СИСТЕМЫ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ


САПР
СИСТЕМЫ
АВТОМАТИЗИРОВАННОГО
ПРОЕКТИРОВАНИЯ

В.Г. Федорук
В.М. Черненький

Информационное и прикладное программное обеспечение

Под редакцией
доктора технических наук, профессора,
лауреата Государственной премии СССР
И. П. НОРЕНКОВА

МОСКВА «ВЫСШАЯ ШКОЛА» 1986

ББК 32.97
С 40
УДК 681.3

Допущено Министерством высшего и среднего специального образования СССР в качестве учебного пособия для студентов технических вузов

Рецензенты:

кафедра «Конструирование и производство интегральных микросхем» Московского института электронной техники (заведующий кафедрой д-р техн. наук, проф. Г. Г. Казенков) и д-р техн. наук, проф. В. Я. Петров (Московский институт инженеров гражданской авиации)

С40 **Системы** автоматизированного проектирования: Учеб. пособие для втузов: В 9 кн. /Под ред. И. П. Норенкова. Кн. 3: В. Г. Федорук, В. М. Черненький. Информационное и прикладное программное обеспечение. — М.: Высш. шк., 1986. — 159 с.: ил.

35 к.

В третьей книге пособия «Системы автоматизированного проектирования» рассматриваются состав и структура информационного обеспечения САПР, организации обмена данными между подсистемами, методологии разработки сложных программных комплексов; даются классификация и примеры использования в САПР банков данных, программ моделирования и оптимизации технических объектов.

С $\frac{2405000000-222}{001(01)-86}$ 156—86

ББК 32.97
6Ф73

Введение

Проектировщик обращается к средствам САПР для выполнения конкретной проектной процедуры или получения сведений, необходимых для принятия обоснованных решений. Эти функции не могут быть выполнены без специального программного (ПО) и информационного (ИО) обеспечений. Функционируя на технических средствах САПР в рамках общего программного обеспечения, ПО и ИО реализуют математическое, лингвистическое, а также поддерживают методическое и организационное обеспечения.

Специальное ПО и ИО — сложные и дорогостоящие составные части САПР, поглощающие до 90% всех средств, выделяемых на разработку САПР в целом. Этим объясняется появление многочисленных приемов и способов организации разработки ПО, повышающих качество и сокращающих сроки создания программ. По современным представлениям программа считается таким же изделием, как и любой материальный продукт, и имеет утвержденные показатели качества. Один из важных показателей качества программы — ее технологичность, т. е. простота составления. Кроме того, программа должна удовлетворять требованиям надежности и правильности, универсальности, эффективности, информационной согласованности.

На качество специального ПО САПР оказывают влияние эффективность алгоритмов математического обеспечения, выбор языка программирования, организация информационного взаимодействия между программными модулями, конфигурация технических средств, структуризация данных, взаимодействие с операционной системой и др.

Основная задача ИО САПР — удовлетворение информационных потребностей проектировщика и отдельных компонентов САПР. Основу ИО САПР составляют банки данных — специальным образом организованные храни-

лища информации. Сведения, содержащиеся в банках данных, должны удовлетворять требованиям полноты и достоверности, а банки данных призваны обеспечить быстрый и удобный доступ к этим сведениям. В свою очередь, сведения, хранящиеся в банках данных, должны быть организованы в структуры того или иного вида. Правильным образом выполненная структуризация данных во многом определяет степень удовлетворения ИО упомянутым требованиям. Вместе с тем проблема информированности проектировщика в процессе принятия им проектных решений выдвигается в настоящее время на первый план.

Общение проектировщика с отдельными системами САПР происходит в рамках диалоговых систем, входящих в состав САПР. Диалоговые системы позволяют упростить работу проектировщика и обеспечивают оперативность обработки его запросов. Для этих целей в диалоговых системах используют развитые лингвистические средства вплоть до применения естественного языка. Диалоговые системы позволяют повысить эффективность расчетных работ благодаря оперативному вмешательству человека в процесс выполнения проектных процедур на ЭВМ. Организация удобного и быстрого диалога — залог эффективного применения САПР в практике проектирования.

Глава 1

ПРИНЦИПЫ ПОСТРОЕНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ СИСТЕМ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ

§ 1.1. Структура данных и управления

Программное обеспечение САПР, как и любое другое ПО, предназначено для обработки информации (данных). Данные, с которыми имеет дело любой компонент ПО (подпрограмма, программа, пакет программ), могут быть входные, выходные и промежуточные.

Входные и *выходные данные* всегда расположены вне тела данного программного компонента [в оперативной (ОП) или внешней (ВП) памяти].

Промежуточные данные могут располагаться либо в области ОП, занимаемой программной единицей, либо вне ее (в том числе и на внешних носителях).

Если какие-либо данные являются одновременно и входными, и выходными, то их называют *модифицируемыми данными* (рис. 1.1). В общем случае входные данные *1* подготавливаются пользователем или являются результатом работы раннее выполненных программных единиц. Выходные данные *2* могут предназначаться пользователю или (и) для последующей программной обработки.

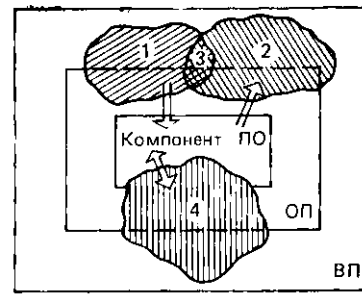


Рис. 1.1. Информационная среда компонента программного обеспечения:

1—4 — соответственно входные, выходные, модифицируемые, промежуточные данные; *ВП* — внешняя память

Простые типы данных. Структуры данных, обрабатываемых ПО САПР, чрезвычайно разнообразны. Поэтому эффективная

реализация методов и алгоритмов автоматизированного проектирования (АП) требует глубокого знания основных способов представления данных в ЭВМ.

■ Примечание. Под *структурой данных* понимают типы и взаимосвязи элементов.

Простыми типами данных, представленными в современных языках программирования, являются: 1) ЦЕЛОЕ, 2) ВЕЩЕСТВЕННОЕ, 3) БУЛЕВО (ЛОГИЧЕСКОЕ), 4) СТРОКА, 5) УКАЗАТЕЛЬ. В большинстве ЭВМ эти типы данных — «встроенные», т. е. имеются машинные команды, непосредственно их обрабатывающие.

Данные типа ЦЕЛОЕ и ВЕЩЕСТВЕННОЕ описывают десятичные числа (как положительные, так и отрицательные) и различаются способом представления в ЭВМ.

■ Примечание. В большинстве современных ЭВМ операции над данными типа ЦЕЛОЕ выполняются значительно быстрее, чем над данными типа ВЕЩЕСТВЕННОЕ. Особенно существенна разница в быстродействии для тех микро- и мини-ЭВМ, у которых отсутствуют машинные команды арифметики с плавающей точкой.

Диапазоны представления чисел данными этих типов ограничены; кроме того, конечная длина разрядной сетки ЭВМ обусловливает наличие погрешности отображения вещественных чисел. Эти обстоятельства необходимо всегда учитывать при программировании обработки данных вещественного типа с большим разбросом порядков.

■ Примечание. Неприятности, связанные с ограниченной длиной разрядной сетки ЭВМ, на практике устраняются либо представлением вещественных чисел в особых машинных форматах («удвоенной», «учетверенной» точности), либо специальным построением числовых алгоритмов (примером может служить алгоритм решения системы линейных алгебраических уравнений с выбором «ведущего элемента»).

Данные типа БУЛЕВО (ЛОГИЧЕСКОЕ) предназначены для описания двоичных (булевых) величин. Это могут быть либо одиночные биты (разряды) с возможными значениями «истина» или «ложь», либо цепочки битов. Данные этого типа обычно используются для управления ходом вычислительного процесса в программах и в качестве значений логических выражений флажков условий. В ПО структурного синтеза данными этого типа описываются многие свойства проектируемых объектов. Операции над данными типа БУЛЕВО (И, ИЛИ, НЕ и др.) — наиболее быстрые.

К данным типа СТРОКА относятся любые тексты, составленные из допустимого для данной ЭВМ набора литер. В этот набор обычно входят символы естественных языков, десятичные цифры и ряд специальных символов. Каждая литера данных типа СТРОКА представляется в ЭВМ последовательностью битов, определяемой принятым для данной ЭВМ стандартом. Литеры, составляющие текст, обычно располагаются в смежных участках памяти в прямом порядке. Данные типа СТРОКА могут иметь фиксированную или переменную длину (ограниченную некоторым пределом).

Данные типа УКАЗАТЕЛЬ (ССЫЛКА) предназначены для ссылки на элементы данных в тех случаях, когда невозможно использование имен элементов данных. Физически они представляют собой адрес указываемых данных или содержат информацию, позволяющую этот адрес вычислить. Основное назначение данных типа УКАЗАТЕЛЬ — предоставление программисту на алгоритмических языках высокого уровня средств для создания сложных структур данных.

Основные структуры данных. В структурах данных находят свое отражение отношения, связывающие между собой отдельные элементы данных, обрабатываемых каким-либо программным компонентом. Рассмотрим основные структуры данных (константа, переменная, массив, запись, таблица, фрейм, стек, очередь, список, дерево).

Элементарными (вырожденными) структурами данных являются константа и переменная.

Константа характеризуется фиксированными именем, типом и значением.

Переменная имеет фиксированные имя и тип, но переменное значение.

Массив — конечное множество переменных единого фиксированного типа, объединенных единым фиксированным именем. Ссылка на отдельные переменные массива осуществляется посредством *имени массива* и одного или нескольких *индексов*. Важное свойство массивов — возможность прямого доступа к его элементам, обеспечивающая высокую скорость обработки. Эта структура находит широкое применение, например, в ПО подсистем машинной графики и диалогового взаимодействия человека и ЭВМ.

■ Примечание. Одно- и двумерные массивы позволяют непосредственно отобразить в ЭВМ вектора и матрицы. Однако представление больших разреженных матриц в виде двумерных

массивов влечет за собой неоправданные потери памяти и машинного времени из-за необходимости хранения и обработки большого количества нулевых элементов. Поэтому для представления разреженных матриц в ЭВМ необходимо использовать иные структуры данных.

Требование обязательной однородности всех элементов массива (т. е. одинаковости типа всех переменных, составляющих массив) часто неудобно для представления данных, обрабатываемых ПО САПР.

Запись — структура данных, позволяющая группировать данные различных типов. Запись состоит из ряда поименованных *полей*, каждое поле определяется как переменная, массив или запись более низкого уровня иерархии, обладающая своими полями.

● Пример записи. Одной из основных конструкций входного языка ПО схемотехнического проектирования является описание элемента эквивалентной схемы. Такая конструкция может иметь следующий вид [1]:

С15 (6, 8) = 1. 5 МКФ,

С — имя элемента (емкость), 15 — его номер, 6 и 8 — номера узлов подключения, 1.5 — параметр элемента; символы МКФ указывают единицу измерения параметра.

Для внутреннего представления этой информации, содержащей данные различных типов, наиболее удобной структурой является запись. Для нашего примера используя нотацию, близкую к нотации языка ПЛ/1, запись можно определить следующим образом:

```
1 элемент — схемы
  2 идентификатор
    3 имя СТРОКА (3)
    3 номер ЦЕЛОЕ
  2 узлы
    3 нач—узел ЦЕЛОЕ
    3 кон—узел ЦЕЛОЕ
  2 параметр
    3 значение ВЕЩЕСТВЕННОЕ
    3 размерность СТРОКА (3)
```

Доступ к информации, содержащейся в записи, осуществляется с помощью составных имен. Так, составное имя «элемент — схемы. идентификатор» позволяет ссылаться на запись, состоящую из полей имени и номера элемента. Имя «элемент — схемы. параметр. значение» идентифицирует переменную типа ВЕЩЕСТВЕННОЕ. На рис. 1.2 показано размещение записи в памяти ЭВМ (она занимает там сплошной участок).

Таблица — объединение структур данных типа запись. Таблица аналогична двумерному массиву, но ее столбцы могут иметь различные типы.

