всегда нежелательна модель, приводящая к появлению взаимно - сцепленных объектов, поскольку возникающую неопределенность приходится раскрывать путем решения в общем случае систем нелинейных уравнений, что может привести к непреодолимым трудностям. В дальнейшем будем стремиться создавать модели, не приводящие к взаимному сцеплению объектов.

Не следует смешивать отношение сцепления и зависимости. Так, если  $O_m \rightarrow O_l$  и  $O_l \rightarrow O_k$ , то вовсе не обязательно, чтобы  $O_m \rightarrow O_k$ . Таким образом, отношение сцепления не является транзитивным.

Если к отношению сцепления добавить полное транзитивное замыкание, то получим *отношение зависимости*. Если  $O_l$  зависит от  $O_m$ , а  $O_k$  зависит от  $O_l$ , то  $O_k$  зависит и от  $O_m$ . Таким образом, *отношение сцепления можно определить как отношение непосредственной зависимости*.



### 1.3. Алгоритмическая модель процесса (АМП)

Поскольку, в общем случае, задание процесса в виде единого оператора (1) либо затруднительно, либо невозможно, то предлагается задавать оператор  $H$  в виде некоторой алгоритмической структуры.

Рассмотрим дискретный процесс  $Z$ . Поставим в соответствие каждой  $i$ -ой точке процесса (момент времени изменения состояния  $t_i$ ) некоторый оператор  $H_i^c$ . Оператор  $H_i^c$  вычисляет значение состояния  $S_i \in S$  в момент времени  $t_i$ :

$$s_i = H_i^c(A_i, t_i, \omega)$$

Оператор  $H_i^c$  описывает вычисление только одной  $i$ -й точки процесса  $Z$ . В силу этого условия будем в дальнейшем называть этот оператор *элементарным*.

Таким образом, если график процесса содержит  $n$  точек, то мы должны задать линейную последовательность элементарных операторов:

$$h_1^c, h_2^c, \dots, h_i^c, \dots, h_n^c.$$

Введем новый элемент модели - *инициатор*. Будем полагать, что инициатор - это объект, обладающий следующими фундаментальными свойствами:

- а) независимостью: может существовать самостоятельно без операторов;
- б) динамичностью: инициатор имеет возможность перемещаться от оператора к оператору; будем называть попадание инициатора на оператор *сцеплением инициатора с элементарным оператором*;

в) инициативностью: в момент сцепления инициатора с оператором происходит выполнение (инициирование) элементарного оператора, что соответствует вычислению нового состояния процесса. Будем в дальнейшем полагать, что выполнение элементарного оператора происходит мгновенно.

Будем в дальнейшем полагать, что выполнение элементарного оператора происходит *мгновенно*. Это ограничение не сужает применимости предлагаемой модели, поскольку, если необходимо описать процесс, где вычисление нового состояния требует затрат реального времени, то можно ввести два элементарных оператора, ограничивающих начальный и конечный момент времени интервала вычислений. Таким образом, описание процесса может быть выполнено путем задания линейной последовательности операторов  $\langle H_i^c \rangle_{i=1}^n$  и перемещения по этой последовательности инициатора  $I$ , сцепляющегося с элементарными операторами  $h_i^c$  в заданные моменты времени  $t_i$  изменения состояния процесса.

Предлагаемая модель описания процесса предполагает, что моменты сцепления инициатора с элементарными операторами определяют сами элементарные операторы. С этой целью введем в состав элементарного оператора  $h_i^c$  оператор  $h_i^y$ , который

определяет условие, при выполнении которого инициатор покидает оператор  $h_i^c$  и сцепляется со следующим оператором  $h_{i+1}^c$ . Возможны следующие варианты задания такого условия:

- а) указание момента времени сцепления инициатора с оператором  $h_{i+1}^c$ ;
  - б) определение логического условия, при выполнении которого инициатор сцепляется с оператором  $h_{i+1}^c$ ;
  - в) комбинированная форма, включающая варианты а) и б).
- Таким образом:

$$h_i^y \in \{h_i^t, h_i^n, h_i^{t,n}\}, \text{ где}$$

- $h_i^y$  - оператор условия продвижения инициатора;
- $h_i^t$  - оператор временного условия (соответствует варианту а);
- $h_i^n$  - оператор логического условия (соответствует варианту б);
- $h_i^{t,n}$  - оператор комбинированного условия (соответствует варианту в).

Расширим понятие элементарного оператора, добавив к нему помимо оператора  $h_i^c$  оператор  $h_i^y$ . Таким образом, определим элементарный оператор  $h_i$ , как двойку:

$$h_i = \langle h_i^c, h_i^y \rangle.$$

Теперь можно определить понятие *алгоритмической модели процесса* (в дальнейшем АМП), в виде тройки:

$$\text{АМП} = \langle \{h_i\}_{i=1}^n, \beta, I \rangle,$$

где:  $\{h_i\}_{i=1}^n$  - множество элементарных операторов;

$\beta$  - линейный порядок на  $\{h_i\}_{i=1}^n$ ;

$I$  - инициатор.

Следует обратить внимание на то, что АМП содержит *один и только один* инициатор, т.е. каждому процессу соответствует один инициатор. В этом смысле инициатор является представителем процесса, при его потери либо отсутствии развитие процесса прекращается.

Линейную последовательность элементарных операторов назовем треком TR:

$$TR = \langle \{h_i\}_{i=1}^n, \beta \rangle$$

Тогда можно АМП определить также как двойку:

$$\text{АМП} = \langle TR, I \rangle$$

### 1.4. Структура трека

Пусть задан некоторый трек TR. В реальных приложениях трек содержит достаточно много элементарных операторов, выполняющих одни и те же операции над аргументами. Операторы эквивалентны, если при одних и тех же значениях аргументов они вычисляют одинаковые результаты. Это свойство трека позволяет задать отношение эквивалентности на множестве  $\{h_i\}_{i=1}^n$  элементарных операторов трека TR.

Назовем *структурой* свертку трека TR по отношению эквивалентности элементарных операторов.

*Пример.* Пусть задан некоторый трек TR (рисунок 1.1).

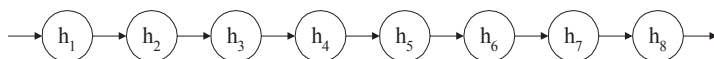


Рисунок 1.1. Пример трека

Пусть отношение эквивалентности *элементарных операторов* имеет вид:

$$\{(h_1, h_3), (h_2, h_5, h_6, h_8), (h_4, h_7)\}$$

Тогда структура имеет вид графа (рисунок 1.2).

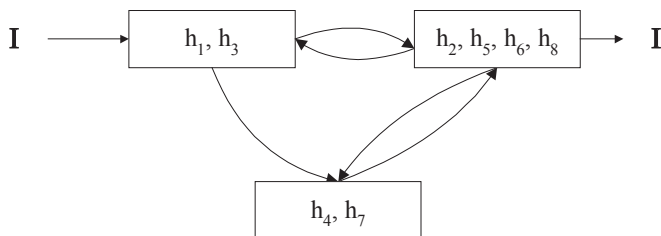


Рисунок 1.2. Свертка трека в структуру

Очевидно, если заданы трек и отношение эквивалентности операторов, то всегда возможно построение структуры. Однако обратное восстановление трека по структуре является неоднозначной операцией. Эта операция относится к классу операций развертки. С тем, чтобы операцию построения трека из структуры сделать однозначной, введем еще один тип элементарного оператора - *навигационный* элементарный оператор. Навигационный оператор определяется так же, как и элементарный оператор, однако в результате его выполнения определяется тот элементарный оператор в структуре, который должен выполняться следующим. Выполнение навигационного оператора инициируется инициатором. Поскольку время на выполнение навигационного оператора, как и всех элементарных операторов, равно нулю, то использование его не сказывается на времени

реализации процесса. В общем случае навигационный оператор должен следовать за каждым элементарным оператором в структуре.

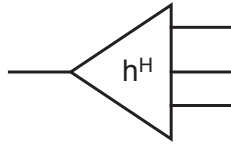


Рисунок 1.3. Обозначение навигационного оператора

Если навигационный оператор обозначить, как показано на рисунке 1.3, то структура из вышеописанного примера будет иметь вид (рисунок 1.4):

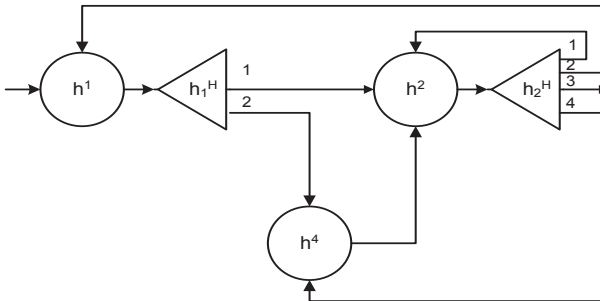


Рисунок 1.4. Вид структуры с навигационными операторами

Здесь операторы  $h_1$ ,  $h_2$ ,  $h_4$  являются представителями своего класса эквивалентности. Как видно, после операторов  $h_1$  и  $h_2$  стоят навигационные операторы  $h_1^H$  и  $h_2^H$ , в то время как после оператора  $h_4$  нет необходимости в использовании навигационного оператора. Навигационный оператор используется также и для организации циклов (оператор  $h_2^H$ , выход 1).

Использование структуры по сравнению с треком позволяет значительно снизить размерность описания процесса. Однако необходимо иметь в виду, что процесс определен только в случае задания трека, а поэтому структура есть лишь способ более компактного описания трека, генерация самого трека остается необходимой операцией. На практике задание структуры с навигационными операторами для последующей генерации трека используется часто и повсеместно, где необходима генерация процесса.

Определение элементарного оператора в составе структуры можно представить в виде:  $h = \langle h^c, h^v, h^u \rangle$ .

## 1.5. Подобные процессы

Элементарный оператор  $h$  оперирует с параметрами и изменяет состояние объекта. По отношению к оператору параметры могут быть входными, выходными и рабочими (смотри рисунок 1.5)

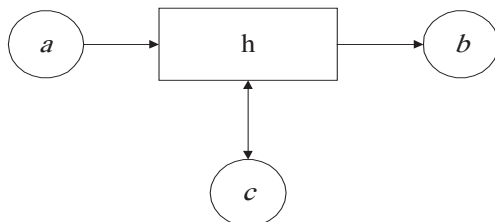


Рисунок 1.5. Отношение параметра к оператору

где:  $a$  - входной параметр;  $b$  - выходной параметр;  $c$  - рабочий параметр.

Входной параметр означает его принадлежность к множеству  $A$ , выходной - к формированию состояния  $s$ , что говорит о его принадлежности к множеству  $O$ , рабочий - к тому и другому множеству одновременно.

Если на трек элементарных операторов указать используемые этими операторами параметры и их взаимосвязи, то получим операторно-параметрическую схему.

Пример такой схемы приведен на рисунке 1.6. Двойными линиями показан путь инициатора, а одинарными - отношение параметров к операторам. Такие схемы дают наглядную картину взаимодействия параметров в ходе реализации процесса.

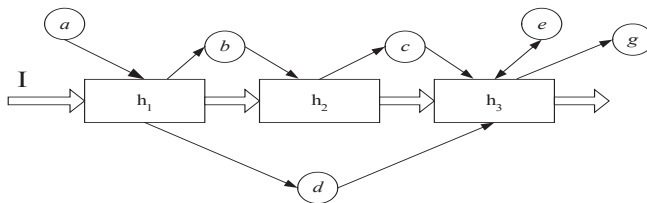


Рисунок 1.6. Операторно-параметрическая схема

Как видно из приведенного примера, операторно-параметрическая схема дает достаточно полное представление о способе описания процесса с использованием АМП.

Рассмотрим случай задания двух близких по описанию процессов  $Z_1$  (трек А) и  $Z_2$  (трек В), показанных на рисунке 1.7. И в том и в другом треке используются элементарные операторы  $h_1$  и  $h_2$ , но они взаимодействуют с разными параметрами как входными, так и выходными. Было бы желательно найти способ объединения описаний таких процессов. Для решения поставленной задачи дополним определение инициатора, добавив к его

фундаментальным свойствам возможность включать в себя параметры. Таким образом, инициатор наряду с фундаментальными свойствами приобретает некоторое “тело” в виде совокупности параметров. Параметры в этой совокупности должны быть упорядочены. Назовем эту совокупность параметров *локальной средой* процесса. Будем в дальнейшем считать, что “тело” инициатора представляет собой *ссылку на локальную среду*. Таким образом, движение инициатора по треку есть движение ссылки на локальную среду по треку, а сама локальная среда может быть неподвижна.

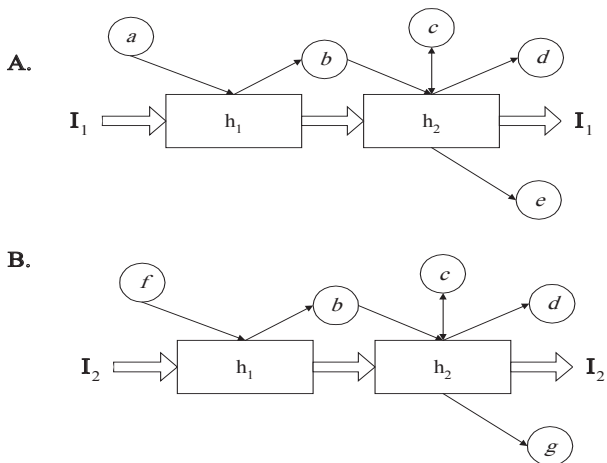


Рисунок 1.7. Пример подобных описаний процессов

Тогда можно предложить следующую схему свертки описаний двух процессов (рисунок 1.7) в одно общее описание (рисунок 1.8):

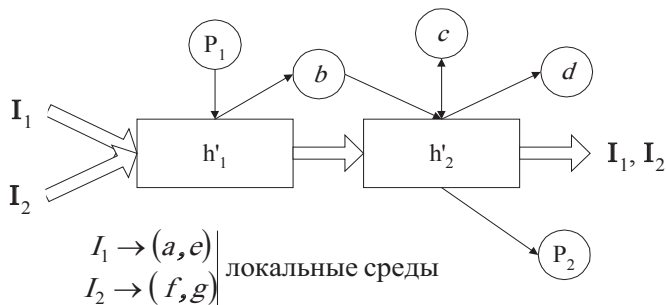


Рисунок 1.8. Объединенное описание процессов  $Z_1$  и  $Z_2$

Поскольку элементарные операторы в такой схеме работают не только с явно заданными параметрами, но и с параметрами в локальных средах, то назовем такие элементарные операторы *объединенными* элементарными операторами.

Эти рассуждения могут быть распространены на случай  $n$  параллельно протекающих процессов. Процессы, сгенерированные треком или структурой, использующими объединенные элементарные операторы и локальные среды, называются *подобными*.

Таким образом, удастся снизить размерность описания множества процессов, введя отношение подобия процессов. Для описания совокупности подобных процессов достаточно иметь одно объединенное описание трека или структуры и множество одинаково структурированных локальных сред, привязанных к инициаторам.

### 1.6. Ресурсы, конфликты на ресурсах

Процессы  $Z_i$  в системе  $Q$  развиваются параллельно. Это значит, что они изменяют значения параметров системы в течение одного и того же интервала времени. Достаточно типичны ситуации, когда по логике функционирования системы накладываются ограничения на изменение некоторых параметров несколькими процессами одновременно в течение заданного либо обусловленного интервала времени. Это возможно лишь для пересекающихся объектов. Ограничения развития процесса, накладываемые другими процессами, назовем конфликтными ситуациями, или конфликтами.

Общую область параметров для пересекающихся процессов  $O_k$  и  $O_m$  назовем ресурсом. Таким образом, ресурс  $R_{k,m}$  определяется, как

$$R_{k,m} = O_k \cap O_m$$

Конфликт возможен только при наличии ресурса. Таким образом, необходимо добиться согласования процессов в этой области. В основе способов разрешения конфликтов лежит утверждение об обязательном разделении доступа процессов к ресурсу во времени. Рассмотрим следующие способы разрешения конфликтных ситуаций.

А. **Явное разделение по времени (синхронизация)**. При этом способе разрешения конфликта разнесение во времени производится явным указанием интервалов времени, определенных для каждого процесса. На рисунке 1.9 показан пример выделения таких интервалов для случая конфликта трех процессов.

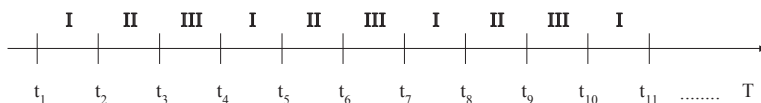


Рисунок 1.9. Синхронизация процессов

Б. Логический способ. В частности – использование семафоров. Если условие захвата ресурса не ограничивает время использования этого ресурса захватившим его процессом, то в этом случае удобно использовать семафоры. Семафор относится к логическим способам разрешения конфликтов. Семафор есть простая логическая переменная, однозначно соответствующая ресурсу. Значение семафора '0' означает, что ресурс может быть захвачен процессом, значение семафора '1' блокирует захват ресурса. На рисунке 1.10 показан пример использования семафора С при захвате ресурса R двумя процессами  $Z_1$  и  $Z_2$ .

Следует заметить, что в качестве семафора может выступать любая логическая функция.

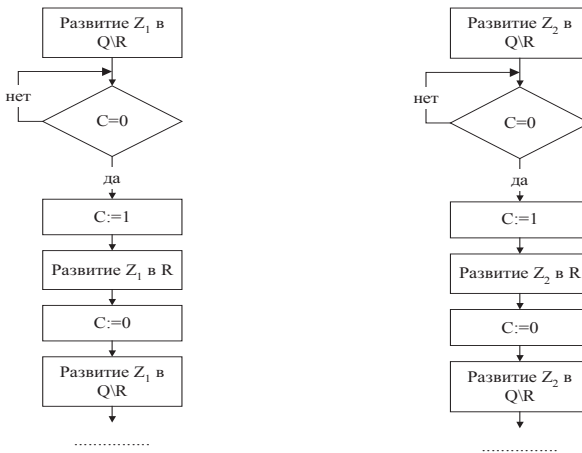


Рисунок 1.10. Применение семафора

В. Алгоритмический способ. Наиболее общий способ управления процессами при захвате ресурсов состоит в использовании блоков - контроллеров. Использование этих блоков для разрешения конфликтов приведено ниже при определении понятия К-блока.

### 1.7. Блоки, типы блоков

На трек можно задать некоторое *плотное* разбиение элементарных операторов на подмножества. Это разбиение обычно выполняется с целью получения функционально однородных подмножеств операторов. Совокупность операторов, входящих в одно подмножество, назовем *обобщенным* оператором.

Пример разбиения приведен на рисунке 1.11. Здесь подмножество операторов  $h_1, h_2, h_3$  объединено в один обобщенный оператор  $H_1$ , а  $h_4, h_5$  - в обобщенный оператор  $H_2$ . В результате получаем трек обобщенных операторов.



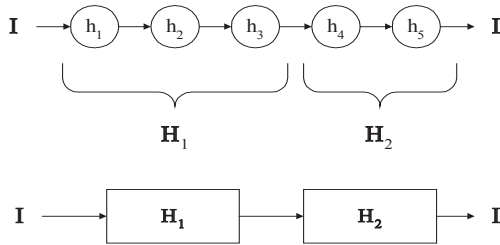


Рисунок 1.11. К понятию обобщенного оператора

Совокупность обобщенных операторов и связанных с ним параметров образует **блок**. Принципиально будем различать два типа блоков: *агрегат* и *процессор*.

*Агрегат*. На практике часто возникает необходимость описывать процесс функционирования некоторой машины по преобразованию значений параметров в соответствии с заданным циклическим алгоритмом. Такой процесс можно описать с помощью блока общего вида, в котором существует один инициатор, а трек циклически замкнут. Блок, в котором развивается один единственный циклический процесс, будем называть агрегатом, или А-блоком. Таким образом, агрегат содержит единственный инициатор и трек элементарных операторов, замкнутый внутри блока. Обмен между агрегатом и другими блоками возможен исключительно посредством параметров.

*Процессор*. Блок, предназначенный для генерации процессов, инициаторы которых являются внешними по отношению к блоку, назовем процессором, или П-блоком. Инициаторы, поступившие извне, сцепляются с блоком, порождая процессы, и затем покидают его. Поскольку процессор генерирует множество одновременно протекающих процессов, в нем используются исключительно объединенные элементарные операторы, а инициаторы должны содержать локальные среды. Таким образом, процессор представляет собой описание произвольной структуры, содержащей объединенные операторы. Процессы порождаются в этом блоке лишь при поступлении в него извне инициаторов, содержащих локальные среды. Из вышесказанного следует, что процессор порождает параллельно протекающие во времени подобные процессы.

*Контроллер*. Рассмотрим вновь блок типа агрегат. Как было показано выше, он не имеет возможности взаимодействовать с внешними инициаторами. С тем, чтобы снять это ограничение, введем над инициаторами операции пассивизации и активизации. Операция пассивизации переводит инициатор в класс обычных параметров. Операция активизации, наоборот, обычный параметр переводит в класс инициаторов. Если агрегат содержит операторы, выполняющие указанные операции, то такой агрегат назовем контроллером, или К-блоком. Контроллер, таким образом, представляет собой агрегат, выполняющий операции над внешними инициаторами в соответствии с собственным алгоритмом

функционирования. Операции над инициаторами суть операции над процессами. Таким образом, контроллер исполняет роль управляющего звена в некоторой блочной схеме.

На рисунке 1.12 показаны обозначения блоков, используемые в дальнейшем.

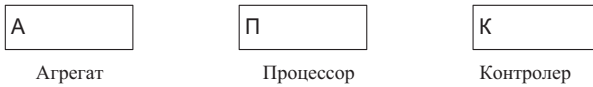


Рисунок 1.12. Обозначения блоков

Пример использования контроллера для разрешения конфликтной ситуации между процессами  $Z_1$  и  $Z_2$  приведен на рисунке 1.13.

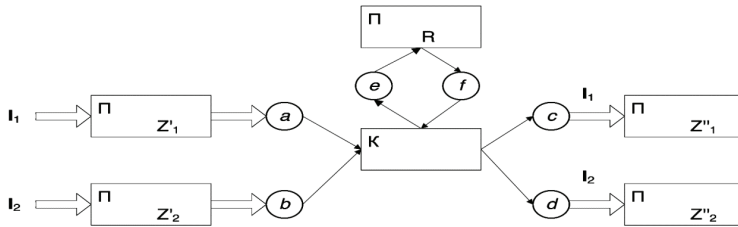


Рисунок 1.13. Применение К-блоков при разрешении конфликтов на R

Здесь первые П-блоки реализуют процессы  $Z_1'$  и  $Z_2'$  в пространстве  $Q \setminus R$ ; затем производится пассивизация инициаторов  $I_1$  и  $I_2$ , они переводятся в параметры  $a$  и  $b$  соответственно. К-блок рассматривает ситуацию с входными параметрами в соответствие с собственным алгоритмом. Приняв решение о захвате ресурса каким-либо процессом, К-блок передает  $a$  или  $b$  в параметр  $e$  и активизирует его, отсылая в П-блок ресурса  $R$ . После завершения процесса в  $R$  выдается сигнал в параметре  $f$ , в ответ на который К-блок передает параметр - инициатор в  $c$  либо  $d$  и активизирует его, отсылая на продолжение процесса  $Z_1''$  либо  $Z_2''$  в соответствующие П-блоки. Использование К-блоков является наиболее универсальным способом управления множеством процессов при захвате ресурсов.

### 1.8. Схемы описаний функционирования систем

Агрегативная схема. Агрегативная схема описания функционирования системы предполагает использование только А-блоков (рисунок 1.14). Как показано выше, в такой схеме взаимодействие может осуществляться лишь через параметры. В результате формируется информационная сеть агрегатов: агрегаты - вершины сети, дуги - информационные связи. Для агрегативных схем показано, что каждый А-блок в

такой модели может представляться некоторым конечным автоматом. Если функционирование системы может быть описано совокупностью конечных автоматов, взаимодействующих между собой через множество входных и выходных параметров, то применение агрегативных схем представляется наиболее рациональным.

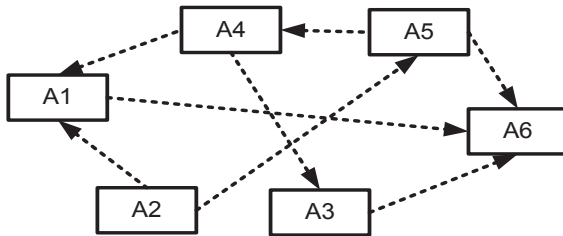


Рисунок 1.14. Пример агрегативной схемы

На рисунке 1.14 показаны 6 агрегатов, выполняющих функции сбора и обработки информации. Например, это могут быть какие-либо агентные структуры. Пунктирными линиями показано взаимодействие через параметры, относящиеся к тому или иному агрегату. Из схемы видно, что агрегат A2 сообщает информацию, агрегаты A1, A4, A3, A5 преобразуют полученную информацию, агрегат A6 собирает всю полученную информацию.

Процессная схема. Если в основе описания функционирования системы лежит использование П-блоков, то такое описание назовем процессным. Взаимодействие в процессных схемах осуществляется как через параметры, так и через локальные среды процессов. Процессный подход наиболее эффективен, когда имеем дело с множеством явно выраженных локальных процессов, например, при описании информационных, биологических, экономических, социальных и т.п. систем. Процессный подход реализован в языках Simula, GPSS и др.

Пример процессной схемы приведен на рисунке 1.15.

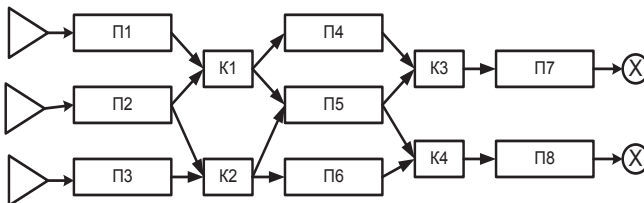


Рисунок 1.15. Пример процессной схемы

На рисунке 1.15 показаны процессоры и контроллеры, обеспечивающие управление потоками инициаторов и разрешение конфликтных ситуаций. Сплошными

линиями показаны пути движения инициаторов. Как видно из схемы, инициаторы, порожденные генераторами, проходят все треки, заданные процессорами и уничтожаются к конце пути. В процессных схемах реализация всех процессов осуществляется только инициаторами, возникшими из генераторов. Контроллеры лишь меняют направление и условия продвижения инициаторов.

Потоковая схема. Потоковая схема возникает из агрегативной, когда среди связей агрегатов появляются связи с управляющей информацией. Когда эта информация поступает в агрегат, она производит инициацию процессов внутри агрегата. Если на графе информационных связей агрегативной схемы выделить лишь связи с управляющей информацией, то получим потоковую сеть агрегатов. Управляющая информация в этом случае может рассматриваться, как *инициаторы сетевого уровня*.

Таким образом, в потоковых схемах существуют два вида инициаторов: сетевые инициаторы, инициирующие запуск агрегата, и блочные инициаторы, реализующие процесс функционирования блока.

Пример потоковой схемы приведен на рисунке 1.16.

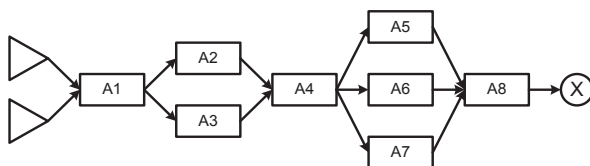


Рисунок 1.16. Пример потоковой схемы

На рисунке стрелками показаны пути движения сетевых инициаторов. Блочные (внутренние инициаторы) находятся внутри агрегатов. Например, сетевой инициатор, поступающий на агрегат A4 от агрегата A2, вызывает выполнение некоторого процесса в A4. Сам инициатор после этого может быть уничтожен или перемещен блоком A4 дальше по сети.

К потоковым схемам относятся сети Наура, E-сети, сети Петри. Наибольшее распространение среди потоковых схем получили сети массового обслуживания (СМО). Эти сети возникают из процессных схем, когда процессоры очень просты, либо отсутствуют вообще. Тогда основным элементом сети становятся контроллеры. Объединение контроллера с простым процессором или агрегатом формируют объект, называемый *системой массового обслуживания* (СМО). Таким образом, возникает сеть из СМО. Согласно определению потоковой схемы, по этой сети перемещаются инициаторы сетевого уровня, которые в теории СМО называются сообщениями или требованиями.

### 1.9. Вопросы для самопроверки

1. Понятие процесса, формальное определение. Определение подпроцесса, системы, объекта.
2. Операция свертки процессов. Пример.
3. Операция развертки процессов. Пример.
4. Операция проецирования процесса. Пример операции проецирования для двухпараметрической системы и двух объектов.
5. Операция сложения процессов. Свойство корректности складываемых процессов и условия его выполнимости.
6. Оператор общего вида описания процесса. Классификация операторов.
7. Оператор общего вида описания процесса. Понятие сцепленности процессов.
8. Алгоритмическая модель описания процесса. Понятия элементарного оператора, инициатора, трека.
9. Алгоритмическая модель описания процесса. Элементарный оператор, его составные части и их назначение.
10. Эквивалентные операторы. Структуры, виды структур.
11. Операторно-параметрическая схема описания процесса. Классификация параметров.
12. Однородные процессы. Понятие объединенного элементарного оператора. Локальные среды процессов.
13. Свертка операторно-параметрических схем однородных процессов.
14. Блоки, типы блоков. Особенности структуры каждого типа.
15. Блок типа агрегат. Назначение, особенности, примеры.
16. Блок типа процессор. Назначение, особенности, примеры.
17. Блок типа контроллер. Назначение, особенности, примеры.
18. Ресурсы, конфликты на ресурсах. Разрешение конфликтов с помощью «семафора». Преимущества и недостатки метода. Примеры.
19. Ресурсы, конфликты на ресурсах. Разрешение конфликтов с помощью «временной синхронизации». Преимущества и недостатки метода. Примеры.
20. Ресурсы, конфликты на ресурсах. Разрешение конфликтов с помощью блоков – контроллеров. Примеры.
21. Понятие блока.

22. Агрегатные схемы.
23. Процессные схемы.
24. Поточковые схемы.

## ГЛАВА 2. ПСЕВДОЯЗЫК ОПИСАНИЯ СЦЕПЛЕННЫХ ПРОЦЕССОВ

### 2.1. Введение

В главе изложена лингвистическая система, позволяющая описать взаимодействие объектов в составе сложной системы в ходе ее функционирования. Лингвистическая система предлагается в форме псевдоязыка, поскольку она ориентирована на создание аппарата описания, а не на машинную реализацию. Поэтому концепция описания должна опираться на некоторую инвариантную к предметной области концепцию. В качестве такой концепции используются результаты исследований, проведенные в главе 1 настоящего пособия. Разработанная лингвистическая система носит название "Псевдоязык Описания Сцепленных Процессов" (ПОСП). Основное назначение этой системы состоит в разработке и использовании достаточно простых, но универсальных, средств описания совокупности процессов в их взаимосвязи. Эти описания должны служить средством общения между специалистами, выполняющими моделирование либо планирование процессов функционирования сложных систем. Поэтому язык описания носит префикс "псевдо-", что подчеркивает его направленность на объяснение, прежде всего собеседнику, идей организации процессов и механизмов их взаимодействия. Требования к синтаксической точности и деталям второго плана в языке значительно ослаблены. Язык предполагает включение макрорасширений, что расширяет возможности описания типовых конструкций. Язык ПОСП не ориентирован на машинную реализацию. Поскольку ПОСП опирается на языковые средства, полученные в результате исследований алгоритмической модели описания процессов, имеющей очень широкий диапазон приложений, то программа, написанная на ПОСП, достаточно просто отображается на широкий класс специализированных языков моделирования.

### 2.2. Основные элементы языка

#### Идентификаторы

В языке ПОСП разрешается использование букв и символов любых алфавитов, в выражениях могут использоваться знаки и символы любых операций из области математики, лингвистики и пр.

*Идентификатор* - последовательность букв, цифр и некоторых символов, начинающаяся с буквы. Набор допустимых символов определяется самим пользователем.

Типы идентификаторов:

а) *простой* - последовательность букв и цифр, начинающаяся с буквы;

б) *составной* - последовательность простых идентификаторов, соединенных символом подчеркивания;

в) *стандартный* - фиксированный простой или составной идентификатор.

Стандартные идентификаторы могут быть пользовательскими и системными. Перечень пользовательских стандартных идентификаторов определяется самим пользователем.

В арифметических выражениях в качестве стандартных пользовательских идентификаторов часто используются имена функций таких, как SIN, COS, EXP, SQRT и т.д. Перечень таких идентификаторов может быть задан пользователем для каждой конкретной программы.

В качестве системных стандартных идентификаторов в языке ПОСП определены следующие:

*ВРЕМЯ* – переменная типа скаляр, ее значение - текущее время в модели;

*RAND* – стандартная функция (оператор). Вычисляет очередное значение псевдослучайной переменной **RAND** в  $[0,1]$  с равномерным законом распределения.

*ИНИЦИАТОР* – переменная типа ссылки, принимающая значение ссылки на локальную среду текущего инициатора.

Запись операторов языка сопровождается служебными словами, которые пишутся русскими буквами и выделяются *курсивом* (в рукописных текстах – подчеркиваются). Запись любого оператора завершается символом «;» .

Запись чисел соответствует общепринятым в математике правилам.

Любой оператор может быть помечен меткой. *Метка* - идентификатор; метка отделяется двоеточием от оператора.

*Список* - линейная последовательность элементов, разделенных запятой.

*Строка символов* - любая последовательность символов, кроме кавычек, помещенная в кавычки.

Блоки типа агрегат и контроллер содержат единственный инициатор по умолчанию. Начальное местонахождение инициатора указывается в блоке в разделе “Описание”.

#### Объекты, типы объектов

Объекты языка: простая переменная; переменная; блок; инициатор.

Каждый объект имеет тип, имя и значение.

Имя объекта есть идентификатор.

Значение объекта есть его содержание:

для простой переменной и переменной - логические, арифметические, текстовые или адресные значения;

для инициатора - адрес локальной среды процесса;

для блока - описание параметров блока и его алгоритм.

Тип простой переменной: скаляр; ссылка; метка.



Тип переменной: скаляр; ссылка; метка; вектор; пространство.

Тип блока: агрегат; процессор; контроллер; параметры.

Тип инициатора: *ссылка*.

### 2.3. Описание типов

Описание скалярных типов:

*Скаляр* - одно неделимое значение. Это - число, логическое значение либо строка символов.

Синтаксис: <список имен простых переменных> - *скаляр*(**ы**)

*Ссылка* - простая переменная типа скаляра, значением которой является адрес объекта.

Синтаксис: <список имен простых переменных> - *ссылка*(**и**)

*Метка* - простая переменная типа скаляра, значением которой является адрес помеченного ею оператора.

Синтаксис: <список имен простых переменных> - *метка*(**и**)

Описание вектора

*Вектор* – линейно - упорядоченная совокупность скаляров либо векторов (определение рекурсивно).

Синтаксис:

<список имен переменных> - *вектор* (<список элементов описания вектора>)

<элемент описания вектора> ::= <левая граница> - <правая граница> - <тип>

<левая граница>, <правая граница> ::= целое число

<тип> ::= скаляр | ссылка | метка | вектор | пространство

Если левая и правая границы совпадают, то указание правой границы может быть опущено.

Примеры:

1. Структура данных, приведенная на рисунке 2.1, может быть описана как: вектор (1-3 - скаляр, 4-5 - ссылка, 6-7 - скаляр)



Рисунок 2.1. Структура данных – вектор

2. Структура данных на рисунке 2.2 может быть описана как: вектор ( 1-вектор ( 1-скаляр , 2-вектор (1-2-ссылка ) , 3-ссылка), 2-3-скаляры, 4-вектор (1-вектор (1-3-скаляры ) , 2-ссылка ) , 5-ссылка, 6-вектор ( 1-4-скаляры ) )

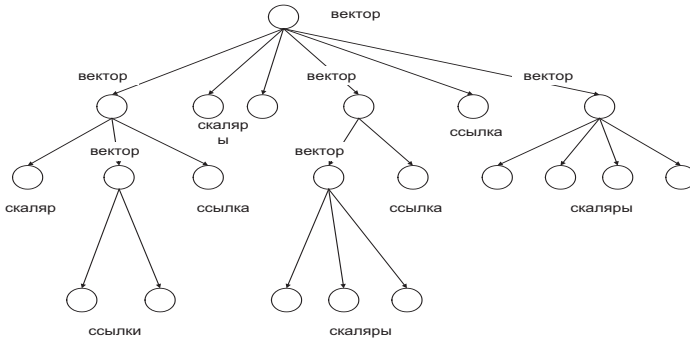


Рисунок 2.2. Структура данных - вектор

*Дополнительное соглашение*

В случае простого описания вектора в виде: *вектор (1-n - скаляры)* будем считать возможным указать имена скаляров в виде списка.