Фрейм — структура данных для представления знаний в конкретных предметных областях. Подобно запи-

си, фрейм состоит из отдельных полей (ячеек), заполненных содержательными понятиями предметной области. Поля фрейма связаны между собой отношениями, реализованными обычно в виде отдельных процедур.

● **Пример фрейма.** Для представления знаний о витой пружине в САПР машиностроения может использоваться фрейм ПРУЖИНА. Поля этого фрейма — диаметр и шаг намотки пружины, диаметр проволоки, количество витков, свойства материала проволоки и др. Отношениями в этом фрейме будут уравнения, составляющие математическую модель пружины. Фреймы ПРУЖИНА, ШТОК, РЫЧАГ и др., объединенные в сеть, составляют модель предметной области САПР машиностроения.

Рис. 1.2. Представление записи в оперативной памяти

В теории программирования фрейм называют *абстрактным типом данных*.

Все рассмотренные выше структуры данных характеризуются сплошным расположением в памяти ЭВМ. Это часто неудобно из-за необходимости заранее фиксировать размер области оперативной памяти, отводимой под размещение этих структур. В большинстве случаев этот размер априорно неизвестен и определяется только в процессе выполнения программ. Поэтому более универсальны структуры данных, ориентированные на цепное представление в памяти ЭВМ. К ним относятся стек, очередь, линейный список, дерево и др. Объединение записей в эти структуры осуществляется с помощью переменных типа УКАЗАТЕЛЬ, размещаемых в полях записей.

Стек характеризуется последовательной организацией и возможностью доступа только с одного края цепочки записей, называемого *вершиной стека*. Возможная физическая реализация стека в памяти ЭВМ показана на рис. 1.3, *а*. В нижнем поле каждой записи располагается указатель на следующую запись стека, указатель

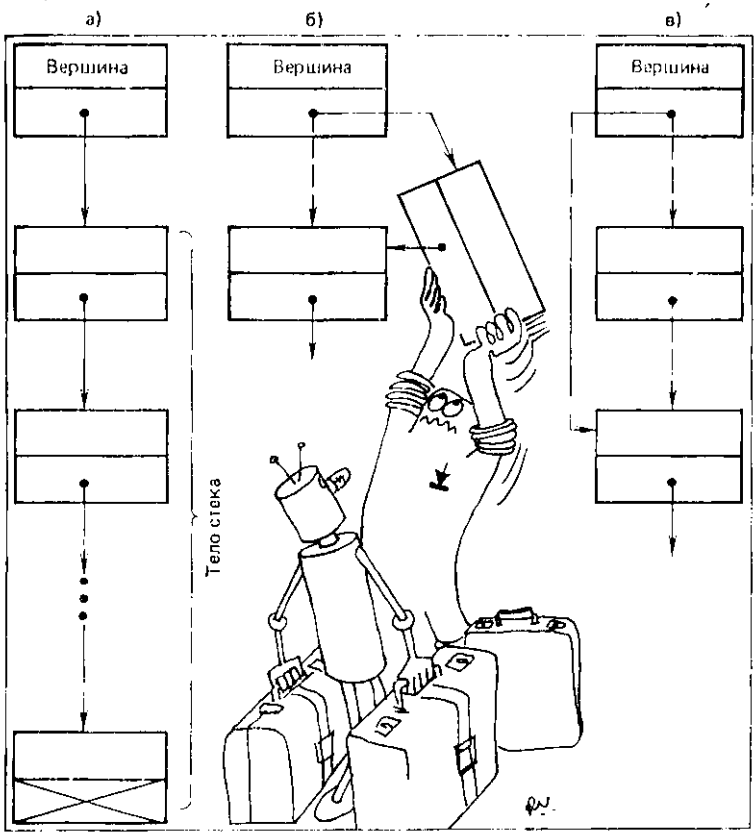


Рис. 1.3.

в последней записи пуст. Основными операциями над стеком являются ЧИТАТЬ ВЕРШИНУ, ДОБАВИТЬ (рис. 1.3, *б*) и УДАЛИТЬ (рис. 1.3, *в*). В стеке реализуется принцип обработки «последним пришел — первым ушел». Наиболее широко в ПО САПР стековая организация данных используется при трансляции входных языков и при управлении ОП.

Очередь — линейная последовательность записей, связанных указателями, но доступ к ее записям осуществляется с начала и конца (рис. 1.4, *а*). Добавлять записи можно только в конец очереди (рис. 1,4, *б*), а читать и удалять записи — только с начала очереди (удаляют самую старую запись) (рис. 1.4, *в*). Обработка данных

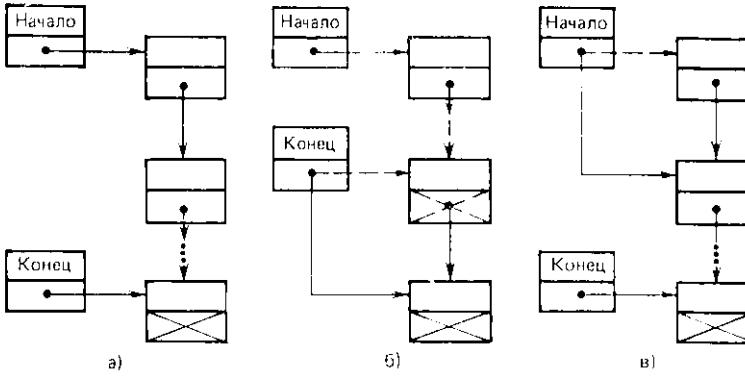


Рис. 1.4. Представление очереди (*а*) и операции над очередью ДОБАВИТЬ (*б*) и УДАЛИТЬ (*в*)

в очереди строится по принципу «первым пришел — первым ушел». Эта структура часто используется для обмена данными между программными компонентами процессоров входных языков САПР. Объединение отдельных записей в очередь — это один из самых распространенных способов организации данных во внешней памяти ЭВМ.

Линейный список — наиболее универсальная структура данных, в нем доступна для чтения и удаления любая запись, более того, новая запись может быть включена между двумя любыми соседними записями списка. На рис. 1.5, *а* показана физическая реализация двунаправленного линейного списка. Встречное направление указателей позволяет осуществить в таком списке поиск записей с обеих сторон.

Во многих алгоритмах САПР требуется упорядочение записей по какому-либо параметру. Линейный список дает возможность реализовывать алгоритмы сортировки (упорядочения) без физического перемещения записей в ОП только путем соответствующей корректировки указателей. В этой структуре легко осуществляются операции удаления и включения новых записей без нарушения упорядоченности списка (рис. 1.5, *б, в*). В отдельных приложениях для повышения скорости обработки необходимо упорядочение записей более чем по одному параметру (в этом случае возможно, не перемещая и не дублируя записи, организовать еще несколько списков, добавляя в записи новые поля с соответствующими указателями).

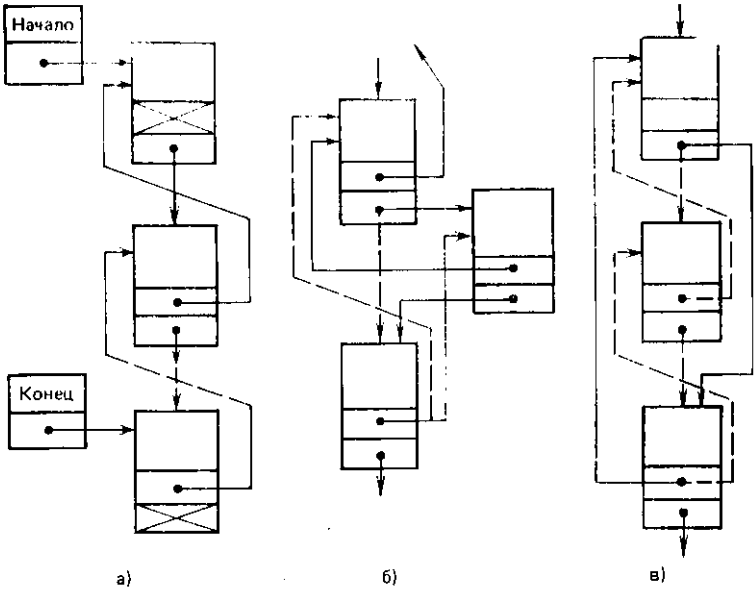
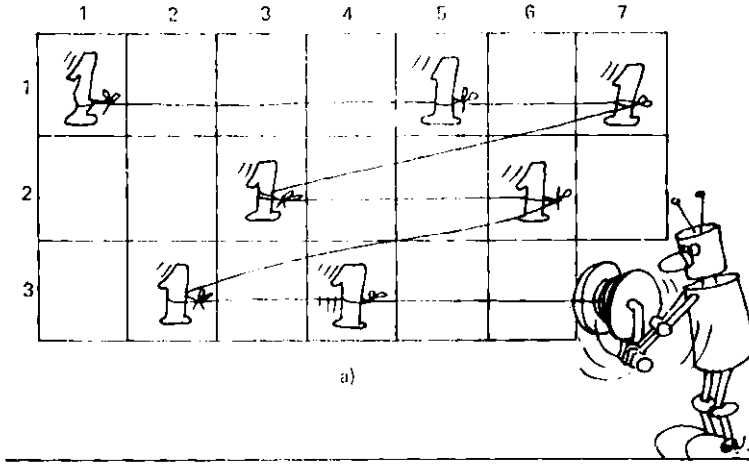


Рис. 1.5. Представление линейного списка (*а*) и операции над линейным списком ДОБАВИТЬ (*б*) и УДАЛИТЬ (*в*)

● **Пример использования линейного списка.** Ключевую роль при событийном моделировании в подсистемах проектирования цифровых устройств выполняет список будущих событий (СБС), содержащий информацию о последовательности переключений логических элементов. Член СБС может содержать разнотипную информацию [момент модельного времени наступления события (действительное число), новый уровень сигнала (логическая переменная), имя логического элемента (строка символов), номер выхода элемента, на котором должно произойти событие (целое число), и т. д.]. Поэтому для представления каждого члена СБС целесообразно использовать структуру данных типа «запись», содержащую поля различных типов, сам же СБС удобно организовать в виде двунаправленного линейного списка, записи которого упорядочены по параметру «модельное время». Такой способ позволяет простыми средствами реализовывать включение в СБС новых членов с произвольным временем наступления события. Член СБС с минимальным модельным временем всегда будет начальным элементом списка; следовательно, для реализации содержащегося в нем события поиск в списке не требуется.

Универсальность линейного списка заключается также и в том, что его элементы, в свою очередь, могут быть линейными списками (в предельном случае единичной длины); такая структура называется просто *списком*.



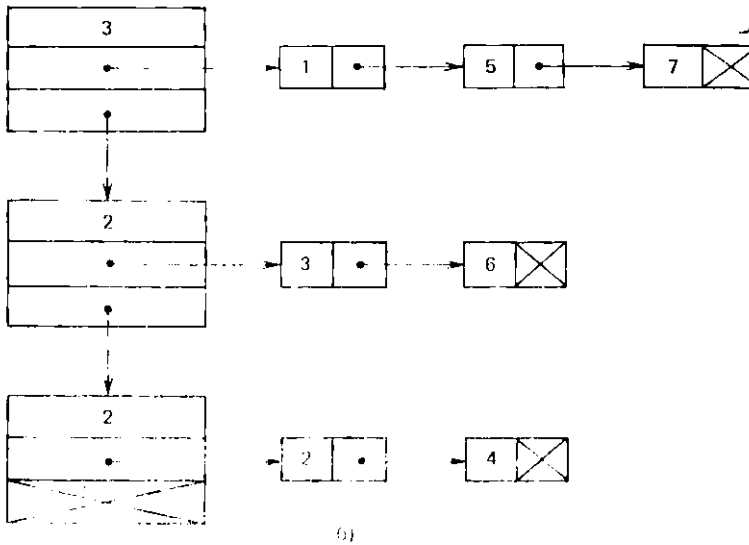


Рис. 1.6.

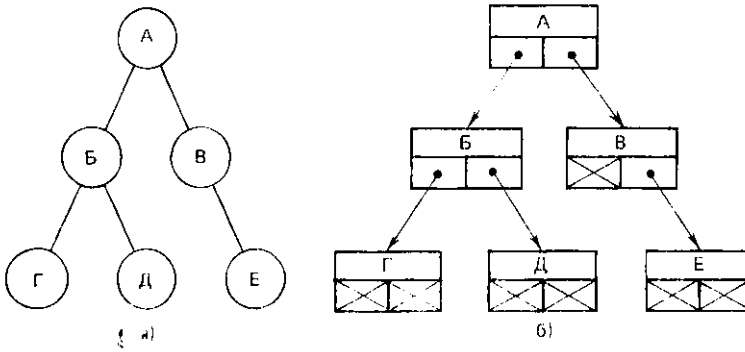


Рис. 1.7. Пример двоичного дерева (*а*) и его физическая реализация (*б*)

■ Примечание. Список позволяет, например, экономно представлять разреженные матрицы, как это показано на рис. 1,6, *а*. Записи линейного списка, изображенного на рис. 1.6, *б* вертикально, содержат в одном из своих полей количество ненулевых элементов в строке матрицы; эта информация может быть использована для перестановки строк матрицы, что требуется в ряде алгоритмов САПР. Записи линейных списков, изображенных горизонтально, содержат номера столбцов ненулевых элементов строк матрицы.

В задачах структурного синтеза, относящихся к задачам искусственного интеллекта, широко используется представление информации в виде дерева, например в методе И-ИЛИ-дерева. Для организации данных в виде дерева может быть применена списковая структура. Однако чаще для этой цели используют двоичные деревья, к которым могут быть сведены общие деревья. *Двоичное дерево* — дерево, у каждой вершины которого не более двух поддеревьев. Один из способов физической реализации двоичных деревьев представлен на рис. 1.7.

Комбинация рассмотренных базовых структур данных позволяет организовывать новые структуры, отражающие сложные отношения между единицами информации, обрабатываемой ПО САПР. Большинство современных языков программирования высокого уровня имеют развитые средства для создания сложных структур данных. Исключение составляет язык ФОРТРАН, среди типов данных которого отсутствует СТРОКА, а единственная «встроенная» структура данных — массив. Поэтому организация более сложных структур при программировании на этом языке является заботой разработчика ПО,

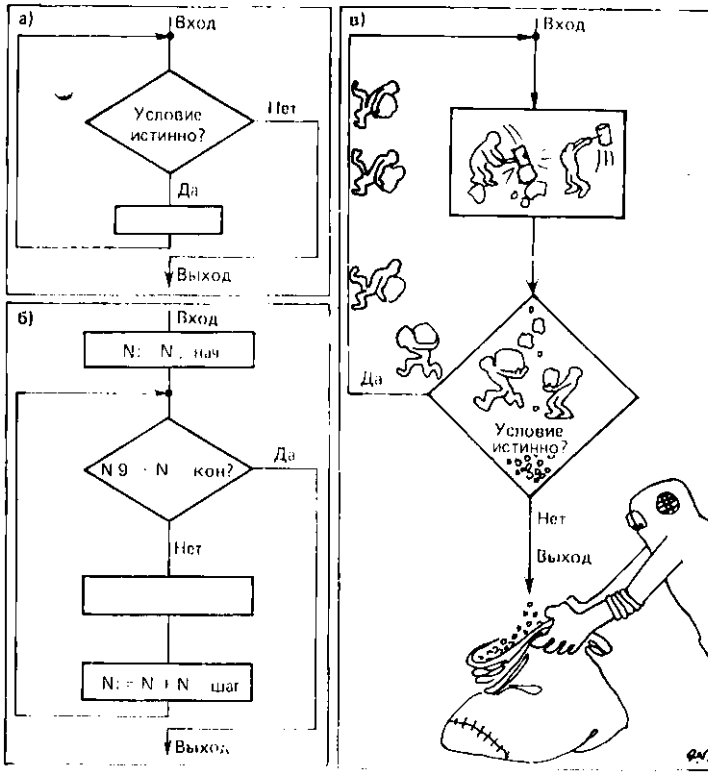


Рис. 1.8.

Структуры управления вычислениями. В ПО реализуются алгоритмы обработки информации. В САПР эти алгоритмы обычно являются весьма сложными и характеризуются итерационностью, многоуровневой вложенностью процедур, множеством точек выбора альтернативных решений. Однако для программной реализации любых алгоритмов достаточно трех базовых *структур управления:* следование, цикл и ветвление.

Следование — структура из нескольких последовательно выполняемых операторов, причем этими операторами в общем случае могут быть операторы вызова подпрограмм.

Цикл — структура, назначение которой — представление многократно повторяющихся вычислительных процессов. Различают циклы с предусловием (рис. 1.8, *а*), с постусловием (рис. 1.8, *в*) и с параметром (рис. 1.8, *б*).

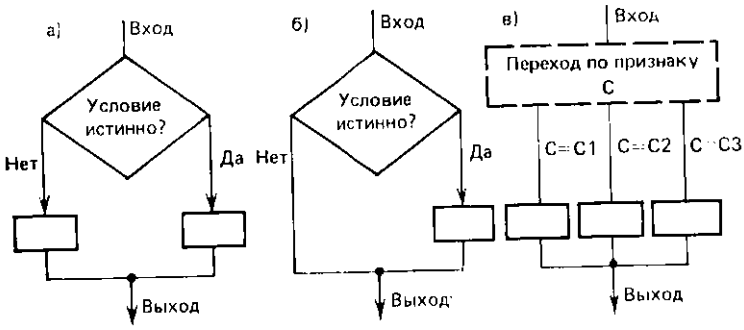


Рис. 1.9. Графическое изображение ветвлений:
а — ЕСЛИ-ТО-ИНАЧЕ; *б* — ЕСЛИ-ТО; *в* — многозначно

Применение последней структуры предпочтительнее в случаях, когда число повторений известно заранее. Циклы с постусловием и параметром могут быть приведены к циклу с предусловием.

Ветвление — структура, предназначенная для принятия решений в ходе вычислительного процесса. Простейшими ветвлениями являются *альтернативные*: ЕСЛИ-ТО-ИНАЧЕ (рис. 1.9, *а*) и ЕСЛИ-ТО (рис. 1.9, *б*). В некоторых алгоритмах возникает задача выбора не из двух, а из нескольких возможностей, в этом случае удобна структура *многозначное ветвление* (рис. 1.9, *в*). Структура ЕСЛИ-ТО-ИНАЧЕ фундаментальна, через нее могут быть представлены две другие структуры ветвления.

Важное свойство всех структур — наличие только одного входа и только одного выхода, как у простого оператора. Поэтому каждый прямоугольный блок на рис. 1.8—1.9, обозначающий какое-либо действие, может быть заменен любой из трех базовых структур. Возможность представления любых алгоритмов с помощью вложенных структур следования, цикла и ветвления составляет основу *метода структурного программирования* (см. § 1.3).

Управление ходом вычислительного процесса с помощью данных. Хотя рассмотренные выше структуры управления универсальны, в некоторых практических приложениях более удобным оказывается передать часть функций по управлению ходом вычислительного процес-

Таблица 1.1

Условие 1	0	0	1	1
Условие 2	0	1	0	1
Действие А	*	*	*	
Действие Б		*		*
Действие В	*	*		*

са данным. Рассмотрим два метода, использующих данные для управления: 1) метод таблиц решений; 2) метод конечного автомата.

Метод таблиц решений целесообразно применять в алгоритмах, характеризующихся большим количеством условий и ограниченным набором действий, выполняемых в различных сочетаниях в зависимости от условий. Табл. 1.1 — таблица решений для простейшего примера, содержащего 2 условия и 3 действия. Символ «1» в клетках таблицы для условий соответствует ситуации, когда значение данного условия истинно. Символ «*» в клетках таблицы для действий означает необходимость выполнения соответствующего действия. Таким образом, обработка таблицы при каждом вхождении заключается в нахождении столбца, соответствующего текущему сочетанию условий, и в выполнении действий, отмеченных символом «*» в этом столбце. Достоинство этого метода управления — легкая модифицируемость ПО под новые условия применения, поскольку реорганизация таблицы не требует изменений в процедурной части программного компонента.

Метод конечного автомата находит широкое применение в языковых процессорах для распознавания цепочек символов [2]. Поясним идею метода на конкретном примере. Пусть из всего множества слов конечной длины, составленных из символов алфавита {А, К, О, П, Р, С, Т, ▷}, допустимыми являются только СТОП▷ и СТРОКА▷, где ▷ — символ конца слова. В задачу программы распознавателя, использующей метод конечного автомата, входит обнаружение из всего множества цепочек символов только двух допустимых. В основе реализации конечного автомата на ЭВМ лежит таблица пе-

Таблица 1.2

	А	К	О	П	Р	С	Т	▷	
0						1		12	
1							2	12	С
2			3		4			12	СТ
3				5				12	СТО
4			6					12	СТР
5								10	СТОП
6		7						12	СТРО
7	8							12	СТРОК
8								11	СТРОКА
9								12	
10	Опознана цепочка СТОП ▷								
11	Опознана цепочка СТРОКА ▷								
12	Цепочка недопустима								

реходов, представленная в табл. 1.2. Ее столбцы помечены символами входного алфавита, строки — номерами состояний. Элементами таблицы являются номера новых состояний, в которые должен переходить автомат из текущего состояния при поступлении входных символов. В таблице переходов примера все пустые клетки должны быть заполнены номером 9 (для наглядности он опущен). Процедурная часть программы распознавателя должна обеспечивать поиск столбца, соответствующего очередному входному символу, и «передвигать» указатель на строку таблицы, отвечающую новому состоянию.

Работа автомата начинается с некоторого исходного состояния (помеченного номером 0). Появление на входе автомата символа С переводит его в состояние 1, любой другой символ (за исключением ▷) вызовет переход автомата в состояние 9, откуда есть выход только по символу ▷ конца строки. Таким образом, любая цепочка символов, не начинающаяся с символа С, будет распоз-

навателем отвергнута. Из состояния 1 допускающим будет только переход в состояние 2 для символа Т, из состояния 2 допускающих переходов два — для символов О и Р. Работа распознавателя завершается обработкой символа !. В табл. 1.2 для состояний автомата 1—8 справа от таблицы записаны подцепочки символов, приводящие в каждое из этих состояний.

Поскольку процедурная часть рассмотренного распознавателя может быть легко реализована как инвариантная по отношению к размерам таблицы переходов, настройка такого распознавателя на новые входной алфавит и множество допустимых цепочек символов осуществляется модификацией только таблицы переходов, т. е. содержимого некоторой структуры данных. Таблица переходов является сильно разреженной матрицей, поэтому в целях экономии ОП для ее представления можно использовать обсужденный выше способ (см. рис. 1.6).

■ Примечание. Таблица переходов может быть сгенерирована автоматически по заданным алфавиту и допустимым цепочкам.

Среди структур управления, не относящихся к базовым, важнейшей является подпрограмма.

Подпрограмма — часть программы, обладающая именем, которое позволяет этой программе (или всякой другой) вызывать ее, чтобы заставить выполнить некоторое действие. В виде подпрограмм целесообразно программировать действия, общие для ряда программ, и универсальные алгоритмы. Подпрограммы позволяют управлять сложностью программ, допуская сжато *именовать* сложные последовательности действий. На этом свойстве подпрограмм базируется методология *нисходящего проектирования программ* (см. § 1.3). Подпрограмма, как и любая другая структура управления, должна иметь один вход и один выход, причем возврат управления из подпрограммы обязательно должен осуществляться в точку ее вызова.

Различают подпрограммы, определенные в теле данной программы и транслируемые вместе с нею, и подпрограммы, определенные и транслируемые раздельно. Подпрограммы первого типа предназначены для обслуживания только данной конкретной программы и другим программам недоступны. Таким подпрограммам обычно предоставлено право манипулировать со всеми данными, имеющимися в основной программе. Программы второго типа могут быть предназначены для коллективного ис-

пользования. В дальнейшем речь будет идти в основном о подпрограммах этого типа.