Например, вместо указанной записи написать:

*вектор (A,B,C,D,E... - скаляры)*

Пространство есть обычно воспринимаемый в моделях данных многомерный массив из однородных элементов.

**Пример.** N-мерное пространство скаляров, где N=5, может быть описано, как: *пространство (1-10, 1-3, 1-5, 1-2, 1-4 - скаляры)*

Такое пространство содержит  $10 \times 3 \times 5 \times 2 \times 4 = 1200$  скаляров.

Обращение к конкретному элементу вектора или пространства осуществляется заданием его координат в соответствии с описанием.

Описание блока близко к описанию класса.

Синтаксис:

*блок - <тип блока> <имя блока>*

*описание*

*<список групп описаний>*

*все описание;*

*алгоритм*

*<программа>*

*все алгоритм;*

*все блок.*

*< тип блока > ::= агрегат | процессор | контроллер | параметры*

Список групп описаний содержит группы описания параметров с указанием: внешние, внутренние, типы, внешние блоки, метки, процедуры, функции, а также необходимые комментарии и договоренности. При описании типов внешних переменных в блоке допустимо после указания типа добавить имя блока, которому они принадлежат. Например: *A,B,C-скаляры блока ЦЕНТР*.

<программа> - последовательность операторов ПОСП.

## 2.4. Операторы ПОСП

### Операции над параметрами

Над параметрами могут выполняться любые арифметические и логические операции.

*Арифметическое (логическое) выражение* есть последовательность арифметических (логических) операций над параметрами, имеющая целью получить некоторое конкретное числовое (логическое) значение.

Полученное значение присваивается переменной с помощью оператора присваивания:

<список переменных> := <выражение>;

### Безусловный навигационный оператор

Синтаксис:

*направить инициатор на <метка> [блок <имя блока>];*

Последняя часть может быть опущена, если эта операция происходит в одном и том же блоке.

### Условный навигационный оператор

Синтаксис:

если <логическое выражение> *то направить инициатор на <метка> [иначе на <метка> ];*

Последняя часть может быть опущена, если инициатор продолжает движение по программе.

### Векторная форма условного навигационного оператора

Синтаксис:

*если V1 направить инициатор на <метка>*

:

*VN направить инициатор на <метка>*

*[иначе направить на <метка>];*

где V1... VN - логические выражения.

### Оператор задержки инициатора (оператор условия продвижения инициатора)

Синтаксис:

*ждать <логическое выражение>;*

Оператор задерживает инициатор до выполнения логического условия. Условие может содержать и переменную *ВРЕМЯ*. Таким образом, этот оператор выполняет функции условного элементарного оператора, совмещая временной, логический и смешанный виды.

#### Векторная форма оператора задержки инициатора

Синтаксис:

*ждать*  $V_1$  *направить инициатор на* <метка>

:

$V_N$  *направить инициатор на* <метка>;

Здесь  $V_1...V_N$  - условные выражения.

Этот оператор совмещает функции двух последовательных операторов: оператора условия продвижения инициатора и векторного навигационного оператора. Его использование позволяет сократить запись в достаточно типичных конструкциях.

#### Оператор активизации инициатора

Синтаксис:

*активизировать инициатор из* <имя простой переменной типа ссылки> *в блок* < имя блока> *на метку* <метка>;

Оператор активизирует параметр типа ссылки, превращая его в инициатор и направляя на сцепление с помеченным оператором указанного блока.

#### Оператор пассивизации инициатора

Синтаксис:

*пассивизировать инициатор в параметр* <имя параметра>;

Этот оператор выполняет действие, обратное предыдущему оператору, лишая текущий инициатор свойства инициализации и направляя оставшуюся от него ссылку на локальную среду в указанный параметр ссылочного типа.

#### Оператор создания объекта

Синтаксис:

*создать* <имя объекта> *типа* <тип>;

Оператор создает объект указанного типа и вводит его в программу.

#### Оператор уничтожения объекта

*уничтожить* <тип объекта> <имя объекта>;

Оператор уничтожает объект с заданным именем и выводит его из программы.

#### Оператор присваивания значения ссылке.

Синтаксис:

<имя ссылки> := *ссылка на* [тип объекта] <имя объекта>

Ссылке присваивается адрес объекта.

#### Оператор разыменования ссылки.

Синтаксис:

<имя ссылки> → <тип переменной>[<местонахождение>]

Операция позволяет по ссылке определять значение переменной указанного типа. *Значение переменной* есть результат операции. Местонахождение необходимо указывать лишь для переменных типа *вектор* и *пространство*. Если операция встречается в выражении, то вышеуказанная запись заключается в круглые скобки.

#### Примеры

1. Вызов значения простой переменной типа скаляр, на которую ссылается ссылка S, имеет вид:

$S \rightarrow \text{скаляр}$

2. Вызов значения 5-го элемента вектора, на который ссылается ссылка S, имеет вид:

$S \rightarrow \text{вектор}(5)$

3. Вызов значения 3-го элемента локальной среды текущего процесса имеет вид:

*ИНИЦИАТОР*  $\rightarrow$  *вектор*(3)

Напомним, что значение инициатора есть ссылка на локальную среду процесса.

#### Оператор обращения к процедуре.

В тексте программы ПОСП можно использовать обращение к любой заранее составленной процедуре. Эта процедура может быть описана на каком - либо алгоритмическом языке или сформулирована содержательно.

Обращение к процедуре:

**процедура** <имя процедуры> (параметры процедуры);

Например, запись скалярного параметра в однонаправленный список:

**процедура** записать &B в список &A;

где &B, &A - макропеременные, подлежащие замене на фактические параметры при обращении. В данном примере предполагается: &A - голова списка, &B - скаляр.

Аналогично может быть использована процедура считывания скаляра из списка:

**процедура** считать из списка &A в &B;

#### Определение комментария.

Комментарии могут вводиться в любом месте программы и отделяться от операторов двойным дефисом ( -- ).

## 2.5. Программа блока ГЕНЕРАТОР

В качестве примера программы на ПОСП приведем описание процесса генерации инициаторов с помощью блока-контроллера ГЕНЕРАТОР. Блок генерирует инициаторы во внешний блок-процессор ОБРАБОТКА на начальную метку этого блока ВХОД. Каждый инициатор имеет локальную среду, содержащую два скаляра: первый скаляр равен времени генерации инициатора, а второй - число 1 или 2 с равной вероятностью. Скаляры для удобства объединены в вектор. Интервал времени между соседними сгенерированными инициаторами, поступающими в блок ОБРАБОТКА, представляет собой случайную величину, равномерно распределенную на интервале [20, 50]. Блок ГЕНЕРАТОР по определению агрегата содержит собственный

инициатор, развивающий процесс в блоке. Таким образом, блок ГЕНЕРАТОР обеспечивает генерацию потока инициаторов с заданными значениями локальных сред на процессор ОБРАБОТКА.

блок – контроллер ГЕНЕРАТОР;

*описание*

ТГЕН - скаляр; -- начальное значение равно 0;

ОБРАБОТКА - внешний блок;

ВХОД - метка блока ОБРАБОТКА;

-- начальное положение собственного инициатора на метке НАЧ;

*все описание;*

*алгоритм*

НАЧ: *ждать* ВРЕМЯ = ТГЕН;

*создать S типа ссылка;*

*создать W типа вектор (1-2-скаляры);*

$W(1) := ВРЕМЯ;$

$W(2) := ЦЕЛОЕ (RAND+0.5)+1;$

$S := ссылка на W;$

активизировать инициатор из S в блок ОБРАБОТКА на метку ВХОД;

$ТГЕН := ВРЕМЯ+RAND*30+20;$

*направить инициатор на НАЧ;*

*все алгоритм;*

*все блок;*

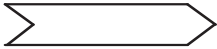
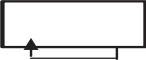



Поскольку блок-контроллер, выполняющий функции генерации инициаторов вовне, является типичной составной частью блочных схем, то для его обозначения введем специальный рисунок - символ, приведенный в таблице обозначений блоков (таблица 2.1, строка 4).

## 2.6. Описание блочных структур

Прежде, чем приступить к составлению алгоритмов на ПОСП, необходимо построить схему блоков, с помощью которых предполагается описать процесс функционирования системы. Как было показано ранее, мы различаем 3 вида блоков, обозначения которых приведены в таблице 2.1 (строки 1,2,3).

В таблице 1 указаны также обозначения для блока агрегатного типа, выполняющего функции генератора инициаторов (строка 4), и параметра общего вида (строка 5)

Таблица 2.1. Схематические обозначения блоков

1		блок - процессор
2		блок - агрегат
3		блок - контроллер
4		блок - генератор инициаторов
5		параметр, имя параметра

Символ блока - контроллера (строка 3) включает в свое обозначение два параметра: входной и выходной, непосредственно связанные с блоком типа агрегат. Такое обозначение отражает основную функцию контроллера - управление внешними процессами посредством приема внешних инициаторов, их пассивизацию, обработку алгоритмом контроллера и последующую активизацию во внешний блок.

Предложенную схему поясняет рисунок 2.3.



Рисунок 2.3. Символ контроллера

В качестве примера приведем блочную схему описания процессов в системе обработки и передачи данных, включающих два канала передачи потоков данных от двух существенно разных источников для последующей обработки переданных данных

по некоторому общему алгоритму. При этом один процессор может периодически выходить из строя, порождая искажение пакетов при передаче данных.

Блочная схема приведена на рисунке 2.4.

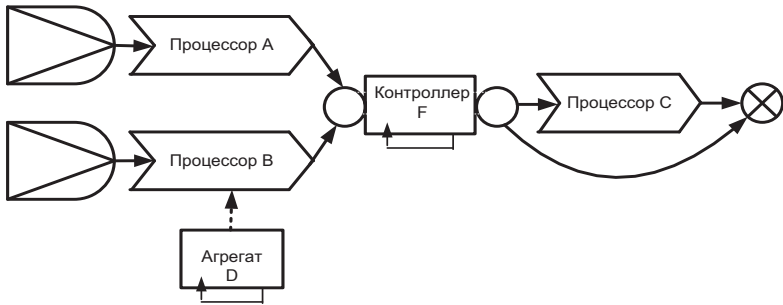


Рисунок 2.4. Блочная схема систем







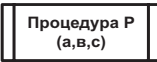



#### Подпроцессный граф

При составлении алгоритма отдельного блока удобно предварительно построить подпроцессный граф этого блока. В этом графе каждой вершине соответствует некоторый функционально - завершённый подпроцесс, а дуге - путь продвижения инициатора. Назовем такой граф *подпроцессным* (ПГ). Степень подробности описания может быть произвольной. Если описание выполняется на уровне содержательного описания совокупности обобщенных операторов, то будем называть такую совокупность подпроцессом и обозначать ее прямоугольником с закругленными углами (таблица 2.2, строка 6). Условие продвижения инициатора может быть указано как в вершине, так и на дуге. Если же мы хотим задать этот граф на уровне элементарных операторов, то операторы состояний будем помещать в прямоугольники (строка 3), операторы условия продвижения инициатора (*ждать*) – в вытянутые шестиугольники (строка 2), навигационные операторы – в треугольники (строка 5) или ромбы (строка 4). При использовании векторных форм условных и навигационных операторов условия указываются на дугах. В случае необходимости внесения каких-либо пояснений возможно выделение части графа пунктирным прямоугольником с указанием соответствующего комментария. При необходимости включения в трек какой-либо процедуры, описанной где-то в тексте, ее название и параметры помещаются в прямоугольник с двойными вертикальными рамками (строка 7). Оператор уничтожения инициаторов обозначается, как (строка 8), направление движения инициатора в виде сплошной стрелки (строка 9), параметр общего вида в виде кружка (строка 1), а его связь с оператором в виде пунктирной стрелки (строка 10).



Для агрегатов и контроллеров, имеющих по определению один инициатор, будем считать, что этот инициатор априорно помещен в блок и в начальном состоянии находится в вершине, помеченной стрелкой с символом **I**.

Таблица 2.2. Схематические обозначения операторов

1		параметр, имя параметра
2		ожидание условия логического или временного
3		оператор ПОСП
1		навигационный оператор если..то..иначе
5		навигационный оператор
6		подпроцесс
7		процедура, имя, параметры
8		уничтожение инициатора
9		направление движения инициатора
10		связь параметра с оператором

## 2.7. Примеры программ блоков

Пример 1. Генератор инициаторов общего вида.

Необходимо создать блок, генерирующий инициаторы и отправляющий их в блок с именем ОБРАБОТКА на оператор с меткой ВХОД. Первый инициатор покидает блок в момент времени  $T_0$ . Далее значения времени между поступлениями инициаторов определяются функцией  $F(\cdot)$ . Всего необходимо сгенерировать  $K$  инициаторов. Инициаторы обладают локальной средой, организованной в виде вектора, содержащего  $N$  скалярных параметров.

Подпроцессный граф блока приведен на рисунке 2.5.

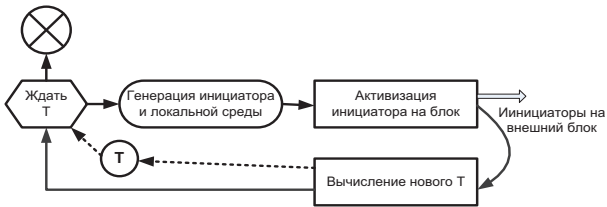


Рисунок 2.5. Подпроцессный граф блока-агрегата ГЕНЕРАТОР

Программа блока на ПОСП:

блок-агрегат ГЕНЕРАТОР;

описание

ТГЕН - скаляры;

ОБРАБОТКА - внешний блок;

ВХОД - метка блока ОБРАБОТКА;

С - скаляр; --начальное значение = 0;

$T_0, K, N, F(\cdot)$  - внешние параметры;

-- значения  $T_0, K, N, F(\cdot)$  - задаются извне;

--начальное положение инициатора на метке НАЧ;

все описание;

алгоритм

ждать  $ВРЕМЯ = T_0$  -- задержка начала генерации;

--генерация инициатора и локальной среды;

НАЧ: создать  $S$  типа ссылка;

создать  $W$  типа вектор (1-N - скаляры);

$S :=$  ссылка на  $W$ ;

-- активизация инициатора во-вне;

активизировать инициатор из  $S$  в блок ОБРАБОТКА на метку ВХОД;

$C := C+1$ ;

```

-- проверка на прекращение генерации;
если С = К то направить инициатор на КОН;
ТГЕН := ВРЕМЯ + F(.);
ждать ВРЕМЯ = ТГЕН;
направить инициатор на НАЧ;
КОН: уничтожить инициатор;
все алгоритм;
все блок;

```

Необходимо иметь в виду, что подобный алгоритм генератора предполагает, что локальную среду вновь сгенерированных инициаторов, посланных генератором на некоторый процессор, должен заполнить первый же оператор этого процессора. На рисунке 2.6. показана эта схема, где оператор А должен выполнить заполнение локальной среды:

```

инициатор → вектор(1) := ...
инициатор → вектор(2) := ...
...
инициатор → вектор(n) := ...

```

Остальные операторы являются собственно функциональными операторами процессора.

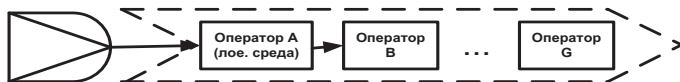


Рисунок 2.6. Требования к заполнению локальной среды

### Пример 2. Одноканальная система передачи данных

Рассмотрим систему передачи пакетов через канал. Поток пакетов поступает на канал передачи данных, который пропускает пакеты по одному, затрачивая  $P(\dots)$  единиц времени на каждую передачу. Блочная схема приведена на рисунке 2.7. Она включает два блока: генератор и процессор.

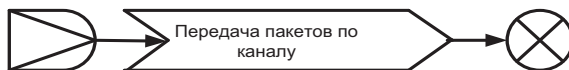


Рисунок 2.7. Блочная схема одноканальной системы

Генератор построен по общей схеме, приведенной на рисунке 2.5.

Вариант подпроцессного графа процессора приведен на рисунке 2.8. В этом варианте элементы графа представлены подпроцессами.

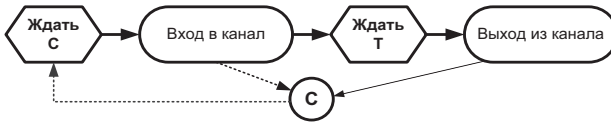


Рисунок 2.8. Подпроцессный граф процессора (вариант 1)

На рисунке 2.9 приведен вариант подпроцессного графа, элементы которого представлены операторами.

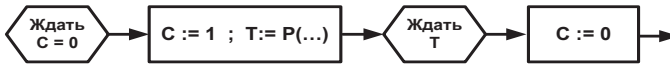


Рисунок 2.9. Подпроцессный граф процессора (вариант 2)

Ниже приведена программа блока процессора на ПОСП.

блок процессор КАНАЛ;

описание

С, Т - скаляры;

все описание;

алгоритм

ВХОД: ждать С=0;

С := 1;

Т := ВРЕМЯ + P(...);

ждать ВРЕМЯ = Т;

С := 0;

уничтожить инициатор;

все алгоритм;

все блок;

Пример 3. Передача пакетов по каналу, подверженному отказам

Система осуществляет передачу пакетов через канал от генератора пакетов. Канал пропускает пакеты по одному, затрачивая P(...) единиц времени на каждую передачу. Канал периодически отказывает и прекращает передачу пакетов. После восстановления отказа передача пакетов по каналу продолжается. Время безотказной работы канала равно Трб, время на устранение отказа равно Трм.

Блочная схема системы приведена на рисунке 2.10. Она включает два блока: генератор и процессор.

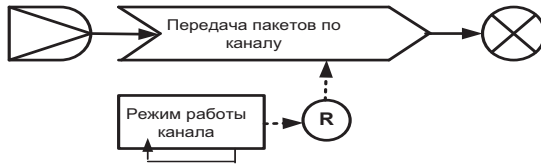


Рисунок 2.10. Блочная схема

Процессор описывает процессы передачи пакетов по каналу, агрегат описывает процесс изменения состояния канала. Индекс состояния канала отражен в параметре R: значение  $R = 1$  означает рабочее состояние канала, значение  $R = 0$  означает состояние канала в ремонте.

Подпроцессный граф процессора приведен на рисунке 2.11.

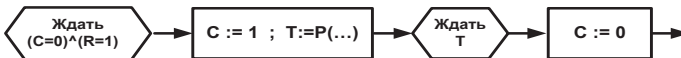


Рисунок 2.11. Подпроцессный граф процессора

Ниже приведена программа блока процессора на ПОСП.

блок процессор КАНАЛСОТК

описание

$C, T$  - скаляры;

$R$  - скаляр, внешний;

все описание;

алгоритм

ВХОД: *ждать*  $(C=0)^(R=1)$  ;

$C := 1$ ;

$T := \text{ВРЕМЯ} + P(\dots)$ ;

*ждать*  $\text{ВРЕМЯ} = T$ ;

$C := 0$ ;

уничтожить инициатор;

все алгоритм;

все блок;

Подпроцессный граф агрегата приведен на рисунке 2.12.

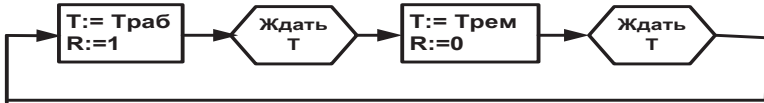


Рисунок 2.12. Подпроцессный граф агрегата

Ниже приведено описание процесса в агрегате на ПОСП

блок агрегат РЕЖИМ;

описание

R, T - *скаляры*;

Траб, Трем - *скаляры*;

все описание;

алгоритм

НАЧ: R:= 1;

T := *ВРЕМЯ* + Траб;

ждать *ВРЕМЯ* = T;

R:= 0;

T := *ВРЕМЯ* + Трем;

ждать *ВРЕМЯ* = T;

направить инициатор на НАЧ;

уничтожить инициатор;

все алгоритм;

все блок;

## 2.8. Макросы и макрорасширения

Предлагаемая лингвистическая система может включать любые расширения и макросы. Поскольку система имеет характер псевдоязыка, то описание макроса может быть выполнено достаточно свободно. Однако, желательно в таком описании придерживаться синтаксиса ПОСП.

Принцип построения макросов не ограничен. Например, в основу построения макросов может быть положен ключевой принцип. Причем ключи могут быть оформлены в виде служебных слов.

Служебные слова макроса будем выделять ***жирным шрифтом***. Макропеременные, определенные внутри макроса, префиксируются символом '&'. Макропеременные, которые заменяются при обращении к макросу на фактические значения, также префиксируются символом '&'. Кроме того, вполне допустимо словесное описание макроса, которое, при необходимости, может быть оформлено средствами ПОСП.

### Пример 1.

Макрос "Задержка инициатора на время, распределенное по экспоненциальному закону":

**задержать инициатор экспоненциально с параметром &A**

Его расширение имеет вид:

алгоритм

&V:=ВРЕМЯ-&A\*LN(RAND);

ждать ВРЕМЯ=&B;

все алгоритм;

### Пример 2.

Макрос генерации нового процесса:

**образовать процесс в блоке &B (метка &M) с локальной средой (&N);**

Расширение:

алгоритм

создать &S типа ссылки;

создать &W типа вектор (1-&N - скаляры);

&S := ссылка на &W;

активизировать инициатор из &S на метку &M блока &B;

все алгоритм;

## **2.9. Модель вычислительного комплекса АСУТП**

Покажем использование макрооператоров на примере описания функционирования вычислительного комплекса автоматизированной системы управления технологическими процессами (АСУТП) компрессорной станции.

Структура вычислительного комплекса АСУТП приведена на рисунке 2.13.

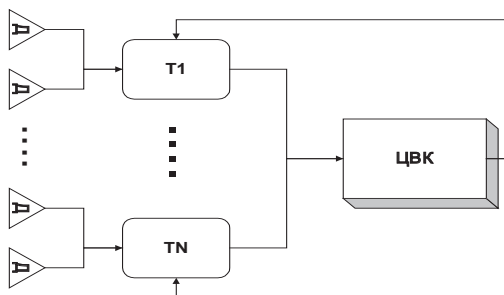


Рисунок 2.13. Структура вычислительного комплекса

Моделируемый комплекс включает:

- центральный вычислительный комплекс (ЦВК), обеспечивающий прием и обработку информации от терминальных устройств;
- терминальные устройства (терминалы), поставляющие технологическую информацию от различных компонент компрессорной станции;
- оконечные устройства;
- каналы связи.

ЦВК решает прикладные задачи по каждому терминалу, необходимые для обеспечения управления технологическими процессами.

Каждый терминал связан с оконечными устройствами, представляющими собой либо датчики, измеряющий технологические параметры элементов компрессорной станции, либо исполнительные механизмы. Для каждого оконечного устройства терминал генерирует запрос к ЦВК для выполнения некоторой обрабатывающей процедуры. Получив ответ, терминал, спустя некоторое время, генерирует новый запрос к ЦВК для следующего оконечного устройства и т.д. ЦВК работает в режиме разделения времени между запросами от терминалов, обрабатывая каждый запрос в течение времени не более, чем заданный временной квант. ЦВК обрабатывает одновременно не более наперед заданного количества запросов, остальные запросы выстраиваются в очередь к ЦВК.

Свернем процессы взаимодействия терминала с оконечными устройствами в один процесс терминала, представляющий собой циклические обращения терминала к ЦВК (рисунок 2.14). Таким образом, в системе протекает  $N$  параллельных процессов с локальной средой, оформленной в виде вектора и содержащей:

время появления нового запроса от терминала к серверу - ТР;

рабочий параметр для сохранения времени задержки инициатора в различных процессах.

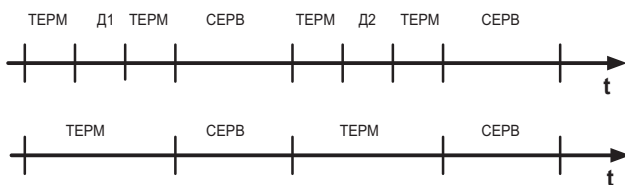


Рисунок 2.14. Преобразование терминальных процессов

Модель обработки терминальных запросов приведена на рисунке 2.15.



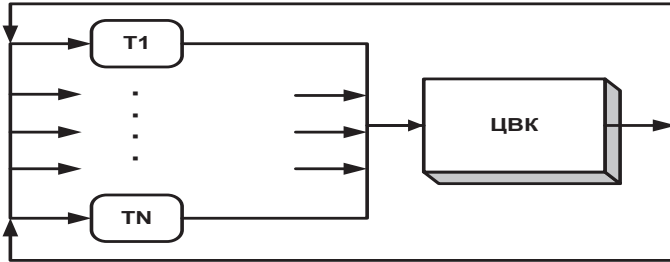


Рисунок 2.15. Модель обработки терминальных запросов

В модели будем использовать определенные ранее макросы. В описании функционирования ЦВК определим локальную среду каждого инициатора, как:

$\&W$  – *локальная среда* ТР, ТЗ, Тобраб;

- где параметр ТР сохраняет момент появления запроса от терминала,

параметр ТЗ – интервал времени задержки между получением ответа от ЦВК и генерацией нового запроса от терминала,

параметр Тобраб – время, необходимое ЦВК для обработки данного запроса.

Параметр КВАНТ равен шагу квантования ЦВК. Параметр  $\Delta TП$  равен затратам времени ЦВК на переключение после окончания времени квантования.

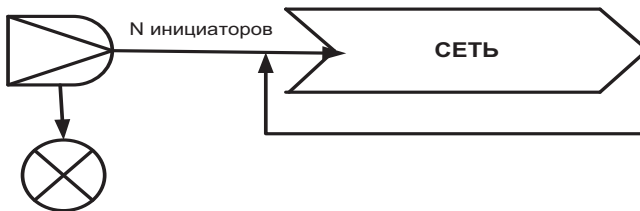


Рисунок 2.16. Блочная схема АСУТП

Представим описание в виде двух блоков: контроллера ГЕНЕРАТОР и процессора СЕТЬ. ГЕНЕРАТОР формирует  $N$  инициаторов в блок СЕТЬ, запуская таким образом  $N$  терминальных процессов (рисунок 2.16).

Генератор инициирует  $N$  замкнутых процессов в процессоре СЕТЬ (рисунок 2.15). После их инициации генератор прекращает работу, его инициатор уничтожается. Блок контроллер ГЕНЕРАТОР имеет подпроцессный граф, представленный на рисунке 2.7.

Алгоритм блока ГЕНЕРАТОР на ПОСП.

блок-агрегат ГЕНЕРАТОР;

описание

$M$  – *скаляр*; -- начальное значение равно 0;  
 $N$  – *скаляр*; -- начальное значение задается в исходных данных;  
 -- инициатор на метке НАЧ;  
 все описание;  
 алгоритм  
 НАЧ :  $M := M+1$ ; -- счетчик;  
 если  $M > N$  то направить инициатор на метку КОНЕЦ;  
 образовать процесс в блоке СЕТЬ (метка НАЧАЛО) с локальной средой (3);  
 направить инициатор на НАЧ;  
 КОНЕЦ : уничтожить ИНИЦИАТОР;  
 все алгоритм;  
 все блок;

ПГ блока процессора СЕТЬ приведен на рисунке 2.17. В блок на метку НАЧАЛО в начале моделирования введено  $N$  инициаторов из блока ГЕНЕРАТОР. Каждый инициатор имеет локальную среду, структура которой описана выше. В ходе функционирования инициаторы не покидают блок, реализуя замкнутый цикл функционирования системы.

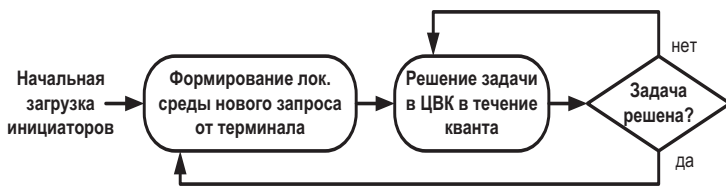


Рисунок 2.17. ПГ блока СЕТЬ

Описание блока на ПОСП:

блок-процессор СЕТЬ;

описание

ТЦВК, Тобр, SERV, Тдообр, Тперекл, ДТП, КВАНТ - *скаляры*;

-- ДТП, КВАНТ - исходные данные;

-- Фтерм(...), Фобраб(...) – функции времени задержки в терминалах и ЦВК;

-- инициаторы имеют локальную среду, описанную выше.

все описание;

алгоритм

-- формирование нового запроса от терминала;

НАЧАЛО: **локальная среда** ТЗ := Фтерм(...) + ВРЕМЯ;

ждать ВРЕМЯ= локальная среда ТЗ;

локальная среда ТР:=ВРЕМЯ;  
**локальная среда** Тобраб := Фобраб(...);

ПРОДОЛЖ:      *ждать* SERV ='свободен'; --очередь на вход в ЦВК;  
                   SERV :='занят';  
                   Тобраб:= **локальная среда** Тобраб;  
                   если КВАНТ $\geq$  Тобраб то направить инициатор на М1;  
                   ТЦВК := КВАНТ+ВРЕМЯ;  
                   Тдообр := Тобраб-КВАНТ;  
                   направить инициатор на М2;

М1:              ТЦВК := Тобраб +ВРЕМЯ;  
                   Тдообр :=0;

М2:              *ждать* ВРЕМЯ = ТЦВК;  
                   локальная среда Тобраб := Тдообр;  
                   Тперекл := ВРЕМЯ+ $\Delta$ ТП;  
                   *ждать* ВРЕМЯ = Тперекл;  
                   SERV := 'свободен';  
                   если Тдообр>0 то направить инициатор на ПРОДОЛЖ;  
                   направить инициатор на НАЧАЛО;

все алгоритм;  
 все блок;

## 2.10. Задания для самостоятельной проработки

1. Поток задач заданной интенсивности поступает на сервер, сервер тратит на решение одной задачи заданное время, сервер периодически переключается на самотестирование. Использовать блок ГЕНЕРАТОР и агрегат СЕРВЕР.

2. Поток пакетов заданной интенсивности поступает на канал, канал тратит на передачу одного пакета заданное время, на канале периодически возникают сбои, задано время их устранения. Использовать блок ГЕНЕРАТОР и агрегат КАНАЛ.

3. Система включает CPU и 2 дисковода, в системе одновременно не может находиться более 3-х задач. Задача решается в CPU и затем поступает на тот дисковод, который в данный момент не занят. Каждая задача проходит таких 4 цикла и затем покидает систему. Все времена заданы. Использовать блок ГЕНЕРАТОР и процессор CPU.

4. ЛВС включает 20 рабочих станций и один сервер, работающие в режиме диалога (вопрос-ответ-вопрос), сервер обрабатывает запрос и отправляет на рабочую станцию ответ, на сервере происходят отказы, на их устранение выделяется время. Все времена заданы. Использовать процессор ЛВС и агрегат РЕЖИМ\_СЕРВЕРА.

5. Поток задач заданной интенсивности поступает на сервер, сервер тратит на решение одной задачи заданное время. После окончания решения задача с некоторой

заданной вероятностью снова поступает на сервер в порядке общей очереди. Сервер периодически переключается на самотестирование. Определить процессы, блоки для этих процессов и дать ПГС каждого блока.

6. На коммутатор поступает поток пакетов 2-х типов. Пакет рассматривать, как неделимую информацию. Каждый тип пакетов поступает в свой буферный накопитель. Коммутатор поочередно подключается к каждому из буферов на 1 единицу времени. За одно подключение передается в канал только один пакет. Если подключенный буфер пуст, то происходит переключение коммутатора к следующему буферу. Заданы: интенсивности потоков, время на переключение коммутатора, время на передачу пакета из буфера в канал. Определить процессы, блоки для этих процессов и дать ПГС каждого блока.

7. Поток пакетов заданной интенсивности поступает на один из двух каналов: основной и запасной. Канал тратит на передачу одного пакета заданное время. Основной канал периодически ломается, задано распределение времени его работоспособного состояния и распределение времени его восстановления. Пакеты, попавшие на нерабочий интервал основного канала, поступают на запасной канал. После восстановления основного канала пакеты снова поступают на него. Определить процессы, блоки для этих процессов и дать ПГС каждого блока.

8. На 4-канальную систему поступает пуассоновский поток пакетов с интенсивностью  $L$ . Если очередь к системе меньше 3, то работает один канал. Если очередь доходит до 6, то работают 2 канала. Если очередь доходит до 10, то работают 3 канала. Если очередь превышает 10 пакетов, то работают все 4 канала. По мере уменьшения очереди каналы отключаются. Время передачи одного пакета в любом из каналов равно  $M$ . На подключение каждого канала требуется  $N$  единиц времени. Величины  $M$  и  $N$  – случайные. Определить процессы, блоки для этих процессов.

9. На складе хранятся комплектующие изделия 3-х типов. Потоки новых поступлений распределены по пуассоновскому закону с интенсивностями  $L_1, L_2, L_3$ . Из цеха периодически через время  $T$  поступает заявка на эти изделия в объеме  $Q_1, Q_2, Q_3$  соответственно. По этой заявке формируется в течение времени  $G$  заказ и отсылается в цех. Если не хватает нужного количества изделий, то выполнение заказа задерживается до появления этих изделий на складе. Величины  $T, Q_1, Q_2, Q_3$  – случайные. Определить процессы, блоки для этих процессов.

10. На коммутатор поступают три потока пакетов, коммутатор передает их в канал, используя способ временной синхронизации: в моменты времени 1, 4, 7, ... передается в канал 1-й поток, в моменты времени 2, 5, 8, ... передается в канал 2-й поток, в моменты времени 3, 6, 9, ... передается в канал 3-й поток. В один момент времени может быть передан только один пакет. Коммутатор имеет отдельный буфер для каждого потока. Использовать блок ГЕНЕРАТОР и агрегат КОММУТАТОР. Предварительно создать агрегат, выдающий сигналы  $C_1, C_2, C_3$  в соответствующие моменты времени для управления потоками.

11. В двух отдельных замкнутых ареалах обитают волки и зайцы. Объем популяции каждого вида задан, как исходные данные. Существует еще один ареал "Водопой", общий для всех. Цикл жизни и волков, и зайцев одинаков: пребывание в своем ареале в течение некоторого времени, а затем выход в ареал "Водопой" и пребывание в нем заданное количество времени. После чего происходит возврат в свой ареал. Если в ареале "Водопой" одновременно находятся волк и заяц, то с заданной вероятностью волк съедает зайца, после чего покидает общую зону. Если какой-либо волк не ел в течение некоторого заданного времени, то он погибает. Если какой-либо заяц прожил более заданного времени, то он порождает еще одного зайца.

Задав все необходимые данные (временами, объемами и вероятностью), опишите на ПОСП функционирование описанной системы.

## ГЛАВА 3. ПОСТРОЕНИЕ ИМИТАЦИОННОГО ПРОЦЕССА

### 3.1. Понятие квазипараллельного процесса

В основе построения имитационной модели лежит понятие квазипараллельного процесса. Квазипараллельный процесс представляет собой один последовательный процесс, реализующий описание совокупности параллельных во времени процессов. Главное требование к квазипараллельному процессу - реализация всех событий без нарушения их причинно - следственных отношений, существующих в системе параллельных процессов.

Совокупность параллельных процессов будем называть исходной системой (или просто системой), для обозначения квазипараллельного процесса будем использовать сокращение КПП. Таким образом, первой задачей при построении КПП является отображение треков исходной системы на один последовательный процесс с целью построения трека КПП.

Пусть исходная система задана в виде двух процессов  $Z_1$  и  $Z_2$ , показанных на рисунке 3.1.

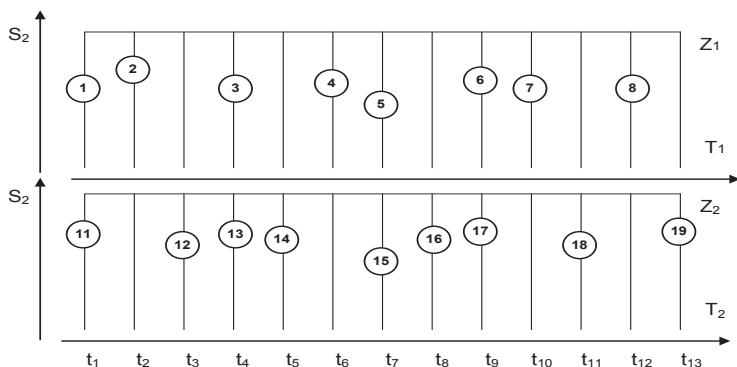


Рисунок 3.1. Пример исходной системы

На рисунке 3.1 показаны треки процессов, имена элементарных операторов указаны в кружочках. Показано также отношение этих операторов ко времени их реализации.