Использование подпрограмм ставит проблему обмена информации между вызывающей и вызываемой подпрограммами. Вызов подпрограммы обычно сопровождается передачей ей фактических параметров, располагаемых в ОП, среди которых различают *аргументы, результаты* и *модифицируемые параметры* (см. рис. 1.1). Отметим, что это не единственный способ обмена данными между программой и подпрограммой (см. гл. 3). Наиболее широко используются два механизма связывания фактических и формальных параметров подпрограмм: 1) *по адресу;* 2) *по значению.*

При использовании первого механизма в подпрограмму передается адрес данных, являющихся фактическим параметром и принадлежащих вызывающей подпрограмме, т. е. вызываемая программа имеет непосредственный доступ к фактическому параметру. Недостаток этого механизма при передаче параметров-аргументов — отсутствие защиты аргумента от модификаций в вызываемой подпрограмме. Передача параметра-аргумента с помощью второго механизма предполагает создание копии передаваемых данных и передачу этой копии в вызываемую подпрограмму по адресу. Аналогично работает механизм возврата параметров-результатов из подпрограммы. Передача параметров по значению обеспечивает защиту данных от несанкционированного доступа со стороны вызываемой подпрограммы, но связана с увеличением затрат машинного времени и оперативной памяти.

Большинство современных языков программирования высокого уровня допускает оба метода передачи параметров. В качестве фактических параметров могут выступать и имена подпрограмм, передача подпрограмм всегда осуществляется передачей адреса точки входа в эти подпрограммы.

Имеется немало ситуаций, когда обмен информацией между подпрограммами через передачу параметров неудобен и неэффективен. В этом случае возможно использование *глобальных структур данных.* Доступ к таким структурам данных может быть осуществлен из любого программного компонента, если только он отредактирован совместно с компонентом, физически содержащим эту структуру. Последнее показывает, что этот способ информационного обмена, несмотря на свое название,

менее общий по сравнению со способом передачи параметрами.

Все языки программирования в той или иной степени предоставляют возможности обобществления данных, однако использовать эти возможности следует весьма осторожно, особенно при программировании параллельных процессов.

■ Примечание. Передача параметров по адресу представляет собой обобществление данных между двумя (а возможно, и более) программными компонентами.

Важными характеристиками подпрограмм (и любых компонентов ПО) являются реентерабельность и повторноиспользуемость. *Реентерабельная подпрограмма* — это такая подпрограмма, к которой возможно повторное обращение до того, как она полностью окончила работу по предыдущему вхождению в нее. Одна копия реентерабельной подпрограммы может обслуживать одновременно несколько разных вызывающих подпрограмм.

Повторноиспользуемая подпрограмма также обеспечивает возможность многократного обращения, но только в том случае, если новый вызов следует после полного завершения ее работы по предыдущему вызову. Повторноиспользуемая подпрограмма — подпрограмма, которая не сохраняет историю своих вызовов. Это условие может быть легко соблюдено при разработке компонентов ПО на любом языке программирования. В ПО САПР все программные компоненты должны быть повторноиспользуемыми.

Создание реентерабельных программ часто требует от их разработчика значительных дополнительных усилий. Поэтому такие программы в ПО САПР находят ограниченное применение — главным образом в подсистемах, допускающих одновременную работу нескольких пользователей.

К понятию реентерабельности подпрограмм близко (но не тождественно) понятие рекурсивности. *Рекурсивная подпрограмма* — подпрограмма, которая вызывает сама себя (либо непосредственно, либо через цепочку модулей). Многие алгоритмы автоматизированного проектирования в области структурного синтеза и параметрической оптимизации по сути рекурсивные. Самым простым примером здесь может служить метод половинного деления, используемый для одномерного поиска экстремума функций. Однако не все алгоритмические языки позволяют писать непосредственно рекурсивные под-

программы, так в языке ФОРТРАН программирование рекурсивных алгоритмов требует использования специальных приемов, усложняющих программу.

§ 1.2. Архитектура программного обеспечения САПР

Основные компоненты ПО САПР. Варианты организации ПО САПР разнообразны и зависят от многих факторов, главными из которых являются:

1) предметная область, аспекты и уровни создаваемых с помощью ПО описаний проектируемых объектов;

2) степень автоматизации отдельных проектных операций и процедур;

3) архитектура и состав технических средств, режим функционирования;

4) ресурсы, отпущенные на разработку ПО.

В качестве основного рассмотрим вариант организации ПО одноуровневой САПР, поясняемый рис. 1.10. Программное обеспечение САПР делится на составные части, которые относятся к проектирующим и обслуживающим подсистемам САПР и в дальнейшем именуются подсистемами ПО.

К обслуживающим подсистемам ПО относятся: диалоговая *ДП*, управления базами данных *СУБД*, инструментальная *ИП*, а также монитор, обеспечивающий взаимодействие всех остальных подсистем и управление их выполнением.

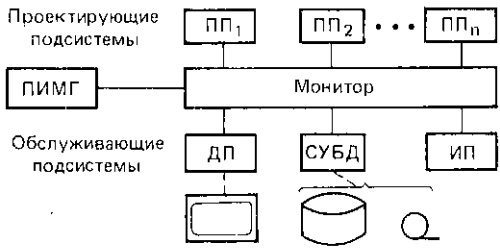
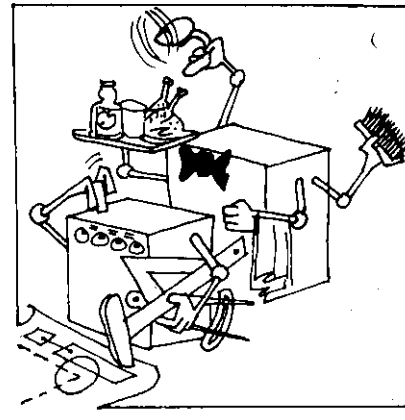


Рис. 1.10. Архитектура специального программного обеспечения САПР:

$ПП_i$ — *i*-я проектирующая подсистема; *ДП* — диалоговая подсистема; *ИП* — инструментальная подсистема; *СУБД* — система управления базами данных; *ПИМГ* — подсистема интерактивной машинной графики



Диалоговая подсистема ПО организует интерактивное взаимодействие пользователя САПР с управляющей и проектирующими подсистемами ПО, подготовку и редактирование исходных данных, просмотр результатов работы проектирующих подсистем, функционирующих в пакетном режиме.

Подсистема управления базами данных СУБД реализует единообразный доступ к общей базе данных (БД) САПР и к индивидуальным БД пользователей. Назначение БД следующее: 1) хранение сведений нормативно-справочного характера (о нормалях, ГОСТах, унифицированных изделиях, типовых проектных решениях, ранее выполненных разработках и т. д.); 2) хранение результатов выполненных этапов текущего проекта; 3) обеспечение информационной согласованности различных подсистем САПР.

■ Примечание. Эти аспекты использования рассматриваются в гл. 2 и 3.

Инструментальная подсистема программирования, основу которой составляет генератор прикладных программ, синтезирующий новые программы из унифицированных модулей и подпрограмм, разработанных пользователем, необходима для достижения открытости ПО САПР. Генератор прикладных программ включает в себя также средства автоматической разработки трансляторов для входных языков проектирующих подсистем САПР.

■ Примечание. Использование генераторов прикладных программ как одного из мощных средств разработки ПО рассматривается в § 1.3.

Проектирующие подсистемы ПО могут быть объектно-зависимыми (проблемно-ориентированными) или объектно-независимыми (методоориентированными, инвариантными). *Объектно-независимые подсистемы* ПО ориентированы на решение задач проектирования при

наличии их предварительно выполненной математической постановки (например, подсистемы параметрической оптимизации, решения систем уравнений в частных производных и систем обыкновенных дифференциальных уравнений и др.). Использование объектно-независимых подсистем ПО менее эффективно, поскольку в них не учитывается специфика задач конкретной предметной области и требуется достаточно высокая математическая подготовка пользователя. Такие подсистемы предназначены для решения задач, для которых в составе САПР отсутствуют соответствующие проблемно-ориентированные подсистемы. Кроме того, они составляют основу для генерации проблемно-ориентированных подсистем ПО.

Проектирующими подсистемами ПО могут быть *простые программы,* ориентированные на узкий класс объектов и использующие простые аналитические модели. Но чаще проектирующие подсистемы ПО представляют собой универсальные пакеты прикладных программ сложной структуры, обладающие своими мониторами, локальными базами данных и средствами их управления, поэтому ниже наряду с термином «проектирующая подсистема ПО» будем использовать и другой термин — «проектирующий пакет ПО». Некоторые из таких пакетов могут реализовывать не только отдельные проектные операции и процедуры, но и законченные их маршруты, а также допускать множественный доступ (т. е. работу одновременно с несколькими пользователями), в последнем случае они должны иметь свои локальные средства поддержки диалогового взаимодействия.

■ Примечание. В гл. 5 рассматриваются принципы построения проектирующих подсистем на примере программного комплекса автоматизации схемотехнического проектирования.

Подсистема интерактивной машинной графики *ПИМГ* (рис. 1.10) занимает промежуточное положение между проектирующими и обслуживающими подсистемами ПО. С одной стороны, средства машинной графики обслуживают ряд проектирующих подсистем (обычно это пакеты функционального проектирования), где они используются в основном для наглядного представления исходной и выходной информации (в виде схем, временны́х диаграмм, гистограмм и т. д.). С другой стороны, во многие подсистемы конструкторского проектирования ПО интерактивной машинной графики входит как основная часть. Поэтому в САПР возможно наличие нескольких пакетов машинной графики (базового в ка-

честве обслуживающего и одного или более в составе проектирующих подсистем конструирования).

Характерная черта рассмотренной архитектуры ПО САПР — строгая разграниченность проектирующих и обслуживающих подсистем. Такой подход к построению ПО обеспечивает, во-первых, его легкую расширяемость и модифицируемость и, во-вторых, переносимость на иные технические средства, поскольку все машинно-зависимые программные компоненты локализованы в обслуживающих подсистемах. Недостатком сосредоточения обслуживающих функций в отдельных подсистемах является сложность в организации к ним множественного доступа.

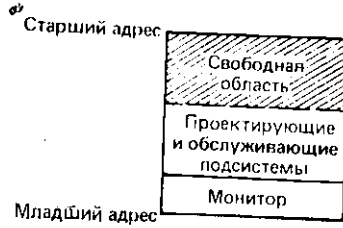


Рис. 1.11. Схема размещения специального программного обеспечения САПР в оперативной памяти

Программное обеспечение САПР ориентировано на раздельное редактирование всех его подсистем и их динамическую загрузку в ОП по мере надобности. На рис. 1.11 показано распределение доступной зоны ОП при функционировании ПО такой структуры. Монитор и диалоговая подсистема ПО резидентны, т. е. постоянно находятся в ОП. В смежную с ними область динамически загружаются обслуживающие и проектирующие подсистемы ПО, при этом обслуживающие подсистемы занимают участки памяти с меньшими адресами. Оставшаяся не занятой область памяти может быть использована для размещения данных. Динамическая структура ПО по сравнению с оверлейной структурой, требующей совместного редактирования всех подсистем ПО, характеризуется легкостью расширения и модификации, а также значительной экономией ОП. Однако для динамической структуры необходимы дополнительные затраты на организацию взаимодействия проектирующих и обслуживающих подсистем.

Монитор САПР. Управление ходом вычислительного процесса и координация взаимодействия подсистем САПР осуществляются монитором. Те же задачи, но в рамках отдельных пакетов, решаются мониторами этих пакетов. В функции мониторов входят:

1) прием и интерпретация обращенных к ним команд пользователя;

2) загрузка и активизация компонентов ПО, организация маршрутов их выполнения;

3) установление взаимодействия между подсистемами;

4) динамическое распределение памяти;

5) обработка прерываний от дисплея пользователя;

6) сервисные функции (регистрация пользователей, сбор статистики, ведение службы времени, обработка сбоев и т. д.).

Язык управления монитором САПР достаточно прост, в его основе лежат команды вызова необходимых проектирующих подсистем ПО и задания им управляющих параметров, а также команды, описывающие способ информационного обмена между подсистемами — через оперативную или внешнюю память, посредством подсистемы управления базой данных. Средства этого языка должны позволять создавать макрокоманды, определяющие маршруты выполнения проектирующих подсистем ПО. Языки управления проектирующих пакетов значительно сложнее, поскольку должны отражать все возможные постановки задач проектирования в конкретных предметных областях, решение которых допускают пакеты. Обычно эти языки имеют процедурный характер (см. § 5.3).