Существует множество вариантов отображения этих треков на один процесс. На рисунке 3.2 в качестве примера приведен один из возможных вариантов отображения исходной системы на КПП.

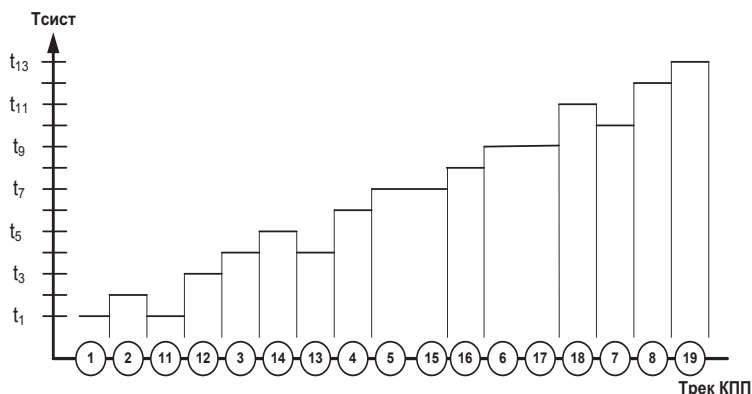


Рисунок 3.2. Вариант отображения исходной системы на КПП

На рисунке показан трек элементарных операторов в составе КПП. Однако, при этом возникает проблема построения обратного преобразования. С этой целью для каждого оператора трека КПП указывается время его реализации в исходной системе. Таким образом, при задании трека на КПП вводится новая координатная ось, отражающая время реализации соответствующих операторов в исходной системе. На рисунке видно, что при принятом варианте построения трека КПП, график времени на оси Тсист имеет произвольный вид.

### 3.2. Анализ отношения сцепленности операторов

Выше было указано, главным требованием к квазипараллельному процессу является реализация всех событий на КПП без нарушения их причинно - следственных отношений, существующих в исходной системе параллельных процессов.

Сформулируем утверждение: *причинно - следственные отношения однозначно определяются отношениями сцепленности операторов, изложенными в главе 1.*

Рассмотрим влияние отношения сцепления на последовательность выполнения операторов при моделировании. Пусть заданы два процесса  $Z_1$  и  $Z_2$ , условно изображенные на рисунке 3.3.

Элементарные операторы процесса  $Z_1$  пронумерованы от 1 до 13, а процесса  $Z_2$  - от 14 до 26. Рассмотрим четыре типовых случая:

а) моделируется один процесс  $Z_1$ , причем его операторы не сцеплены между собой;

б) моделируется процесс  $Z_1$  без ограничения на сцепленность элементарных операторов;

в) моделируются два процесса  $Z_1$  и  $Z_2$ , при этом эти процессы не сцеплены между собой;

г) моделируются два сцепленных процесса  $Z_1$  и  $Z_2$ .

**А. Моделирование процесса  $Z_1$  при несцепленных операторах  $h_i(\forall i)$ .**

Поскольку сцепление  $h_i \rightarrow h_{i+1}$  отсутствует для всех  $i$ , то последовательность вычислений элементарных операторов не имеет значения и операторы  $h_i(\forall i)$  могут вычисляться в любом порядке, т.е. допустим любой вариант отображения трека на КПП. Восстановление исходного трека производится с использованием дополнительной оси времени Тсист.

**Б. Моделирование процесса  $Z_1$  в общем случае.**

В общем случае возможен самый крайний вариант, когда все последовательные элементарные операторы одного процесса сцеплены между собой:

$$h_i \rightarrow h_{i+1}, \text{ для всех } i = \overline{1, n}.$$

Последовательность сцепленных операторов строго следует порядку  $\alpha$  на  $T$ . Следовательно, чтобы не нарушить отношение сцепления ни у одной пары операторов, необходимо производить вычисления операторов в КПП строго в этом же порядке. *Необходимо заметить, что график времени по оси Тсист будет в этом случае монотонно возрастающим.*

**В. Моделирование несцепленных между собой процессов  $Z_1$  и  $Z_2$ .**

Предполагаем, что каждый процесс  $Z_1$  и  $Z_2$  в отдельности представлен общим случаем (т.е. возможен вариант полного сцепления). Если  $h_i^1$  - операторы процесса  $Z_1$ , а  $h_j^2$  - операторы процесса  $Z_2$ , то по исходному предположению отсутствует сцепление



между  $h_i^1$  и  $h_j^2$  для всех  $i$  и  $j$ . Так же, как и в случае **A**, здесь последовательность вычислений элементарных операторов из разных процессов не имеет значения. Однако, поскольку все  $h_i^1$  сцеплены между собой, и все  $h_j^2$  также сцеплены между собой, важно, чтобы в этой последовательности выполнялся порядок  $\alpha_1$  для процесса  $Z_1$  и  $\alpha_2$  для процесса  $Z_2$ . В частности, возможен вариант вычисления сначала всех операторов процесса  $Z_1$ , а затем всех операторов процесса  $Z_2$ . Очевидно, что порядок  $\alpha_1$  и  $\alpha_2$  при этом сохраняется. Очевидно, что существует множество других вариантов построения трека на КПП при сохранении отношений  $\alpha_1$  и  $\alpha_2$ .

#### Г. Моделирование произвольно сцепленных между собой процессов $Z_1$ и $Z_2$ .

Предполагаем также, как и в случае **B**, что каждый процесс  $Z_1$  и  $Z_2$  в отдельности представлен общим случаем. Таким образом, в треке КПП обязательно должен быть выдержан порядок  $\alpha_1$  для  $Z_1$  и  $\alpha_2$  для  $Z_2$ . Кроме того, необходимо выполнить условия сцепленности для любой пары сцепленных операторов из разных процессов. Это означает, что если ( $h_i^1 \rightarrow h_j^2$ ), то первым должен выполняться оператор  $h_i^1$ , поскольку по определению отношения сцепления для вычисления оператора  $h_j^2$  необходимо знание состояния  $h_i^1$ .

Пример системы показан на рисунке 3.3. Стрелками указано отношение сцепленности между операторами разных процессов.

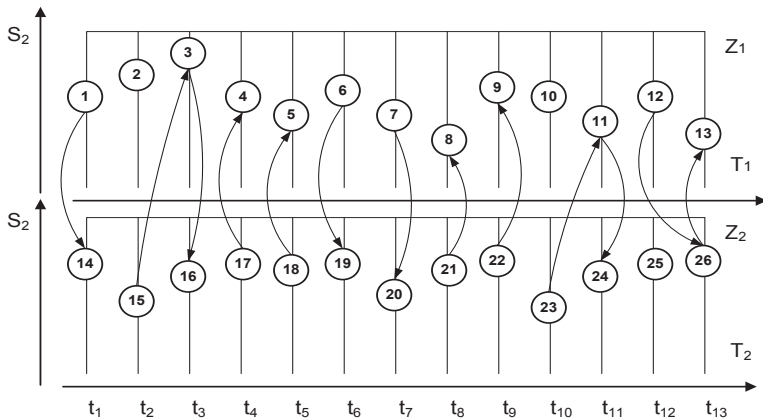


Рисунок 3.3. Пример сцепленных процессов

На приведенном графике видно, что из операторов  $h_1$  и  $h_{14}$  первым должен вычисляться оператор  $h_1$ , из операторов  $h_3$  и  $h_{16}$  первым должен вычисляться оператор  $h_3$ , из операторов  $h_4$  и  $h_{17}$  первым должен вычисляться оператор  $h_{17}$ , из операторов  $h_5$  и  $h_{18}$  первым должен вычисляться оператор  $h_{18}$  и т.д.

Получим возможный порядок вычислений:  $h_1, h_{14}, (h_2, h_{15}), h_3, h_{16}, h_{17}, h_4$  и т.д.

В скобках указаны пары операторов из разных процессов, для которых можно поменять порядок вычислений в силу отсутствия отношения сцепленности между операторами  $h_2$  и  $h_{15}$ .

Построение начального фрагмента КПП для примера системы процессов, приведенных на рисунке 3.3, показано на рисунке 3.4. Следует также подчеркнуть, что и в этом случае график времени по оси  $T_{сист}$  будет монотонно возрастающим.

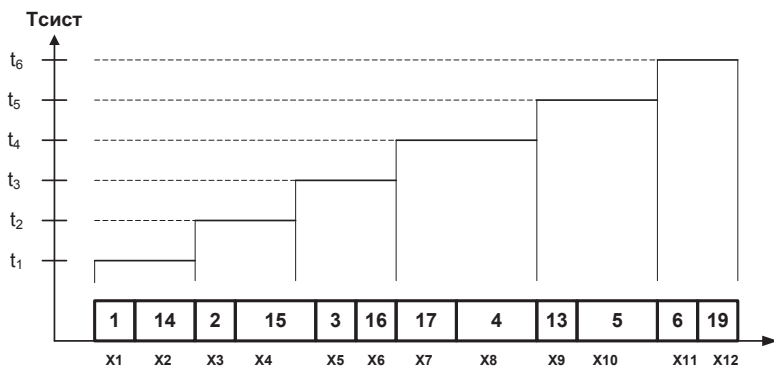


Рисунок 3.4. Фрагмент КПП для исходной системы

На основании проведенного анализа можно сделать следующие выводы:

1. Для каждого процесса в ходе вычисления операторов необходимо строго придерживаться порядка  $\alpha$  на  $T$ .

2. Выполнение п.1. обеспечивает реализацию сцепленности операторов  $h_i \rightarrow h_j$ , если  $t_i < t_j$ , в рамках реализации одного процесса.

3. При выполнении п.1 указание сцепленности операторов с разным значением  $t_i$  в рамках одного процесса не добавляет новых ограничений на порядок расчета и может быть опущено.

Действительно, если вычислены элементарные операторы  $h_{10}$  и  $h_{23}$  (см. рисунок 3.3.) в момент времени  $t_{10}$ , то указание  $h_{23} \rightarrow h_{11}$  не меняет порядок вычислений, поскольку к моменту времени  $t_{11}$  оператор  $h_{23}$  уже будет рассчитан. При этом условие сцепленности выполняется автоматически.

4. Для определения порядка расчета операторов, принадлежащих разным процессам, важно знание отношения сцепления для операторов разных процессов с одинаковым значением времени  $t_i$ .

Таким образом, можно сформулировать алгоритм определения порядка вычисления элементарных операторов в КПП для совокупности исходных параллельных процессов.

Пусть заданы треки процессов  $Z^i (i = \overline{1, n})$ :  $\langle \{h_j^i\}, \beta^i \rangle$  для всех  $i$ . Пусть текущее значение времени равно  $t$  и все элементарные операторы  $h^i$ , у которых  $t^i \leq t$ , вычислены. Для всех  $h_j^i$ , имеющих  $t^i = t$  и обладающих условным временным оператором  $(h_j^i)^{\neq}$ , определим для каждого очередной момент времени  $t_{j+1}^i$  сцепления инициатора по своему треку, задаваемый этим временным оператором. Получим множество  $\{t_{j+1}^i\}$ . Назовем его *активным временным множеством*.

Это множество содержит по одному значению от каждого процесса, остановленного на элементарном операторе, содержащем временное условие продвижения инициатора. Определим очередное значение времени по формуле:

$$t_0 = \min\{t_{j+1}^i\} \quad (2)$$

Последовательное применение формулы 2 строит упорядоченное множество  $T^{\text{кпп}}$ . Легко доказать, что множества  $T^{\text{кпп}}$  и  $(\bigcup_{\forall i} T^i)$  равны.

### 3.3. Классы одновременных событий

Рассмотрим последовательность выполнения элементарных операторов в КПП. Пусть исходная система содержит 9 объектов, в каждом из которых развивается процесс. На рисунке 3.5. приведен пример организации треков на некотором временном интервале. Пусть в некоторый момент времени система находилась в состоянии, показанном на рисунке слева. Назовем его начальным состоянием. Для каждого процесса  $Z_i$  указан текущий элементарный оператор в следующих обозначениях:

$$\langle h_n^{i,c}, h_n^{i,y} \rangle, \quad (3)$$

где  $i$  - номер процесса (он же- номер строки);

$n$  - порядковый номер элементарного оператора в своем треке;

$c$  - символ "состояние";

$y = l$  - символ логического условия продвижения инициатора;

$y = t$  - символ временного условия продвижения инициатора.

Предположим, что к этому моменту времени все элементарные операторы вычислены. Тогда активное временное множество на текущий момент равно  $\{t_{10}^1, t_{11}^3, t_{21}^6, t_{26}^9\}$ . Определим в соответствии с (2) очередной момент времени, как:

$$t_0 = \min \{t_{10}^1, t_{11}^3, t_{21}^6, t_{26}^9\} \quad (4)$$

В нашем примере  $t_0 = t_{10}^1$ . Таким образом, из начального состояния система переходит в новое состояние в момент времени  $t_{10}^1$ , соответствующее временному условию  $h_{10}^{1,c}$ .

В этот момент времени инициатор  $I_1$  процесса  $Z_1$  перемещается в элементарный оператор ( $h_{10}^{1,c}, h_{10}^{1,c}$ ), вычисляя новое состояние объекта  $O_1$ , а значит, и системы в целом. При этом оказывается выполненным условие  $h_{11}^{4,c}$  и  $h_{11}^{7,c}$ . В треке процесса  $Z_4$  выполняется оператор состояния  $h_{16}^{4,c}$  и устанавливается логическое условие  $h_{17}^{4,c}$ , а в треке процесса  $Z_7$  –  $h_{20}^{7,c}$  и временное условие  $h_{21}^{7,c}$ . На рисунке 3.5 соответствующие события отмечены светлыми кружками. Событие по временному условию показано на рисунке 3.5 в виде темного кружка. Так как больше условий не выполняется, то необходимо перейти к новым элементарным операторам в соответствии с треками.

Начальное состояние

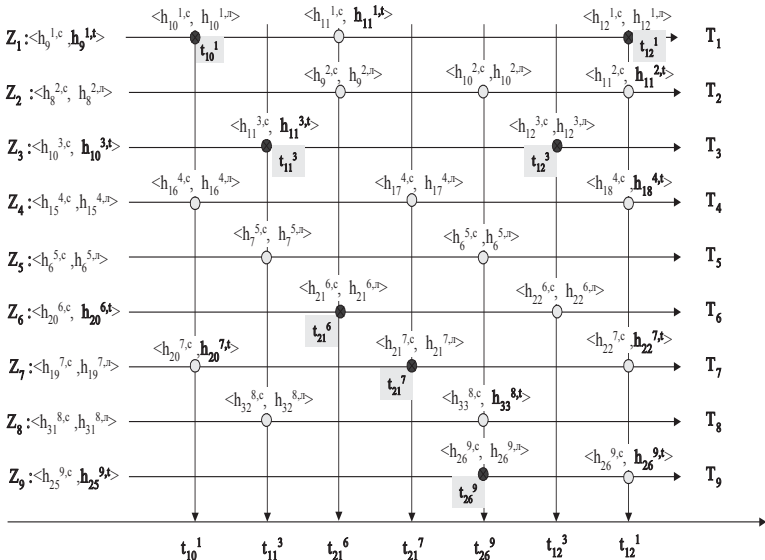


Рисунок 3.5. Пример треков процессов с системе

В рассматриваемый момент времени следующее активное временное множество имеет вид  $\{t_{11}^3, t_{21}^6, t_{21}^7, t_{26}^9\}$ . Согласно (3) необходимо перейти к новому моменту времени

$t_0 = \min\{t_{11}^3, t_{21}^6, t_{21}^7, t_{26}^9\}$ . В нашем случае  $t_0 = t_{11}^3$ . Первым выполняется оператор  $\langle h_{11}^{3,c}, h_{11}^{3,t} \rangle$ , который изменяет состояние объекта  $O_3$  и системы в целом, а также формирует новый заказ времени передвижения своего инициатора  $I_3$  в момент времени  $t_{12}^3$ . При этом выполняются логические условия для процессов  $Z_5$  и  $Z_8$ . Инициаторы  $I_5$  и  $I_8$  вызывают выполнение операторов  $\langle h^{5,c}_7, h^{5,t}_7 \rangle$  и  $\langle h^{8,c}_{32}, h^{8,t}_{32} \rangle$  соответственно. Формируется новое активное временное множество:  $\{t_{12}^3, t_{21}^6, t_{21}^7, t_{26}^9\}$ , наименьшим в котором является  $t_{21}^6$ . С оператора  $(h_{21}^{6,c}, h_{21}^{6,t})$  и начинается следующий цикл вычислений, и т.д.

На этом примере хорошо иллюстрируется алгоритм продвижения модельного времени и порядок выполнения элементарных операторов в каждом процессе. Сделаем некоторые обобщения.

Выполнение каждого элементарного оператора назовем *событием* в системе. *Событие активное, если оно следует в треке за элементарным оператором, содержащем  $h^t$* . На рисунке 3.5 условия, содержащие  $h^t$ , выделены жирным шрифтом. К активным событиям относятся выполнение  $\langle h_{10}^{1,c}, h_{10}^{1,t} \rangle$ ,  $\langle h_{11}^{3,c}, h_{11}^{3,t} \rangle$ ,  $\langle h_{21}^{6,c}, h_{21}^{6,t} \rangle$  и т.д.

*Событие пассивное, если оно следует в треке за элементарным оператором, содержащем  $h^c$* . На рисунке 3.5 к пассивным событиям относятся выполнение  $\langle h_{11}^{1,c}, h_{11}^{1,t} \rangle$ ,  $\langle h_9^{2,c}, h_9^{2,t} \rangle$ ,  $\langle h_{17}^{4,c}, h_{17}^{4,t} \rangle$  и т.д.

Множество событий, происходящих в один и тот же момент модельного времени, назовем *классом одновременных событий* (КОС). На рисунке 3.5 класс одновременных событий в момент времени  $t_{10}^1$  включает события для 1, 4 и 7-го процессов; в момент времени  $t_{11}^3$  КОС составляют события для 3, 5 и 8-го процессов и т.д.

Рассмотрение КОС примера на рисунке 3.5 показывает, что в каждом КОС содержится активное событие. Введем следующие допущения:

*Допущение 1.* В каждом КОС содержится одно и только одно активное событие.

Тогда: а) все остальные события в КОС, если они есть, являются пассивными;

б) количество КОС равно мощности объединенного множества времен  $T$ .

*Допущение 2.* Первым событием в любом КОС является активное событие.

Если это так, то оставшиеся пассивные события для определения последовательности своего выполнения требуют знания отношения сцепления. Представим отношение сцепления в КОС в виде направленного графа. На рисунке 3.6. приведен пример некоторого КОС с заданным на нем отношением сцепления.

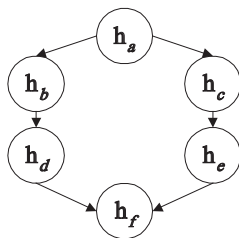


Рисунок 3.6. Пример класса одновременных событий

Здесь:  $h_a$  - активное событие;  $h_b, h_c, h_d, h_e, h_f$  - пассивные события. Из рисунка следует, что после  $h_a$  необходимо выполнить  $h_b$  либо  $h_c$ ; затем  $h_d$ , либо  $h_e$ ; и, наконец,  $h_f$ .

Задание отношения сцепления для каждого КОС является самостоятельной задачей. Однако можно предложить таким образом определять условие выполнения каждого пассивного события, чтобы оно содержало условие выполнения предыдущего события, т.е. содержало описание отношения сцепления с предыдущим событием.

Так, условие выполнения события  $h_d$  должно содержать выражение, проверяющее выполнение события  $h_b$ . Если это удастся сделать, то алгоритм формирования КОС выглядит следующим образом:

1. Выполняется активное событие;
2. Проверяются условия всех возможных в системе пассивных событий;
3. Если выполняется условие пассивного события, то оно вычисляется.

Алгоритм продолжается с п.2.

Важно найти признак, по которому можно определить завершенность КОС.

**Теорема А:**

*КОС завершен, если все условия, заданные операторами  $h^i$  во всех треках, равны 0.*

Поскольку КОС содержит одно активное событие и оно выполняется первым, то все остальные события пассивные. Пассивное событие по определению выполняется, если определяющее его условие равно 1. Однако, по предположению, все условия, заданные  $h^i$  равны 0. Таким образом, выполнение пассивного события невозможно, а единственное активное событие уже выполнено. Поскольку не выполняется ни один элементарный оператор, состояние системы и параметры условных операторов не могут быть изменены. Состояние системы зафиксировано и не будет изменено вплоть до нового КОС. Что и требовалось доказать.

**Теорема В:**

*Первым событием в КОС является активное событие.*

Доказывая теорему А, мы показали, что когда завершен КОС, то все условия в условных операторах системы равны 0, и состояние системы не может быть изменено

ни одним пассивным событием. Значит, следующий КОС может начаться только с активного события. Что и требовалось доказать.

Таким образом, наше допущение 2 о начальной функции активного события в КОС доказано строго в теореме В.

### 3.4. Моделирующий алгоритм сканирующего типа

Моделирующий алгоритм в любом случае, как это следует из выше изложенного, должен включать следующие составные части:

- подпрограммы событий, реализующие элементарные операторы;
- алгоритм формирования модельного времени;
- алгоритм выбора очередного КОС;
- алгоритм генерирования КОС.

*Подпрограмма события* представляет собой программную реализацию одного элементарного оператора, включающего оператор состояния, оператор условия продвижения инициатора и навигационный оператор. В этих подпрограммах, в общем случае, все параметры являются глобальными. В каждом же конкретном случае часть параметров может быть локализована. Однако все параметры, через которые осуществляется обмен, являются глобальными. Если подпрограмма события реализует объединенный элементарный оператор, то она должна иметь доступ к значению инициатора, определяющего локальную среду данного процесса.

В самом общем виде моделирующий алгоритм представлен на рисунке 3.7.

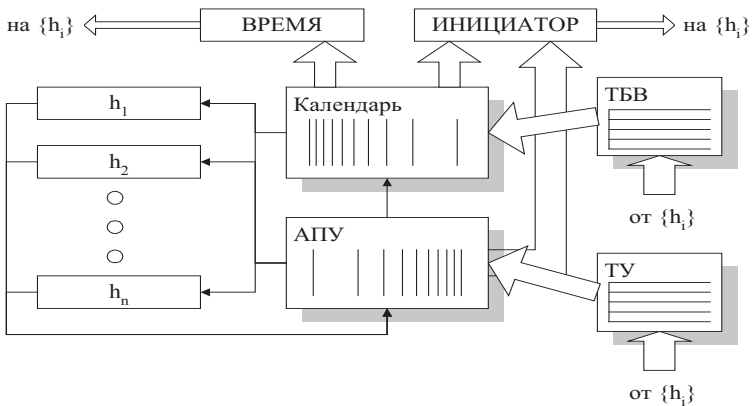


Рисунок 3.7. Моделирующий алгоритм сканирующего типа

Здесь  $(h_1, h_2, \dots, h_n)$  – неупорядоченная совокупность подпрограмм событий, реализующих элементарные операторы треков всех процессов в системе.

Параметр **ВРЕМЯ** - содержит текущее значение модельного времени.

Параметр **ИНИЦИАТОР** - содержит значение текущего инициатора (ссылку на локальную среду процесса).

**КАЛЕНДАРЬ** - алгоритм, реализующий монотонно возрастающее продвижение модельного времени и начало нового КОС в системе.

**АПУ** - Алгоритм Проверки Условий, обеспечивающий построение КОС для текущего значения модельного времени.

**ТБВ** - Таблица Будущих Времени, структура которой приведена на рисунке 3.8. Каждая строка ТБВ соответствует одному процессу и содержит следующие элементы описания будущего *активного* события:

столбец 1 - значение момента времени активизации инициатора, определяемого предшествующим оператором  $h^i$  ;

столбец 2 – инициатор процесса;

столбец 3 - имя подпрограммы активного события в треке данного процесса.

-1-	-2-	-3-
момент активизации	значение инициатора	имя активной подпрограммы
• • •	• • •	• • •

Рисунок 3.8. Структура таблицы ТБВ

Совокупность значений атрибута “момент активизации” таблицы ТБВ составляет в любой момент модельного времени активное временное множество.

**ТУ** - Таблица Условий, структура которой приведена на рисунке 3.9. Каждая строка ТУ соответствует одному процессу и содержит следующие элементы описания будущего *пассивного* события:

столбец 1 – логическое условие активизации инициатора, определяемое предшествующим оператором  $h^i$  ;

столбец 2 – инициатор процесса;

столбец 3 - адрес подпрограммы пассивного события в треке данного процесса.

-1-	-2-	-3-
условие логическое	значение инициатора	адрес очередной подпрограммы
• • •	• • •	• • •

Рисунок 3.9. Структура таблицы ТУ



КАЛЕНДАРЬ определяет *первое активное событие* в новом КОС в соответствии. Его алгоритм выглядит следующим образом:

1) Поиск минимального значения в столбце 1 ТБВ. Пусть это значение равно  $t_k$ , где  $k$  - номер строки ТБВ.

2) **ВРЕМЯ** :=  $t_k$

3) **ИНИЦИАТОР** := <значение столбца 2 в ТБВ по строке  $k$ >

4)  $C$  := <значение столбца 3 в ТБВ по строке  $k$ >

5) Затирание  $k$ -ой строки ТБВ.

6) Передача управления подпрограмме, хранящейся в  $C$ .

Из алгоритма видно, что КАЛЕНДАРЬ ведет модельное время и инициирует выполнение активного события - первого события в каждом КОС.

АПУ генерирует пассивные события КОС и его алгоритм имеет следующий вид:

1) Просчет всех логических условий, заданных в столбце 1 ТУ. В зависимости от результата, полученного при вычислении логического условия  $j$ -ой строки ТУ, выполняется шаг 2 либо шаг 3.

2) Если логическое условие равно 0 (ложь), то  $j:=j+1$  и повторяется шаг 1.

3) Если логическое условие равно 1 (истина), то происходит разбор  $j$ -ой строки:

- **ИНИЦИАТОР** := <значение столбца 2 в ТУ по  $j$ -ой строке>;
- $D$  := <значение столбца 3 в ТУ по  $j$ -ой строке>;
- затирание  $j$ -ой строки ТУ;
- передача управления по адресу, хранящемуся в  $D$ .

4) Если все логические условия в столбце 1 равны 0 и нет ни одного условия, равного 1, управление передается программе КАЛЕНДАРЬ, так как исчерпаны все события текущего КОС и необходим переход к новому КОС, начинающемуся с активного события (см. теорему А.)

Как видно из рисунка 3.7, подпрограммы событий после своего выполнения передают управление в алгоритм АПУ. Каждая подпрограмма  $h_i$  в ходе своего выполнения меняет состояние системы и определяет условие продвижения инициатора своего процесса по треку. Если подпрограмма  $h_i$  содержит условие типа  $h^1$ , то  $h^1$  заполняет строку в ТБВ, определяет момент активизации инициатора. Навигационный оператор  $h''$  в составе  $h^1$  определяет адрес следующей по треку подпрограммы событий. В эту же таблицу помещается и значение текущего инициатора (ссылка на локальную среду). Если подпрограмма  $h_i$  содержит условие типа  $h^2$ , то  $h^2$  заносит строку в ТУ, помещая в столбец 1 логическое условие, а в остальные - инициатор и адрес следующей по треку подпрограммы событий аналогично вышеописанному.

Таким образом, предложенный моделирующий алгоритм реализует все необходимые действия в соответствии с алгоритмической моделью процесса. При этом задача генерации трека по ходу моделирования возлагается на подпрограммы событий.

### **3.5. Вопросы для самопроверки**

1. Определение отношения сцепленности элементарных операторов.
2. Понятие квазипараллельного процесса.
3. Правила корректности отображения параллельных процессов на квазипараллельный процесс.
4. Активное временное множество.
5. Алгоритм формирования модельного времени.
6. Классификация событий. Активные и пассивные события.
7. Класс одновременных событий.
8. Свойства класса одновременных событий.
9. Моделирующий алгоритм сканирующего типа.
10. Алгоритм календаря, структура таблицы будущих времен.
11. Алгоритм проверки условий, структура таблицы условий.

## ГЛАВА 4. СИСТЕМА ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ GPSS

### 4.1. Основные принципы

Программа на языке GPSS, как и во многих алгоритмических языках, представляет собой последовательность операторов. Особенностью реализации языка GPSS является описание развития процессов посредством обращений к некоторым фиксированным процедурам с помощью движущихся по этим описаниям динамических объектов, называемых *транзактами*. Видимо, в силу задания оператора в виде обращения к некоторой стандартной подпрограмме, исполняемые операторы в GPSS называются *блоками*. Таким образом, программа имитационной модели задается в виде последовательности блоков. Эта последовательность называется в дальнейшем *треком*. Под *оператором* в языке GPSS понимается некоторое описание, не входящее в трек, но описывающее некоторые параметры блоков или характеристики управления. Кроме того, в GPSS существует ряд ключевых слов, выполняющих служебные функции, структура которых фиксирована и оговаривается заранее. Концепция языка GPSS опирается на понятия *объектов*, которые являются основным инструментом создания имитационной модели. Система GPSS имеет развитую библиотеку процедур.

В учебном пособии излагаются основные положения системы имитационного моделирования GPSS. Содержание пособия структурировано по объектно-ориентированному принцип, т.е. для каждого типа объекта дается:

- описание структур данных объекта;
- синтаксис и семантика блоков и операторов, работающих с этими структурами;
- особенности их реализации;
- перечень стандартных (системных) атрибутов, соответствующих данному типу объекта;
- примеры обращения.

Кроме того, даны пояснения с использованием понятий Псевдоязыка Описания Сцепленных Процессов (ПОСП). Концепция построения системы имитационного моделирования GPSS в терминах ПОСП выглядит следующим образом:

1. Существует объект «транзакт», полностью соответствующий понятию «инициатор» в ПОСП и обладающим теми же свойствами:

- независимость
- динамичность

- инициативность
  - наличие локальной среды
2. В основу построения программы положен процессный принцип, когда модель описывается с помощью процессорной схемы.
3. Модель процессора описывается в виде структуры элементарных операторов, при этом элементарный оператор набирается из последовательности стандартных (библиотечных) подпрограмм, называемых в GPSS блоками, и завершается блоком условия задержки продвижения транзакта.
4. Выполнение программы блока происходит лишь при условии сцепления этого блока с каким-либо транзактом.
5. Моделирующий алгоритм GPSS построен по сканирующему принципу и содержит Таблицу Будущих Времени (список активных событий) и Таблицу Условий (список пассивных событий), являющиеся системными элементами.
6. Активным транзактом называется транзакт, который в данный момент времени имитационного процесса сцепляется с каким-либо блоком в рамках активного или пассивного событий.

## 4.2. **Общее описание**

### Объекты GPSS

Объекты являются концептуальными единицами, с помощью которых создается имитационная модель. Объекты либо создаются автоматически – при ссылке на них в модели, либо должны быть обязательно описаны перед их использованием.

В языке GPSS существуют следующие объекты:

#### Основные:

- транзакты
- блоки
- операторы

#### ресурсы:

- приборы
- памяти
- ключи

#### группы и списки:

- семейства транзактов
- группы транзактов
- числовые группы
- списки пользователя

#### вычислительные:

- датчики случайных чисел
- функции

- переменные
- булевские переменные
- сохраняемые величины
- матрицы сохраняемых величин

#### статистические:

- очереди
- таблицы

Каждый представленный объект имеет набор стандартных числовых атрибутов (СЧА) и стандартных логических атрибутов (СЛА), определяющий свойства объекта.

#### Именованние объектов

Для идентификации объектов в модели используются имена. Имена могут быть числовыми и символическими. Основными именами являются числовые имена. Числовое имя - целое положительное число. Символическое имя - последовательность символов. Символы включают прописные буквы A-Z, строчные буквы a-z, цифры 0-9 и символ \_ (подчеркивание).

Правила создания символических имен:

- длина имени от 1 до 250 символов
- имя должно начинаться с символа
- имя не должно быть ключевым словом GPSS.

Система GPSS не различает в обозначениях верхний и нижний регистры (прописные или строчные буквы). Нельзя присваивать объектам имена операторов и блоков, а также стандартных числовых атрибутов, используемых в системе. Имена используются также в качестве меток операторов или блоков GPSS.

#### Типы данных

В системе выделяют три типа данных: целочисленный (Integer), вещественный (Real) и строковый (String). Первые два относятся к числовому типу данных.

Целочисленный тип - 32-разрядные целые числа. Если во время арифметических операций происходит переполнение целого числа, то выполняется его преобразование к вещественному числу.

Вещественный тип - это числа с двойной точностью с плавающей запятой. Они имеют точность 15 десятичных цифр и диапазон экспоненты от -306 до 306.

Строковая константа - последовательность символов ASCII, взятая в двойные кавычки. Строковая константа может иметь любой размер. Для создания и управления строковыми константами в системе имеются строковые процедуры, которые находятся в библиотеке процедур. Строковые константы используются при выводе результатов моделирования в файл результата и формирования собственных сообщений.

Типы данных преобразуются либо явным образом при помощи вызова соответствующих процедур, либо неявно при вычислении выражения.

### 4.3. Стандартные числовые и логические атрибуты

Каждый тип объектов имеет набор стандартных числовых (СЧА) и стандартных логических (СЛА) атрибутов, определяющий свойства объектов данного типа.

Атрибуты являются параметрами блоков или системы в целом с фиксированными, или определенными некоторыми правилами, именами. Обычно имя атрибута начинается с *префикса*, однозначно соответствующего типу объекта.

Стандартные числовые и логические атрибуты конкретного объекта данного типа могут быть заданы одним из следующих способов:

1) <имя СЧА/СЛА типа объекта><j> – где j – положительное целое число, определяющее числовое имя объекта.

Префикс отражает принадлежность объекту. Так, Q - соответствует объекту QUEUE, S - объекту STORAGE, F - объекту FACILITY и т.д.

Пример: Q1 – текущая длина очереди с именем 1

2) <имя СЧА/СЛА типа объекта >\$ <имя> – где имя – символическое имя объекта

Пример: Q\$SERVER – текущая длина очереди с именем SERVER.

3) <имя СЧА/СЛА типа объекта >\*<j> – где j – положительное целое число, определяющее числовое имя параметра активного транзакта, который содержит числовое имя объекта (косвенная адресация).

Пример: Q\*1 – текущая длина очереди, числовое имя которой содержится в первом параметре транзакта.

4) <имя СЧА/СЛА типа объекта >\*< имя >(или <имя СЧА,СЛА типа объекта >\*\$<имя >) – где имя – символическое имя параметра активного транзакта, который содержит числовое имя объекта (косвенная адресация).

Пример: Q\*\$CHANNEL (или Q\*\$CHANNEL) – текущая длина очереди, числовое имя которой содержится в параметре транзакта с именем CHANNEL.

*Примечание:* Системные СЧА и СЧА транзактов (кроме СЧА P (значение параметра активного транзакта) являются «атомарными». Такие СЧА являются законченными, для их вычисления не требуется задания имени объекта.

### Системные СЧА

Таблица 4.1 – Системные СЧА (некоторые)

C1	Текущее значение условного времени. Автоматически изменяется в модели и устанавливается в 0 управляющими операторами CLEAR или RESET. Вещественное значение.
RN1 .. RN7	Число, вычисляемое датчиком случайных чисел. Датчик генерирует последовательность равномерно распределенных целочисленных случайных чисел в интервале 0 – 999. При использовании датчика в качестве аргумента функции или объекта в переменной значение будет дробью от 0 до 0.999999.

#### 4.4. Арифметические и логические выражения

##### Арифметическое выражение

Арифметическое выражение представляют собой комбинацию арифметических операторов, стандартных числовых атрибутов, библиотечных функций и констант, которая удовлетворяет правилам элементарной алгебры. Выражение вычисляется согласно приоритетам арифметических операций. Вычисление происходит слева направо. Порядок вычислений может быть изменен с помощью круглых скобок. При использовании арифметического выражения в качестве операнда блока, оно должно быть записано в круглых скобках.

Пусть имеем запись некоторой операции в виде:  $A < \text{знак операции} > B$

Результат такой операции представлен в таблицах 4.1 и 4.2.

Таблица 4.2 - Арифметические операции в порядке возрастания приоритета.

Знак операции	Операция	Результат
+	Сложение	$A+B$ возвращает значение суммы $A$ и $B$
-	Вычитание	$A-B$ возвращает значение разности $A$ и $B$
#	Умножение	$A\#B$ возвращает значение произведения $A$ и $B$
/	Деление	$A/B$ возвращает значение частного от деления $A$ на $B$
\	Целочисленное деление	$A\backslash B$ возвращает значение целочисленного деления $A$ на $B$
@	Целый остаток	$A@B$ возвращает целый остаток от деления $A$ на $B$
^	Возведение в степень	$A^B$ возвращает значение $A$ , возведенное в степень $B$

##### Логическое выражение

Логическое выражение принимает одно из двух значений: 1 или 0.