В общем случае загруженные проектирующие подсистемы ПО могут функционировать либо как обычные подпрограммы, подчиненные управляющей подсистеме ПО, либо как «параллельно» выполняемые подзадачи, способные соревноваться между собой и монитором за управление. Функционирование нескольких пакетов одновременно в качестве подзадач оправдано только в случаях, когда каждый из них в отдельности не способен загрузить процессор ЭВМ и «распараллеливание» не сказывается на эффективности и удобстве работы каждого из пользователей. Очевидно, что при этом каждая из проектирующих подсистем ПО должна иметь свою локальную подсистему диалогового взаимодействия. Создание подзадач — один из способов обеспечения множественного доступа пользователей к САПР, однако его реализация значительно усложняет управляющую подсистему: во-первых, возникает задача динамического распределения ресурсов ЭВМ; во-вторых, появляется потребность в механизме, разрешающем каким-либо образом конфликты в работе подзадач. Такие конфликты могут возникнуть, например, при одновременном обращении

нескольких проектирующих пакетов к подсистеме управления базой данных. Конфликты могут быть устранены использованием очередей запросов к СУБД, в которых запросы на обслуживание подсистем ПО базой данных располагаются в порядке поступления и приоритетности.

Для обеспечения доступа к одной проектирующей подсистеме ПО нескольким пользователям может быть использован более экономичный в смысле затрат ОП способ, основанный на реализации основных программных компонентов пакета реентерабельными.

■ Примечание. Этот способ еще более сложен, чем предыдущий, но его применение в подсистемах, ориентированных на интенсивный диалог с пользователем, дает значительный эффект (если, конечно, речь идет об ЭВМ с небольшим объемом ОП).

Взаимодействие подсистем. Взаимодействие управляющей подсистемы ПО и мониторов проектирующих пакетов осуществляется через стандартный интерфейс, представляющий собой формальные правила передачи фактических параметров. В проектирующие подсистемы ПО передаются:

параметры, задающие режим функционирования;

адреса точек входа в обслуживающие подсистемы ПО;

адреса динамически распределенных областей памяти, предназначенных для информационного обмена между различными подсистемами ПО.

Каждый проектирующий пакет, входящий в состав САПР, имеет паспорт, хранящийся в базе данных САПР. Паспорт содержит следующие сведения о проектирующем пакете: 1) размер занимаемой области ОП; 2) имена требуемых обслуживающих подсистем ПО; 3) имена режимных параметров и их значения по умолчанию; 4) имя языка программирования, в стандарте которого пакет использует представление структур данных; 5) местонахождение в пакете обработчика прерываний от дисплея пользователя (если он предусмотрен); 6) указатели на возможные способы обмена информацией с другими проектирующими подсистемами ПО (через ОП, базу данных или файловую систему ЭВМ) и т. д.

Монитор САПР, получив команду на активизацию какой-либо проектирующей подсистемы ПО, считывает из базы данных ее паспорт, проверяет корректность команды и возможность загрузки подсистемы. Далее он помещает в ОП необходимые обслуживающие подсистемы ПО (если их там еще нет), вслед за ними — требуе-

мую проектирующую подсистему, а затем в строгом соответствии с данными из паспорта строится обращение к этой подсистеме. После окончания работы подсистемы она удаляется из ОП.

Некоторые проектирующие подсистемы ПО для решения задач высокой размерности требуют больших затрат машинного времени и ОП, например задачи анализа сложных динамических объектов, их параметрическая оптимизация, синтез тестов для цифровых устройств, трассировка печатных плат и т. д. Использование интерактивного режима на этапе «счета» таких задач нецелесообразно, но он необходим на подготовительных стадиях и при интерпретации результатов. Для таких случаев в составе ПО САПР необходимо иметь обслуживающую подсистему образования *фоновых заданий.* Если САПР функционирует на вычислительной установке, имеющей связь с другими ЭВМ, то такая подсистема должна обеспечивать возможность передачи фоновых заданий на одну из этих ЭВМ. После завершения фонового задания его результаты могут быть просмотрены и обработаны пользователем средствами проектирующей подсистемы ПО, породившей это задание.

Важной функцией управляющей подсистемы САПР и мониторов проектирующих пакетов является динамическое распределение ОП, необходимое всегда, когда пакет предназначен для работы с данными переменного объема.

Монитор САПР выделяет ОП для обеспечения информационного обмена между подсистемами; мониторы пакетов делают это по запросам модулей пакета, если средства языка, применявшегося для их программирования, недостаточны для эффективного использования ОП.

Средства динамического распределения памяти — обязательные компоненты всех современных операционных систем (ОС) и имеются во многих языках программирования (за исключением языков ФОРТРАН и КОБОЛ).

Для эффективной работы коллектива пользователей необходим множественный доступ к САПР. Выше были рассмотрены два способа организации одновременной работы нескольких пользователей, однако для обоих способов характерно то, что активизация заданий пользователей происходит в хаотическом порядке и на неопределенное время.

■ Примечание. Хаотический порядок активизации заданий сказывается на скорости реакции проектирующих пакетов на команды пользователей и, следовательно, на удобстве работы.

Эту проблему решает режим разделения времени, но его реализация средствами прикладного ПО САПР представляет собой сложную задачу. Более целесообразна реализация режима разделения времени с помощью общего программного обеспечения — соответствующих ОС ЭВМ.

Рассмотренный вариант архитектуры ПО САПР сравнительно прост, он пригоден для создания САПР средних размеров. Крупные промышленные САПР, функционирующие на сетях ЭВМ, имеют сложные, распределенные по ЭВМ мониторы, специальные обслуживающие подсистемы информационного обмена, управления технологическим оборудованием, планирования и управления ходом проекта. Такие САПР интегрированы с автоматизированными системами научных исследований, технологической подготовки производства, испытаний и с гибкими автоматизированными производствами. Их ПО отражает специфику конкретных предметных областей, принятые в них маршруты проектирования и структуру имеющихся на предприятии технических средств.

§ 1.3. Методы разработки программного обеспечения

Программное обеспечение САПР представляет собой сложную программную систему, включающую в себя десятки и сотни компонентов (см. § 1.2). Создание ПО САПР — трудная научно-техническая задача, для решения которой требуются большие материальные затраты. Так, известны САПР, ПО которых насчитывает до 500 тыс. операторов языка программирования. Разработка такого ПО требует сотен и тысяч человеко-лет [3]. Затраты на разработку и сопровождение ПО составляют подавляющую долю всех затрат на создание и эксплуатацию САПР.

Высокая стоимость ПО объясняется низкой скоростью роста производительности труда программистов по сравнению с темпами роста производительности труда в других сферах производства. Средняя производительность труда программистов в организациях, занимающихся промышленной разработкой ПО, составляет 1000—2000 операторов/год.

В США цена одного оператора программы колеблется в зависимости от степени сложности ПО от 15 до 700 долл.; по данным на 1985 г. 1 ч работы программиста стоит в 5 раз дороже 1 ч работы ЭВМ быстродействием 300 тыс. операций/с. Приведенные данные касаются ПО, представляющего собой *законченный программный продукт,* поставляемый как промышленное изделие. В отличие от программ индивидуального пользования, предназначенных для обслуживания только их разработчика, программный продукт:

1) имеет универсальное назначение, ориентирован на применение многими пользователями и в ряде организаций;

2) предназначен для работы в комплексе с другими компонентами ПО;

3) имеет специальные средства модификации и расширения;

4) всесторонне отлажен;

5) описан в тщательно составленной документации.

Стоимость программного продукта приблизительно в 9 раз выше стоимости программы индивидуального назначения и с увеличением его сложности растет по квадратичному закону [3].

Для оценки сложности ПО используются два основных показателя:

1) количество операторов;

2) количество и типы взаимосвязей компонентов ПО между собой. Этот показатель более важный, так как именно он определяет эффективность декомпозиции исходной задачи разработки ПО в целом на ряд вложенных подзадач разработки его компонентов. Поэтому, в частности, трудоемкость разработки управляющих программ выше (приблизительно в 4 раза) трудоемкости разработки прикладных программ.

С конца 60-х годов, когда очевидным стало противоречие между потребностями общества в системах электронной обработки информации и возможностями создания ПО, началось становление новой научной дисциплины «Методы разработки программного обеспечения», основные цели которой — разработка методов управления сложностью систем ПО; повышение характеристик надежности и правильности ПО; развитие методов планирования и прогнозирования затрат на разработку ПО [4].

В рамках дисциплины «Методы разработки ПО» предлагается рассматривать цикл жизни (цикл разра-

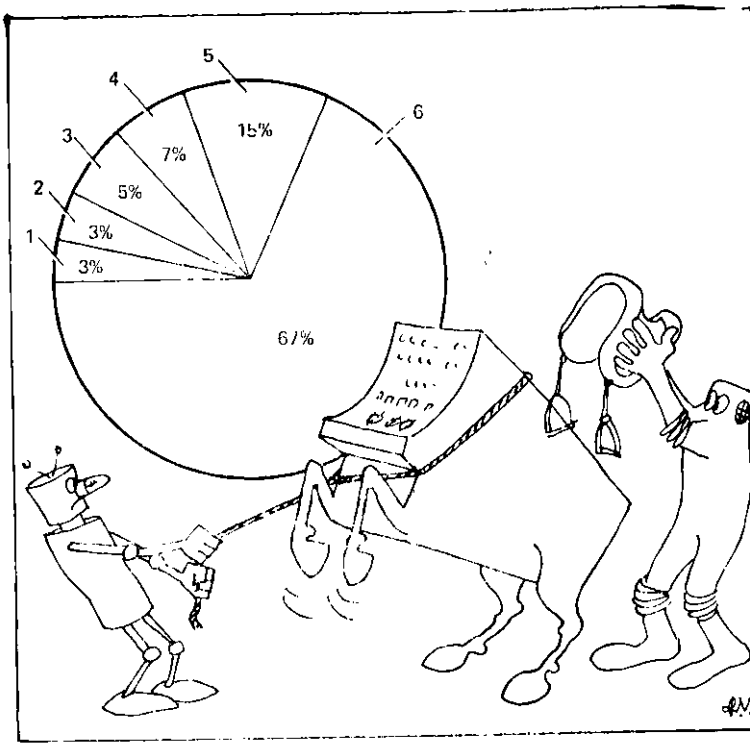


Рис. 1.12.

ботки) ПО состоящим из шести основных этапов: 1) анализ требований, предъявляемых к системе; 2) разработка спецификаций на систему; 3) проектирование; 4) кодирование; 5) тестирование; 6) сопровождение и эксплуатация. На рис. 1.12 приведено соотношение временных затрат по этапам цикла жизни ПО: *1* — анализ требований; *2* — определение спецификаций; *3* — проектирование; *4* — кодирование; *5* — тестирование; *6* — сопровождение.

Этап 1. Анализ требований, предъявляемых к ПО САПР. Основу этих требований составляют требования, перечисленные выше (см. Введение). Они дополняются требованиями заказчика, включающими пространственно-временные ограничения, необходимые функции и возможности, режимы функционирования, требования точности и надежности и т. д., т. е. вырабатывается описание системы с точки зрения пользователя. Эта работа

выполняется группой системных аналитиков, конечным результатом деятельности которых должно стать полное и непротиворечивое описание требований к проектируемой системе на языке, понятном разработчикам ПО.

На данном этапе выявляются проектные процедуры и операции, автоматизация которых возможна и целесообразна, изучаются особенности математических моделей (ММ) проектируемых объектов, выбирается или разрабатывается математическое обеспечение. Принимается решение о типах используемых ЭВМ и операционных систем, рассматривается возможность использования готовых компонентов ПО. Здесь же решаются вопросы планирования работ, устанавливается их очередность, этапность сдачи подсистем САПР в эксплуатацию. Особое внимание уделяется исследованию путей создания открытого ПО.