Таблица 4.3 - Логические операции

Знак операции	Операция	Результат
&, 'AND'	Логическое «И»	$A\&B$ возвращает 1, если числовые значения $A$ и $B$ отличны от нуля, в противном случае - 0
, 'OR'	Логическое «ИЛИ»	$A\text{ OR }B$ возвращает 1, если хотя бы одно из числовых значения $A$ или $B$ отлично от нуля, в противном случае - 0

Таблица 4.4 - Условные операции отношения

Знак операции	Операция	Результат
>, 'G'	Больше	$A > B$ возвращает 1, если числовое значение A больше B, в противном случае - 0
$\geq$ , 'GE'	Больше или равно	$A \geq B$ возвращает 1, если числовое значение A больше или равно B, в противном случае - 0
=, 'E'	Равно	$A = B$ возвращает 1, если числовое значение A равно B, в противном случае - 0
$\neq$ , 'NE'	Не равно	$A \neq B$ возвращает 1, если числовое значение A не равно B, в противном случае - 0
<, 'L'	Меньше	$A < B$ возвращает 1, если числовое значение A меньше B, в противном случае - 0

#### 4.5. Транзакты

Транзакт представляет собой динамический объект, полностью аналогичный понятию «инициатор». Содержательно транзакт есть ссылка (указатель) на локальную среду. Локальная среда транзакта представляет собой вектор, содержащий скалярные элементы. Вектор содержит системную и пользовательскую части. Элементы пользовательской части локальной среды транзакта называются параметрами транзакта. Пользовательская часть локальной среды транзакта имеет префикс P. Транзакты не могут непосредственно ссылаться друг на друга и могут обмениваться информацией только через другие объекты. Если транзакт в активном состоянии, интерпретатор пытается его продвинуть по блокам своего трека. При этом выполняются операции, соответствующие проходимому транзактом блоку.

Таблица 4.5 – СЧА транзакта

P	Значение параметра активного транзакта: целочисленное, вещественное или строковое значение.
PR	Приоритет активного транзакта: целочисленное значение.
M1	Время пребывания в модели активного транзакта. Равно разности текущего значения абсолютного времени и времени рождения активного транзакта: вещественное значение.
XN1	Номер активного транзакта. Целочисленное значение.

Алгоритм продвижения транзактов по модели работает следующим образом:

- если транзакт в активном состоянии, то интерпретатор пытается его продвинуть по треку блоков; при этом выполняются операции, соответствующие алгоритму блоков;
- если в выполняемом блоке не задана явным образом задержка транзакта, то моделирующий алгоритм сразу же пытается продвинуть транзакт через следующий блок.



Этот процесс продолжается до тех пор, пока транзакт не получает отказ при попытке войти в блок и будет задержан.

#### 4.6. Блоки генерации и задержки транзактов

Блоки – это объекты, описывающие изменение состояния модели. Блок характеризуется именем и параметрами, называемыми операндами блока. Блок выполняется только тогда, когда на него поступает транзакт. Блок может принять транзакт или отказать ему во входе, если не выполняются условия входа транзакта в блок. В случае отказа транзакт остается в предыдущем блоке. Если блок принял транзакт, то выполняются операции, соответствующие данному блоку.

Таблица 4.6 – СЧА блока

N	Общее число транзактов, которые вошли в блок: целочисленное значение.
W	Текущее число транзактов в блоке: целочисленное значение.

#### Блок GENERATE

Блок является единственным системным агрегатом и выполняет функции генерации транзактов. Блок генерирует транзакты и отправляет их вниз под себя на начало трека. Алгоритм блока соответствует алгоритму, изложенному на ПОСП в п.2.7.

*Синтаксис:* GENERATE [A],[B],[C],[D],[E]

Операнд	Назначение	Значение	Значение по умолчанию
A	<u>Среднее</u> время между моментами генерации новых транзактов	Число, СЧА (кроме СЧА транзактов)	0
B	Модификатор, задающий <u>разброс</u> времени между моментами генерации новых транзактов . Два типа модификаторов: - модификатор-интервал - модификатор-функция. <u>Модификатор-интервал</u> появляется, когда значением В является числовой скаляр. Тогда система предполагает задание <u>равномерного</u> закона распределения времени между моментами генерации новых транзактов. <u>Нижняя</u> граница интервала вычисляется как $(A - B)$ , <u>верхняя</u> граница как $(A + B)$ . <u>Модификатор-функция</u> появляется, когда значением В является функция. Тогда время между моментами генерации новых транзактов определяется как произведение значения операнда А и значения функции, заданной операндом В	Число, СЧА (кроме СЧА транзактов)	0

<b>C</b>	Начальная задержка. Задаёт момент генерирования первого транзакта в блоке GENERATE при первом выполнении модели, и после выполнения операции CLEAR. Поля A и B на задержку транзакта не влияют. Начальная задержка может быть меньше, равна или больше среднего времени, заданного в поле A.	Число, СЧА (кроме СЧА транзактов)	Если операнда нет или равен 0, то момент появления первого транзакта определяются операндами A и B
<b>D</b>	Предел генерации. Задаёт максимальное число транзактов, которое будет создано в блоке GENERATE. Если поле D пусто, блок генерирует неограниченное число транзактов. Предел генерации устанавливается повторно оператором CLEAR.	Число, СЧА (кроме СЧА транзактов)	$\infty$
<b>E</b>	Приоритет транзакта, задаваемый при генерации.	Число, СЧА (кроме СЧА транзактов)	0

Блок GENERATE при генерации транзакта привязывает к нему вектор параметров, извлекаемый из динамической области памяти системы моделирования. Он содержит ряд системных параметров (момент рождения, приоритет и др.) и пользовательские параметры, доступные пользователю.

Примеры:

1) GENERATE 25,10

- приращение времени для генерации очередного транзактов будет равно случайному числу, равномерно распределенному в интервале от 15 до 20 единиц модельного времени .

2) GENERATE 25, FN\$EXP

- приращение времени для генерации транзактов вычисляется как произведение значения числа 25 и текущего значения функции EXP.

3) GENERATE ,,25

- генерируются 25 транзактов в момент времени, равный 0.

4) GENERATE 720

- генерируются транзакты с интервалом 720 единиц модельного времени, причем первый транзакт появится в момент времени, равный 720.

Блок PRIORITY

Блок изменяет приоритет активного транзакта.

Синтаксис: PRIORITY A

Операнд	Назначение	Значение	Значение по умолчанию
<b>A</b>	Задаёт новое значение приоритета.	Имя, число, СЧА	Обязательный параметр

Пример: PRIORITY 10

- Вошедшему в блок транзакту присваивается приоритет, равный 10.

Блок ASSIGN

Блок заменяет, увеличивает или уменьшает значение параметра активного транзакта., производит формирование локальной среды транзакта.

Синтаксис: ASSIGN A,B[,C]

Операнд	Назначение	Значение	Значение по умолчанию
A	Определяет номер или имя параметр транзакта, которому присваивается значение. Если значение параметра нужно увеличить или уменьшить, то справа в операнде A ставится знак сложения (+) или вычитания (-)	Имя, число, СЧА	Обязательный параметр
B	Определяет значение, которое следует добавить или вычесть из значения параметра транзакта, заданного операндом A, или заменить его	Число, СЧА	Обязательный параметр
C	Задаёт имя модификатора-функции. При использовании операнда C значение операнда B умножается на значение модификатора-функции. Полученное произведение становится значением, которое изменяет значение параметра транзакта, заданного в операнде A.	СЧА функции	Нет

Примеры:

1) ASSIGN 2,15

- При поступлении транзакта к этому блоку в параметр номер 2 транзакта будет занесено значение 15.

2) ASSIGN 2+,3

- При поступлении транзакта к этому блоку значение параметра 2 этого транзакта будет увеличено на число 3 (инкремент).

3) ASSIGN 2,5, FN\$EXP

- При поступлении транзакта к этому блоку происходит обращение к функции FN\$EXP, полученное значение функции умножается на 5, а затем результат всей операции записывается во 2-й параметр активного транзакта.

Блок TERMINATE (базовое описание)

Блок уничтожает активный транзакт и все его параметры.

Синтаксис: TERMINATE

Полная транскрипция оператора приведена в приложении.

### Блок ADVANCE

Блок задерживает продвижение активного транзакта на заданное в параметрах блока время. Блок в основном соответствует оператору ЖДАТЬ (временное условие) языка ПОСП. Точнее говоря, этот оператор полностью эквивалентен макросу ЗАДЕРЖАТЬ ИНИЦИАТОР НА ...

*Синтаксис:* ADVANCE A[,B]

Операнд	Назначение	Значение	Значение по умолчанию
A	Среднее время задержки транзакта в блоке ADVANCE	Число, СЧА	Обязательный параметр
B	Модификатор, задающий <u>разброс</u> времени задержки Два типа модификаторов: - модификатор-интервал - модификатор-функция. <u>Модификатор-интервал</u> появляется, когда значением B является числовой скаляр. Тогда система предполагает задание <u>равномерного</u> закона распределения времени задержки транзакта. <u>Нижняя</u> граница интервала вычисляется как $(A - B)$ , <u>верхняя</u> граница как $(A + B)$ . <u>Модификатор-функция</u> появляется, когда значением B является функция. Тогда время задержки определяется как произведение значения операнда A и значения функции, заданной операндом B	Число, СЧА	0

#### Примеры:

- 1) ADVANCE 10  
- продвижение активного транзакта будет задержано на 10 единиц модельного времени
- 2) ADVANCE 15, 5  
- время задержки транзакта будет равно случайному числу, равномерно распределенному в интервале от 10 до 20 единиц модельного времени
- 3) ADVANCE 10, FN\$TOBR  
- время задержки транзакта будет равно произведению числа 10 и текущего значения функции с именем TOBR

### 4.7. Ресурсы

Ресурсы – это часть трека, содержащая некоторые условия блокировки этого участка для вхождения в него транзактов. В связи с появлением блокировок к этим ресурсам организуются очереди транзактов, управляемые системой моделирования.

### Устройство (FACILITY)

С позиций ПОСП этот тип объектов соответствует понятию «семафор – простая логическая переменная». Структурой данных этого типа объектов в GPSS является вектор, элементами которого являются простые логические переменные, принимающие значения 0 либо 1. Будем полагать, что 1 - соответствует состоянию устройства "занят", 0 - соответствует состоянию устройства "свободен". Префиксом этого одномерного массива в GPSS является символ F. Номер элемента массива F является именем соответствующего устройства: например, F3.

Вектор F : 

1	2	3	4	5	6	...
---	---	---	---	---	---	-----

В интерпретации GPSS под устройством понимается такой участок трека (ресурс), который заблокирован *одним логическим семафором*. Если семафор открыт, то транзакт может войти на этот участок трека (захватить ресурс). Если семафор закрыт, то все подошедшие транзакты выстраиваются в очередь типа FIFO к этому участку трека (очередь к ресурсу). Параметры, связанные с устройством, имеют префикс F.

Таблица 4.7 – СЧА устройств

F	Состояние прибора. Равно 0, если прибор свободен, и 1 - во всех остальных случаях. Целочисленное значение.
FI	Флаг прерывания прибора : 1 - если прибор находится в состоянии прерывания, 0 - в противном случае. Целочисленное значение.
FR	Коэффициент использования прибора в долях тысячи. Вещественное значение.
FC	Общее число занятий прибора. Целочисленное значение.
FT	Среднее время использования прибора одним транзактом. Вещественное значение.

Таблица 4.8 - СЛА устройств

NU	Равен 1, если прибор свободен, иначе 0
U	Равен 1, если прибор занят, иначе 0

### Блоки взаимодействия с устройствами

#### Блок SEIZE

Блок позволяет активному транзакту занять устройство (захватить ресурс) или встать в очередь к ресурсу.

Синтаксис: SEIZE A

Операнд	Назначение	Значение	Значение по умолчанию
A	Имя занимаемого устройства	Имя, число, СЧА	Обязательный операнд

Например, SEIZE 3 означает захват устройства под номером 3

Сцепление транзакта с блоком SEIZE 3 соответствует следующей записи на ПОСП:

*ЖДАТЬ* F(3) = 0;  
F(3) := 1;

Особенности выполнения.

1) При попытке транзакта войти в блок SEIZE всегда проверяется, свободен ли ресурс (открыт ли semaфор).

2) Если ресурс свободен (semaфор открыт), то транзакт, занявший устройство, пытается перейти к следующему по номеру блоку, а semaфор закрывается. Устройство остается занятым до тех пор, пока занимающий его транзакт не войдет в соответствующий блок RELEASE. Прежде чем освободить ресурс, транзакт может пройти через произвольное число блоков.

3) Блок SEIZE отказывает во входе транзакту, если ресурс занят (semaфор закрыт). При этом транзакт помещается в список задержки устройства в конец очереди своего приоритетного класса.

Пример: SEIZE Server

Активный транзакт пытается занять устройство с именем Server.

#### Блок RELEASE

Блок позволяет активному транзакту освободить занятое устройство (открыть semaфор).

Синтаксис: RELEASE A

Операнд	Назначение	Значение	Значение по умолчанию
A	Имя освобождаемого устройства	Имя, число, СЧА	Обязательный операнд

Например, RELEASE 3 означает освобождение устройства под номером 3.

Сцепление транзакта с блоком RELEASE 3 соответствует следующей записи на ПОСП: F(3) := 0;

Особенности выполнения:

1. Если ресурс занят активным транзактом, он освобождает ресурс и пытается войти в следующий по порядку блок.
2. Если активный транзакт освобождает ресурс, следующий транзакт выбирается из списка задержки и пытается занять ресурс. Если в списках транзакты отсутствуют, ресурс становится не занятым.
3. Активируются транзакты из списка задержки блоков GATE NU.

#### Памяти (STORAGE, многоканальный сервер)

С позиций ПОСП этот тип объектов соответствует понятию «семафор – логическая функция». Для объектов типа «память» логическая функция имеет вид:  $T \leq M$ . Структурой данных этого типа объектов в GPSS является двумерная матрица, номер строки которой соответствует номеру памяти, а два столбца описывают состояние памяти. Элементами матрицы являются целые числа. В первом столбце помещается максимальный объем памяти  $M$ , во втором – текущее значение занятой памяти  $T$ . Префиксом этого массива в GPSS является символ  $S$ .

	МАТРИЦА S	
	M	T
ПАМЯТЬ НОМЕР 1	10	6
ПАМЯТЬ НОМЕР 2		
ПАМЯТЬ НОМЕР 3		
ПАМЯТЬ НОМЕР 4		
...		

С позиций GPSS под памятью понимается такой участок трека (ресурс), который заблокирован *одним арифметическим семафором*. Это означает, что семафор пропускает не более заданного в этом семафоре количества транзактов в ресурс. Если количество подошедших к ресурсу транзактов превышает заданное ограничение, то семафор закрывается, а оставшиеся транзакты выстраиваются к ресурсу (памяти) в очередь с дисциплиной FIFO.

Параметры, связанные с памятьями, имеют префикс  $S$ .

Таблица 4.9 – СЧА памяти

S	Текущее содержимое памяти. Целочисленное значение.
R	Число свободных единиц памяти. Целочисленное значение.
SR	Коэффициент использования памяти в долях тысячи. Вещественное значение.
SA	Взвешенное по времени среднее содержимое памяти. Вещественное значение.

SM	Максимальное содержимое памяти. Целочисленное значение.
SC	Общее количество использовавшихся элементов памяти. Целочисленное значение.
ST	Среднее время пребывания транзактов в памяти. Вещественное значение.
SE	Флаг незанятости памяти: 1 - свободна, 0 – занята. Целочисленное значение.
SF	Флаг заполненности памяти: 1 - заполнена, 0 - не заполнена. Целочисленное значение.
SV	Флаг готовности памяти: 1 - готова , 0 - не готова. Целочисленное значение.

Таблица 4.10 – СЛА памяти

SE	Равен 1, если память пуста (нулевое содержимое) , иначе 0
SNE	Равен 1, если память не пуста (ненулевое содержимое) , иначе 0
SF	Равен 1, если память заполнена, иначе 0
SNF	Равен 1, если память не заполнена, иначе 0
SV	Равен 1, если память используется, иначе 0
SNV	Равен 1, если память не используется, иначе 0

#### Блоки взаимодействия с памятьми

##### Оператор STORAGE

Оператор STORAGE определяет максимальное значение емкости в памяти (размер семафора). Оператор относится к декларирующему типу и ставится в начале программы.

Синтаксис: <имя памяти> STORAGE N

Имя памяти – символическое или числовое имя памяти, ставится в поле меток. Операнд N определяет емкость данной памяти: обязательный операнд целого типа

*Пример:* RAM\_Workstation STORAGE 1024

- Оператор определяет память с именем RAM\_Workstation с общей емкостью 1024 элементов.

##### Блок ENTER

Блок позволяет активному транзакту либо занять определенное число элементов памяти, либо встать в очередь к данной памяти

Синтаксис: ENTER A,[B]

Операнд	Назначение	Значение	Значение по умолчанию
<b>A</b>	Имя занимаемой памяти	Имя, число, СЧА	Обязательный параметр
<b>B</b>	Число занимаемых элементов памяти	Имя, число, СЧА	1



Сцепление транзакта с блоком ENTER 3 соответствует следующей записи на ПОСП:

*ЖДАТЬ* S(3,2) ≤ S(3,1);

S(3,2) := S(3,2) + 1;

Особенности выполнения:

1. Операнд А должен указывать на заранее определенную оператором STORAGE память.

2. При попытке транзакта войти в блок ENTER всегда проверяется, существует ли необходимое число свободных элементов данной памяти и находится ли память в состоянии «готова к использованию». Для определения необходимого числа свободных элементов памяти используется операнд В.

3. Если необходимое число свободных элементов есть и память готова, она занимает. Число свободных элементов памяти уменьшается на заданную в параметре В величину.

4. Транзакт, занявший память, пытается перейти к следующему по треку блоку.

5. Элементы памяти остаются занятыми до тех пор, пока занимающий их транзакт не войдет в соответствующий блок LEAVE.

6. Блок ENTER отказывает во входе транзакту, если нет необходимого числа свободных элементов данной памяти. При этом устанавливается его индикатор задержки и транзакт помещается в список задержки памяти в конец своего приоритетного класса

Примеры:

1. ENTER RAM

Транзакт пытается занять один элемент памяти с именем RAM

2. ENTER RAM, P\$PAM

Транзакт пытается занять память с именем RAM. Необходимое число элементов памяти содержится в параметре транзакта с именем PAM.

Блок LEAVE

Блок позволяет активному транзакту освободить определенное число элементов памяти.

Синтаксис: LEAVE A,[B]

Операнд	Назначение	Значение	Значение по умолчанию
А	Имя освобождаемой памяти	Имя, число, СЧА	Обязательный параметр
В	Число освобождаемых элементов памяти	Имя, число, СЧА	1

Сцепление транзакта с блоком LEAVE 3,4 соответствует следующей записи на ПОСП:

$S(3,2) := S(3,2) - 4;$

Особенности выполнения:

1. Операнд А должен указывать на заранее определенную оператором STORAGE память.

2. Для определения числа освобождаемых элементов памяти используется операнд В.

3. Транзакт, освободивший элементы памяти, пытается перейти к следующему по треку блоку.

4. После освобождения памяти, список задержки памяти просматривается в порядке убывания приоритетов, определяются транзакты, потребность в памяти которых может быть удовлетворена. Используется правило «первый подходящий с пропусками». Успешные транзакты входят в блок ENTER.

Примеры:

1. LEAVE RAM

Транзакт освобождает один элемент памяти с именем RAM

2. LEAVE RAM, P\$PAM

Транзакт освобождает элементы памяти с именем RAM. Освобождаемое число элементов памяти содержится в параметре транзакта с именем PAM.

#### 4.8. Логические ключи

Логические ключи представляют собой логические переменные, предназначенные выполнять функцию логического семафора на любом участке трека.

Таблица 4.11 – СЛА ключей

LR	Равен 1, если логический ключ "выключен", иначе 0
LS	Равен 1, если логический ключ "включен", иначе 0

Блок LOGIC

Блок изменяет состояние логического ключа.

Синтаксис: **LOGIC X A**

X – логический оператор;

A – имя (номер) логического ключа.

Операнд	Назначение	Значение	Значение по умолчанию
X	Логический оператор	S,R или I	Обязательный параметр
A	Имя логического ключа	Имя, число, СЧА	Обязательный параметр

Особенности выполнения:

1. Блок LOGIC предназначен для того, чтобы устанавливать, сбрасывать или инвертировать (менять на противоположное) состояние логического ключа.

2. Логический ключ может находиться в двух состояниях: **S** (включен или 1) или **R** (выключен или 0).

3. Если логический оператор равен «S» или «R», то логический ключ, заданный операндом A, устанавливается во включенное или выключенное состоянии соответственно.

4. Если логический оператор равен «I», то логический ключ, заданный операндом A, инвертируется. Это значит, что если он был включен, он будет выключен и наоборот.

*Примеры:*

1. LOGIC S SWITCH
2. LOGIC R SWITCH
3. LOGIC I SWITCH

#### 4.9. Блоки и операторы организации вычислений

Вектор сохраняемых величин – это совокупность параметров, предназначенная для сохранения скалярных значений, организованная в виде вектора. Адресация такой величины имеет вид XI, где X – стандартное имя вектора, I – номер или имя элемента вектора X.

##### Блок SAVEVALUE

Блок присваивает, увеличивает или уменьшает значение сохраняемой величины.

Синтаксис: SAVEVALUE A[±],B

Операнд	Назначение	Значение	Значение по умолчанию
<b>A</b>	Имя изменяемой сохраняемой величины	Имя, число, СЧА	Обязательный операнд
<b>B</b>	Значение, которое присвоено, прибавлено или вычтено из сохраняемой величины	Имя, число, СЧА	Обязательный операнд

Примеры.

1. SAVEVALUE 25, 7

- Значение сохраняемой величины X25 становится равным 7.

2. SAVEVALUE WES+, 2

- Значение сохраняемой величины X\$WES увеличивается на 2.

##### Оператор MATRIX

*Матрица сохраняемых величин* – это совокупность параметров, предназначенная для сохранения скалярных значений, организованная в виде двумерной матрицы.

Адресация такой величины имеет вид  $MX(I, J)$ , где  $MX$  – стандартное имя матрицы;  $I, J$  – номера или имена элемента матрицы  $MX$ .

Оператор определяет матрицу сохраняемых величин. Имеет декларирующий характер и располагается выше описаний треков.

Синтаксис: `<имя матрицы> MATRIX A, B, C`

имя матрицы – числовое или символическое имя, обязательный параметр. Расположен в поле метки.

A - неиспользуемое поле (для совместимости с ранними версиями GPSS).

B - максимальное количество элементов в первом измерении. (количество строк матрицы). Допустимое значение – целое число. Обязательный операнд

C - максимальное количество элементов во втором измерении (количество столбцов матрицы). Допустимое значение – целое число. Обязательный операнд.

*Пример:*

`REZULT MATRIX ,15,3`

Оператор определяет матрицу с именем REZULT с 15 строками и 3 столбцами.

#### Блок MSAVEVALUE

Блок присваивает, увеличивает или уменьшает значение элемента матрицы сохраняемых величин.

Синтаксис: `MSAVEVALUE A[±], B, C, D`

Операнд	Назначение	Значение	Значение по умолчанию
<b>A</b>	Имя изменяемой матрицы сохраняемых величин	Имя, число, СЧА	Обязательный операнд
<b>B</b>	Номер строки матрицы	Имя, число, СЧА	Обязательный операнд
<b>C</b>	Номер столбца матрицы	Имя, число, СЧА	Обязательный операнд
<b>D</b>	Значение, которое присвоено, прибавлено или вычтено из элемента матрицы	Имя, число, СЧА	Обязательный операнд

Особенности выполнения:

1. Матрица должна быть предварительно определена с помощью оператора определения данных MATRIX.

2. Если операнд A сопровождается знаком «+», то к значению элемента матрицы, определенного операндами A, B и C, прибавляется значение, определенное операндом D.

3. Если операнд A сопровождается знаком «-», то из значения элемента матрицы, определенного операндами A, B и C, вычитается значение, определенное операндом D.

4. Если за операндом A нет знака, то значение элемента матрицы, определенного операндами A, B и C, заменяется значением, определенным операндом D.

*Пример:* MSAVEVALUE DATA, 4, 5, P7

Элементу матрицы DATA с номером строки 4 и номером столбца 5 присваивается значение седьмого параметра активного транзакта.

Обращение к этому элементу в арифметических и других выражениях имеет вид **MX(4, 5)**.

#### Оператор VARIABLE

Оператор VARIABLE является оператором- функцией, определяющим арифметическую целую переменную, имеющую имя V I, где V – стандартное обозначение переменной типа оператор-функция, I – номер или имя функции **VARIABLE**.

Синтаксис: <имя переменной> VARIABLE <арифметическое выражение>

- Имя переменной - символическое или числовое имя переменной, ставится в поле меток;

- Арифметическое выражение определяет значение данной переменной.

*Примеры:*

1) 5 VARIABLE 46+P6

- Арифметическая переменная с числовым именем 5 равна сумме числа 46 и значения параметра 6 активного транзакта. Обращение к этой переменной имеет вид V5.

2) SUM VARIABLE (P3+P4)/5

- Арифметическая переменная с символическим именем SUM равна сумме значений 3-го и 4-го параметров активного транзакта, деленной на 5. Обращение к этой переменной имеет вид V\$SUM.

Арифметическая переменная может использоваться как:

- элемент другой арифметической переменной, булевой переменной;
- аргумент функции, таблицы;
- операнд блока.

#### Оператор FVARIABLE

FVARIABLE является оператором-функцией, определяющим арифметическую переменную с фиксированной точкой, имеющую имя FV I, где FV – стандартное обозначение, I – номер или имя функции **FVARIABLE**.

Синтаксис: <имя переменной> FVARIABLE <арифметическое выражение>

Имя переменной - символическое или числовое имя переменной, ставится в поле меток. Арифметическое выражение определяет значение переменной.

Пример: PBR FVARIABLE (SI-\$SCAN)/5 +3.6

Ссылка на арифметическую переменную с фиксированной точкой выполняется так же, как и на арифметическую переменную.

### Оператор BVARIABLE

Оператор BVARIABLE определяет булеву (логическую) переменную.

Синтаксис: <имя переменной> BVARIABLE <булево выражение>

Выражения в операторе BVARIABLE кроме логических операторов могут включать операции отношений и вызовы библиотечных процедур.

*Пример:* FLAG BVARIABLE BV\$CAN1'AND'BV\$CAN2

Значение переменной FLAG равно 1, если булевы переменные CAN1 и CAN2 имеют значение TRUE, и равна 0 в других случаях

### Оператор FUNCTION (базовое описание)

Оператор определяет функцию GPSS, заданную таблично.

Синтаксис: <имя функции> FUNCTION A,BN

- имя функции – числовое или символическое имя, обязательный параметр.

- A - аргумент функции; обязательный операнд. Допустимые значения – имя, число, СЧА.

- B - тип функции (одна буква) и N - количество пар данных в списке данных функции. Обязательный операнд.

Обращение к функции выглядит, как FN I, где I – имя функции.

Существует несколько типов функций. Тип определяется операндом B оператора FUNCTION. За строкой FUNCTION A,BN сразу же должна следовать строка, содержащая список пар данных, разделенных символом «/». Каждая пара данных определяет значения аргумента X и значения функции Y, разделенные запятой. Список данных используются для вычисления значения функции по заданным значениям аргумента.

В данном описании оператора рассмотрим два типа функций. Полное изложение вариантов задания функции приведено в Руководстве GPSS.

#### 1. Функции типа C – непрерывные числовые функции.

В списке данных функций типа C значения X и Y должны быть целочисленными (Integer) или вещественными (Real). Значения X и Y хранятся, как числа с плавающей точкой двойной точности.

Вычисление функции начинается с вычисления аргумента. Далее определяется интервал ( $X_i$ ;  $X_{i+1}$ ), на котором находится вычисленное значение аргумента и на этом интервале выполняется линейная интерполяция двойной точности с использованием соответствующих значений  $Y_i$  и  $Y_{i+1}$ . Результатом является значение функции двойной точности. Если аргумент попадает за предельные значения

области определения функции, возвращается значение функции в ближайшей предельной точке.

Примеры:

1) ART FUNCTION X1,C3  
1.1,10.1/20.5,98.7/33.3,889.2

- Оператор определяет кусочно-линейную функцию с двумя линейными участками. Если мы обращаемся к функции FNSART, то по значению сохраняемой величины X1 вычисляется функция в соответствии с заданной совокупностью точек.

2) Xpdis FUNCTION RN1,C24  
0,0/1.,104/2, 222/3, 355/4, .509/5, .69/6,915/7,1.2/75,1.38  
.8, 1.6/.84, 1.83/.88, 2.12/.9, 2.3/.92, 2.52/.94, 2.81/.95, 2.99/.96,3.2  
97, 3.5/.98, 3.9/.99, 4.6/.995, 5.3/.998,6.2/.999,7/.9998,8

- Пример приближенного представления обратного экспоненциального распределения со средним, равным 1.

2. Функции типа D - дискретные функции.

В списках данных функции типа D значения X должны быть целочисленными или вещественными, а значения Y – целочисленными, вещественными или именами.

Функция типа D задает одно и то же значение функции Y<sub>i</sub> для всех значений аргумента X<sub>i-1</sub> < X ≤ X<sub>i</sub>. Значения X в списке данных функции должны быть неубывающими. Внутренне они сохраняются, как числа двойной точности. Когда вычисляется функция, значения X в списке данных функции просматриваются от наименьшего к наибольшему. Когда найдено значение X, которое больше или равно текущему значению аргумента, возвращается соответствующее ему значение Y. Если такое значение X отсутствует, возвращается значение Y или именованная величина, соответствующая самому большому значению X.

Примеры:

1) LIR FUNCTION XSA2, D5  
1.1,6.9/2.1,7/6.33,9.4/7,10/9.9,12.01

2) RAF FUNCTION RN1, D5  
0,0/2,7.2/4,6.667/.8,9.92/1.0,10

Оператор INITIAL

Оператор INITIAL задает начальное значение сохраняемым величинам, элементам матрицы, логическим ключам.

Синтаксис: INITIAL A, B

- A - логический ключ, сохраняемая величина, элемент матрицы, определенные как СЧА. Операнд A должен иметь форму СЧА классов LS, X, MX или имени

матрицы. В операнде А нельзя использовать параметры транзакта;

- В - присваиваемое значение

Если операнд А указывает на логический ключ, присваиваются только значения 0 или 1. Если операнд В явно задан как 0, то присваивается значение 0. В противном случае присваивается 1. Если операнд А задает имя матрицы, всем ее элементам присваивается значение, указанное операндом В. По умолчанию это 1. Для задания элемента двумерной матрицы может использоваться СЧА класса МХ.

*Примеры:*

INITIAL X21, 17

INITIAL L\$KLU, 1

#### 4.10. Блоки управления движением транзактов

Блок TRANSFER (базовое описание)

Блок передает транзакт на указанный блок.

Синтаксис: TRANSFER [P],В

Операнд	Назначение	Значение	Значение по умолчанию
<b>Р</b>	Вероятность Р продолжения движения транзакта по треку	число, СЧА	Режим безусловной передачи
<b>В</b>	Номер или метка блока, куда переходит транзакт с вероятностью (1-Р)	Имя, число, СЧА	Нет

Таким образом, блок TRANSFER является навигационным оператором с несколькими режимами работы.

##### 1. Режим статистической передачи

В этом режиме активный транзакт переходит к блоку, заданному в операнде В, с вероятностью, обратной заданной в операнде А. Операнд А может быть положительной дробью, меньшей единицы или целым положительным числом. Если операнд А - целое число, оно интерпретируется как доля от тысячи. С вероятностью, заданной в операнде А, транзакт продолжает движение по треку.

Пример: TRANSFER 0.3, LAB1

- Транзакт с вероятностью 0.3 продолжает движение дальше по треку, с вероятностью 0.7 – поступает на блок, помеченный меткой LAB1.

##### 2. Режим безусловной передачи

Когда операнд А отсутствует, блок TRANSFER функционирует в режиме безусловной передачи. В этом режиме активный транзакт всегда переходит к блоку, заданному в операнде В.

Пример: TRANSFER ,NO\_SERV



- Транзакт из блока TRANSFER передается в блок с меткой NO\_SERV.

### Блок TEST

Блок выполняет навигацию транзакта в зависимости от результата операции отношения, заданной в этом же блоке.

Синтаксис: TEST X A, B [, C]

Операнд	Назначение	Значение	Значение по умолчанию
X	Операция отношения (см.выше)	Условная операция отношения	Обязательный операнд
A	Левая часть отношения	Имя, число, СЧА	Обязательный операнд
B	Правая часть отношения	Имя, число, СЧА	Обязательный операнд
C	Метка или номер блока, в который будет передан транзакт	Имя, число, СЧА	Режим отказа

Блок TEST функционирует в двух режимах:

1. Если операнд C не используется, блок TEST функционирует в режиме отказа. Когда транзакт пытается войти в блок TEST, работающий в режиме отказа, и заданное условие не выполняется, транзакт блокируется, и ему не позволяется войти в блок TEST, проверка повторяется до тех пор, пока условие не будет выполнено. После выполнения заданного условия активный переходит к следующему по порядку блоку.

2. Если операнд C используется, блок TEST функционирует в режиме передачи транзакта. Когда транзакт пытается войти в такой блок TEST и проверяемое условие не выполняется, транзакт переходит к блоку, указанному в операнде C. Если проверяемое условие выполняется, активный транзакт входит в блок TEST и затем переходит к следующему по порядку блоку.

Таким образом, в терминах ПОСП блок работает в качестве навигационного оператора совместно с оператором ЖДАТЬ (логическое условие).

Примеры:

1. TEST L Q\$SERVER,100

- После входа в блок TEST транзакт при выполнении условия (Q\$SERVER<100) перейдет к следующему блоку по треку, иначе транзакт будет задержан вплоть до выполнения указанного условия.

2. TEST L Q\$SERVER,100, MET1

- После входа в блок TEST транзакт при выполнении условия (Q\$SERVER<100) перейдет к следующему блоку по треку, иначе транзакт будет передан на блок, помеченный меткой MET1.

### Блок GATE

Блок передает транзакт в зависимости от состояния объекта.

Синтаксис: GATE X A[,B]

Операнд	Назначение	Значение	Значение по умолчанию
X	Определяет проверяемый логический атрибут объекта	СЛА (по таблице СЛА)	Обязательный параметр
A	Имя или номер проверяемого объекта	Имя, число, СЧА	Обязательный параметр
B	Определяет блок для режима перехода	Имя, число, СЧА	Режим отказа

Блок GATE работает в двух режимах:

1. Если операнд B не используется, блок GATE работает в режиме отказа. Когда транзакт пытается войти в блок GATE, работающий в режиме отказа, и условие, указанное в операнде X не выполняется, транзакт задерживается, пока условие не будет выполнено. Если условие выполняется, активный транзакт переходит к следующему блоку по треку.

2. Если операнд B используется, блок GATE работает в режиме перехода. Когда транзакт пытается войти в блок GATE, и условие не выполняется, транзакт направляется к блоку, помеченному операндом B. Если условие выполняется, активный транзакт входит в блок GATE и затем переходит к следующему блоку по треку.

Таким образом, в терминах ПОСП блок работает в качестве оператора ЖДАТЬ (логическое условие) совместно с навигационным оператором.

#### Примеры:

1. GATE FV SERVER

В режиме отказа активный транзакт войдет в блок GATE, если устройство с именем SERVER доступно, в противном случае транзакт блокируется до выполнения условия.

2. GATE SE RAM, NO\_RAM

В режиме перехода, если память с именем RAM пуста, транзакт входит в блок GATE и переходит к следующему блоку по треку. В противном случае транзакт переходит к блоку с меткой NO\_RAM.

### Блок LOOP

Блок передает транзакт и уменьшает значение параметра транзакта – счетчика итераций. Используется для организации циклических процессов.

Синтаксис: LOOP A,B

Операнд	Назначение	Значение	Значение по умолчанию
<b>A</b>	Имя параметра транзакта, содержащего уменьшаемое число	Имя, число, СЧА	Обязательный параметр
<b>B</b>	Метка или номер блока, в который будет передан транзакт	Имя, число, СЧА	Обязательный параметр

Особенности выполнения:

1. Блок всегда принимает транзакт.
2. После входа транзакта в блок числовое значение параметра транзакта, заданного операндом А, уменьшается на единицу.
3. Если новое значение параметра больше нуля, то транзакт передается в блок, имя которого содержится в операнде В. В противном случае транзакт переходит к следующему блоку по треку.

*Пример:* LOOP 5, CYCLE

Значение пятого параметра транзакта уменьшается на единицу и, если оно остается больше нуля, транзакт передается на блок с меткой CYCLE.

#### 4.11. Блоки и операторы сбора статистики

Блок QUEUE (очередь) – регистратор статистики. Предназначен для сбора и обработки статистики на некотором фрагменте трека. Блок отмечает место входа на треке транзакта в регистратор, что соответствует началу сбора статистики для данного транзакта.

Синтаксис: QUEUE A

Операнд	Назначение	Значение	Значение по умолчанию
<b>A</b>	Имя регистратора (очереди)	Имя, число, СЧА	Обязательный операнд

Пример: QUEUE STAT1

- В регистраторе с именем STAT1 отмечается начало сбора статистики для подошедшего транзакта.

#### Блок DEPART

Блок отмечает место выхода транзакта из регистратора на треке, что соответствует концу сбора статистики для данного транзакта.

Синтаксис: DEPART A

Операнд	Назначение	Значение	Значение по умолчанию
A	Имя регистратора (очереди)	Имя, число, СЧА	Обязательный операнд

Пример: DEPART SYSTEM

- В регистраторе с именем SYSTEM отмечается окончание сбора статистики для подошедшего транзакта.

Результаты по собранной статистике размещаются в СЧА регистратора (очереди). Собираемая статистика и имена ее параметров приведены в таблице 4.12.

Таблица 4.12 – СЧА очередей

Q	Текущая длина очереди. Целочисленное значение.
QA	Взвешенная по времени средняя длина очереди. Вещественное значение.
QM	Максимальная длина очереди. Целочисленное значение.
QC	Общее число входов в очередь. Целочисленное значение.
QZ	Число нулевых входов в очередь. Целочисленное значение.
QT	Среднее время пребывания транзактов в очереди (включая нулевые входы). Вещественное значение.
QX	Среднее время пребывания сообщения в очереди (без нулевых входов). Вещественное значение.

#### Оператор TABLE

Оператор TABLE – декларирующий оператор, определяет таблицу плотности распределения случайной величины, ее интегральных относительных частот, среднего и стандартного отклонения.

Синтаксис: <имя таблицы> TABLE A, B, C, D

Длина имени таблицы ограничена 32 символами.

A - аргумент таблицы - случайная величина, плотность распределения которой находится, как частота попадания на интервалы таблицы.

B – левая граница таблицы (левая граница первого частотного интервала).

C - размер частотных интервалов.

D - количество частотных интервалов (включая интервал от  $-\infty$  до левой границы таблицы и интервал от правой границы таблицы до  $+\infty$ ).

Пример: GIST TABLE P\$TOA, 3.62, 10, 10

- В этом примере в таблице с именем GIST регистрируется распределение значений параметра TOA транзакта. Оператор TABLE создает таблицу с десятью частотными интервалами. Все значения TOA, меньшие или равные 3.62, приводят к изменению первого частотного интервала таблицы. (Обычно частота увеличивается на 1. Однако в операнде B блока TABULATE может использоваться весовой коэффициент, что приводит к добавлению весового коэффициента к значению

частоты. Весовой коэффициент также применяется для среднего и стандартного отклонения, что равнозначно нескольким входам в блок TABULATE).

Если значение параметра TOA больше 36.2, будет изменено значение частоты в десятом (последний) частотном интервале. Если значение параметра TOA не попадает ни в первый, ни в последний частотный интервал, оно используется для изменения частоты в интервалах со 2-го по 9-й.

Статистика, собранная в таблице, выводится в стандартный отчет системы GPSS.

#### Блок TABULATE

Блок строит таблицу и добавляет в нее данные.

Синтаксис: TABULATE A [, B]

Операнд	Назначение	Значение	Значение по умолчанию
<b>A</b>	Имя таблицы, в которую заносится табулируемая величина (аргумент) в момент входа транзакта в данный блок	Имя, число, СЧА	Обязательный операнд
<b>B</b>	Весовой коэффициент	Имя, число, СЧА	1

Особенности выполнения:

Таблица должна быть предварительно определена с помощью оператора определения данных TABLE.

Если задан операнд B, то он задает количество раз, которое табулируемая величина (аргумент) должна быть занесена в таблицу при каждом входе в блок. Операнд B должен быть положительным.

В результате моделирования таблица с именем A содержит гистограмму, а также значения среднего и стандартного отклонения регистрируемой величины (аргумента).

Блок TABULATE является частью трека и срабатывает лишь при попадании в него транзактов.

Пример: TABULATE GIST

- Когда транзакт входит в блок TABULATE, происходит занесение в таблицу GIST величины (аргумента), определенной в операторе TABLE.

Таблица 4.13 – СЧА табуляции

TB	Среднее значение аргументов таблицы. Вещественное значение.
TC	Общее число аргументов таблицы. Целочисленное значение.
TD	Среднеквадратичное отклонение для аргументов таблицы. Вещественное значение.

## 4.12. Блоки работы с семейством транзактов

### Блок SPLIT

Блок генерирует транзакты того же семейства, что и активный транзакт.

Синтаксис: SPLIT A, [B], [C]

Операнд	Назначение	Значение	Значение по умолчанию
A	Количество генерируемых транзактов-потомков семейства	Число, СЧА	Обязательный параметр
B	Имя или номер блока, куда переходят транзакты-потомки	Имя, число, СЧА	Следующий блок
C	Параметр транзакта, значение которого увеличивается на единицу для каждого транзакта семейства (для транзакта-родителя на 1, для первого потомка семейства на 2 и т.д.).	Имя, число, СЧА	Нет

#### Примеры:

1) SPLIT 2

Создаются два транзакта-потомка, которые вместе с транзактом-родителем переходят к следующему блоку.

2) SPLIT 3,fork,5

Создаются три транзакта-потомка, которые переходят к блоку с меткой fork. Параметр 5 родителя и потомков будет модифицирован. Если параметр 5 родителя не был определен, то он будет создан и после выполнения блока SPLIT его значение у родителя станет равным 1, у первого потомка – 2, у второго – 3, у третьего – 4.

### Блок ASSEMBLE

Блок объединяет транзакты одного семейства в один.

Синтаксис: ASSEMBLE A

Операнд	Назначение	Значение	Значение по умолчанию
A	Количество членов семейства, подлежащих объединению	Число, СЧА	Обязательный параметр

Пример: ASSEMBLE 5

- В результате выполнения блока происходит объединение пяти членов каждого ансамбля, члены которых будут входить в данный блок

### Блок MATCH

Блок синхронизирует движение транзактов одного семейства.

Синтаксис: MATCH A

Операнд	Назначение	Значение	Значение по умолчанию
A	Имя или номер блока MATCH, сопряженного с данным.	Имя, число, СЧА	Обязательный параметр

Пример:      PROC1 MATCH    PROC2  
                  :  
                  :  
                  PROC2 MATCH    PROC1

В этом случае два транзакта одного семейства могут пройти через сопряженные блоки MATCH только одновременно.

#### 4.13. Останов процесса моделирования

Процедура останова модели включает 3 компонента:

- счетчик останова;
- занесение начального значения в счетчик останова;
- изменение значения счетчика останова.

Останов моделирования происходит, когда содержимое счетчика останова принимает значение 0. Счетчик останова – системный параметр TG1. Работа со счетчиком останова выполняется с помощью оператора START и блока TERMINATE. Оператор START в параметре A содержит начальное значение счетчика останова.

Изменение содержимого счетчика останова выполняется блоком TERMINATE, содержащим параметр A. При поступлении активного транзакта на блок TERMINATE происходит не только его уничтожение, но и выполняется вычитание из счетчика останова содержимого параметра A блока TERMINATE.

*Пример управления окончанием моделирования по числу транзактов:*

```
GENERATE    1000
<сегмент модели>
TERMINATE    1
START 20
```

Моделирование завершится, когда через сегмент модели пройдет 20 транзактов. Эта величина первоначально задается оператором START и уменьшается каждый раз при входе транзакта в блок TERMINATE. После поступления 20 транзактов на блок TERMINATE счетчик останова станет равным 0 и моделирование завершится.

Пример управления окончанием моделирования по времени:

```
GENERATE    15,5
<сегмент модели>
TERMINATE
GENERATE    720
```

TERMINATE 1

START 1

Основной сегмент модели содержит блоки TERMINATE, у которых отсутствует параметр А. Для останова по времени создается отдельный сегмент, содержащий указанные блоки. Моделирование завершится, когда модельное время будет равно 720 единицам модельного времени, поскольку именно в этот момент времени из генератора выйдет первый транзакт, и счетчик останова обнулится.

#### 4.14 Сводный список СЧА объектов

Таблица 4.14 - Стандартные числовые атрибуты (СЧА) объектов GPSS

Типы объектов	Имя СЧА	Назначение
Системные СЧА (атомарные)	C1	Текущее значение условного времени. Автоматически изменяется в модели и устанавливается в 0 управляющими операторами RESET. Вещественное значение.
	AC1	Текущее значение абсолютного времени. Автоматически изменяется в модели. Устанавливается в 0 под действием оператора CLEAR. Вещественное значение.
	TG1	Текущее значение счетчика числа завершений. Целочисленное значение.
	RN1	Случайное число в интервале 0...999.
Транзакты	P	Значение параметра активного транзакта. Целочисленное, вещественное или строковое значение.
	PR	Приоритет активного транзакта. Целочисленное значение.
	A1	Семейство активного транзакта. Целочисленное значение.
	MP	Транзитное время пребывания в модели активного транзакта. Равно разности текущего значения абсолютного времени и содержимого параметра активного транзакта. Вещественное значение.
	XN1	Номер активного транзакта. Целочисленное значение.
Блоки	N	Общее число транзактов, которые вошли в блок. Целочисленное значение.
	W	Текущее число транзактов в блоке. Целочисленное значение.
Устройства	F	Состояние ресурса. Равно 0, если ресурс свободен, и 1 - во всех остальных случаях. Целочисленное значение.
	FI	Флаг прерывания: 1 - если ресурс находится в состоянии прерывания, 0 - в противном случае. Целочисленное значение.
	FV	Флаг готовности ресурса к использованию: 1 - если готов, 0 - в противном случае. Целочисленное значение.
	FR	Коэффициент использования ресурса в долях тысячи. Вещественное значение.
	FC	Общее число занятий ресурса. Целочисленное значение.
	FT	Среднее время использования ресурса одним занятием. Вещественное значение.



Типы объектов	Имя СЧА	Назначение
Памяти	S	Текущее содержимое памяти. Целочисленное значение.
	R	Число свободных единиц памяти. Целочисленное значение.
	SR	Коэффициент использования памяти в долях тысячи. Вещественное значение.
	SA	Взвешенное по времени среднее содержимое памяти. Вещественное значение.
	SM	Максимальное содержимое памяти. Целочисленное значение.
	SC	Общее количество использовавшихся элементов памяти. Целочисленное значение.
	ST	Среднее время пребывания транзактов в памяти. Вещественное значение.
	SE	Флаг незанятости памяти: 1 - свободна, 0 – занята. Целочисленное значение.
	SF	Флаг заполненности памяти: 1 - заполнена, 0 - не заполнена. Целочисленное значение.
	SV	Флаг готовности памяти: 1 - готова , 0 - не готова. Целочисленное значение.
Ключи	LS	Состояние логического ключа: 1 - установлен, 0 - не установлен. Целочисленное значение.
Очереди	Q	Текущая длина очереди. Целочисленное значение.
	QA	Взвешенная по времени средняя длина очереди. Вещественное значение.
	QM	Максимальная длина очереди. Целочисленное значение.
	QC	Общее число входов в очередь. Целочисленное значение.
	QZ	Число нулевых входов в очередь. Целочисленное значение.
	QT	Среднее время пребывания транзактов в очереди (включая нулевые входы). Вещественное значение.
	QX	Среднее время пребывания сообщения в очереди (без нулевых входов). Вещественное значение.
Таблицы	TB	Среднее значение аргументов таблицы. Вещественное значение.
	TC	Общее число аргументов таблицы. Целочисленное значение.
	TD	Вычисленное среднеквадратичное отклонение для аргументов таблицы. Вещественное значение.
Датчики случайных чисел	RN1 ... RNx	Число, вычисляемое датчиком случайных чисел. Датчик генерирует последовательность равномерно распределенных целочисленных случайных чисел в интервале 0 – 999. При использовании датчика в качестве аргумента функции или объекта в переменной значение будет дробью от 0 до 0.999999.
Функции	FN	Вычисленное значение функции. Вещественное значение.
Переменные	V	Вычисленное значение целочисленной переменной или переменной с плавающей точкой. Вещественное значение.
Булевские переменные	BV	Вычисленное значение булевой переменной. Вещественное значение.
Сохраняемые величины	X	Значение сохраняемой величины. Целочисленное, вещественное или строковое значение.

Типы объектов	Имя СЧА	Назначение
Матрицы сохраняемых величин	MX (a,b)	Содержимое элемента матрицы сохраняемых величин, расположенного в строке a, столбце b. Целочисленное, вещественное или строковое значение.
Числовые группы	GN	Текущее число членов в числовой группе. Целочисленное значение.
Группы транзактов	GT	Текущее число членов в группе транзактов. Целочисленное значение.
Семейства транзактов	A1	Номер семейства активного транзакта. Целочисленное значение.
	MB	Флаг синхронизации: 1 - если транзакт в некотором блоке принадлежит тому же семейству, что и активный транзакт; 0 - в противном случае. Целочисленное значение.
Списки пользователя	SN	Текущее число транзактов в списке пользователя. Целочисленное значение.
	CA	Взвешенное по времени среднее число транзактов в списке пользователя. Вещественное значение.
	CM	Максимальное число транзактов в списке пользователя. Целочисленное значение.
	CC	Общее число входов транзактов в список пользователя. Целочисленное значение.
	CT	Среднее время пребывания транзактов в списке пользователя. Вещественное значение.

Таблица 4.15 - Стандартные логические атрибуты (СЛА) объектов GPSS

Тип объектов	Имя СЛА	Значение
Приборы	NU	Равен 1, если прибор свободен, иначе 0
	U	Равен 1, если прибор занят, иначе 0
	NI	Равен 1, если прибор не прерван, иначе 0
	I	Равен 1, если прибор прерван, иначе 0
	FV	Равен 1, если прибор доступен, иначе 0
	FNV	Равен 1, если прибор недоступен, иначе 0
Памяти	SE	Равен 1, если память пуста (нулевое содержимое), иначе 0
	SNE	Равен 1, если память не пуста (ненулевое содержимое), иначе 0
	SF	Равен 1, если память заполнена, иначе 0
	SNF	Равен 1, если память не заполнена, иначе 0
	SV	Равен 1, если память используется, иначе 0
	SNV	Равен 1, если память не используется, иначе 0
Ключи	LR	Равен 1, если логический ключ "выключен", иначе 0
	LS	Равен 1, если логический ключ "включен", иначе 0

#### 4.15 Задания для самостоятельных работ по созданию имитационных моделей

##### 1. ЛВС

Дано: 1 сервер, 10 рабочих станций (АРМ). Запросы поступают от АРМ на сервер, тот готовит ответ и посылает его на АРМ. Спустя некоторое время обдумывания, АРМ снова посылает запрос на сервер и т.д. (режим замкнутого цикла). Времена обдумывания на АРМ, а также решения на сервере заданы. Сеть обеспечивает в любой момент времени одно и только одно соединение.

Определить: среднее время реакции системы в зависимости от времени решения на сервере (быстродействие сервера). Законы распределения случайных параметров задать самостоятельно.

##### 2. Справочная служба.

Дано: бригада операторов, время обслуживания ответа у каждого одинаковое, заданы параметры потока звонков от клиентов, вопросы разной сложности. Дисциплина очереди: если занято, то клиент уходит.

Определить: какой процент клиентов не обслужен в зависимости от количества операторов в бригаде. Законы распределения случайных параметров задать самостоятельно.

##### 3. Передача в сети с подтверждением.

Дано: сеть из 2-х вершин и 2-х каналов (по одному в каждую сторону), поток пакетов из каждой вершины пуассоновский с подтверждением. Заданы также: пропускные способности каналов; распределение времени обработки полученного пакета в вершине для формирования подтверждения; значение «скользящего окна». Подтверждение поступает в общую очередь при передаче в соседнюю вершину.

Найти: пропускную способность сети, как функцию от размера скользящего окна. Законы распределения случайных параметров задать самостоятельно.

##### 4. Сеть с отказами.

Дано: сеть из 2-х вершин: А и В. Поток пакетов из А в В задан. Пропускная способность канала задана. Задан поток отказов канала и время восстановления отказа.

Найти зависимость времени пребывания пакетов в системе от интенсивности отказов.

Законы распределения случайных параметров задать самостоятельно.

##### 5. Ремонт станков.

Дано: в цеху непрерывно работают 100 станков. Задано распределение времени работоспособности станка до поломки. В цехе есть бригада мастеров по ремонту станков, которая работает с 8 до 17 ч. Задано распределение времени ремонта станка. Каждый станок ремонтируется одним мастером.

Найти: среднее число работающих станков в цеху в зависимости от численности ремонтной бригады. Законы распределения случайных параметров задать самостоятельно.

6. Сеть из 3-х узлов с отказами.

Задана сеть из 3-х узлов: А, В, С. Задан поток пакетов из А. Пакеты поступают в В, если канал АВ работает, иначе по каналу АС в С. Заданы пропускные способности каналов. Задан поток отказов на канал АВ и время восстановления отказа. Считаем канал АС безотказным. Пакет, оказавшийся в канале АВ в момент отказа, считается пропавшим.

Найти: коэффициент полезной загрузки канала АВ и число потерянных пакетов в зависимости от интенсивности отказов. Законы распределения случайных параметров задать самостоятельно.

7. Магазин.

Задан поток покупателей на входе в магазин. В магазине 2 продавца. Время обслуживания одного покупателя задано и одинаково для каждого продавца.

Сравнить по среднему времени пребывания в магазине покупателей для следующих организационных вариантов: - разные очереди к каждому из продавцов. При этом покупатели с равной вероятностью занимают очереди; - одна общая очередь к двум продавцам. Законы распределения случайных параметров задать самостоятельно.

## ГЛАВА 5. СРЕДА МОДЕЛИРОВАНИЯ SIMIO

### 5.1. Объекты

Среда моделирования Simio (© SIMIO LLC [18]) ориентирована на описание потоковых агрегативных систем (см. главу 1). Поэтому Simio поддерживает концепцию объектного моделирования во всех аспектах модельной среды и использует понятие «Умные объекты» (Intelligent Objects). Умные объекты создаются разработчиками и могут быть использованы в разных проектах. Объекты хранятся в библиотеках и позволяют совместное использование. Начинаящий разработчик, возможно, захочет применять уже созданные объекты из библиотек, тем не менее, система разработана для помощи начинающим в создании их собственных умных объектов. Объект может моделировать машину, робота, самолет, клиента, доктора, танк, автобус, корабль или иные вещи, которые потребуются моделировать.

Модель строится путём комбинирования объектов, представляющих физическую трактовку системы. Модель Simio визуально может выглядеть как реальная система. Логический алгоритм и анимация строятся с помощью интуитивно понятного интерфейса. На объект может быть наложена анимация для отображения изменяющегося состояния объекта. Например, погрузчик, который опускает и поднимает свои вилы; робот, который открывает или закрывает захват; танк, поворачивающий свою башню. Модель с анимацией показывает меняющееся изображение системы в динамике. В отличие от других объектно-ориентированных систем моделирования процесс создания объекта достаточно прост и является полностью визуальным. В Simio нет необходимости писать код или скрипт для создания новых объектов.

Процедура разработки объекта в Simio близка к процедуре разработки модели, т.к. на самом деле нет отличия между объектом и моделью. Это основная идея в Simio. Построив новую модель, по определению, мы получим объект готовый к встраиванию в любую другую модель. Например, при объединении двух станков и робота в модель производственной ячейки эта ячейка – сама по себе объект, который может быть импортирован в другие модели. Производственная ячейка – теперь такой же объект, как и станок или робот в отдельности. В Simio невозможно разделить идею создания модели от идеи создания объектов. Каждая модель, которая строится в Simio, автоматически является строительным блоком для создания моделей более высокого уровня.

Каждый объект в Simio имеет свои собственные Процессы, Элементы, Свойства, Состояния и События. Он также имеет Изображение, показывающее как объект будет выглядеть, когда будет вставлен в другую модель в базовом окне другой модели. По этой причине каждый объект в стандартной библиотеке Simio имеет свои собственные Процессы, Элементы, Свойства, Состояния, События и Изображение. Главная модель тоже является объектом, и она тоже имеет Процессы, Элементы, Свойства, Состояния и События.

Когда пользователь работает с несколькими моделями внутри проекта, важно представлять, какая модель является текущей активной моделью. Это можно выяснить, посмотрев в панель навигации, расположенной в верхнем правом углу интерфейса и найдя выделенное название модели. Это и будет активной моделью. Это означает, что окна, расположенные в главной части интерфейса являются окнами, ассоциированными с этой активной моделью. Для просмотра окон другой модели, нужно сменить модель, щелкнув на другой модели в окне навигации, и вы увидите иной набор вкладок (окон) в главной части интерфейса.

Хотя каждый объект в стандартной библиотеке имеет свои собственные Процессы, Элементы, Свойства, Состояния и События, пользователь может не увидеть эти компоненты объекта. В Simio эти данные скрыты, чтобы защитить пользователя от внесения случайных изменений в стандартные объекты. Стандартные объекты были сделаны для стандартной функциональности, которые удовлетворяют большинству задач моделирования, поэтому скрыв стандартные компоненты, мы получили более простую рабочую среду. Если пользователь захочет посмотреть Процессы, Элементы, Свойства, Состояния и События стандартного объекта, то ему следует переместить объект из Стандартной Библиотеки в его собственную библиотеку. Как только объект оказывается в пользовательской библиотеке, окна настроек модели и поведения этого объекта становятся доступными для изучения.

#### Понятие и описание объекта

Simio использует объектный подход для моделирования, поэтому модели строятся комбинированием объектов, которые представляют физические компоненты системы. Объект – самодостаточный конструкторский компонент моделирования, который определяет характеристики, данные, поведение, пользовательский интерфейс и анимацию. Объекты – самые популярные компоненты для построения моделей. Набор объектов для общих целей поставляется с Simio в Стандартной Библиотеке.

Объект имеет своё собственное поведение, которое отвечает на события в системе, так как определено его внутренней моделью. Например, модель производства строится из объектов, которые представляют станки, конвейеры, погрузчики и работников, а госпиталь может быть смоделирован комбинацией объектов, которые представляют персонал, палаты пациентов, кровати, медицинские устройства и операционные. Помимо использования стандартной библиотеки объектов при

проектировании модели, вы можете организовать свои собственные библиотеки объектов, предназначенные для специального применения. Также можно расширять объекты стандартной библиотеки, изменяя алгоритм логики процессов.

Особым внутренним свойством Simio является использование тройственной концепции объекта, которая разделяет объект на описание объекта, экземпляр объекта и реализацию объекта.

Описание объекта – это либо объект библиотеки, который поставлялся с Simio, либо объект, созданный разработчиком. Описание объекта – это определение как объект будет вести себя в модели с использованием Процессов, Событий, Состояний и Свойств для определения взаимодействий с Транзактами или другими объектами. Описание объекта указывает на поведение этого объекта, и оно используется всеми экземплярами данного объекта во всех моделях.

Описания объектов хранятся в библиотеках: стандартной, пользовательской или библиотеке проекта. Новые описания объектов могут быть созданы при проектировании новой модели. Дополнительно описание нового объекта может быть создано путем извлечения (аналог наследования) какого-то существующего объекта в библиотеки.

### Экземпляр объекта

Экземпляр объекта – это простое воспроизведение описания объекта в объекте-родителе, т.е. модели или определении другого объекта. Это то, что вы получаете, когда размещаете описание в базовом окне модели.

Каждый создаваемый по описанию объекта экземпляр отличается собственными значениями Свойств или созданием процессов внутри экземпляра. Например, каждый Сервер имеет различное «Время обработки». Другой пример - некоторые Трансферные узлы могут иметь значение «Ждать Транспорт» установленное в Да, некоторые в Нет. Экземпляр данных в свою очередь совместно используется всеми реализациями объекта.

Обычно моделирующие системы используют понятие «атрибут» или «переменная», но в Simio объекты используют близкие по смыслу понятия «Свойство» (Property) и «Состояние» (State) для передачи информации между объектами и получения выходных данных. Свойства и Состояния добавляются к объектам через интерфейс окна Определений (DefinitionWindow).

### Типы объектов

Таблица 5.1. Основные типы объектов

Тип	Чей наследник	Описание
Умный объект (Intelligent Object)	Нет	Базовый объект с возможностью изменений в размерах, независимостью и поведением по определённому графику изменений (плану).
Фиксированный (Fixed)	Intelligent Object	Обычно используется для отображения целой системы (например, фабрика) или подсистемы, имеющей фиксированное положение (например,

		станок, оборудование, рабочее место). Фиксированный объект имеет стационарное положение в базовом окне модели.
Агент (Agent)	Intelligent Object	Добавляет поведение для моделирования объектов, которые могут быть динамически созданы и уничтожены, могут перемещаться в непрерывном или дискретном пространстве (на сетке) и которые могут следить, обнаруживать и перехватывать другие объекты. Этот тип модели полезен при агентном подходе к моделированию, когда большое число (например, несколько тысяч) независимо действующих агентов взаимодействуют для создания целого поведения системы. В текущей версии Simio пользователь не может добавлять новые объекты класса Агент. Тем не менее, как Транзакты, так и Транспортеры наследуются от объекта класса Агент.
Транзакт (Entity)	Agent	Определяет поведение объектов моделирования, которые подчиняются рабочему процессу в системе, включая возможность использовать сеть связей для перемещения между объектами, возможность посещать, входить и выходить в другие объекты через узлы, и возможность быть забранным, перенесенным или выброшенным транспортными объектами. Объект Транзакт может быть динамически создан и уничтожен, может входить и выходить в Фиксированные (Fixed) объекты. Транзакты могут иметь несколько графических символов. Транзакт в модели (ModelEntity) по умолчанию автоматически добавляется вместе с первым Фиксированным объектом в проекте.
Транспортер (Transporter)	Agent	Объект Транспортер – специальная сущность, которая может забирать объекты Транзакты, перевозить эти объекты через сеть связей или в свободном пространстве, доставлять и выгружать их у конечной цели.
Связь (Link)	Intelligent Object	Настраивает перемещение в модели фиксированных объектов, для задания путей движения транзактов / транспортеров. Объект Связь имеет длину, которая может быть разделена на равные доли (ячейки) и должна иметь начальный (стартовый) узел и конечный узел, и может входить в состав одной или нескольких сетей. Объект Связь перемещает транзакты по пути, определенном линией, соединяющей два объекта типа Узел в окне модели. Узел при отрисовке подсвечивается красным.
Узел (Node)	Intelligent Object	Определяет поведение в модели на пересечении между объектами Связь или входными/выходными точками для использования в фиксированном объекте. Транзакты могут быть забраны или выгружены Транспортером в Узле. Пользователи могут управлять логикой узла, движения по модели сети и указывать точки загрузки/выгрузки.





Рисунок 5.1 - Граф наследования классов объектов в Simio

В Simio реализовано шесть основных классов объектов, а также есть несколько способов создания новых объектов. Объекты могут быть созданы из эскиза, для этого пользователю необходимо выбрать один из классов объекта или воспользоваться API. Более общий и простой способ создания нового объекта – это копирование (создание подкласса) из стандартной библиотеки Simio и использование стандартной логики в своих собственных нуждах. Можно добавлять модели из других библиотек в подклассы и использовать в качестве основы для создания нового объекта. Как только новый объект создан внутри проекта, его можно добавлять в модели путем перетаскивания из библиотеки проекта в базовое окно модели. Библиотека проекта расположена под стандартной библиотекой объектов с левой стороны окна Simio.

Понятия «модель» и «объект» используются во этом документе как синонимы. Это возможно, поскольку в Simio модель также является объектом, который может быть использован как часть другой модели. И сам объект является моделью, потому что у него есть своя собственная внутренняя логика и в его составе могут быть другие объекты, поэтому на него тоже можно ссылаться как на модель. Таким образом, в принципе модель и объект – это фактически одно и то же.

Главная модель внутри проекта является обычно объектом класса Fixed (фиксированный объект). Самый простой способ добавить свой логический алгоритм в модель: использовать стандартную библиотеку объектов в новом объекте класса Fixed. Это делается в ходе создания новой модели и расположения объектов из стандартной библиотеки в базовом окне модели. Можно создать большое разнообразие вариантов модельных систем с помощью размещения типовых объектов, таких как Source, Server, Combiner, Separator, Sink, и соединения их различными типами связей.

Определение объекта содержит пять основных компонентов: свойства, состояния, события, внешний вид и логика. Существует три подхода к определению логики модели для объекта. Первый подход – создать подкласс существующего определения объекта и затем заменить/расширить его поведение, настроив его свойства

и логику. Этот подход обычно используется, когда есть существующая модель, чье поведение похоже на желаемый объект, и которая может быть настроена с помощью имен, описаний и поведения, чтобы удовлетворить требованиям создаваемого объекта. Второй подход – создать модель иерархически, используя модельные объекты. Этот подход можно расширить путем использования подключаемых (add-on) процессов для определения особенностей поведения внутри объекта. Этот метод обычно используется для построения компонентов более высокого уровня, например, таких как производственный центр, включающий два станка, рабочего и инструменты. Третий, более гибкий подход – создание определения поведения объекта с чистого листа, используя процессы поведения модели. Этот подход был использован для создания стандартной библиотеки.

Simio работает со стандартными объектами, которые расположены в стандартной библиотеке (Standard Object Library) и их поведение может быть расширено при помощи подключаемых (add-on) процессов. Если необходимое поведение не может быть создано за счет подключаемых процессов, пользователю, возможно, потребуется доработать внутреннюю логику объекта. Это делается при помощи создания подкласса объекта и последующей замены процессов. Объект можно отправить в подкласс в окне проектов (Project window). Как только объект занесен в подкласс, процесс должен быть переписан путем выбора иконки Override на панели инструментов для доступа к изменению стандартного описания.

## 5.2. Процессы

Процесс – это последовательность действий (например, назначение состояния, задержка по времени, выделение ресурса и т.д.), которая может длиться во времени и менять состояние модели. Процесс в Simio состоит из шагов, элементов и токенов. Токены проходят через процесс, выполняя шаги, которые изменяют состояние одного или нескольких элементов. В Simio есть автоматическая функция, создающая потоки процессов. Процессы создаются и изменяются в окне процессов в закладке Processes.

Процесс может быть либо включенным (разрешенным), либо выключенным (запрещенным). Свойство процесса InitiallyEnabled позволяет пользователю указывать, включать ли процесс при инициализации системы. Процесс имеет свойство Enabled, которое показывает включен ли процесс в данный момент. Отключенный процесс игнорирует любые попытки его запуска. Процесс может быть включен с использованием шага Assign для установки состояния ProcessName.Enabled в 1, где ProcessName – имя процесса. Отключение процесса происходит установкой этого состояния в 0. Любые попытки запуска отключенного процесса будут проигнорированы (например, Execute step). Любые триггерные события процесса будут отменены, если он отключен. Однако, если процесс отключен, но в нем есть активные работающие токены, эти токены продолжат свою работу до завершения. Отключенный процесс (или процесс без логики) при запросе вернет возвращаемое значение

ReturnValue (например, если процесс OnEvaluating опрашивается системой). ReturnValue – это состояние, связанное с процессом, которое отдает возвращаемое значение стандартного токена данного процесса. При этом сам процесс не может быть изменен во время работы модели.

Существует несколько особых процессов внутри стандартных объектов Simio, которые называются «процесс принятия решения» (Decision Process, процесс ПР). Эти процессы используются в принятии решений потока событий, т.к. исполняющей системе требуется узнать из модели определенные значения, и для поиска ответа моделирующая система запускает эти процессы. Такие процессы могут содержать шаги Assign\_Decide, Search и Execute. Если исполняющая система сталкивается с другим типом шага при запуске процесса ПР, она возвращает ошибку. Если процесс ПР содержит шаг Execute, любые дополнительные вызванные процессы тоже должны содержать только шаги Assign\_Decide, Search и Execute. Так как процесс ПР должен обработать до своего полного выполнения без прерываний, он игнорирует отладочные прерывания (Breakpoints), приостановки и пошаговое выполнение.

Процессы используются для изменения поведения существующих объектов или для создания новых объектных определений. Если пользователю необходимо незначительно изменить поведение того, что предоставляется объектами стандартной библиотеки, модификация производится с помощью настройки процессов Simio. Вторая основная роль для моделирования процессов в Simio – это создание новых объектов. Описание объекта – это модель того, как экземпляр объекта должен себя вести.

Существует несколько типов процессов в Simio. Стандартные процессы автоматически запускаются Simio в определенных моментах работы логики модели. Процессы ПР – это стандартные процессы с мгновенным исполнением (0-time Standard Process), которые рассчитывают возвращаемое значение для принятия решения. Процессы событий (Event-triggered Process) выполняются каждый раз в момент, когда происходит определенное указанное пользователем событие. Дополнительные процессы (Add-on Process) могут быть встроены в объект в определенных точках логики модели.

#### Подключаемые процессы (Add-On Process)

При добавлении объекта из стандартной библиотеки в базовое окно модели можно заставить объект запускать дополнительные процессы в определенные моменты логики поведения. Подключаемый процесс (Add-On Processes) – логический процесс, который создается для выполнения действий, таких как назначение переменной состояния, захват ресурса, получение информации из таблицы и т.д.

В целом, процесс может ссылаться только на элементы, которые находятся в пределах его владельца. Объект ничего «не знает» о модели, в которой он находится, так что любой процесс, определенный внутри объекта, имеет те же ограничения.

Например, процесс, определенный внутри объекта Server, может ссылаться только на данные, которые были определены внутри данного объекта Server. Один из способов, с помощью которых объект и его процессы могут взаимодействовать с другими объектами – это указание объектом свойств, которые используются для передачи информации из внешней среды. Например, объект Server имеет свойство Process Time, которое может использовать или «передать» такие данные, как таблицы и функции, определенные вне объекта Server.

Подключаемые процессы определяются не только в пределах базового объекта (например, Server). Подключаемые процессы являются частью модели и находятся в ее пределах. Например, если обработка процесса определена для Server1, который находится в модели Factory, предел действия этого процесса - модель Factory. Важно отметить, что код, определенный в подключаемом процессе при перемещении в описание объекта, может перестать работать, т.к. его пределы изменятся от модели к пределам определения самого объекта.

Процесс может быть построен так, что вся логика, которая должна быть выполнена в данный момент исполнения модели, будет выполняться внутри определенного процесса. Однако в некоторых случаях части логики в разных точках модели могут быть похожи. Шаг Execute может быть использован для того, чтобы «переместиться» от текущего процесса к другому процессу. При использовании шага Execute новый токен входит в процесс и проходит по логическим шагам алгоритма. Внутри исходного процесса (в котором был вызван шаг Execute), исходный токен может продолжать работу, а может ждать окончания выполнения вызванного процесса.

Если используется произвольный токен с произвольными состояниями, состояния получающего токена в вызванном процессе получают такой же набор (подобно передаче аргументов функции). Если у токена установлено возвращаемое значение ReturnValue в исполняющем процессе, то это же значение будет установлено (возвращено) токеноу, который начал выполнение (подобно возврату значения от функции). Это позволяет пользователю передавать информацию между процессами (подобно работе с функциями).

Подключаемые процессы – частный случай процессов. Подключаемые процессы, как правило, определяются пользователем для помощи в настройке поведения объекта или добавлении особого поведения в модель. Объекты в стандартной библиотеке содержат триггеры подключаемых процессов (Add-On Process Trigger). Если подключаемый процесс определен в свойстве триггера объекта, то объект при запуске определенного события будет выполнять подключаемый процесс.

#### Триггеры подключаемых процессов в стандартном объекте

Для создания подключаемого процесса нужен объект и триггер процесса. Выберите объект в базовом окне и затем раскройте список триггеров подключаемых процессов. Кликните на триггер, который вы хотите использовать и выберите «Создать новый» (Create New) из выпадающего списка.