Этап 2. Определение спецификаций на ПО САПР. На этом этапе производится точное описание функций САПР, разрабатываются и утверждаются входные и промежуточные языки, форма выходной информации для каждой из подсистем, описывается возможное взаимодействие с другими программными комплексами, специфицируются средства расширения и модификации ПО, разрабатываются интерфейсы обслуживающих и проектирующих подсистем, решаются вопросы организации базы данных, утверждаются основные алгоритмы, реализуемые в подсистемах. Итогом выполнения данного этапа являются эксплуатационные и функциональные спецификации, содержащие конкретное описание ПО. Разработка спецификаций требует тщательной работы системных аналитиков, постоянно контактирующих с пользователями, которые в большинстве случаев не способны самостоятельно сформулировать четкие и реальные требования к системе.

Эксплуатационные спецификации содержат сведения о быстродействии ПО, затратах памяти, требуемых технических средствах, надежности и т. д. [5].

Функциональные спецификации определяют функции, которые должна выполнять САПР, т. е. в них определяется, что́ надо делать, но без указания, как это делать.

Спецификации должны быть полными, точными и ясными (понятными) [6]. Полная спецификация исключает необходимость получения разработчиками ПО в процессе их работы от пользователей иных сведений, кроме содержащихся в спецификациях. Точная спецификация

не позволяет толковать ее в разных смыслах. Ясная спецификация должна быть легко написана и прочитана как пользователем ПО (возможно, через системного аналитика), так и его разработчиком, причем оба они должны понимать ее совершенно однозначно. Ниже будут рассмотрены некоторые формальные средства разработки спецификаций.

Значение спецификаций при создании ПО определяется тремя факторами:

1) спецификации являются заданием на разработку ПО и строгое их выполнение — закон для разработчика;

2) спецификации используются для проверки готового ПО (выполняет ли оно требуемые функции и в какой степени);

3) спецификации, являясь неотъемлемой частью программной документации, облегчают сопровождение и модификацию ПО.

Завершается второй этап цикла жизни подготовкой тестов, на которых будут проводиться испытания при приемке САПР. На подготовленные тестовые данные в дальнейшем не будет оказывать влияние конкретная реализация системы.

Этап 3. Проектирование ПО САПР. На этом этапе формируется структура ПО и разрабатываются алгоритмы, задаваемые спецификациями. Устанавливается состав модулей с разделением их на иерархические уровни на основе изучения схем алгоритмов для типовых задач проектирования [7]. Выбирается структура информационных массивов, составляющих базу данных. Фиксируются межмодульные интерфейсы.

Для эффективной реализации данного этапа разработки ПО предложены методы, наиболее известные из которых будут рассмотрены ниже. Эти методы имеют одну общую цель, реализуемую разными способами, — иерархическое разбиение сложных задач создания ПО на подзадачи меньшей сложности. Результатом работ на этом этапе являются спецификации на отдельные модули, дальнейшая декомпозиция которых на подмодули нецелесообразна.

Этап 4. Кодирование модулей. На данном этапе производится программирование модулей на каком-либо алгоритмическом языке, т. е. перевод разработанных алгоритмов на язык программирования. Этот этап менее сложен по сравнению со всеми остальными этапами цикла жизни ПО, для его реализации широко используется

метод структурного программирования. Одна из задач, которую необходимо решить на данном этапе, — обоснованный выбор языков программирования.

Этап 5. Тестирование. Тестированием называется процесс проверки ПО (или его компонентов), имеющий целью обнаружение ошибок в ПО. При этом используются тестовые наборы, включающие в себя специально подобранные исходные данные и эталонные результаты. Тестирование включает в себя три шага: автономное, комплексное и системное. При автономном тестировании контролируется каждый компонент ПО изолированно от других на данных, подготовленных его разработчиком. В процессе комплексного тестирования всего ПО выявляются ошибки, связанные с нарушением спецификаций на все ПО САПР, при этом используются тесты, подготовленные на этапе 2. Системное тестирование — испытание ПО САПР на технических средствах и данных пользователя в производственных условиях. Тестирование является составной частью *отладки* — процесса обнаружения ошибок, их локализации и устранения.

Необходимо отметить, что надежность ПО закладывается на более ранних этапах его цикла жизни, правильное тестирование позволяет лишь выявить большинство из относительно немногочисленных вкравшихся ошибок. Повысить надежность плохо спроектированных программ никакое тестирование не способно.

Этап 6. Сопровождение. Как показано на диаграмме рис. 1.12, наибольшие затраты в цикле жизни ПО приходятся именно на этап сопровождения. Эти затраты для ПО САПР складываются в основном из затрат: 1) на устранение ошибок, не выявленных на этапе тестирования; 2) на внесение изменений в первоначальную версию ПО, вызванных взаимным недопониманием заказчика и разработчика ПО (в лице системных аналитиков) на первых двух этапах создания ПО, что привело к разработке «не тех» спецификаций; 3) на адаптацию ПО к быстроизменяющимся требованиям к САПР.

Устранение ошибки во время эксплуатации ПО обходится по крайней мере в два раза дороже, чем на этапе тестирования. На рис. 1.13 показана динамика изменения количества обнаруживаемых ошибок N в ПО с момента сдачи его в эксплуатацию [3]. Первоначально обнаруживаются наиболее «простые» ошибки, затем некоторое время все идет нормально, но с накоплением опыта и повышением квалификации пользователей САПР, с ее

полной загрузкой начинают выявляться наиболее «тонкие» ошибки. И если при проектировании ПО не были приняты соответствующие меры (такие, например, как использование принципа модульности), то оно со временем становится все менее упорядоченным и менее жизнеспособным (верхняя часть заштрихованной области на рис. 1.16). Одна из проблем сопровождения состоит в том, что даже для «правильно» спроектированного ПО исправление одной ошибки влечет за собой внесение новой ошибки с вероятностью 0,2—0,5. Третья составляющая затрат на сопровождение для ПО САПР наиболее существенна и связана с задачей продления срока эффективной эксплуатации САПР, поскольку изменение технологий промышленного производства и развитие математического обеспечения АП происходят обычно более быстрыми темпами, чем может создаваться ПО новых САПР.

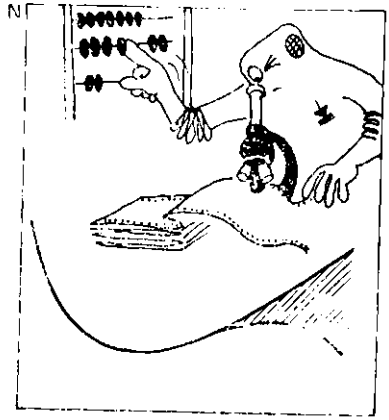


Рис. 1.13.

Для замедления морального старения ПО САПР предлагают три возможных способа:

1) разработка САПР совместно с разработкой новых промышленных технологий;

2) разработка САПР с учетом прогнозов на будущее;

3) создание САПР, открытых как к новым элементам математического обеспечения, так и по отношению к новым технологиям и предметным областям.

Наиболее перспективен третий способ, хотя он и наиболее сложен в реализации.

■ Примечание. В § 1.2 предлагается архитектура ПО некрупной САПР, обладающей свойством открытости; в гл. 5 рассматриваются некоторые подходы к созданию адаптируемых проектирующих пакетов САПР. Однако для полной разработки этого вопроса необходимы дальнейшие совместные усилия специалистов в области методов разработки ПО и математического обеспечения САПР.

Особенно остро задача сопровождения ПО стоит для САПР, эксплуатируемой в нескольких организациях, по-

скольку процесс исправления обнаруженных ошибок, включения новых программных компонентов различными пользователями очень скоро приводит к появлению стольких версий ПО, что их учет и контроль становится невозможным. Чтобы этого не происходило, все коррективы и дополнения необходимо вносить только в версию разработчика ПО и передавать ее всем пользователям одновременно через достаточно крупные промежутки времени, называемые *периодом обновления*.

Рассмотренная модель цикла жизни ПО весьма условна. Реальный процесс разработки носит итерационный характер, многие этапы перекрывают друг друга, выполняются параллельно. Рассмотрим некоторые конкретные методы разработки ПО.

Методы формализации разработки ПО. В последнее время разработчиками больших программных систем все большее внимание уделяется созданию методов и средств формализации первых двух этапов цикла разработки ПО, поскольку ошибки, внесенные в проект на этих этапах, самые «дорогие». Данные методы и средства позволяют вручную или автоматически выявлять и устранять такие недостатки функциональных спецификаций, как неполнота, противоречивость и неоднозначность.

Пример технологии ручной разработки спецификаций на ПО дан К. Хеннинджером в [6]. Суть этого подхода к документированию требований выражается в следующих трех принципах:

1. Ставить вопросы прежде, чем пытаться на них ответить.

■ Примечание. Разработан специальный список вопросов, на которые необходимо получить ответы заказчика ПО. По мере работы с потенциальными пользователями список расширяется новыми, уточняющими вопросами.

2. Разделять аспекты. Этот принцип обеспечивает возможность каждому участнику проекта сосредоточиться на хорошо определенной совокупности вопросов и позволяет легко модифицировать документ.

3. Как можно больше формализации.

■ Примечание. Для представления требований и спецификаций разработаны формальные способы, чисто словесные описания сведены к минимуму.

Из автоматизированных средств анализа требований и документирования

спецификаций наиболее известны системы PSL/PSA и SREM [6].

Система PSL/PSA позволяет: а) описывать проектируемые информационные системы в форме, пригодной для обработки на ЭВМ, для чего используются язык PSL; б) записывать и хранить такое описание в базе данных; в) корректировать описание в базе данных; г) выполнять анализ описания и выдавать отчеты. Три последние функции реализуются с помощью пакета программ, называемого *анализатором формулировок задач* PSA. Кроме описания на языке PSL в базе данных могут храниться таблицы, массивы, матрицы, а также графические диаграммы и схемы. В язык PSL намеренно не включены средства процедурного типа, чтобы не провоцировать преждевременное начало проектирования ПО. Анализатор формулировок задач PSA поставляет системным аналитикам отчеты, содержащие результаты различного рода анализов сведений, содержащихся в базе данных. Он может, например, обеспечить анализ выходных и входных данных контактирующих компонентов проектируемого ПО или анализ взаимодействия данных и компонентов ПО.

Методология разработки требований к ПО, реализованная в системе SREM, схожа с PSL/PSA и включает в себя язык описания требований RSL и подсистему REVS обработки и проверки требований. Язык PSL ориентирован на спецификацию требуемых шагов обработки данных в виде потоковых графов.

Подсистема REVS обработки и проверки требований состоит из: 1) транслятора с языка описаний требований RSL; 2) центральной базы данных, содержащей модель проектируемой программной системы; 3) автоматизированных средств обработки информации в базе данных. Подсистема REVS имеет средства машинной графики, позволяющие работать с изображениями потоковых графов, а также обеспечивает *динамическое моделирование* разрабатываемого ПО, используя для этого имитаторы отдельных компонентов ПО. Такие имитаторы могут применяться для проверки системных требований и для определения набора алгоритмов, «наилучшим» образом отвечающих этим требованиям.

Процесс проектирования ПО, как и любых других сложных объектов, — блочно-иерархический. Систематическое применение декомпозиции позволяет свести задачу большой размерности к совокупности простых подза-

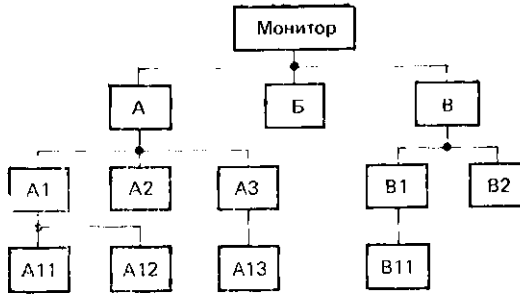


Рис. 1.14. Вариант структуры программы

дач. Принципиально возможны два подхода к проектированию ПО: нисходящий и восходящий.

Нисходящее проектирование (пошаговая детализация) представляет собой последовательность шагов, уточняющих проект. Первый шаг — определение способа решения задачи в самых общих чертах. За первым шагом следуют мелкие шаги в направлении детализации алгоритмов и структур данных. В ходе этого процесса выделяются отдельные модули решения и данных и дальнейшая конкретизация каждого модуля может производиться независимо. Специально для реализации стратегии нисходящего проектирования разработан язык проектирования программ PDL [4]. Он состоит из двух частей: 1) заданного набора операторов, построенных по образцу того языка программирования, на котором планируется вести кодирование компонентов ПО; 2) предложений естественного языка. Для описания логики проектируемой программы используются управляющие структуры языка программирования (цикл, ветвление, вызов подпрограмм), а для описания данных и процедур их обработки — естественный язык.