Для добавления этапов в процесс кликните кнопку Processes, и в окне появится новый Add-On Process. Кликните и перетащите этап из списка в процесс между начальным и конечным этапом. Функциональность этапа может быть настроена путем изменения свойств в окне свойств в нижнем правом углу окна процессов.

#### Пример подключаемого процесса

Объект Источник имеет четыре триггера подключаемых процессов:

- Initialized (Инициализирован) срабатывает, когда объект Source инициализирован;
- Creating Entities (Создание транзакта) срабатывает, когда объект Source вот-вот создаст поступление одного или нескольких транзактов;
- Created Entity (Создан транзакт) срабатывает, когда транзакт создается объектом Source;
- Exited (Вышел) срабатывает, когда транзакт покинул объект Source.
- 

#### Итоговая статистика

Расчет переменных итоговой статистики (Tally) доступен для объектов Узлов BasicNode и TransferNode, а также для внешних узлов объектов Source, Sink, Server, Combiner, Separator и Workstation. Переменная итоговой статистики может быть определена, когда объект входит в узел (On Entered) и когда объект выходит из узла (On Exited).

При помощи редактора свойств пользователь может выбирать ситуации, при которых будет подсчитана итоговая статистика. Свойство Tally\_If определяет, когда будет подведен итог: когда транзакт входит/выходит из узла или когда транспортер (транспорт или исполнитель) входит/выходит из узла. В качестве альтернативы итог может быть подсчитан, когда любой объект входит ('No\_Condition') или когда встречаются специальные условия ('Custom\_Condition'). Множественные итоги могут подсчитываться с помощью определения множества строк в группе.

Для выполнения шага Tally внутри окна процессов, любые итоги, которые подводятся в разделе объекта Tally Statistics, должны иметь связанный элемент Tally Statistic, определенный внутри панели элементов в окне определений (Definitions). В примере на рисунке должны быть определены имена элементов итоговой статистики 'Blue\_In\_Server' и 'Total\_In\_Server'.

В любом объекте типа узел итоговая статистика On\_Entered рассчитывается сначала для подключаемого процесса и затем назначается новое значение ресурса. Итоговая статистика On\_Exited рассчитывается сначала для освобождаемого состояния ресурса и затем для подключаемых процессов Exited.

### **5.3. Очереди в Simio**

Очереди (списки транзактов, ожидающие обработки) присутствуют во многих местах в Simio. В модели обеспечивается большое разнообразие работы с очередями в

различных частях объектов. В объектах стандартной библиотеки есть очередь входного буфера, очередь обработки и очередь выходного буфера.

Создание очереди в Simio возможно либо с помощью варианта отделенной очереди (Detached Queue), которая не будет привязана к какому-либо объекту, либо очереди, привязанной к определенному объекту. Присоединенная (Attached Queue) очередь будет двигаться в окне вместе с объектом. Иконка, используемая для создания отделенной очереди, находится на вкладке анимации (Animation).

Для создания отделенной очереди необходимо снять выделение со всех объектов и выбрать иконку Detached Queue на вкладке Animation. Клик левой кнопкой в базовом окне модели добавляет промежуточные вершины линии очереди, клик правой определяет замыкающую вершину.

Для создания присоединенной очереди необходимо выбрать объект, к которому должна быть привязана очередь, и выбрать иконку Queue на вкладке Symbols. После создания очереди в базовом окне модели, выберите подходящее состояние очереди QueueState в окне свойств (Properties) на вкладке определений (Definitions). Состояния QueueStates отсортированы по алфавиту в соответствии с объектом, к которому они привязаны.

При создании присоединенной очереди Simio предоставляет ярлык. Если для очереди выбран объект, вместо клика по иконке Queue, необходимо нажать на стрелку вниз для показа выпадающего меню Quick Create. Это меню предоставляет доступные пользователю способы создания очереди. Если пользователь выбирает один из вариантов данного меню, а потом создает очередь через базовое окно модели, то автоматически заполнятся свойства для данной очереди. Например, если пользователь выбирает InputBuffer.Contents из выпадающего меню и потом создает очередь через базовое окно модели, свойство «AttachedTo» будет равно имени выбранного объекта, а свойство QueueState будет равно значению InputBuffer.Contents. В результате в очередь будут запущены транзакты, которые в данный момент находятся в InputBuffer выбранного объекта.

Транзакты могут быть удалены из определенных очередей при выполнении в процессах шага Remove.

#### Создание очередей в узлах

При размещении узла в Simio его очереди парковки (Parking Queues) создаются автоматически, даже если в модели нет транспорта. Это свойство помогает при использовании транспортера (Transporter) и избавляет пользователя от ручного создания очереди для проверки, припаркован ли к узлу транспорт. Если выбрать узел, кликнув его, (включая узлы Input и Output, привязанные к стандартным объектам), можно увидеть, что будет выделена иконка Parking Queue. При нажатии на эту иконку будет отключено автоматическое создание Parking Queue для данного узла. Simio будет предоставлять возможность создания очереди, позволяя пользователю создать

присоединенную очередь для данного узла, включая Parking Queue, если пользователя не устраивает стандартное расположение Parking Queue.

Стандартное поведение очереди – это линейная очередь. Это значит, что объекты будут появляться и располагаться в соответствии с созданной линией. Опции None и Inline вкладки Appearance предоставляют два типа линейной очереди по отношению к транзактам в очереди. При выборе опции Inline объекты всегда выстраиваются по направлению линии очереди. В обоих типах линейной очереди в момент, когда объект ее покидает, все остальные объекты продвигаются вперед. Число объектов визуальнo отображающихся на линии очереди зависит от размера линии очереди и размера объектов.

Линейная очередь может быть изменена в виде точечной очереди, если ее выделить и выбрать кнопки Point (Oriented Point) на вкладке Appearance. При использовании обоих типов точечных очередей объекты визуальнo показываются в очереди на вершинах линии очереди. При изменении типа очереди на точечную, стрелки будут привязаны к каждой вершине очереди. При использовании точечной очереди при ее выделении одна из конечных стрелок становится вершиной. Она может быть перемещена для изменения ориентации всех остальных вершин линии очереди. При использовании очереди Oriented Point при ее выделении каждая из вершин на линии имеет направление соответствующей вершины, причем ориентацию можно направить в любую сторону индивидуально.

При использовании обоих типов Point или Oriented Point пользователь может выбрать опцию Keep In Place. Это позволит объектам в очереди оставаться в той точке, в которой они были при постановке в очередь, вместо того, чтобы продвигаться вперед к следующей доступной позиции, когда один из объектов будет удален из очереди.

При использовании любых типов очереди могут быть добавлены или удалены дополнительные вершины линии с помощью кнопок Add Vertex и Remove Vertex на вкладке Appearance. Чтобы добавить дополнительную вершину в конец очереди необходимо выделить очередь и нажать Add Vertex. Вершину можно расположить в любом месте базового окна модели. Если новая вершина размещается между двумя другими, то она будет добавлена в промежуток между ними. Если новая вершина добавляется вне имеющейся линии, то она станет замыкающей вершиной очереди. Для удаления вершины выделите ее и нажмите Remove Vertex, и линия очереди будет обновлена.

## **5.4. Транзакты и токены**

### Токены

Токен выполняет шаги (действия) в процессах Simio. Токен может содержать несколько состояний, определенных пользователем для передачи информации от шага к шагу. Токен может ссылаться на ассоциированный с ним объект, такой как транзакт или родительский объект. Токен находится внутри процесса Simio – он создается в

начале процесса и уничтожается в его конце. Следовательно, токен создается в первый момент исполнения процесса. Новый токен можно создать из сегмента Create шага Create и сегмента Found шага Search. Тип токена, создаваемого внутри процесса Simio, определяется свойством процесса Token Class Name.

В Simio предоставляется стандартный токен (Token), который можно использовать в большинстве процессов. Новый токен следует добавлять только, если необходим токен с одним или несколькими произвольными состояниями.

### Транзакты

В Simio транзакты являются частью модели объекта и могут иметь свое собственное «разумное» поведение. Они могут принимать решения, отклонять запросы, «отдыхать» и т.д. Транзакты содержат объектные определения (свойства, состояния) так же, как и остальные объекты модели. Транзакты могут быть динамически созданы и уничтожены, могут перемещаться по сети связей или узлов, проходить через 3D-пространство и перемещаться внутрь фиксированных объектов или из них. Например, транзакты в модели могут представлять клиентов и участников работы. Транзакты не проходят через внутренние процессы, как это делают токены. Перемещение транзактов в объект или из него может создавать определенное событие, которое может запускать какой-либо процесс. В момент исполнения процесса создается токен, который проходит через все шаги процесса.

У транзактов есть физическое расположение внутри базового окна, и они могут находиться в пространстве (Free Space), в стадии обслуживания в объекте (Station), на линии связи (Link) или в узле (Node). Примеры позиций объектов в Simio: ParkingStation в узле (TransferNode1.ParkingStation), InputBuffer в объекте Server (Server1.InputBuffer), Processing Station в объекте Workstation (Workstation1.Processing).

### Различия токенов и транзактов

В свете концепции, изложенной в главе 1, транзакты это сетевые инициаторы, а токены - блочные инициаторы.

Рассмотрим транзакт, который входит в стандартный объект Server, чтобы понять роль транзактов и токенов в Simio.

В примере показаны процессы, которые запускаются двумя триггерами Processing (обработка) и Processed (обработан) в объекте Server. Экземпляр транзакта, который находится в базовом окне, попадает на входной узел (InputNode) объекта Server. Он переходит в InputNode и далее сразу перемещается в позицию InputBuffer объекта Server (предполагается, что в InputBuffer имеется необходимая доступная емкость). Если необходимая емкость доступна в объекте Server, она предоставляется транзакту и триггер Processing запускает процесс Process1. Создается токен, ассоциированный с транзактом, в процессе Process1 и проходит через шаг Assign. После того, как Process1 завершается, объект Транзакт перемещается в позицию Processing объекта Server. Когда в объекте Server заканчивается время на обработку, а емкость стадии обработки Server еще не освобождена, запускается процесс Process2.



Объект Транзакт остается в позиции Processing объекта Server, создается новый токен внутри процесса Process2. Этот токен проходит через шаг Assign процесса. Токен завершает свою работу, когда достигает конца процесса Process2. В этой точке Транзакт перемещается из позиции Processing объекта Server в позицию OutputBuffer (предполагается, что в OutputBuffer имеется необходимая доступная емкость). Далее транзакт перемещается в выходной узел (OutputNode) объекта Server, где уже направляется в сеть с помощью логики узла Output. Как видно из примера, транзакт в базовом окне перемещается между позициями и узлами, а токены, связанные с этим транзактом, создаются для прохода через процессы и выполнения логики шагов.

Важно заметить, что внутри логики Server шаг Execute используется для каждого дополнительного процесса. Внутри шага Execute находится действие WaitUntilComplete, таким образом, когда в процессе создается токен, ассоциированный с транзактом, транзакт остается физически на одном месте во время всего процесса, а не только при использовании шага Delay.

#### Транзакт по умолчанию (Model Entity)

Один транзакт (ModelEntity) автоматически создается в новой библиотеке проектов (Project Library). Объект ModelEntity автоматически добавляется в ваш проект. Пользователю не обязательно перетаскивать его в базовое окно модели. Транзакт под названием DefaultEntity уже является частью проекта и, следовательно, будет создан объектом Источник (Source) или при помощи шага процесса «Создать» (Create Step). В то же время, если вы хотите изменить символ транзакта или, например, изменить любое свойство, вам нужно перетащить ModelEntity в базовое окно из библиотеки проектов. Дополнительные экземпляры этого класса тоже можно перетащить в модель.

#### Маршрутизация и перемещение транзактов

Транзакты в Simio маршрутизируются несколькими различными способами, например, по последовательности, движению к определенным узлам, следованию вдоль путей и по их выборочному весу для определения, куда двигаться дальше. Маршрутизация транзакта, как правило, определяется трансферными узлами (Transfer Nodes), через которые проходит транзакт. Свойство Entity Destination определяет, каким образом транзакт будет перемещаться, когда он покинет узел.

Стандартный транзакт (ModelEntity) имеет унаследованный скоростной режим, называемый "Movement". Он возвращает общее линейное расстояние, которое этот объект проходит в свободном пространстве или по текущей сетевой связи. У него есть определенные параметры состояния, такие как:

- Начальное значение состояния (Initial State Value)
- Номинальное значение (Rate Value)
- Ускорение (Acceleration)
- Продолжительность ускорения (Acceleration Duration)
- Заголовок (Heading)

- Шаг (Pitch)
- X, Y, Z координаты.

Можно увидеть информацию о движении транзакта `ModelEntity` во время работы, обращаясь к таким свойствам как `Movement.Rate`, `Movement.X`, `Movement.Y`, `Movement.Z`, `Movement.Pitch` и `Movement.Heading`. Эти функции будут возвращать текущее значение, например, `Movement.X` вернет текущее положение транзакта вдоль оси X.

Сеть в модели представляет собой набор из одной или нескольких связей, по которым перемещаются транзакты. В Simio можно определить столько сетей, сколько нужно, и объект Связь может быть частью нескольких сетей. Этот подход важен для моделирования ситуаций, когда различные транзакты перемещаются по собственным сетям, но разделяют общий путь, например, рабочие и вилочные погрузчики, имеющие общий проезд. Для перемещения по сети транзакт должен быть привязан к конкретным сетям, по которым разрешено «путешествовать». Совокупность всех связей автоматически присваивается специальной сети под названием Глобальная сеть (Global). По умолчанию все транзакты назначаются для перемещения по Глобальной сети (т.е. они могут перемещаться по любой связи). Транзакт может изменить свою сеть перемещений в любое время в процессе моделирования.

#### Ассоциированный и родительский объекты

Взаимосвязь объектов поясним на примере процесса 'OnEnteredProcessing'. Процесс включается триггером, когда транзакт перемещается в стадию обработки (Processing) объекта Server. Когда транзакт попадает в стадию, в точке Begin создается токен. Он работает от имени транзакта. Далее, токен перемещается к шагу первого процесса и выполняет определенное действие по алгоритму. В данном примере первый шаг – Decide. Токен перемещается от шага к шагу за нулевое время.

Ассоциированный объект – это объект, который включил выполнение определенного процесса. В данном примере Ассоциированный объект – это транзакт, так как он задействует процесс при его перемещении в стадию Processing объекта Server. Объект Транзакт останется в стадии Processing, пока токен не пройдет все шаги в процессе. Другими словами, шаг Delay действительно удерживает токен от перехода на следующий шаг, но он не задерживает транзакт от выхода из позиции. Транзакт должен ожидать завершения выполнения процесса токеном.

Родительский объект – это объект, в котором дополнительный процесс был создан и будет реализоваться в модели. Так как процесс OnEnteredProcessing является частью объекта Server, то Server – это родительский объект для процесса OnEnteredProcessing. Чтобы понять, какой объект является родительским объектом, надо посмотреть какой объект необходимо выделить (Select), чтобы увидеть процесс. Для того, чтобы изменить или показать подробности процесса OnEnteredProcessing, необходимо создать подкласс объекта Server и открыть окно процессов этого объекта.

Следовательно, для всех дополнительных процессов чаще всего родительским процессом является основная модель, так как дополнительные процессы создаются внутри основной модели и могут быть просмотрены или изменены в окне процессов модели.

### 5.5. Стандартная библиотека объектов

Система моделирования Simio поставляется со стандартной библиотекой объектов, включающей 15 встроенных определений объектов, используемых для моделирования различных систем.

Таблица 5.2 – Библиотека объектов

Название	Класс	Описание
Источник (Source)	Фиксированный	Создает транзакты, которые появляются в системе
Поглотитель (Sink)	Фиксированный	Уничтожает транзакты и записывает статистику
Сервер (Server)	Фиксированный	Моделирует многоканальный процесс обслуживания с входной/выходной очередями
Ресурс (Resource)	Фиксированный	Создает (моделирует) ресурс, который может быть использован другими объектами
Комбинатор (Combiner)	Фиксированный	Объединяет транзакты в группы
Разделитель (Separator)	Фиксированный	Разделяет группы на транзакты
Рабочая станция (Workstation)	Фиксированный	Моделирует рабочую станцию с тремя стадиями: установка, обработка и демонтаж
Транспорт (Vehicle)	Транспортер	Перемещает транзакты между фиксированными объектами
Работник (Worker)	Транспортер	Перемещает транзакты между фиксированными объектами и обрабатывает транзакты в фиксированном объекте
Основной узел (Basic Node)	Узел	Простое пересечение связей
Трансферный узел (Transfer Node)	Узел	Пересечение, где транзакты устанавливают направления и могут ждать Транспортера
Соединитель (Connector)	Связь	Первичное соединение между двумя узлами
Путь (Path)	Связь	Траектория, по которой перемещение транзакта зависит от его скорости
Временной путь (Time Path)	Связь	Путь с указанным временем прохождения
Конвейер (Conveyor)	Связь	Конвейер с накоплением или без накопления

Большинство объектов библиотеки, такие как транзакты (Entities), источники (Source), поглотители (Sink) и серверы (Server) имеют свойство «размер». Оно обеспечивает связь между отображаемым (мнимым) и физическим размерами в модели (если необходимо). Узлы являются исключениями и не имеют никакого физического размера в модели. Для прохождения узла обычно не требуется времени (хотя время может потребоваться для выполнения определенного логического алгоритма, указанного в узле).

В стандартной библиотеке время обработки задержки объектов сервера, комбинатора и сепаратора помечены как прерываемые. Процесс в этих объектах, осуществляющих обработку задержки, называется OnEnteredProcessing. Таким образом, чтобы прервать процесс обработки в сервере, комбинаторе или сепараторе, можно использовать шаг Прерывание (Interrupt), чтобы прервать процесс 'OnEnteredProcessing' целевого объекта.

#### Основной узел (Basic node)

Основной узел является простым узлом для поддержки маршрутов между связями. Большинство объектов, поддерживающих входящие соединения, имеют связанный (встроенный) основной узел. Основные узлы используются в качестве исходных и/или конечных точек ссылок. Они также могут использоваться в качестве точки пересечения для транспортных сетей. Триггеры процесса могут реализовать дополнительную логику, которая должна быть выполнена при использовании основного узла.

#### Трансферный узел (Transfer node)

Трансферный узел является логически сложным узлом, который поддерживает подключение к сети, а также выбор места назначения, пути и устройства передачи. Большинство объектов, поддерживающих исходящее соединение, имеют связанный (встроенный) трансферный узел. Триггеры дополнительного процесса допускают дополнительную логику при использовании трансферного узла.

#### Источник (source)

Источник является объектом, который позволяет создавать (порождать) транзакты с определенной интенсивностью, при прибытии определенного события или по исходу определенного события. Источник имеет выходной буфер (Output Buffer), в котором хранятся транзакты до покидания источника через узел (рисунок 5.2). Из данного узла транзакты выходят по связям до места, куда их направляет существующая логика.

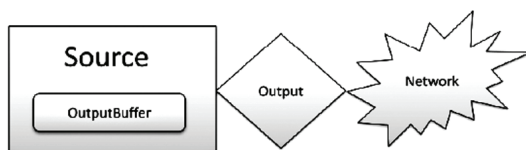


Рисунок 5.2. Схема объекта Source

Присвоение свойств может быть произведено при выходе из Источника. Триггеры дополнительного процесса позволяют настроить дополнительную логику использования Источника.

### Сервер

Сервер используются для моделирования сервисных операций с ограничениями. Основной объект Сервер имеет два прикрепленных объекта: входной узел (Basic node) и выходной узел (Transfer node).

Объект Сервер имеет три позиции: InputBuffer (Входной буфер), Processing (Процессор) и OutputBuffer (Выходной буфер) (рисунок 5.3).. Прибывший транзакт обычно приходит из маршрутной сети через узел Input (Вход) во входной буфер, обрабатывается в процессоре, и затем ожидает в выходном буфере, чтобы пройти через узел Output (Выход) и попасть в маршрутную сеть. Отметим, что сеть может блокировать попытку перемещения из выходного узла в нее (например, когда в канале передачи может возникнуть затор и не будет свободного места). Аналогично, когда Сервер заполнен, он может блокировать сеть на входной стороне.

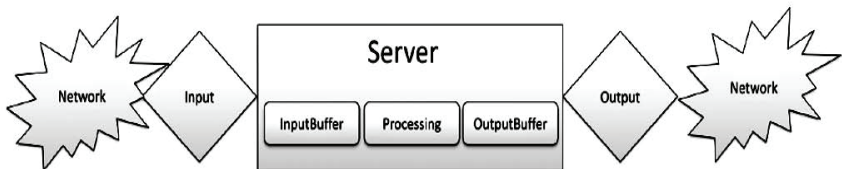


Рисунок 5.3. Схема объекта Server

И у входного буфера, и у выходного буфера можно установить емкость равную нулю, что приведет к исключению их из логики. Следовательно, если для выходного буфера установлена емкость равная нулю, то транзакт, обрабатываемый в процессоре, подождет в нем перемещения в маршрутную сеть через выходной узел. Точно также, если для входного буфера установлена емкость равная нулю, то транзакты, которые приходят из сети через входной узел, должны переходить сразу в процессор на Сервере. Если процессор заполнен, то он будет блокировать попытки транзактов попасть на Сервер.

Всякий раз, когда освобождается место в процессоре, следующий транзакт для обработки берётся из входного буфера. Входной буфер ранжирует на основе свойства Правило ранжирования (Ranking Rule), в котором может быть указано FIFO («первым пришел, первым ушел»), LIFO («последним пришел, первым ушел»), SVF («сначала маленькие значения») или LVF («сначала большие значения»). В последних двух случаях укажите выражение, на котором будет основываться ранжирование – например, можно ранжировать очередь, основываясь на времени обработки или сроке доставки. Если не указано никакое динамическое правило выбора (Dynamic Selection

Rule), тогда для обработки берётся первый транзакт в статически ранжированной очереди. Причём если динамическое правило выбора указано, то все транзакты в очереди выбираются для обработки согласно этому правилу. Такое динамическое ранжирование - гибкий и мощный инструмент, поскольку каждый раз при принятии решения оно пересматривает выражение для каждого транзакта. Например, выражение, вычисляющее запоздалый транзакт (сравнивая его срок с текущим временем), будет правильно использовать текущее время, принимая решение, а не время прибытия в очередь.

Свойство TransferInTime (Время на перемещение) на Сервере указывает время, которое понадобится транзакту, чтобы переместиться внутрь Сервера. Одновременно может перемещаться только один транзакт, и обработка транзакта может не начаться до тех пор, пока это перемещение не завершено. Свойство Processing Time (Время обработки) указывает время, необходимое для обработки транзакта в Процессоре.

Сервер может использоваться для моделирования или одиночного сервера, или центра с множеством одинаковых серверов, в зависимости от производительности, указанной для Процессора. Процессор регулируется «Типом емкости» (Capacity Type), который указывается как Фиксированный (Fixed) или на основе «Расписания работы» (Work Schedule). Заметим, что на одно расписание работы могут ссылаться несколько серверов. Чтобы задать производительность на основе Расписания работы, сначала создадим расписание, кликнув на окно Data, выбрав панель Schedules (Расписания) слева, и затем щелкнув Create Work Schedule в ленте Schedule. Чтобы определить новое расписание, нужно сначала создать один или более шаблонов дня, задающих шаблон работы в определенный день.

При моделировании нескольких серверов всегда есть выбор моделировать каждый сервер как отдельный объект Сервер, или использовать один Сервер с ёмкостью равной числу серверов. Последний вариант гораздо проще и выдаст такой же результат, как и первый, если у всех серверов одинаковые характеристики производительности, и ждущие транзакты всегда отправляются к любому свободному серверу.

Сервер также предоставляет возможные настройки для учета отказов, которые останавливают обработку сервером на время отказа. Варианты отказов включают: CalendarTimeBased (основанные на календарном времени), ProcessingCountBased (основанные на подсчёте обработок), ProcessingTimeBased (основанные на времени обработки) и EventCountBased (основанные на подсчёте событий). Во всех случаях длительность отказа определяется свойством «Время на восстановление» (Time To Repair).

Транзакты, проходящие через Сервер, всегда будут захватывать Сервер как основной ресурс, который нужен для обработки транзакта. Также возможно указать второстепенный ресурс (инструменты, операторы и т.д.), который нужен в процессе обработки. Этот ресурс, если он указан, захватывается до начала обработки и

освобождается сразу после её окончания. Если указанный ресурс определен как подвижный (например, Рабочий), то может понадобиться его посещение, связанного с Сервером узла до начала обработки. Гибкость настроек обеспечивается разрешением другим ресурсам быть индивидуально захваченными и/или освобожденными в выбранные моменты в пределах Сервера. Дополнительные моменты захвата и освобождения предоставляются при входе транзакта на Сервер, до начала и после окончания процесса обработки. Заметим, что это позволяет ресурсам быть захваченными на одном Сервере, а затем освобожденными на другом.

По умолчанию у Сервера есть анимированные очереди для входного буфера, процессора и выходного буфера. Это просто конструкции, не влияющие на логику Сервера. Иногда может понадобиться увеличить длину анимированной очереди, чтобы была возможность увидеть все транзакты, которые на самом деле находятся на обслуживании.

#### Соединитель (Connector)

Соединитель - объект класса Link, который может использоваться для определения не случайных траекторий между позициями двух узлов, перемещение по которым осуществляется *мгновенно*. Его свойства DrawnToScale и LogicalLength имеют значения 'False' и '0.0' соответственно и жестко заданы. Транзакты никогда фактически не входят, не перемещаются через соединители и не выходят из них, но всегда выполняют непосредственный перенос между узлами вдоль такой линии связи. Транзакты перемещаются вдоль соединителя поодиночке, что отличается от стандартного объекта Path класса Связь. Транзакты, ожидающие входа в соединитель, ставятся в очередь. Соединитель может быть анимирован при помощи декорации (Decorator).

#### Путь (Path)

Path - объект класса Link, который может использоваться для определения траектории между позициями двух узлов, где время перемещения определяется длиной пути и скоростью транзакта. Передача может быть разрешено или запрещена. Путь может быть анимирован при помощи декорации (Decorator).

Для определения триггеров дополнительных процессов (Triggers) во время использования связи Путь допускается использование дополнительной логики.

Свойство Type позволяет пользователю определить, будет ли путь однонаправленным или двунаправленным. Необходимо заметить, что поток данных не может перемещаться в двух разных направлениях по одной связи в одно и то же время. Таким образом, даже если установлена бесконечная емкость связи, если есть поток данных, перемещающийся в одном направлении, и транзакт или транспортер собирается войти в связь и пройти в противоположном направлении, транзакт или транспортер будет ждать, пока перемещающиеся транзакты выйдут из связи, и только тогда войдут сами. Для того, чтобы позволить потоку данных перемещаться в

противоположном направлении в одно и то же время, добавьте две однонаправленных связи рядом, работающих в противоположных направлениях.

Во время использования двунаправленных путей, вам могут понадобиться временные «обходные» однонаправленные пути, чтобы минимизировать вероятность блокировки.

Каждый раз, когда используется двунаправленный путь, возникает вероятность возникновения блокировки. Это означает, что движение транзакта, исполнителя или носителя может остановиться из-за того, что один объект у узла пытается пройти двунаправленную связь, в то время как другой объект пытается пройти через этот же узел в другую связь. Если это происходит, имитационная модель выглядит «зависшей» в этом конкретном узле. Simio проверяет наличие потенциальных блокировок в узле. Если потенциальная блокировка найдена, отображается предупреждение.

Для предотвращения блокировки существует несколько моделирующих решений. Рядом с узлом, где транзакт может приближаться к двунаправленной связи, должна быть область для транзакта, достаточная для ожидания, чтобы выход из связи не блокировался («обходная» область). Размер этой обходной области зависит от величины и размера входящих транзактов. Если у вас сеть из путей (Path), то любое ответвление в этой сети должно иметь связь емкостью 1. Это позволит транзактам входить и выходить из ответвления.

Свойство Selection Weight (вес связи) может быть использовано для определения, какой транзакт должен пройти по этому пути. Ему можно задать числовое значение (например, 0,2), в случае выбора между этим путем и другим, 20% транзактов пройдут этим путем. Весу можно присвоить выражение, например, ModelEntity.Priority == 2, так что все транзакты, у которых состояние приоритета равно 2, будут проходить этим путем. В этом примере, если это будет единственный путь, по которому могут пройти транзакты, они все будут проходить по этому пути, независимо от того, равен ли их атрибут Priority 2 или нет.

#### Временной путь (TimePath)

Связь TimePath - объект класса Link, который может использоваться для определения траектории между позициями двух узлов, где время перемещения задается пользователем. Связь TimePath может быть анимирована при помощи декорации (Path Decorator).

### **5.6. Пример модели**

Поясним шаги для создания небольшой тестовой модели Источник-Обработчик-Поглотитель (Source-Server-Sink) (рисунок 5.4).

Сначала создайте новую модель внутри проекта. Это может быть сделано как нажатием на кнопку «Создать новую модель» внизу Начальной страницы Simio, или при нажатии на кнопку Новая Модель в панели. После того как модель создана, вы увидите базовое окно (Facility) новой модели.



Сначала щелкните на объекте Source в стандартной библиотеке (Standard Library), и перетащите его в левую верхнюю часть Базового окна. После того как вы отпустите его, вы должны увидеть объект Source, названный Source1. Таким же образом переместите объект Server в середину базового окна и объект Sink в нижнюю правую часть окна. Для выделения, перемещения или изменения свойств щелкните на имени (например, Sink1). Для этого примера мы оставим свойства всех объектов по умолчанию.

На следующем шаге соедините все объекты. Ромбики около объектов называются узлы (Node). Они обозначают места, где могут входить или выходить транзакты. Несмотря на то, что сейчас здесь ничего не будет меняться, для того, чтобы посмотреть или изменить значения узла, необходимо щелкнуть на ромбике, и увидеть его свойства в окне свойств (Property Window) внизу справа. Свойства узлов используются для указания места назначения транзакта и транспортной логики.

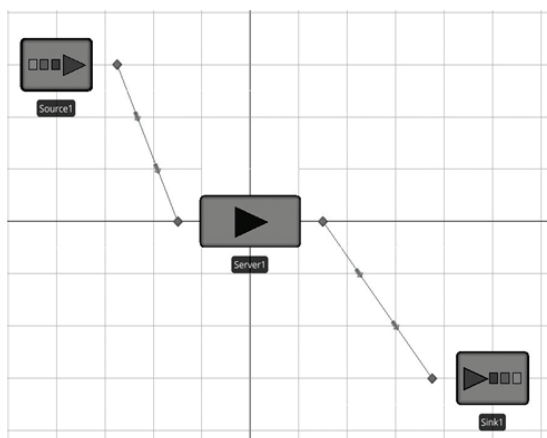


Рисунок 5.4. Схема модели

Выбор узла в базовом окне немного отличается от других объектов, потому что есть три способа выбора:

*Щелкнуть* – Выбрать Узел и отобразить его свойства;

*Щелкнуть (выделить) и переместить* – Перемещает узел в другое место на экране;

*Ctrl + Shift + Щелкнуть и переместить* – Иницирует создание связи между узлами. Свойство связи (Link) может быть перевыбрано в панели библиотеки или связь можно будет удалить, щелкая по узлу.

Соединим объекты связями типа Путь (Path). Нажмите и держите Ctrl и Shift на клавиатуре, щелкните левой кнопкой мыши на трансферный узел (Transfer node) в

Source1. Переместите курсор к левому узлу Server1 и щелкните снова левой кнопкой мыши. Появится меню, в котором выберите Path (Путь) в Типе связи (Link Type).

Теперь нажмите клавиши Ctrl+Shift, а затем нажмите на правом узле в Server1. Можно добавлять промежуточные точки, кликая в различных местах в базовом окне или же можно закончить соединение, щелкнув на узле в Sink1. Если вы передумаете при размещении соединения перед его окончанием и захотите удалить его, то щелкните правой кнопкой мыши. Появится меню для выбора типа связи (Link type), где надо выбрать Путь (Path).

Существует еще один метод для создания связей. Щелкните на Тип связи в стандартной библиотеке и затем левой кнопкой мыши на узле, где нужно ее создать.

Двойное нажатие на объекте Путь (Path) переведет вас в режим повторения действий. Например, двойное нажатие на Path (Путь) позволит соединить Источник (Source) с Сервером (Server), а затем сервер (Server) с поглотителем (Sink) без повторного выбора из библиотеки. Этот способ работает со всеми объектами из стандартной библиотеки и некоторыми командами интерфейса.

На данном этапе создание модели закончено. Для сохранения проекта, выберите зеленую треугольную кнопку, расположенную на панели в верхнем левом углу главного окна Simio. Затем выберите “Save As (Сохранить проект как)” из выпадающего вниз меню, и используйте окно Проводника (Window Explorer) для сохранения файла. Существует альтернативный способ добавить стандартные Source (источник), Server (сервер) и Sink (поглотитель) – в базовом окне можно просто выбрать Source/Server/Sink из надстроек (Select Add-Ins) на вкладке Дом проекта (Project Home tab). Это действие сразу добавит 3 объекта, соединенные путями.

Для запуска модели щелкните на кнопке «Запустить» (Run), расположенной на вкладке Дом проекта (Project Home).

### Пользовательский интерфейс

Пользователь может работать с моделью через несколько интерфейсных окон. Для их использования нужно выбрать соответствующие вкладки или панели.

Окно проекта может быть вызвано нажатием на название проекта в окне навигации. Оно содержит все модели, эксперименты, символы, текстуры и декораторы, которые имеются в своем проекте.

Панель модели активируется по умолчанию при первом открытии проекта. Все модели, которые содержатся в проекте, отображаются в панели и организованы по типам. Чтобы просмотреть свойства каждой модели, просто выберите соответствующую модель, и ее свойства отобразятся в панели Свойства в нижнем правом углу окна интерфейса. После выбора модели щелкните правой кнопкой мыши для редактирования настроек или удаления модели. Если пользователь выбирает редактирование, можно переходить к следующим окнам проекта и панели этой модели отобразятся для обработки.

Модель может быть защищена паролем, если выбрать иконку защиты на ленте меню. Новые модели могут быть добавлены к проекту несколькими способами. Одним из вариантов является добавление новой пустой модели, т.е. не содержащей никакой логики. Это делается путем выбора соответствующего объекта с иконкой «Новая модель» на ленте меню.

Базовое окно (Facility) определяет модель как совокупность анимированных объектов. Объекты могут быть помещены в базовое окно перетаскиванием их либо из стандартной, либо из проектной библиотеки.

Добавить проект в модель можно перетащив его из Стандартной библиотеки (к примеру, Source, Server, Sink) или из Проектной библиотеки (к примеру, MyServer, MySubmodel). Добавить несколько объектов в модель можно двойным щелчком на модели в библиотеке, тем самым перейдя в режим повторяющегося добавления. В этом режиме каждый щелчок левой кнопки создает новый объект. Выйти из этого режима можно, кликнув правой кнопкой мыши или нажав клавишу Esc. Добавление связи между узлами (к примеру, Path, Connector) можно выполнить, щелкая по одному из видов связи в библиотеке и затем по начальному узлу, тем самым перейдя в режим рисования линий. Затем добавьте узлы, соединяемые связью, последовательно кликая на каждом из них, и завершите рисование связи, кликнув по конечному узлу. Можно войти в режим рисования линий, дважды кликнув на стартовом узле.

В базовом окне модели может быть изменен внешний вид объекта для анимации. Вкладка «Символы» на ленте меню позволяет изменять внешний вид символов, добавлять и импортировать символы. Вкладка «Рисование» позволяет добавлять статические символы, линии, прямоугольники, эллипсы и т.д. Вкладка «Вид» позволяет изменить вид базового окна с 2D- на 3D-проекцию, а также предоставляет некоторые другие опции, связанные с отображением этого окна. Вкладка «Прогон» (Run) дает пользователю возможность запустить модель в интерактивном режиме, добавлять точки останова, менять скорость анимации и задавать начальное и конечное модельное время.

Визуальные элементы метки (Status Label), графики (Status Plot), диаграммы (Status Pie), надписи (Floor Label), индикаторы (Circular Gauge, Linear Gauge), а также кнопки могут быть помещены в базовое окно с помощью команд вкладки «Анимация». Они могут быть прикреплены к объектам модели в базовом окне. Если они прикреплены к транзактам (Entities) или транспорту (Transporter), то они будут перемещаться вместе с этими динамическими объектами.

Каждый объект (за исключением связей) имеет свойство, называемое Location (расположение), которое показывает, где этот объект расположен в базовом окне по отношению к нулевой точке (0,0). Свойство Location представляет собой центральную точку для узла и нижний центр для фиксированного объекта. Например, если значение  $Y=0$ , то объект находится «на земле».