● **Пример нисходящего проектирования с использованием языка PDL.** Программа, структура которой представлена на рис. 1.14, может быть описана средствами языка PDL на первом шаге нисходящего проектирования следующим образом:

```
монитор: ПРОЦЕДУРА;
  выполнить действие А;
  ЕСЛИ (условие 1 истинно) ТО;
     выполнить действие Б;
  КОНЕЦ — ЕСЛИ;
  ЦИКЛ ПОКА (условие 2 истинно);
     выполнить действие В;
  КОНЕЦ — ЦИКЛ;
КОНЕЦ — ПРОЦЕДУРА;
```

Следующим шагом детализации может быть принятие решения о выделении отдельных подпрограмм А, Б и В, реализующих действия А, Б, и В, и определение данных, передаваемых в эти подпрограммы. Кроме того, в целях возможной экономии ОП принимается, что подпрограмма Б загружается в нее динамически. В результате проект программы принимает такой вид:

```
монитор: ПРОЦЕДУРА;
    определить данные α, β, γ, ... ;
    подготовить α, β, γ
    CALL A (α, β, γ)
  ЕСЛИ (условие 1 истинно) ТО;
    загрузить в ОП подпрограмму Б;
    CALL Б (γ);
  КОНЕЦ — ЕСЛИ;
  ЦИКЛ ПОКА (условие 2 истинно);
    CALL В (...)
  КОНЕЦ — ЦИКЛ;
КОНЕЦ — ПРОЦЕДУРА;
```

После того как принято решение о том, что какая-либо часть поставленной задачи будет решаться отдельным модулем, необходимо исчерпывающе и строго определить функции этого модуля и его интерфейс, не рассматривая деталей его реализации. Процедура пошаговой детализации монитора продолжается до тех пор, пока на очередном шаге не станет очевидной его реализация операторами выбранного языка программирования. Далее аналогичным образом проектируются модули А, Б и В (возможно, параллельно разными программистами). В конце этапа проектирования имеем логическую структуру ПО в виде дерева, как это показано на рис. 1.11, и детальные спецификации и проекты всех его модулей.

Достоинство нисходящего проектирования состоит в том, что оно позволяет разработчикам ПО сосредоточиться на основных для данного момента проблемах и отложить принятие всех тех решений, которые не должны приниматься на данном этапе проектирования.

Для реализации нисходящего проектирования разработаны технологии, наиболее известной из которых является *методология* HIPO *на основе графических диаграмм* (фирма «IBM»). В этой методологии предусмотрено два вида диаграмм. С помощью *иерархических диаграмм* (подобных диаграмме, показанной на рис. 1.14) представляется полная структура ПО. Каждый модуль этой диаграммы, в свою очередь, раскрывается *диаграммой* «вход — обработка — выход» (HIPO — input — process — output), изображенной на рис. 1.15.

Использование нисходящего проектирования ставит перед разработчиками ПО две серьезные проблемы.

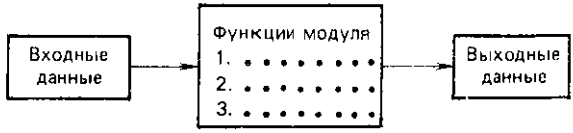


Рис. 1.15. IPO-диаграмма

Первая проблема связана с начальным выбором алгоритмов и структур данных, реализуемых в ПО. Если, например, выбор структуры данных сделан преждевременно, то на более низких уровнях проектирования может выявиться ее неприменимость для эффективной реализации ряда алгоритмов — возникает необходимость перепроектирования всего ПО. Поэтому нисходящее проектирование требует с самого начала ставить и решать наиболее фундаментальные задачи, откладывая частные вопросы для последующего рассмотрения.

Вторая проблема связана с тем, что нисходящее проектирование приводит к древовидной структуре ПО, и вполне вероятно, что в разных поддеревьях этой структуры окажутся модули с похожими, но не одинаковыми спецификациями. Конечно же, целесообразно объединить спецификации и разработать один универсальный модуль, однако это приведет к перепроектированию вызывающих модулей.

Восходящее проектирование альтернативно методу пошаговой детализации. В то время как при нисходящем проектировании первоначальная задача разбивается на подзадачи, которые потом пытаются реализовать, при восходящем проектировании, прежде чем приступить к решению задачи, создаются средства, позволяющие ее решить. Например, если необходимо спроектировать восходящим методом программу, аналогичную программе, представленной на рис. 1.14, сначала проектируют модули A11, и A12, затем добавляют модуль A1, использующий характеристики A11 и A12, и т. д., пока не будет получен законченный проект ПО, отвечающий всем спецификациям.

Основной недостаток восходящего проектирования связан с проблемой интеграции модулей одного уровня иерархической структуры посредством модуля более высокого уровня. Такая интеграция может быть чрезвычайно сложна, поскольку спроектированные раздельно (и, возможно, разными программистами) нижние моду-

ли в общем случае имеют несогласованные интерфейсы и используют различные способы представления данных, подлежащих обобществлению. По этой причине восходящее проектирование находит на практике ограниченное применение, его используют для создания программных систем, имеющих прототипы, структура которых известна и хорошо теоретически разработана (ОС, языковые процессоры и т. п.).

В действительности при разработке ПО нисходящее и восходящее проектирования не используются по отдельности, а взаимно дополняют друг друга. Например, по восходящему методу первоначально создается «инструментарий» программистов — общие подпрограммы ввода — вывода, обслуживания структур данных, динамического распределения памяти и т. д. Дальнейшее проектирование ведется нисходящим методом. С использованием нисходящего метода проектируется состав модулей ПО, их функции и связи по управлению. Проектирование связей по информации на этой стадии носит эскизный характер. Детальная разработка межмодульных интерфейсов производится с использованием восходящего метода. Объясняется это следующим: чем ниже уровень иерархической структуры ПО, занимаемый модулем, тем чаще этот модуль применяется, и, следовательно, обстоятельство, «удобен» или нет интерфейс модуля для его разработки, оказывает значительное влияние на эффективность всего ПО.

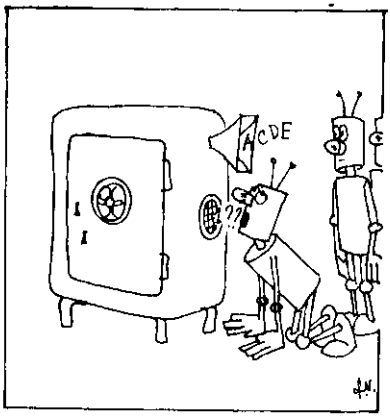
Программирование модулей ПО. Выше многократно использовался термин «модуль» как синоним понятия «часть ПО». *Модуль* — программная единица, для создания которой нужен минимум знаний о других программных единицах, составляющих ПО; перекомпоновка и замена модулей не должна вызывать перекомпоновки ПО в целом [8]. Таким образом, центральная концепция модульности — *независимость*. Модуль:

1) должен реализовывать единственную функцию, иметь один вход и один выход;

2) должен возвращать управление тому, кто его вызывал, иметь возможность обращаться к другим модулям;

3) не должен сохранять историю своих вызовов;

4) должен быть сравнительно невелик (рекомендуемая специалистами длина исходного текста модуля — одна страница распечатки АЦПУ) [9].



Для достижения независимости модулей часто при проектировании ПО используется *принцип информационной локализованности*, который состоит в том, что вся информация о структуре данных, требующихся многим компонентам ПО, сосредоточивается («упрятывается») в одном модуле. Доступ к данным осуществляется только через этот модуль. Таким образом, конкретное представление структур данных скрывается от всех модулей-пользователей. При необходимости изменения структуры данных все связанные с этим модификации будут сосредоточены только в одном модуле.

Подобно проектированию, кодирование (программирование) модулей ПО может осуществляться в двух направлениях: 1) восходящем; 2) нисходящем.

При восходящем кодировании после завершения проектирования всего ПО приступают к кодированию модулей самого нижнего уровня иерархической структуры ПО, а затем, после их проверки, переходят к модулям более высокого уровня и т. д., пока не будет изготовлена вся система. Недостаток этого подхода в том, что убедиться в правильности функционирования ПО с точки зрения пользователя можно только после завершения кодирования самого верхнего модуля.

Нисходящее кодирование свободно от этого недостатка. В его простейшей форме подразумевается, что кодирование начинается после завершения проектирования всего ПО. Но чаще под нисходящим кодированием понимается процесс, идущий параллельно с проектированием: после завершения проектирования модулей какого-либо уровня следует их кодирование и тестирование совместно с модулями верхних уровней и только потом переходят на следующий нижний уровень и т. д.

Кодирование сверху вниз подразумевает использование *нисходящего тестирования* вновь закодированных

модулей с уже отлаженными модулями верхних уровней, при этом еще не разработанные модули нижних уровней имитируются специальными подыгрывающими подпрограммами — *заглушками*. Тестированием «последнего» модуля завершается комплексное тестирование всего ПО. К основным достоинствам нисходящего тестирования относят: 1) исключение тестирования на уровне системы; 2) тестирование в первую очередь важнейших интерфейсов между модулями; 3) ознакомление с «макетом» будущего ПО пользователя на относительно ранних стадиях разработки [10].

Использование принципа пошаговой детализации при программировании отдельных модулей ПО называют *структурным программированием*. Цель структурного программирования — заставить программиста мыслить ясно, писать программы минимальной сложности, облегчать восприятие программ [5]. Эта цель может быть достигнута в первую очередь за счет использования для выражения логики программ небольшого набора простых структур управления [следование, ветвление и цикл (см. § 1.1)]. С этим методом хорошо согласуется использование языка PDL.

Структурное программирование — это итерационный процесс, на первом шаге которого весь программный модуль представляется тремя операторами PDL:

```
модуль А: ПРОЦЕДУРА;
    выполнить спецификации модуля А;
КОНЕЦ — ПРОЦЕДУРА;
```

На следующем шаге второй оператор заменяется одной из управляющих структур — следование, ветвление или цикл, на третьем и последующих шагах предложения на русском языке последовательно заменяются допустимыми структурами, пока не будут исключены полностью. В результате получается программа, характеризующаяся простотой, ясностью, легкой модифицируемостью.

■ Примечание. Неупорядоченное использование оператора ПЕРЕЙТИ — НА (GOTO) не позволяет осуществить такую процедуру программирования, поэтому часто структурное программирование называют программированием «без GOTO».

Для программ, полученных структурным методом, можно осуществить строгое и формальное доказательство их правильности с точки зрения выполнения спецификаций.

Выбор языка программирования. На этапе кодирования модулей ПО большое значение имеет выбор языков

программирования. На выбор языка программирования влияют:

1) возможность создания произвольных структур данных средствами языка;

2) наличие конструкций структурного программирования;

3) переносимость программ с ЭВМ одного типа на ЭВМ другого типа, из одной ОС в другую;

4) удобство освоения и использования, производительность кодирования;

5) эффективность закодированных на языке программ.

В настоящее время для создания ПО САПР наибольшее распространение получили алгоритмические языки: ФОРТРАН, ПЛ/1, ПАСКАЛЬ, ассемблера.

Алгоритмический язык ФОРТРАН предназначен только для научно-технических расчетов; прост в освоении, позволяет легко и быстро кодировать формулы и итерационные процессы над векторами и матрицами целого и вещественного типов. Трансляторы с языка ФОРТРАН имеются практически во всех ОС и обеспечивают высокую эффективность объектного кода. Однако примитивность этого языка в отношении типов и структур данных, отсутствие динамического распределения памяти существенно ограничивают его применение при разработке ПО САПР. Кроме того, структурное программирование на языке ФОРТРАН возможно только с использованием специальных *препроцессоров*, осуществляющих перевод с «расширенного» языка ФОРТРАН, включающего в себя конструкции структурного программирования, в стандартный язык ФОРТРАН.