Визуальные элементы Grids, Labels, Axes, Arrows, Nodes, Queues могут быть сделаны видимыми или невидимыми в базовом окне модели. Этот режим управляется на вкладке «Вид».

Кнопка Приложение располагается в самом левом верхнем углу интерфейса вкладок. Она служит для открытия, закрытия или сохранения проекта. Тут можно ввести ключ лицензии к продукту. На панели располагаются несколько различных вкладок, и внутри каждой вкладки сгруппированы иконки или панели. Состояние вкладки панелей зависит от того, кто запускает приложение. Основные панели меню представлены ниже.

Окно по умолчанию, которое появляется при создании нового проекта, – это базовое окно модели (Facility Window). Вкладки окон проекта используются для переключения между окнами, относящимися к текущей модели или экспериментам. Доступные окна зависят от типа объекта выбранной модели.

Вкладка Дом проекта – единственная вкладка, которая доступна вне зависимости от того, какие окна открыты. Эта вкладка позволяет пользователю создать новую модель, новый эксперимент, добавить новый символ, импортировать новую текстуру, получить доступ к построителю отчетов или загрузить библиотеку к текущему проекту. Это место, где пользователь может вырезать, копировать и вставлять иконки, включить и выключить Трассировку (Trace) или показать/скрыть окно ошибок (Error window). Иконка Наблюдение отобразит окно наблюдения, позволяющее пользователю просматривать значение состояний, функций и элементов, связанных с выбранным экземпляром объекта, во время прогона модели. Иконка Поиск запускает поиск определенной текстовой строки. Окно покажет все вхождения строки, которые при двойном щелчке мышью переместят интерфейс на объект, где присутствует данная строка.

Окно навигации, расположенное в верхнем правом углу интерфейса, используется для переключения между различными моделями, существующими внутри проекта. Иконка SimBits используется для доступа к окну поиска фрагментов моделей (SimBits). Начальная страница Simio может быть открыта и из окна навигации, щелчком на начальной странице (Start Page). Страница содержит ссылки на SimBits, обучающее видео, страницу Simio, замечания к выпуску, сайт веб-поддержки и дополнительную информацию для знакомства с продуктом.

Из окна навигации может быть получен доступ к свойствам модели. Для этого щёлкните правой кнопкой мыши и выберите свойства модели (Model Properties). Свойства проекта можно просмотреть, если в окне навигации выбрать проект.

Щелкнув правой кнопкой мыши в базовом окне, получим небольшое меню, где можно установить точки останова в объекте или открыть инструментальную панель к выбранному объекту. По щелчку правой кнопкой мыши на объекте типа Транспорт (Vehicle), пользователь может установить этот транспорт по умолчанию.

Щелчок на объекте типа Связь выводит небольшое меню, позволяющее добавить точку останова, добавить или удалить связь из сети или преобразовать связь в иной тип связи. Связь может быть добавлена к существующей сети или можно создать новую сеть. Идея в разрешении пользователю менять тип связи простым щелчком дает следующее преимущество: сделав всего несколько нажатий можно быстро и просто изменить логику модели.

Помимо возможности устанавливать точки останова и наблюдать за объектами через контекстное меню, существует возможность делать объекты видимыми внешне, или уменьшать до реальных (графических) размеров. Когда объект Внешне видимый (по умолчанию), то он виден во Внешнем окне (External Window) и будет виден, если используется в иерархии объектов. Уменьшение до графических размеров доступно, если размер объекта был ранее изменён. Уменьшение до графических размеров может быть использовано для возврата графики к первоначальному виду.

Сетка, представленная в базовом окне, внешнем окне и окне нового символа, включает в себя маркеры и обозначения, чтобы прояснить расположение и наличие свободного места вокруг объекта. В сетке Simio по умолчанию 1 блок равен 1 метру. Сетка Simio отображает десятки, а при дальнейшем удалении – сотни метров. Если продолжить удалять, то следующий масштаб будет тысячи. Функция увеличения (zoom) выполняется щелчком правой кнопкой мыши на сетке и передвижением мышью вверх-вниз. Если у вас мышь с колесиком, то просто прокрутите колесом.

В Simio есть возможность отображения нескольких окон одновременно, таких как базовое окно и окно процессов. Для доступа и организации этих окон, щелкните правой кнопкой мыши на заголовке любого окна; выпадет контекстное меню. На примере ниже, базовое окно было скрыто за окном процессов, т.к. пользователь работал в окне процессов, что является стандартным вариантом отображения. Щелкнув на верхушке вкладки Процессы, пользователь выбирает «Новая вертикальная группировка вкладок» (New Vertical Tab Group).

## 5.7. Вопросы для самопроверки

1. Какие модельные объекты включены в стандартную библиотеку среды SIMIO?
2. Какие объекты в среде SIMIO используются для создания транзактов (entity)?
3. Какие объекты в среде SIMIO используются для моделирования устройств-обработчиков?
4. Какие объекты в среде SIMIO используются для моделирования разделяемых ресурсов?
5. Какие объекты в среде SIMIO используются для моделирования процессов размножения-объединения транзактов?
6. Какие объекты в среде SIMIO используются для маршрутизации транзактов?
7. Какие элементы в среде SIMIO используются для связей в трекобъектах?
8. Сколько может использоваться внутренних очередей в объекте Server в среде SIMIO?
9. Сколько предусмотрено модельных стадий обработки (processing) в объекте Workstation в среде SIMIO ?
10. Какие объекты в среде SIMIO используются для моделирования подвижных ресурсов?
11. Какие объекты используются в среде SIMIO для уничтожения транзактов (entity)?
12. Как в среде SIMIO можно задать трек развития процесса?
13. Каким образом можно в среде SIMIO создавать собственные модельные объекты?
14. Как настраивается динамическая анимация развития процессов в среде SIMIO?
15. Как можно в среде SIMIO моделировать события отказов и поломок объектов-обработчиков?
16. Чем отличаются различные варианты связей между объектами, применяемые в среде SIMIO?

## ПРИЛОЖЕНИЕ 1. Лабораторный практикум по моделированию на GPSS

В процессе выполнения лабораторных работ студенты индивидуально выполняют поставленные задачи. Методическая схема выполнения лабораторных работ предполагает первоначальное изучение студентом готовых программ, размещенных в библиотеке моделей. Затем в заключительной части лабораторного занятия студенту предлагается задание для самостоятельной работы по составлению программы имитационной модели.

Рассматриваемые модели размещаются в библиотеке Samples в составе дистрибутивной установки GPSS World Student Version. Имена моделей указываются в задании по теме.

Основные требования к выполнению лабораторной работы:

- открыть модель из библиотеки в GPSS World;
- изучить алгоритм модели;
- провести несколько вариантов моделирования с изменением параметров модели;
- отобразить графически результаты моделирования.

Все действия студенты выполняют в интерактивном режиме в среде GPSS World или в редакторе GPSS Studio. После выполнения задания студенты демонстрируют работу модели, полученные отчеты, графические результаты и защищают работу, отвечая на контрольные вопросы преподавателя. Отчет подготавливается в текстовом процессоре (типа Word) с включением необходимых иллюстраций.

Задания сгруппированы по сходным предметным тематикам. В зависимости от учебной программы на изучение одного цикла может выделено несколько лабораторных работ. При организации лабораторных работ на каждое задание может быть отведено определенное время, за которое студент должен выполнить работу. Также может быть ограничено общее время проведения всех занятий по одной тематике.

### Тема 1. Цикл «Функции, многоканальные обработчики»

**Задание 1.** Изучите пример модели SCHR4A.GPS, где приведена типовая модель обслуживания потока транзактов.

```
Mean FUNCTION Q$Wait,D4 ;Среднее время обслуживания
0,330/2,300/5,270/6,240
GENERATE (Exponential(1,0,300)) ;Приход клиентов
QUEUE Wait ;Вход в очередь
SEIZE Survu ;Начало обслуживания
DEPART Wait ;Выход из очереди
ADVANCE FN$Mean,(Exponential(1,0,1));Время обслуживания
```

```

RELEASE Survu ;Освобождение сервера
TERMINATE 1

```

В этом примере среднее время обслуживания устройством Survu зависит от длины очереди ожидания обслуживания (например, ситуация обслуживания в буфете). Эта зависимость в модели задана функцией MEAN. Необходимо построить график функции MEAN и график изменения длины очереди WAIT.

**Задание 2.** Изучите пример модели SCHR4B.GPS, где приведена модель магазина самообслуживания.

```

Carts STORAGE 100
Ayl1 FUNCTION RN1,C2 ;количество покупок в отделе 1
0,2/1,5
Ayl2 FUNCTION RN1,C2 ;количество покупок в отделе 2
0,3/1,6
Ayl3 FUNCTION RN1,C2 ;количество покупок в отделе 3
0,4/1,7
Cotym FUNCTION P1,C2 ;время обслуживания у кассира
0,3/18,54
Impul FUNCTION RN1,C2 ;количество покупок перед кассой
0,1/1,4
* модельный сегмент 1
GENERATE (Exponential(1,0,75)) ;появление покупателей
ENTER Carts ;взять тележку
TRANSFER .25,,Try2 ;будем покупать в отделе 1?
ADVANCE 120,60 ;ходим в отделе 1
ASSIGN 1,FN$Ayl1 ;Запишем в P1 кол-во покупок
Try2 TRANSFER .45,,Try3 ; будем покупать в отделе 2?
ADVANCE 150,30 ; ходим в отделе 2
ASSIGN 1+,FN$Ayl2 ;Запишем в P1 кол-во покупок
Try3 TRANSFER .18,,Out ; будем покупать в отделе 3?
ADVANCE 120,45 ; ходим в отделе 3
ASSIGN 1+,FN$Ayl3 ;Запишем в P1 кол-во покупок
Out QUEUE Checker ;Очередь в кассу
ASSIGN 1+,FN$Impul ;Запишем в P1 кол-во покупок у кассы
SEIZE Checker ;Занять кассира
DEPART Checker ;Выйти из очереди
ADVANCE FN$Cotym ;Время скана покупок
RELEASE Checker ;Уйти от кассира

```



```

LEAVE Carts ;Вернуть тележку
TERMINATE ;Выйти из магазина
* модельный сегмент 2
GENERATE 28800 ;Таймер через 8 ч
TERMINATE 1

```

В магазине 3 отдела (aisle). С некоторой вероятностью покупатели делают покупки в некоторых отделах. Типовое количество покупок для каждого отдела задано через функции (Ayl1, Ayl2, Ayl3, Impul) и накапливается у транзактов в параметре локальной среды P1.

Время обслуживания в кассе (устройство checker) зависит от количества покупок и задано функцией Cotum. Необходимо построить график количества покупок на одного покупателя за 8 часов модельного времени.

**Задание 3.** Изучите пример модели MANUFACT.GPS, где приведена модель производственного участка со складом.

```

Sizeorder FUNCTION RN1,D7 ;Размер заказа
.10,6/.35,12/.65,18/.80,24/.92,30/.97,36/1.0,48
Transit TABLE M1,.015,.015,20
Number TABLE X1,100,100,20 ;Кол-во упаковок в день
Ptime VARIABLE .0028#P1+0.0334 ;Время упаковки
Amount EQU 1000 ;Нач.сост.склада
Stock STORAGE 4000 ;Емкость склада 4000 шт
* модельный сегмент 1
GENERATE (Exponential(1,0,0.25)) ;пришел заказ
ASSIGN 1,1,Sizeorder ;P1=размер заказа
TEST GE S$Stock,P1,Stockout ;Есть на складе?
LEAVE Stock,P1 ;Взять P1 шт со склада
QUEUE Packing
SEIZE Machine ;Занять станок
DEPART Packing
ADVANCE V$Ptime ;Время упаковки
RELEASE Machine ;Освободить станок
SAVEVALUE 1+,P1 ;Счетчик упаковок
TABULATE Transit ;Запись времени упаковки
TERMINATE
Stockout TERMINATE ;увидим здесь, если нет на складе
* модельный сегмент 2
GENERATE 0.75,0.08334,1 ;Транзакт каждые 45+/-5 мин
ENTER Stock,60 ;Добавить 60 на склад

```

TERMINATE

```
* модельный сегмент 3
GENERATE 8 ;Транзакт в конце дня
TABULATE Number
SAVEVALUE 1,0
TERMINATE 1

* модельный сегмент 4
GENERATE ,,1,10 ;инициализация склада
ENTER Stock,Amount ;задать нач.объем
TERMINATE
```

Необходимо построить график изменения количества товара на складе и график времени изготовления деталей за 40 часов модельного времени.

**Задание 4.** Изучите пример модели STOCKCTL.GPS, где приведена модель функционирования складов дистрибутора.

```
INITIAL X1,3400 ;Нач.сост.склада фабрики
INITIAL X2,2100 ;Крит.уровень запасов фабрики
INITIAL X3,2300 ;Реком.объем поставки на склад
INITIAL X$Stock1,430 ;Нач.сост.склада 1 филиала
INITIAL X$Stock2,600 ;Нач.сост.склада 1 филиала
INITIAL X$Stock3,1000 ;Нач.сост.склада 1 филиала
INITIAL X$EOQ1,115 ;Реком.объем поставки на склад 1
INITIAL X$EOQ2,165 ;Реком.объем поставки на склад 2
INITIAL X$EOQ3,200 ;Реком.объем поставки на склад 3
INITIAL X$Point1,240 ;Крит.уровень запасов на складе 1
INITIAL X$Point2,430 ;Крит.уровень запасов на складе 2
INITIAL X$Point3,630 ;Крит.уровень запасов на складе 3
Demand1 VARIABLE (Normal(2,64,24)) ;Объем спроса в регионе 1
Demand2 VARIABLE (Normal(3,128,32)) ;Объем спроса в регионе 2
Demand3 VARIABLE (Normal(4,192,48)) ;Объем спроса в регионе 3
Total VARIABLE P1+P2+P3
Sales TABLE X5,200,200,20
Region_1 TABLE X$Stock1,0,40,20
Region_2 TABLE X$Stock2,0,40,20
Region_3 TABLE X$Stock3,0,40,20
Factory TABLE X1,0,200,20

* модельный сегмент 1
GENERATE ,,1,2 ;Запуск процесса на фабрике
```

Backhere TEST LE X1,X2 ;Требуется пополнение склада?  
 ADVANCE 4 ;время поставки 4 недели  
 SAVEVALUE 1+,X3 ;пополнили склад  
 TRANSFER ,Backhere ;цикл опроса

\* модельный сегмент 2  
 GENERATE 1,,,1 ;Запуск процесса в регионе 1

Distr1 TEST L X\$Stock1,X\$Point1 ; Требуется пополнение склада?  
 ADVANCE 1 ; время поставки 1 неделя  
 SAVEVALUE 1-,X\$EOQ1 ; забрали с фабрики  
 SAVEVALUE Stock1+,X\$EOQ1 ; пополнили склад 1  
 TRANSFER ,Distr1 ; цикл опроса

\* модельный сегмент 3  
 GENERATE 1,,,1 ; Запуск процесса в регионе 2

Distr2 TEST L X\$Stock2,X\$Point2 ; Требуется пополнение склада?  
 ADVANCE 1 ; время поставки 1 неделя  
 SAVEVALUE 1-,X\$EOQ2 ; забрали с фабрики  
 SAVEVALUE Stock2+,X\$EOQ2 ; пополнили склад 2  
 TRANSFER ,Distr2 ; цикл опроса

\* модельный сегмент 4  
 GENERATE 1,,,1 ; Запуск процесса в регионе 3

Distr3 TEST L X\$Stock3,X\$Point3 ; Требуется пополнение склада?  
 ADVANCE 1 ; время поставки 1 неделя  
 SAVEVALUE 1-,X\$EOQ3 ; забрали с фабрики  
 SAVEVALUE Stock3+,X\$EOQ3 ; пополнили склад 3  
 TRANSFER ,Distr3 ; цикл опроса

\* модельный сегмент 5  
 GENERATE 1,,,3 ;Еженедельный спрос  
 ASSIGN 1,V\$Demand1 ;P1 = Продажи в регионе 1  
 ASSIGN 2,V\$Demand2 ;P2 = Продажи в регионе 2  
 ASSIGN 3,V\$Demand3 ;P3 = Продажи в регионе 3  
 SAVEVALUE Stock1-,P1 ; Уменьшим запасы для 1  
 SAVEVALUE Stock2-,P2 ; Уменьшим запасы для 2  
 SAVEVALUE Stock3-,P3 ; Уменьшим запасы для 3  
 SAVEVALUE 5+,V\$Total ;Суммируем все продажи  
 TABULATE Region\_1 ; Запишем состояние склада 1  
 TABULATE Region\_2 ; Запишем состояние склада 2  
 TABULATE Region\_3 ; Запишем состояние склада 3  
 TABULATE Factory ; Запишем состояние склада фабрики

```

TERMINATE 1
* модельный сегмент 6
GENERATE 4,,1 ;Ежемесячная статистика
TABULATE Sales
SAVEVALUE 5,0 ;Сброс накопленных продаж
TERMINATE

```

Необходимо построить график изменения количества товара на складах за 50 недель модельного времени.

**Задание 5.** Постановка задачи для самостоятельной работы.

Сделать модель обслуживания заказов клиентов дистрибутора в его филиалах (с применением объектов типа storage) по данным складского процесса из задания 4. Построить графики изменения складских запасов для всех 4-х складов за 90 дней модельного времени.

## **Тема 2. Цикл «логические ключи, порождение дочерних процессов»**

**Задание 1.** Изучите пример модели LOCKSIMN.GPS, где приведена модель работы речного шлюза.

```

Upbarge FUNCTION X$Upcount,D6
1,.967/2,.767/3,.767/4,.767/5,.767/6,.767
Downbarge FUNCTION X$Downcount,D6
1,.967/2,.767/3,.767/4,.767/5,.767/6,.767
Upq QTABLE Upq,.25,.25,20
Downq QTABLE Dnq,.25,.25,20
Upcount TABLE X$Upcount,2,2,20
Dncount TABLE X$Downcount,2,2,20
INITIAL X$Uplimit,6 ;Предел барж вверх по реке
INITIAL X$Downlimit,6 ; Предел барж вниз по реке
* модельный сегмент 1
GENERATE 1.67,.5,.67 ;Появление баржи вверх
QUEUE UPQ ;встать в очередь
GATE LR Lock ;Ждать открытия шлюза
SEIZE Lock ;Занять ворота шлюза
SAVEVALUE Upcount+,1 ;Посчитаться
DEPART Upq ;выйти из очереди
ADVANCE FN$Upbarge ;время причаливания в шлюзе
TEST GE X$Uplimit,X$Upcount,Swh1 ;Не превышена емкость?
TEST NE Q$Upq,0,Swh1 ;Есть еще в очереди?

```

```

RELEASE Lock          ;Освободить ворота
TERMINATE
*   ветвь переключения шлюза 1
Swh1 LOGIC S Lock      ;Включить режим шлюза
RELEASE Lock          ;Освободить ворота
TABULATE Upcount      ;Записать статистику
SAVEVALUE Upcount,0   ;Сброс счетчика
TERMINATE
*   модельный сегмент 2
GENERATE 1.67,,5,1    ; Появление баржи вниз
QUEUE Dnq             ; встать в очередь
GATE LS Lock          ; Ждать открытия шлюза
SEIZE Lock            ; Занять ворота шлюза
SAVEVALUE Downcount+,1 ; Посчитаться
DEPART Dnq           ; выйти из очереди
ADVANCE FN$Downbarge ; время причаливания в шлюзе
TEST GE X$Downlimit,X$Downcount,Swh2 ; Не превышена емкость?
TEST NE Q$Dnq,0,Swh2 ; Есть еще в очереди?
RELEASE Lock          ; Освободить ворота
TERMINATE
*   ветвь переключения шлюза 2
Swh2 LOGIC R Lock     ; Включить режим шлюза
RELEASE Lock          ; Освободить ворота
TABULATE Dncount      ; Записать статистику
SAVEVALUE Downcount,0 ; Сброс счетчика
TERMINATE
*   модельный сегмент 3
GENERATE 24           ; Один раз в сутки
TERMINATE 1          ; пускаем транзакт

```

Шлюз соединяет два уровня речного канала. Баржи подходят с двух сторон — сверху и снизу. Емкость шлюза ограничена шестью баржами. Время входа в шлюз первой баржи из очереди больше, чем для последующих из-за учета времени освобождения шлюза.

Направление движения барж определяется логическим ключом с названием LOCK. Так, если он выключен (состояние reset), означает что происходит движение вверх по течению, если включен (состояние set), то вниз по течению. Необходимо промоделировать работу шлюза в течение месяца (30 дней). Получите и

проанализируйте таблицы распределения количества барж и времен ожидания в очереди для обоих направлений.

**Задание 2.** Изучите пример модели SCHR6C.GPS, где представлена модель станка с периодической заменой запчасти.

```
* модельный сегмент 1
GENERATE ,,1 ;пришел оператор станка
Again SEIZE Mac ;включили станок
ADVANCE (Normal(1,3500,700)) ;закончился период работы сверла
RELEASE Mac ;выключили станок
ADVANCE 40 ;сняли сверло
SPLIT 1,Fetch ;отдали в ремонт
SEIZE Fixer ;ремонтник начал заточку
ADVANCE (Normal(1,80,5)) ;время на заточку
RELEASE Fixer ;освободили ремонтника
SAVEVALUE 1+,1 ;посчитали запчасть
TERMINATE ;работа выполнена!
Fetch TEST G X1,0 ;ждем запчасть
SAVEVALUE 1-,1 ;забрали запчасть
ADVANCE 60 ;установили деталь
TRANSFER ,Again ;идем включать станок
* модельный сегмент 2
GENERATE (Exponential(1,0,90)),,,1 ;заказы для ремонтника
ADVANCE ; в очередь с приоритетом!
SEIZE Fixer ;заняли ремонтника
ADVANCE 80,40 ;сделали заказ
RELEASE Fixer ;освободили ремонтника
TERMINATE ;закончили
* модельный сегмент 3
GENERATE 104000 ;засаеаем год работы
TERMINATE 1 ;выключили модель
```

В этой задаче моделируется работа сверлильного станка. На сверлильном станке сверло периодически выходит из строя (тупится). Токарь-станочник меняет его на запасное, если оно есть, и передает слесарю-инструментальщику для восстановления (заточки). Слесарь, в основном, занят изготовлением техоснастки и выполняет заточку сверла в свободное от основной работы время.

Необходимо промоделировать работу станка в течение года. Найти загрузку слесаря и станка, количество замен сверла. Построить график ожидания сверлом операции по заточке.

**Задание 3.** Изучите пример модели SCHR5D.GPS, где приведена модель автозаправки.

```

lat FUNCTION RN1,C7 ;распределение времени приезда
0,0/.25,100/.48,200/.69,300/.81,400/.9,500/1,600
Stime FUNCTION RN1,C7 ; распределение времени обслуживания
0,100/.06,200/.21,300/.48,400/.77,500/.93,600/1,700
Store1 EQU 1 ;надо для TEST
Store1 STORAGE 1 ;кол-во заправочных колонок
Net VARIABLE SC$store1#1000-1000-500#R$Store1
* модельный сегмент 1
GENERATE FN$lat,,,,1 ;появление клиента
GATE LR Lock,Bybye ;проверяем шлагбаум
ASSIGN 1,FN$Stime ;P1 =время обслуживания
TEST LE Q1,Store1,Bybye ;длина очереди<=число колонок?
Goin QUEUE 1 ;=да – встаем в очередь
ENTER Store1 ;занимаем колонку
DEPART 1 ;выходим из очереди
ADVANCE P1 ;обслуживаемся
Done LEAVE Store1 ;освобождаем колонку
Bybye TERMINATE ;уезжаем (=нет)
* модельный сегмент 2
GENERATE 43200 ;закрываемся на инкассацию
LOGIC S Lock ;закрываем шлагбаум
TEST E N$Goin,N$Done ;ждем дообслуживания всех въехавших
SAVEVALUE 1,V$Net ;фиксируем прибыль
ADVANCE 100
LOGIC R Lock ;открываем шлагбаум
TERMINATE 1 ;работаем

```

На АЗС приезжают автомобили в интервале от 0 до 600 сек, это распределение задано функцией IAT. Длительность заправки распределена от 100 до 700 сек. Размер очереди не может быть больше количества заправочных колонок. Через 12 час приезжает инкассация, обслуживание клиентов прекращается, дообслуживаются все приехавшие, и фиксируется финансовый результат.

Необходимо провести моделирование при количестве колонок, меняющемся от 1 до 5, сравнить результаты моделирования.

**Задание 4.** Изучите пример модели TVREPAIR.GPS, где приведена модель работы ремонтной мастерской по телевизорам.

\* модельный сегмент 1

```

GENERATE 2400,480,,,1 ;сложный ремонт
QUEUE Overhaul ;очередь таких работ
QUEUE Alljobs ;очередь всех работ
SEIZE Maintenance ;занимаем ремонтника
DEPART Overhaul ;выходим из очереди
DEPART Alljobs ;выходим из очереди всех работ
ADVANCE 600,60 ;длительность 10+/-1 час
RELEASE Maintenance ;освобождаем ремонтника
TERMINATE ;готово!
* модельный сегмент 2
GENERATE 90,10,,,3 ;простая диагностика
QUEUE Spot ; очередь таких работ
QUEUE Alljobs ; очередь всех работ
PREEMPT Maintenance,PR ;срочно занимаем ремонтника
DEPART Spot ; выходим из очереди
DEPART Alljobs ;выходим из очереди всех работ
ADVANCE 15,5 ;время на диагностику
RETURN Maintenance ; освобождаем ремонтника
TERMINATE
* модельный сегмент 3
GENERATE 300,60,,,2 ;несложный ремонт
QUEUE Service ; очередь таких работ
QUEUE Alljobs ; очередь всех работ
PREEMPT Maintenance,PR ; занимаем ремонтника вне очереди
DEPART Service ; выходим из очереди
DEPART Alljobs ;выходим из очереди всех работ
ADVANCE 120,30 ;время ремонта
RETURN Maintenance ; освобождаем ремонтника
TERMINATE
* модельный сегмент 4
GENERATE 480 ; засекаем одну раб.смену 8 час
TERMINATE 1
*
Overhaul QTABLE Overhaul,10,10,20
Spot QTABLE Spot,10,10,20
Service QTABLE Service,10,10,20
Alljobs QTABLE Alljobs,10,10,20

```



В задаче смоделирован процесс ремонта телевизоров одним мастером. Мастер выполняет три вида ремонтов – сложные (редко, но долго), простые (недолго, вне очереди) и диагностику (без очереди и быстро). Характерные времена появления клиентов и их обслуживания показаны в параметрах операторов задачи. Необходимо промоделировать работу мастерской и построить график очереди 3х типов работ в течение календарного месяца (4 недели).

#### **Задание 5. Постановка задачи самостоятельной работы**

Сделать модель работы мастерской по ремонту ноутбуков с двумя мастерами – системщиком и электроником. При диагностике системщиком отсеивается 8% посетителей как «неремонтируемые случаи». Основная часть ремонтов — исправление системщиком софтверных проблем (upgrade firmware, установка OS, анти-вирусы, драйверы). Работы по ремонту электроники появляются в 25% случаев ремонта («сложный» ремонт), но их выполняют оба мастера совместно. Недостающие сведения можно взять из описания процесса в задании 4.

Необходимо промоделировать работу мастерской и построить график очереди 3х типов работ в течение календарного месяца.

### **Тема 3. Цикл «Логические условия, семейства транзактов»**

**Задание 1.** Изучите пример моделей SCHR7A1.GPS и SCHR7A2.GPS, в которых приведены разные варианты модели обслуживания клиентов в операционном зале банка.

Вариант 1. Одна очередь к 8 окошкам.

Mean FUNCTION RN1,D5 ; распределение времени обслуживания  
.1,450/.29,750/.61,1000/.85,1500/1,3000

Telrs STORAGE 8 ; в наличии 8 операционистов

\* модельный сегмент 1

GENERATE (Exponential(1,0,180)) ; пришел клиент

ASSIGN 1,(Exponential(1,0,FN\$Mean)) ; P1 = время обслуживания

QUEUE One ; встал в очередь

GATE SNF Telrs,Wait ; если все заняты, переходим в очередь Line

Grab ENTER Telrs ; занимаем свободного операциониста

DEPART One ; выходим из очереди

ADVANCE P1 ; обслуживаем клиента

LEAVE Telrs ; освобождаем операциониста

UNLINK Line,Grab,1 ; призываем следующего из очереди

TERMINATE ; уходим

Wait LINK Line,FIFO ; здесь наша очередь

\* модельный сегмент 2

GENERATE 216000 ; засекаем 6 часов (1 смена)

## TERMINATE 1

Вариант 2. Восемь очередей к 8 окошкам.

Mean FUNCTION RN1,D5 ; распределение времени обслуживания  
.1,450/.29,750/.61,1000/.85,1500/1,3000

\* модельный сегмент 1

GENERATE (Exponential(1,0,180)) ; пришел клиент

ASSIGN 1,(Exponential(1,0,FN\$Mean)) ;P1 = время обслуживания

SELECT E 2,1,8,0,F,Queue ;Кто из 8 свободен?

Line QUEUE P2 ;Занимаем к нему очередь

QUEUE 10 ;записываем в общий учет

SEIZE P2 ;занимаем своего операциониста

DEPART 10 ;выходим из общей очереди

DEPART P2 ;выходим из своей очереди

ADVANCE P1 ; обслуживаем клиента

RELEASE P2 ; освобождаем операциониста

TERMINATE ; уходим

Queue SELECT MIN 2,1,8,,Q ;если все заняты ищем короткую очередь

TRANSFER ,Line ;встаем в очередь

\* модельный сегмент 2

GENERATE 216000 ; засекаем 6 часов (1 смена)

TERMINATE 1

В 1 варианте очереди организованы отдельно к каждому операционисту, а в варианте 2 - очередь общая для всех. Необходимо открыть модели, получить результат моделирования за 25 рабочих смен. Оценить статистику по времени ожидания в очереди Line (среднее время и стандартное отклонение). Сравнить варианты ожидания в общей очереди и в очередях к каждому операционисту. Получить сведения о загрузке операционистов. Разобрать алгоритм выборки значения в блоках SELECT.

**Задание 2.** Изучите пример модели FOUNDRY.GPS, в котором приведена модель производственных процессов в литейной мастерской мелкосерийного художественного литья.

Weight FUNCTION RN1,C8 ;распределение по весу в кг

0.0,3/.13,6/.25,11/.50,20/.70,28/.85,35/.95,42/1.0,50

Ordertype FUNCTION RN1,D2 ;если новый P1=1: повторный P1=2

0.3,1/1.0,2

Size VARIABLE RN1@19+6 ;размер серии заказа

Ddate VARIABLE V\$Mtime#P2+RN1@121+40+C1 ;срок изготовления

Mtime VARIABLE (P3#2) ;время заливки детали

Day VARIABLE (C1/480) ;счетчик дней (по 8 ч)  
 Total VARIABLE P3#P2 ;вес всего заказа  
 Times TABLE M1,400,400,20 ;статистика времени изготовления  
 Cast TABLE X\$Wtmold,400,400,20 ; статистика веса заказов  
 Molders STORAGE 18 ;всего литейщиков  
 \* модельный сегмент 1  
 GENERATE (Exponential(1,0,60)) ;заказы каждый час (+-)  
 ASSIGN 1, FN\$Ordertype ;тип работы 1/2  
 TEST E P1,2, Newjob ;если уже было то начинаем  
 ADVANCE 300,180 ;ищем шаблон  
 Commence ASSIGN 2, V\$Size ;размер серии  
 ASSIGN 3, FN\$Weight ;вес компонента  
 ASSIGN 4, V\$Mtime ;время заливки компонента  
 ASSIGN 5, V\$Ddate ;срок изготовления  
 ASSIGN 6, V\$Total ;общий вес заказа  
 GATE SNF Molders, Wait ;есть свободный рабочий?  
 Beg ENTER Molders ;занимаем одного  
 ASSIGN 7, P2 ;P7=повторы для серии  
 Next ADVANCE P4 ;время на 1 компонент  
 LOOP 7, Next ;цикл повтора на серию  
 LEAVE Molders ;освобождаем, подготовлено  
 SAVEVALUE Wtmold+, P6 ;общий веес по всем заказам  
 UNLINK 1, Beg, 1 ;берем след.заказ  
 TABULATE Times ;пишем статистику по времени  
 TERMINATE ;закончили заказ  
 Newjob ADVANCE 4320,1440 ;= для нового нужно делать шаблон  
 TRANSFER ,Commence ;шаблон готов, ставим в очередь  
 Wait LINK 1, P5 ;очередь заказов в порядке скорости изготовления  
 \* модельный сегмент 2  
 GENERATE 420,,,1,2 ;в конце смены заливаем металл  
 Again SUNAVAIL Molders ;все рабочие заняты заливкой!  
 ADVANCE 60 ;цикл заливки на 60 мин  
 SAVAIL Molders ;рабочие могут продолжить свои работы  
 ADVANCE 420 ;ждм 420 мин до след.заливки  
 TABULATE Cast ;пишем статистику по весу металла  
 SAVEVALUE Totcast+, X\$Wtmold ;запишем в общий счетчик  
 SAVEVALUE Wtmold, 0 ;сбросим счетчик  
 TRANSFER ,Again ;перезапустим процесс заливки

\* модельный сегмент 3  
 GENERATE 4800,,,,4 ;один раз в 10 дней  
 SAVEVALUE V\$Day,X\$Totcast ;запишем сколько металла ушло  
 TERMINATE 1

В литейной мастерской над мелкосерийными заказами, поступающими в среднем раз в час, работают 18 литейщиков. Мастерская работает в одну смену с 8-часовым рабочим днем. 30% заказов — новые, а 70% — повторные. Для новых заказов нужны новые формы и модели, которые делают в модельной мастерской за  $72 \pm 24$  ч. Формы повторных заказов требуется найти и подготовить, что занимает  $5 \pm 3$  ч. В заказе может быть от 6 до 24 штук деталей. Масса одной детали варьируется от 3 до 50 кг. Один заказ выполняет (формирует) один рабочий. Формовка занимает 2 мин на кг массы детали. Срок выполнения заказа определяется общим временем формовки плюс технологическое время от 40 до 160 ч на заказ. Заливка металла происходит раз в день в последний час смены. Заливку металла в заготовленные формы выполняют одновременно все рабочие.

Необходимо открыть модель, получить результат моделирования за 30 смен, собрать статистику по времени выполнения заказов (среднее время и станд. отклонение).

Оцените ежедневный расход металла (средняя масса и станд.отклонение), найдите сведения о загрузке рабочих, постройте графики очередей выполнения заказов по типам.

**Задание 3.** Изучите пример модели ASSEMBLY.GPS, где приведена модель производственного участка сборки центробежных насосов, сборка которых осуществляется по заказу клиентов. Заказы прибывают в случайные моменты времени. Интервалы времени между поступлениями двух последовательных заказов распределены по экспоненциальному закону с математическим ожиданием 30 мин.

Когда прибывает заказ, делается две его копии. Оригинал заказа используется для получения двигателя со склада и подготовки его для сборки. Время выполнения этой операции является экспоненциально распределенной случайной величиной со средним значением 8 мин. Первый экземпляр копии используется для заказа и адаптации насоса (время  $10 \pm 2$  мин), А второй экземпляр используется для начала изготовления плиты основания (время 15 мин).

Когда насос и плита основания готовы, производится пробная подгонка (время  $5 \pm 1$  мин). Все три компонента собираются вместе (время распределено по нормальному закону с математическим ожиданием 6 мин и стандартным отклонением 1 мин), когда они имеются налицо. Затем установка разбирается, насос и двигатель подвергаются окраске. Время покраски двигателя  $2 \pm 0,5$  мин, А время покраски насоса распределено по экспоненциальному закону со средним значением 1,5 мин. Плита основания гальванизируется 4 мин. После этого производится окончательная сборка. Время

сборки – нормально распределенная случайная величина с математическим ожиданием 8 мин и стандартным отклонением 1 мин.

Промоделировать сборку 100 центробежных насосов и оценить среднее время их сборки, используя для этого таблицу. Опишем кратко логику работы модели. Транзакты имитируют заказы покупателей. Когда транзакт входит в блок SPLIT, создается еще два транзакта копии. Это позволяет одновременно продолжить выполнение индивидуальных заказов на мотор, насос и плиту основания.

Транзакты, имитирующие насос и плиту, ожидают друг друга в блоках MATCH с метками PUMP(насос) и PLATE(плита). Если и насос, и плита прибыли, то имитируется задержка на их поверочную сборку. После того, как придут все три заказа в блок GATHER, блок ADVANCE имитирует тестовую сборку трех компонентов изделия друг к другу. Затем три детали снова разделяются для окончательной отделки. Блок ASSEMBLE (сборка) с меткой BUILD вызывает отсрочку окончательной сборки, пока не поступят все компоненты. В таблице TRANSIT собирается распределение времени выполнения заказов. Единица модельного времени 0,1 мин.

Transit TABLE M1,200,200,20

\* модельный сегмент 1

GENERATE	(Exponential(1,0,300))	;поступил новый заказ
SPLIT	2,Factory,1	;сделаем копии под 3 детали
QUEUE	Motor	;это очередь за двигателем
SEIZE	Motor	;занят участок двигателя
DEPART	Motor	;выходим из очереди
ADVANCE	200,100	;берем мотор со склада
RELEASE	Motor	;свободен участок двигателя
TRANSFER	,Tryout	;идем на сборку
Factory TEST E	P1,2,Baseplate	;это насос P1=2 ?
QUEUE	Pumps	;это очередь за насосом
SEIZE	Pumps	;занят участок насосов
DEPART	Pumps	;выходим из очереди
ADVANCE	180,120	;готовим насос
Pump MATCH	Plate	;ждем станину
ADVANCE	50,10	;проверяем на станине
RELEASE	Pumps	;свободен участок насосов
TRANSFER	,Tryout	; идем на сборку
Baseplate QUEUE	Base	; это станина P1=3
SEIZE	Base	; занят участок плит
DEPART	Base	; выходим из очереди
ADVANCE	80,20	; готовим плиту
Plate MATCH	Pump	; ждем насос

	ADVANCE	50,10	; проверяем на насос на станине
	RELEASE	Base	; свободен участок плит
Tryout	GATHER	3	; ждем 3 детали для проверки
	ADVANCE	60	; пробная сборка/разборка
	TEST E	P1,1,Finish	;это мотор?(P1=1)
	SEIZE	Paint1	;красим мотор
	ADVANCE	100,20	; красим мотор
	RELEASE	Paint1	; покрасили мотор
	TRANSFER	,Build	;идем на сборку
Finish	TEST E	P1,2,Basplate	;это насос?(P1=2)
	SEIZE	Paint2	;красим насос
	ADVANCE	120,30	; красим насос
	RELEASE	Paint2	; покрасили насос
	TRANSFER	,Build	; идем на сборку
Basplate	SEIZE	Galvanize	;оцинкуем станину
	ADVANCE	120,30	; гальванизация!
	RELEASE	Galvanize	; готова станина
Build	ASSEMBLE	3	;ждем все 3 детали
	ADVANCE	150,30	;собираем агрегат
	TABULATE	Transit	;запишем время работы
	TERMINATE	1	;заказ готов

Необходимо запустить модель; провести моделирование 100 заказов; исследовать коэффициенты использования сборочных участков; определить время выполнения заказов; построить график времени ожидания заказов до начала сборки.

**Задание 4.** Изучите пример модели BICYCLE.GPS, где приведена модель работы велосипедной мастерской.

```

Orders FUNCTION P$Department,L6
1,Order/2,Frame/3,Saddle/4,Handlebars/5,Wheels/6,Pedals
Transit TABLE M1,100,100,20
Clerks STORAGE 2
Framers STORAGE 3
Saddlers STORAGE 1
Handlers STORAGE 1
Wheelers STORAGE 1
Pedalers STORAGE 1
Builders STORAGE 4
Packers STORAGE 3 ; нужны ли упаковщики?

```

\* модельный сегмент 1  
GENERATE 50,10 ;поступление заказа на велосипед  
SPLIT 5,Factory,Department ;делаем копии для 6 процессов

Order ENTER Clerks  
ADVANCE 80,10 ;1 – подготовка бумаг  
LEAVE Clerks

Invoice MATCH Bicycle ;ждем велосипед  
TERMINATE ;заказ готов

\* модельный сегмент 2  
Factory TRANSFER FN,Orders ;маршрутизация по работникам

\* модельный сегмент 3  
Frame ENTER Framers  
ADVANCE (Exponential(1,0,65)) ;2 – делаем раму  
ADVANCE 12,2 ;проверяем раму  
LEAVE Framers  
TRANSFER ,Build ;идем на сборку

\* модельный сегмент 3  
Saddle ENTER Saddlers  
ADVANCE 6,3 ;3 – делаем седло  
ADVANCE 3,1 ;проверяем седло  
LEAVE Saddlers  
TRANSFER ,Build ;идем на сборку

\* модельный сегмент 4  
Handlebars ENTER Handlers  
ADVANCE 4,2 ;4 – делаем руль  
ADVANCE 3,1 ;проверяем руль  
LEAVE Handlers  
TRANSFER ,Build ;идем на сборку

\* модельный сегмент 5  
Wheels ENTER Wheelers  
ADVANCE 3,1 ;5 – делаем колеса  
ADVANCE 3,1 ;проверяем колеса  
LEAVE Wheelers  
TRANSFER ,Build ;идем на сборку

\* модельный сегмент 6  
Pedals ENTER Pedalers  
ADVANCE 5,1 ;6 – делаем педали  
ADVANCE 3,1 ;проверяем педали

LEAVE Pedalers  
 TRANSFER ,Build ; идем на сборку  
 \* модельный сегмент 7  
 Build ASSEMBLE 5 ; собираем  
 ENTER Builders  
 ADVANCE (Normal(1,90,10)) ; время на сборку  
 ADVANCE 35,5 ; время на проверку  
 LEAVE Builders  
 Bicycle MATCH Invoice ; ждем договор/счет/деньги  
 ENTER Packers  
 ADVANCE 40,5 ; упаковываем  
 LEAVE Packers  
 TABULATE Transit  
 TERMINATE ; сдали заказ  
 \* модельный сегмент 8  
 GENERATE 480 ; засекаем 1 раб.смену  
 TERMINATE 1

В мастерской собирают велосипеды индивидуально под заказ клиентов. Заказы поступают в среднем каждые 50 минут. При получении заказа начинается 6 процессов подготовки узлов. В завершении процесса все узлы собираются в велосипед.

Необходимо открыть модель; провести моделирование 500 заказов; исследовать коэффициенты использования сборочных участков; определить время выполнения заказов.

Теперь, измените модель таким образом, чтобы операцию упаковки выполнял свободный от работы и наименее загруженный специалист вместо специально выделенных упаковщиков. Дополнительно сбалансируйте загрузку всех специалистов, изменив количество каналов в Storage.



## ПРИЛОЖЕНИЕ 2. Дополнительное описание операторов и блоков GPSS

### Блоки работы со списками пользователя

#### Блок LINK

Назначение. Блок помещает активный транзакт в список пользователя.

Синтаксис: LINK A, B[, C]

Операнд	Назначение	Значение	Значение по умолчанию
<b>A</b>	Список пользователя, в который должен быть помещен входящий в блок транзакт.	Имя, число, СЧА	Обязательный параметр
<b>B</b>	Дисциплина помещения нового транзакта в список пользователя.	LIFO, FIFO, СЧА транзакта	Обязательный параметр
<b>C</b>	Имя блока, куда переходит транзакт, если индикатор компоновки списка пользователя находится в выключенном состоянии (сброшен)	Имя, число, СЧА	Нет

Особенности выполнения:

При помещении транзакта в список он удаляется из всех других списков, кроме групп транзактов и списков прерываний.

Транзакт остается в списке пользователя до тех пор, пока какой-либо другой транзакт не войдет в блок UNLINK и не считает его из списка.

Если в операнде B указано FIFO (первым вошел, первым вышел), транзакты помещаются в конец списка. Если в операнде B указано LIFO (последним вошел, первым вышел), вновь прибывшие транзакты помещаются в начало списка.

Если в операнде B не используется LIFO или FIFO, то может применяться СЧА транзакта - PR, M1 или P. Может быть использована косвенная адресация. Если задано PR, транзакты помещаются в список пользователя в приоритетном порядке. Если задан номер параметра, транзакт помещается в список пользователя позади тех транзактов, значение соответствующего параметра которых меньше, чем у входящего транзакта.

Если задан операнд C, то используется флаг «Индикатор компоновки». Если индикатор компоновки списка пользователя выключен(сброшен), блок LINK не поместит транзакт в список. Вместо этого транзакт перейдет к блоку указанному в операнде C, после чего индикатор компоновки будет включен (установлен). Следующие транзакты, входящие в блок LINK, будут помещены в список пользователя. Индикатор компоновки управляется блоками LINK и UNLINK. Он выключается (сбрасывается), когда блок UNLINK определяет, что список пользователя пуст.

Примеры:

1. LINK CHANNEL, FIFO

Транзакт, вошедший в блок LINK, будет добавлен в конец очереди с именем CHANNEL.

2. LINK CHANNEL, PR, LABEL\_CH

Если список пользователя CHANNEL пуст (индикатор компоновки списка пользователя выключен), транзакт перейдет к блоку с именем LABEL\_CH и индикатор компоновки будет включен. Следующие транзакты, входящие в блок LINK, будут помещены в список пользователя. Индикатор компоновки выключится, когда блок UNLINK определит, что список пользователя пуст.

## 1.2. Блок UNLINK

Назначение. Блок считывает транзакты из списка пользователя.

Синтаксис. UNLINK [X] A,B,[C],[D],[E],[F]

Операнд	Назначение	Значение	Значение по умолчанию
X	Операция сравнения для операндов D и E.	Условная операция отношения	E
A	Список пользователя, из которого будет считан один или несколько транзактов.	Имя, число, СЧА	Обязательный операнд.
B	Блок, куда переходят считанные транзакты.	Имя, число, СЧА	Обязательный операнд
C	Максимальное количество транзактов, которые будут считаны	Имя, число, СЧА, ALL	ALL
D	Определяет условия считывания транзактов из очереди	Имя, число, СЧА, BACK	Нет
E	Используется совместно с операндом D и X для определения условия считывания транзактов из списка	Имя, число, СЧА	Нет
F			Нет

### Особенности выполнения.

Блок UNLINK считывает транзакты из списка пользователя и направляет их в указанный блок. Транзакты, которые необходимо исключить, можно выбирать, также можно наложить ограничение на количество исключаемых транзактов.

Если транзакт входит в блок UNLINK, когда в списке нет транзактов, индикатор компоновки списка пользователя сбрасывается. Можно ограничить число считываемых из списка транзактов с помощью операнда C. Если операнд C опущен, то берется ALL.

Если операнды D, E и условный оператор опущены, исключаются все транзакты с начала списка, пока список не будет исчерпан, или не будет достигнут предел исключаемых транзактов (операнд C).

Операнд D может быть булевой переменной, номером параметра или словом «BACK». Если операнд D является булевой переменной, он вычисляется относительно транзакта, находящегося в списке пользователя, и если результат не нулевой, транзакт исключается. Если в операнде D указано BACK, транзакты исключаются, начиная с конца списка пользователя, пока не будет достигнут предел.

В противном случае операнд вычисляется относительно транзакта, находящегося в списке пользователя, и используется в качестве номера параметра, значение которого возвращается членом списка пользователя, как конечный результат. Это конечное значение сравнивается с результатом вычисления операнда E.

Если операнд D задает параметр, а E не используется, параметр транзакта из списка пользователя сравнивается с таким же параметром активного транзакта. Если они равны, транзакт, находящийся в списке, считается из списка.

Операнд E используется только в том случае, если используется оператор отношения. В этом случае обязательно требуется операнд D. Список пользователя проверяется, начиная с начала. Если условие, заданное оператором отношения, выполняется для операндов D и E, исключается каждый транзакт, (вплоть до предела (операнд C)). Если в операнде E используется СЧА транзакта, он вычисляется относительно активного транзакта. Оператор отношения используется для определения соотношения между атрибутом транзакта (операнд D) и значением операнда E. Если отношение выполняется, то транзакт считается из списка. В качестве оператора отношения могут быть использованы E, G, GE, L, LE или NE. По умолчанию в качестве оператора отношения используется E (равно).

Операнд F используется для определения блока, куда будет направлен входящий транзакт, в случае, если предел исключения транзактов (операнд C) не может быть достигнут, или из списка пользователя не может быть удален ни один транзакт.

#### Примеры.

1. UNLINK CHANNEL, LABEL\_CH,1

Считывается один транзакт из начала списка CHANNEL и он направляется в блок с именем LABEL\_CH. Транзакт, вошедший в UNLINK , переходит к следующему блоку

2. UNLINK BUFFER, FAC\_2,1,BACK

Считывается из списка пользователя с именем BUFFER один транзакт с конца списка и он направляется в блок с именем FAC\_2.

3. UNLINK E P\$BUF,MET\_1,ALL,COND,P\$COND,MET\_2

Считываются из списка пользователя, номер которого записан в параметре BUF вошедшего транзакта, и направляются в блок с именем MET\_1 все транзакты,

содержимое параметра COND которых равно содержимому одноименного параметра вошедшего транзакта. Если таких транзактов в списке нет, то вошедший транзакт будет направлен в блок с именем MET\_2, в противном случае - к следующему блоку.

## 2. Работа с приборами в режиме прерывания

### 2.1. Блок PREEMPT

Назначение. Блок позволяет транзакту, в зависимости от условий, заданных в операндах блока, занять прибор, даже если он занят другим транзактом (абсолютный приоритет).

Синтаксис: PREEMPT A,[B],[C],[D],[E]

Операнд	Назначение	Значение	Значение по умолчанию
A	имя занимаемого прибора	Имя, число, СЧА	Обязательный операнд
B	Условия занятия прибора	PR	Режим прерывания
C	Блок, куда должен перейти прерванный транзакт	Имя, число, СЧА	Нет
D	Параметр прерванного транзакта, в который записывается оставшееся время, если транзакт удаляется из списка будущих событий	Имя, число, СЧА	Нет
E	Режим удаления	RE	Нет

#### Особенности выполнения:

1. Если прибор находится в состоянии «не готов к использованию», то транзакт помещаются в список задержки прибора в конец своего приоритетного класса.
2. Операнд B задает приоритетный режим (PR) или режим прерывания, если операнд опущен.
3. При работе в приоритетном режиме транзакт, занимающий уже прибор, может быть прерван только транзактом, приоритет которого выше приоритета данного транзакта. Если приоритет входящего транзакта ниже, то он помещается в список задержки в конец своего приоритетного класса.
4. В режиме прерывания, если прибор уже используется, поступивший транзакт помещается в список отложенных прерываний. Транзактам из списка отложенных прерываний право занять прибор предоставляется раньше, чем транзактам из списков прерываний или задержки.
5. Прерванный транзакт теряет управление прибором, но может претендовать на дообслуживание, когда прервавший его транзакт входит в соответствующий блок RETURN (если только не задан операнд E).
6. Прерванные транзакты помещаются в список прерываний в порядке приоритета. Операнд C задает блок, куда должен попытаться перейти прерванный

транзакт в этот же момент модельного времени.

7. Если прерываемый транзакт находится в списке будущих событий(вошел в блок ADVANCE), то для него выполняются следующие действия:

- вычисляется остаток времени, в течение которого транзакт должен был находиться в блоке ADVANCE, равный разнице планируемого времени выхода транзакта из блока ADVANCE и текущего значения абсолютного условного времени; Операнд D задает номер параметра прерванного транзакта, куда записывается вычисленный остаток времени. Если такой параметр не существует, то он создается.

- транзакт удаляется из списка будущих событий;
- транзакт рассматривается, как находящийся в состоянии прерывания и помещается в список прерывания;

- счетчик прерываний увеличивается на единицу.

9. Прерываемый транзакт может находиться в списке текущих событий (например, когда блок ADVANCE имеет нулевую задержку). В этом случае удаление транзакта из списка текущих событий и перевод его в состояние прерывания производится не сразу. Сначала устанавливается индикатор состояния прерывания. Транзакт, занимающий прибор, будет обрабатываться интерпретатором как обычно и перейдет в состояние прерывания только тогда, когда оно войдет в блок ADVANCE с ненулевой задержкой.

10. Поле E задает один из следующих режимов:

- режим удаления(RE). Задание этого режима означает, что прерванный транзакт более не претендует на пользование прибором. Прерванный транзакт пытается войти в блок, заданный полем C (если в поле E стоит RE, то должно быть указано и поле C). При использовании RE прерванный транзакт не должен входить в блоки RELEASE и RETURN, связанные с прерванным транзактом;

- если режим RE не задан, т.е. поле E - пусто, то прерванный транзакт по возвращении в список текущих событий будет вновь пытаться занять прибор.

11. Прерванный транзакт борется за прибор, даже если он перемещен операндом C (если RE не используется в операнде E). Если прерванный транзакт все еще борется за прибор, то попытка транзакта войти в блок TERMINATE приводит к ошибке. Такой транзакт перед входом в блок TERMINATE должен войти в блок RELEASE или блок RETURN.

12. Транзакт может быть прерван на любом количестве приборов и продолжать циркулировать в модели при выполнении следующих двух условий:

- транзакт вошел в блок ADVANCE с положительным временем;
- транзакту не разрешается покинуть блоки ASSEMBLE, GATHER или MATCH до тех пор, пока в них не войдет заданное число транзактов из того же семейства.

13. Прибор может быть захвачен любое количество раз, но не два раза подряд одним транзактом.

14. При использовании операндов C, D, E следует учитывать следующее:

- при задании полей D и (или) E, поле C также должно быть задано;
- если приоритетный режим не задан (PR в операнде B), то операнды C, D и (или) E игнорируются.

Примеры.

1. PREEMPT UNIT

В режиме прерывания, если прибор UNIT уже занят, поступивший транзакт помещается в список отложенных прерываний

2. PREEMPT UNIT,PR,NODE2,P\_Time

В приоритетном режиме, если приоритет поступившего транзакта выше приоритета транзакта, занимающего уже прибор, возникает прерывание. Если приоритет входящего транзакта ниже, то он помещается в список задержки в конец своего приоритетного класса. Прерванный транзакт пытается перейти в блок с меткой NODE2, а остаток времени до окончания обслуживания будет помещен в параметр транзакта с именем P\_Time.

## 2.2. Блок RETURN

Назначение. Блок позволяет активному транзакту освободить занятый прибор или исключить транзакт из списка прерываний прибора.

Синтаксис. RETURN A

Опе-ранд	Назначение	Значение	Значение по умолчанию
A	Имя освобождаемого прибора	Имя, число, СЧА	Обязательный операнд

Пример RETURN WorkStation

Освобождается прибор с именем WorkStation

## 3. Расширенные описания основных блоков

### 3.1. Блок TRANSFER (полное описание)

Назначение. Блок передает транзакт на указанный блок.

Синтаксис. TRANSFER [A],[B],[C],[D]

Операнд	Назначение	Значение	Значение по умолчанию
A	Режим блока	BOTH, ALL, PICK, FN, P, SBR, SIM, Имя, число, СЧА	Режим безусловной передачи
B	Номер или метка блока. Номер или имя параметра в режиме P	Имя, число, СЧА	Нет

<b>C</b>	Номер или метка блока. Приращение в режимах FN и P	Имя, число, СЧА	Нет 0 для режима P
<b>D</b>	Приращение номера блока для режима ALL	Имя, число, СЧА	1

### Особенности выполнения.

Блок TRANSFER может функционировать в одном из 9 режимов. Операнд А используется для определения режима, в котором функционирует блок. Значение операндов В и С зависит от режима. Если не задан операнд, определяющий блок, куда будет передан транзакт, то используется блок, следующий за блоком TRANSFER.

#### **Режим безусловной передачи**

Когда операнд А отсутствует, блок TRANSFER функционирует в режиме безусловной передачи. В этом режиме активный транзакт всегда переходит к блоку, заданному в операнде В.

TRANSFER ,NO\_SERV

При входе транзакта в блок TRANSFER, он передается в блок с меткой NO\_SERV.

#### **Режим статистической передачи**

В этом режиме активный транзакт переходит к блоку, заданному в операнде С, с вероятностью, заданной в операнде А. Операнд А может быть положительной дробью, меньшей единицы или целым положительным числом. Если операнд А - целое число, оно интерпретируется как доля от тысячи. Альтернативный переход задается в операнде В. Если операнд В пропущен, транзакт переходит к следующему по порядку блоку.

TRANSFER .75,CHANNEL\_2

При входе транзакта в блок TRANSFER, с вероятностью 0.75 он переходит к блоку с именем CHANNEL\_2. С вероятностью 0.25 он переходит к следующему по порядку блоку.

#### **Режим BOTH (ОБА)**

В данном режиме проверяется возможность входа транзакта в блок, метка или номер которого указаны в операнде В. Если транзакту отказано во входе в данный блок, проверяется блок, метка или номер которого указаны в операнде С. Транзакт направляется в первый блок, в который ему будет позволено войти. Если ни один из блоков не принимает транзакт, он остается в блоке TRANSFER до тех пор, пока не сможет войти в один из них.

TRANSFER BOTH, CHANNEL\_1, CHANNEL\_2

При входе транзакта в блок TRANSFER, проверяется блок с меткой CHANNEL\_1. Если транзакт не может войти в него, проверяется блок с меткой CHANNEL\_1. Если транзакту отказано во входе и второй блок, он остается в блоке TRANSFER до тех пор, пока не сможет войти в один из блоков.

### **Режим ALL (ВСЕ)**

В этом режиме проверяется блок, метка или номер которого указаны в операнде В. Если этот блок не может принять активный транзакт, то последовательно проверяются все блоки до тех пор, пока не будет достигнут блок, метка или номер которого заданы в операнде С, или один из проверенных блоков не примет транзакт до достижения блока, заданного в операнде С. Номер каждого последовательно проверяемого блока вычисляется путем добавления операнда D к номеру ранее проверенного блока. Если операнд D не используется, проверяется каждый блок между блоками, заданными в операндах В и С. Если не используется операнд С, проверяется только один блок. Блоки с большим номером, чем операнд С, не проверяются. Транзакт направляется в первый блок, принявший его. Если блок, принимающий транзакт, отсутствует, транзакт остается в блоке TRANSFER до тех пор, пока не сможет войти в один из блоков.

TRANSFER ALL, CHANNEL\_1, CHANNEL\_2, 2

Когда транзакт входит в данный блок TRANSFER, проверяется блок с меткой CHANNEL\_1. Если транзакт не может войти в него, проверяется каждый блок с номером на 2 больше предыдущего. Если все проверенные блоки отказывают транзакту во входе, проверка заканчивается на блоке с меткой CHANNEL\_2 или на блоке перед ним. Если ни один из блоков не принял транзакт, он остается в блоке TRANSFER, пока не сможет покинуть его.

### **Режим PICK (выборочный)**

В режиме PICK номер блок для перехода транзакта выбирается случайно из интервала, заданного операндами В и С.

TRANSFER PICK, CHANNEL\_1, CHANNEL\_2

Когда транзакт входит в этот блок TRANSFER, номер нового блока выбирается случайно в интервале между блоками с именами CHANNEL\_1 и CHANNEL\_2.

### **Режим FN (функциональный)**

В режиме FN номер блок для перехода транзакта выбирается путем вычисления функции, заданной в операнде В, с прибавлением к этому значению необязательного приращения, заданного в операнде С.

TRANSFER FN,Exp,5

Когда транзакт входит в данный блок TRANSFER, номер блок для перехода вычисляется как текущее значение функции с именем Exp плюс 5.

### **Режим P (параметрический)**

В параметрическом режиме активный транзакт переходит к блоку, вычисленному, как сумма значения параметра, заданного операндом В, и значения, заданного операндом С.

TRANSFER P,End\_Proc,1

Когда транзакт входит в этот блок TRANSFER, он переходит в блок, следующий за блоком, номер которого задан в параметре транзакта с именем End\_Proc.



### Режим SBR (подпрограммный)

В режиме SBR активный транзакт всегда переходит к блоку, заданному в операнде В. Номер блока TRANSFER помещается в параметр, указанный в операнде С.

TRANSFER SBR,Proc, End\_Proc

Когда транзакт входит в данный блок TRANSFER, он переходит в блок с меткой Proc. Номер блока TRANSFER помещается в параметр с именем End\_Proc.

### Режим SIM (одновременный)

В режиме SIM активный транзакт переходит к одному из двух местоположений в зависимости от индикатора задержки транзакта. Если индикатор задержки установлен, транзакт переходит к блоку, заданному в операнде С, а если индикатор задержки сброшен, транзакт переходит к блоку, заданному в операнде В.

Индикатор задержки транзакта устанавливается всегда, когда транзакту отказывается во входе в какой-либо блок. Индикатор задержки остается установленным до тех пор, пока транзакт не войдет в блок TRANSFER SIM.

TRANSFER SIM,Nodelay\_Place,Delay\_Place

Когда транзакт входит в этот блок TRANSFER, он немедленно направляется в блок с меткой Delay\_Place, если его индикатор задержки установлен, или в блок с меткой Nodelay\_Place, если его индикатор задержки сброшен. После перехода транзакта его индикатор задержки всегда сбрасывается

## 3.2. Блок SELECT

Блок выбирает первый объект определенного типа, который удовлетворяет заданному условию. Номер объекта записывается в параметр активного транзакта.

Синтаксис: SELECT X A,B,C,[D],[E],[F]

Операнд	Назначение	Значение	Значение по умолчанию
X	Определяет условный или логический режим функционирования блока.	Условный оператор или СЛЖ	Обязательный операнд
A	Номер параметра активного транзакта, в который записывается номер объекта, удовлетворяющего заданному условию	Имя, число, СЧА	Обязательный операнд
B	Минимальный номер объекта данного типа, для которого проверяется заданное условие	Имя, число, СЧА	Обязательный операнд
C	Максимальный номер объекта данного типа, для которого проверяется заданное условие	Имя, число, СЧА	Обязательный операнд
D	Величина для сравнения в режиме отношения. Не используется в режиме выбора максимального или минимального	Имя, число, СЧА	Обязательный операнд для режима отношения
E	СЧА типа объекта для режима отношения и выбора максимального или минимального	СЧА типа объекта	Обязательный операнд для режима отношения и выбора максимального или

			МИНИМАЛЬНОГО
F	Определяет блок для перехода транзакта, если объект не выбран. Не используется в режиме выбора максимального или минимального	Имя, число, СЧА	Следующий блок

#### Примеры.

1. SELECT E 1,10,20,0,F,NO\_UNIT

В режиме отношения по стандартному числовому атрибуту F (состояние прибора) среди приборов с номерами от 10 до 20 ищется первый прибор со значением F=0 (прибор свободен). Номер найденного прибора записывается в первый параметр транзакта. Если такого прибора нет, то в первый параметр записывается 0 и транзакт переходит в блок с меткой NO\_UNIT

2. SELECT MAX 1,5,15,,Q

В режиме выбора максимального элемента по стандартному числовому атрибуту Q (длина очереди) среди очередей с номерами от 5 до 15 ищется очередь с максимальным значением данного атрибута.

3. SELECT SE 3,1,7

В логическом режиме по стандартному логическому атрибуту SE (память пуста) среди памятей с номерами от 1 до 7 ищется первая пустая. Номер найденной памяти записывается в третий параметр транзакта. Если такой памяти нет, то в третий параметр записывается 0.

#### **4. Оператор FUNCTION (полное описание)**

Оператор определяет функцию GPSS, заданную таблично.

Синтаксис. <имя функции> FUNCTION A, B

Имя функции – числовое или символическое имя, обязательный параметр.

A - Аргумент функции; обязательный операнд. Допустимые значения – имя, число, СЧА.

B - Тип функции (одна буква) и количество пар данных в списке данных функции. Обязательный операнд.

Последующие обращения к СЧА типа FN вычисляют функцию и возвращают результат. Существует несколько типов функций. Тип определяется операндом B команды FUNCTION.

За каждой командой FUNCTION сразу же должен следовать список пар данных, разделенных символами «/». Каждая пара данных определяет значения аргумента X и значения функции Y, разделенные запятой. Списки данных используются для вычисления значения функции по заданным значениям аргумента.

Существует 5 различных типов функций.

1. Функции типа C – непрерывные числовые функции.

В списке данных функций типа С значения X и Y должны быть целочисленными (Integer) или вещественными (Real). Значения X и Y хранятся, как числа с плавающей точкой двойной точности.

Вычисление функции начинается с вычисления аргумента. Далее определяется интервал  $(X_i; X_{i+1})$ , на котором находится вычисленное значение и на этом интервале выполняется линейная интерполяция двойной точности с использованием соответствующих значений  $Y_i$  и  $Y_{i+1}$ . Результатом является значение функции двойной точности. Если аргумент попадает за предельные значения области определения функции, возвращается значение функции в ближайшей предельной точке.

Примеры.

- 1) ART FUNCTION X1, C3  
1.1,10.1/20.5,98.7/33.3,889.2

Оператор определяет кусочно-линейную функцию с двумя линейными участками. Если мы обращаемся к функции FN\$ART, то по значению сохраняемой величины X1 вычисляется функция в соответствии с заданной совокупностью точек.

2) Пример приближенного представления обратного экспоненциального распределения со средним, равным 1.

```
Xpdis FUNCTION RN1, C24  
0,0/.1,.104/.2, .222/.3, .355/.4, .509/.5, .69/.6, .915/.7, 1.2/.75, 1.38  
.8, 1.6/.84, 1.83/.88, 2.12/.9, 2.3/.92, 2.52/.94, 2.81/.95, 2.99/.96, 3.2  
.97, 3.5/.98, 3.9/.99, 4.6/.995, 5.3/.998, 6.2/.999, 7/.9998, 8
```

2. Функции типа D - дискретные функции.

В списках данных функций типа D значения X должны быть целочисленными или вещественными, а значения Y – целочисленными, вещественными или именами.

Функции типа D задают одно и то же значение функции  $Y[i]$  для всех значений аргумента  $X[i-1] < X \leq X[i]$ . Значения X в списке данных функции должны быть неубывающими. Внутренне они сохраняются как числа двойной точности. Когда вычисляется функция, значения X в списке данных функции просматриваются от наименьшего к наибольшему. Когда найдено значение X, которое больше или равно текущему значению аргумента, возвращается соответствующее ему значение Y. Если такое значение X отсутствует, возвращается значение Y или именованная величина, соответствующая самому большому значению X.

Примеры:

- 1) LIR FUNCTION X\$A2, D5  
1.1,6.9/2.1,7/6.33,9.4/7,10/9.9,12.01
- 2) RAF FUNCTION RN1, D5  
0,0/.2,7.2/.4,6.667/.8,9.92/1.0,10

### 3. Функции типа E - дискретные атрибутивные функции.

В качестве значений Y в списке данных дискретных атрибутивных функций используются стандартные числовые атрибуты.

Функции типа E вычисляются тем же способом, что и функции типа D, за исключением того, что значение функции вычисляется косвенно через стандартный числовой атрибут.

Пример:

```
Edisc FUNCTION P7, E4
```

```
1,FR2/2,FR7/3,FR9/4,FR11
```

Значение функции Edisc равно коэффициенту использования приборов 2, 7, 9, 11 в зависимости от значения аргумента, содержащегося в параметре транзакта с именем 7.

### 4. Функции типа L - списковые функции.

Функции типа E вычисляются тем же способом, что и функции типа D, за исключением того, что значения X должны начинаться с 1 и увеличиваться на 1 для каждой последующей пары данных. Если аргумент меньше 1 или превосходит наибольшее заданное значение X, происходит останов по ошибке. Списковые функции требуют меньшего времени выполнения по сравнению с дискретными.

Пример:

```
Listtype FUNCTION Q$Barber, L5
```

```
1,PAR1/2,PAR2/3,PAR3/4,PAR4/5,PAR5
```

### 5. Функции типа M - списковые атрибутивные функции.

Функции типа M вычисляются тем же способом, что и функции типа L, за исключением того, что в качестве значений Y в списке данных используются стандартные числовые атрибуты и значение функции вычисляется косвенно через стандартный числовой атрибут.

Пример:

```
Mlist FUNCTION X$Namel, M5
```

```
1, Q$Nnaml/2, Q$NnamX/3, Q$Nnam4/4, Q$Nnam6/5, F$Tanl
```

Правила использования функций

- Значения X в списке данных функции должны быть неубывающими;
- Спискам данных функций никогда не присваиваются номера строк;
- Все поля команды FUNCTION обязательны;
- Все значения X и Y в списке данных функции обязательны;
- Количество пар данных, указанное в операнде B команды FUNCTION, должно совпадать с количеством пар, разделенных символами «/» в списке данных функции;
- Списки данных функций не имеют полей комментария;
- В списке данных функций за значением X следует запятая, за которой следует значение Y, за ним «/» или перевод строки, затем опять значение X;
- Функции типа C, L и D не могут иметь СЧА в качестве значений Y;

- Функции типа E и M должны содержать СЧА или выражение в качестве значений Y;
- Функции типа L и M не могут иметь случайные аргументы;
- Списки данных функций типа L и M должны иметь последовательно возрастающие значения X, начинающиеся с 1.

## СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Хоар Ч. Взаимодействующие последовательные процессы: М.: Мир. 1989. 264 с.
2. Семиков В.Л., Ушаков В.Д. Теория организации. М.: Рид Групп. 2011. 368 с.
3. Бусленко Н.П., Калашников Н.Н., Коваленко И.Н. Лекции по теории сложных систем. М.: Советское радио. 1973. 441 с.
4. Крупский В.Н., Плиско В.Е. Теория алгоритмов. М.: Изд. центр «Академия». 2005.
5. Черненький В.М. Алгоритмический метод описания дискретных процессов функционирования систем /Информационно-измерительные и управляющие системы, 2016, - Т. 14 , № 12 .- С. 11 - 21
6. Бусленко Н.П. Моделирование сложных систем. М.: Наука. 1978.
7. Аляев Ю.А., Тюрин С.Ф. Дискретная математика и математическая логика. М.: Финансы и статистика. 2006. 368 с.
8. Советов Б.Я. Моделирование систем: Учебник для бакалавров / Б.Я. Советов, С.А. Яковлев. - М.: Юрайт, 2013. - 343 с.
9. Лоу А., Кельтон В. Имитационное моделирование [Simulation Modeling and Analysis]. СПб.: Питер. 2004. 848 с.
10. Духанов А.В. Имитационное моделирование сложных систем. Владимир: Изд-во Владим. гос. ун-та. 2010. 115 с.
11. Кобелев Н.Б., Половников В.А., Девятков В.В. Имитационное моделирование: Учебное пособие // Под общей редакцией д.э.н. Н.Б. Кобелева. / КУРС, НИЦ ИНФРА-М, 2013, 368 с. ISBN 978-5-905554-17-9
12. Карпов Ю. Имитационное моделирование систем. Введение в моделирование с AnyLogic 5 / Ю. Карпов. - СПб.: BHV, 2009. - 400 с.
13. Боев В. Моделирование систем. Инструментальные средства GPSS World. / В. Боев. - СПб.: BHV, 2012. - 368 с.
14. Черненький В.М. Процессно - ориентированная концепция системного моделирования АСУ: Дисс. док. техн. наук - М., 2000. – 350 с.
15. Методология и технология имитационных исследований сложных систем: современное состояние и перспективы развития: Моногр./ В.В. Девятков - М.: Уз. учеб.: ИНФРА-М, 2013. - 448 с.: 60х90 1/16. - (Научная книга). (п) ISBN 978-5-9558-0338-8, 200 экз.

16. Якимов И.М., Кирпичников А.П., Исаева Ю.Г. Сравнение систем имитационного моделирования вероятностных объектов с графическим вводом структурных схем /Известия Самарского научного центра Российской академии наук, том 18, №2(3), 2016 С. 977-981
17. The Simio Reference Guide, version 8 / SIMIO LLC, 2017 [электронный ресурс]
18. Rapid Modeling Solutions: Introduction to Simulation and Simio / Dennis Pegden, David Sturrock. SIMIO LLC, 2014 [электронный ресурс]
19. Миронов А.М. Теория процессов. Изд-во НОУ Институт программных систем. – Университет г. Переславля им. А.К. Айламазяна, 2008. – 346 с.
20. Боев В. Д., Кирик Д. И., Сыпченко Р. П. Компьютерное моделирование: Пособие для курсового и дипломного проектирования. — СПб.: ВАС, 2011. — 348 с.

**Черненко В.М.**

Доктор технических наук, заведующий кафедрой  
“Системы управления” МГТУ им. Н.Э. Баумана

**Черненко М.В.**

Доцент кафедры “Системы обработки информации  
и управления” МГТУ им. Н.Э. Баумана

Рецензенты:

Карпов Валерий Иванович д.т.н., профессор,  
Девятков Владимир Васильевич д.т.н., профессор

**ПРОЦЕССНО-АГРЕГАТИВНЫЕ СИСТЕМЫ  
ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ**

Подписано в печать

Формат 60х90/16. Печ. л. 10.

Печать офсетная. Бумага офсетная.

Тираж 500 экз. Заказ №