Алгоритмический язык ПЛ/1 имеет конструкции структурного программирования и богатые средства для создания произвольных структур данных. Но он сложен в освоении, его трансляторы имеются в составе не всех ОС, генерируемый ими объектный код уступает ассемблерному по быстродействию и затратам ОП в 2—3 раза.

Алгоритмический язык ПАСКАЛЬ наиболее часто используется для создания ПО САПР. Язык ПАСКАЛЬ значительно проще языка ПЛ/1, хотя и обладает всеми его разумными возможностями. В нем специально исключены конструкции, приводящие к неэффективному объектному коду. Имеется опыт использования этого языка для создания не только прикладных программ, но и ОС. Трансляторы с языка ПАСКАЛЬ есть на

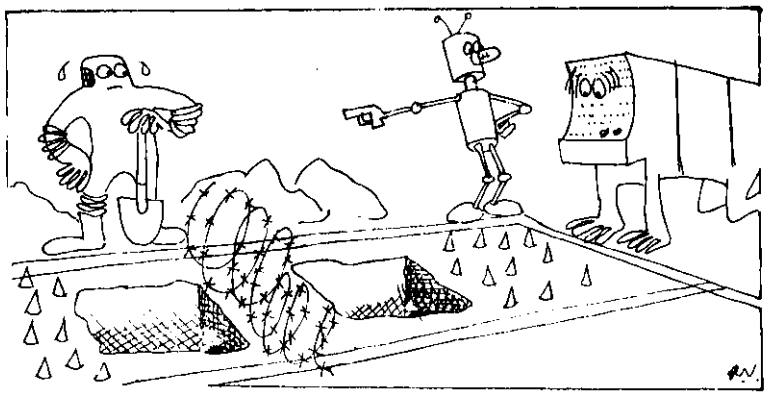
большинстве ЭВМ, выпускаемых в СССР. Сказанное выше позволяет рекомендовать этот язык в качестве основного языка программирования ПО САПР.

■ Примечание. В ближайшее время ожидается появление трансляторов с универсального языка программирования АДА. Существует мнение, что этот язык будет наиболее целесообразен для создания ПО САПР.

Алгоритмический язык ассемблера используется при создании сложных программных систем. Область применения этого языка в ПО САПР должна быть ограничена только отдельными модулями управляющей и обслуживающих подсистем, поскольку его жесткая привязка к типу ЭВМ и ОС усложняет адаптацию САПР к новым условиям применения. Возможно, что с распространением трансляторов с языка СИ потребность в языке ассемблера при разработке ПО САПР отпадет совсем. Однако знание языка ассемблера, как и особенностей работы используемой ЭВМ, обязательно для программиста-профессионала, с каким бы языком он ни работал.

Тестирование и отладка. Выше было отмечено, что для программ, созданных методом структурного программирования, допустимо математическое доказательство их правильности. Однако полное доказательство правильности сложных программ неосуществимо из-за очень большого объема самого доказательства (например, текст доказательства требует места на порядок большего, чем текст программы, для которой проводится доказательство). Кроме того, доказательство, являясь видом человеческой деятельности, само не защищено от ошибок. В настоящее время ведутся работы по автоматизации доказательств, но по-прежнему основным способом проверки правильности ПО остается тестирование. Важно, однако, осознавать, что положительные результаты тестирования являются необходимым, но не достаточным условием правильности программ.

Тестирование — один из наименее формализованных этапов цикла жизни ПО. Из автоматизированных средств здесь можно назвать только генераторы тестовых данных, имитаторы заглушек, компараторы эталонных и реальных результатов, т. е. удается автоматизировать сам процесс тестирования, но выбор тестовых данных является творческой функцией разработчиков ПО. И в зависимости от того, как им удается справиться с этой задачей, зависит стоимость и успешность тестирования и в конечном счете жизнеспособность всего ПО.



В [5] даются некоторые аксиомы тестирования:

Аксиома 1. Тестирование должно начинаться с постановки цели.

Аксиома 2. Считайте тестирование ключевой задачей разработки.

Аксиома 3. Невозможно тестировать свою собственную программу.

Аксиома 4. Хорошим считается тот тест, для которого высока вероятность обнаружить ошибку, а не тот, который демонстрирует правильную работу программы.

Аксиома 5. Готовьте тесты как для правильных, так и для неправильных входных данных.

Отладка ПО, в которую составной частью входит тестирование, очень сложна. Наибольшую трудность в ней составляет локализация местонахождения ошибки. Обычными здесь являются средства ОС и трансляторов с языков программирования [11]: 1) аварийная печать содержимого памяти ЭВМ; 2) отладочная печать указанных программистом переменных в указанных точках программы; 3) трассировка (печать значений переменных при каждом их изменении, а также меток операторов в порядке их выполнения).

В последние годы широкое распространение получают интерактивные отладчики, дающие возможность прерывать выполнение программы, просматривать и изменять любые ее элементы. Такие отладчики позволяют по крайней мере в два раза сократить время индивидуальной отладки отдельных модулей и подсистем ПО САПР.

Бригадная организация разработки ПО САПР обусловлена его сложностью, разнородностью входящих в

его состав подсистем, при этом для разработки каждой подсистемы создается своя бригада, оптимальная численность которой составляет 3—5 человек. Увеличение числа членов бригады ведет обычно к снижению ее производительности за счет увеличения доли «непроизводительного» времени программистов, уходящего на их взаимодействие, работу за «слабого», обучение новичков и т. д.

Организация и средства разработки ПО. Иерархическая структура взаимодействия бригад отражает архитектуру ПО САПР (см. рис. 1.10). Бригада системного архитектора занимается разработкой архитектуры ПО САПР, проектирует, кодирует и тестирует управляющую подсистему САПР. Основу этой бригады составляют системотехник САПР и системный программист (оба должны быть программистами высокой квалификации). По окончании проектирования монитора бригада разрабатывает спецификации на обслуживающие подсистемы, утверждает унифицированный интерфейс с проектирующими пакетами. Далее вступают в работу бригады, разрабатывающие обслуживающие подсистемы ПО (диалоговую, машинной графики, управления базами данных). В эти бригады входят системные и прикладные программисты. Разработка проектирующих пакетов может начинаться после завершения этапа проектирования обслуживающих подсистем и вестись независимо (возможно, в различных организациях). Бригады, занятые разработкой прикладных пакетов, включают в себя кроме системных и прикладных программистов специалистов по автоматизации проектирования в конкретных предметных областях [1].

Рассмотренные выше передовые методы разработки ПО (ИПРО — технология, нисходящее проектирование, структурное проектирование, нисходящее тестирование, бригада главного программиста) были использованы фирмой «IBM» для создания программной системы объемом свыше 80 тыс. операторов языка программирования, при этом была достигнута производительность труда 65 операторов/день на каждого программиста и 35 операторов/день на каждого члена бригады. Если учесть, что бригада возглавлялась программистом чрезвычайно высокой квалификации, а проект поддерживался фирмой с колоссальными возможностями, то можно предположить, что эти показатели близки к предельным. Однако темпы выпуска ЭВМ во всем мире продолжают

расти (так, в США в настоящее время количество ежегодно выпускаемых ЭВМ превышает количество студентов, оканчивающих вузы), усиливаются потребности общества в системах ПО. Многие специалисты по электронной обработке данных связывают возможность разрешения этого противоречия с созданием и широким использованием *генераторов прикладных программ*. Например, такие интерактивные генераторы, как ADF и DMS, позволяют на несколько порядков повысить производительность труда программистов при разработке диалоговых прикладных программ для решения экономических задач. Практически для создания прикладного пакета требуется всего лишь несколько сеансов совместной работы системного аналитика и будущего пользователя за экраном дисплея, во время которых главным образом создаются и опробуются входные и выходные языки создаваемого ПО.

В качестве генератора простейших проектирующих пакетов САПР может выступать инструментальная система программирования ПРИЗ, разработанная в Институте кибернетики АН ЭССР. Ядром системы является семантическая память, в которую может быть записана модель соответствующей предметной области. Модель предметной области составляют переменные, соответствующие содержательным понятиям этой области, и отношения, связывающие эти понятия (переменные). Например, понятиями в механике будут сила, скорость, масса, трение, гибкость и т. д., а отношениями — законы Ньютона, Гука и др. При создании проектирующего пакета на основе ПРИЗ вручную происходит только заполнение семантической памяти и программирование отдельных модулей, выражающих отношения в ней. Этапы создания проблемно-ориентированного языка, разработка монитора пакета, обеспечение связи с диалоговой подсистемой и базой данных автоматизированы в ПРИЗ в очень высокой степени. Однако универсальная ориентация этой системы существенно ограничивает ее область применения в САПР пакетами, использующими простые аналитические модели проектирующих объектов.

Более широкое применение для создания проектирующих пакетов САПР могут найти генераторы, основанные на инвариантности многих элементов математического обеспечения автоматизированного проектирования к предметным областям [7]. Такие генераторы в качестве ядра будущей проектирующей подсистемы ПО использу-

ют модули одного или нескольких методоориентированных пакетов, снабжая их монитором.

Разработка трансляторов с проблемно-ориентированных языков на языки методоориентированных пакетов осуществляется генератором трансляторов. Для настройки созданного таким образом пакета на предметную область пользователю необходимо лишь заполнить базу данных пакета подпрограммами моделей элементов соответствующей физической природы. Этот подход дает возможность создавать в короткое время пользователям-прикладникам, имеющим малый опыт в программировании, очень эффективные проектирующие пакеты, но применим он главным образом для генерации пакетов функционального проектирования на микро- и макроуровнях, в меньшей степени — на метауровне.

■ Примечание. В гл. 5 будет рассмотрена подсистема ПО САПР, которая может быть использована в качестве ядра пакетов функционального проектирования динамических объектов, описываемых системами обыкновенных дифференциальных уравнений.

КРАТКИЕ ВЫВОДЫ

Специальное и базовое программное обеспечение САПР реализует алгоритмы обработки информации для выполнения проектных операций и процедур и представляет собой сложную программную систему.

В состав специального и базового ПО САПР входит ПО проектирующих и обслуживающих подсистем. Проектирующие подсистемы непосредственно предназначены для автоматизации проектных операций и процедур. Различают объектно-зависимые (проблемно-ориентированные) и объектно-независимые (методоориентированные) проектирующие подсистемы. Обслуживающие подсистемы поддерживают функционирование проектирующих подсистем, к ним относятся монитор (управляющая подсистема), подсистема диалогового взаимодействия, инструментальная, интерактивной машинной графики и управления базами данных.

Создание ПО САПР — сложная научно-техническая задача, решение которой возможно лишь с привлечением современных методов разработки ПО. Процесс создания ПО состоит из шести основных этапов: 1) анализ требований; 2) определение спецификаций; 3) проектирование; 4) кодирование модулей; 5) тестирование; 6) сопровождение. Наиболее ответственны ранние этапы разработки, на последний этап приходятся наибольшие затраты. Для повышения производительности труда разработчиков ПО предложен ряд методов и средств: анализаторы требований, нисходящее проектирование, модульное и структурное программирование, генераторы прикладных программ и др.

ПРИНЦИПЫ ПОСТРОЕНИЯ БАНКОВ ДАННЫХ

§ 2.1. Общие сведения

Современные системы обработки информации (СОИ) выполняют преобразование больших массивов данных. Синонимом понятия «данные» служат термины «информация» и «сведения». Как показал опыт разработки и эксплуатации СОИ, способы хранения и обработки данных оказывают решающее влияние на показатели функционирования системы в целом, ее практическую эффективность. Сказанное относится и к САПР, оперирующим с большим числом данных различного типа и назначения. Систематическая организация данных и способов их обработки осуществляется в банках данных.

Банк данных (БНД) — совокупность базы данных и системы управления базами данных.

База данных (БД) — структурированная совокупность данных. Наименьшая единица описания данных называется *элементом описания*. Совокупность элементов описания, объединенных отношением принадлежности к одному описываемому объекту, называется *записью*. Если элементы описания соответствуют отдельным свойствам объекта, то запись описывает объект в целом. Например, код типа микросхемы, логическая функция, мощность потребления, коэффициент разветвления в совокупности составляют запись и описывают свойства конкретного объекта — микросхемы.

Система управления базами данных (СУБД) состоит из языковых и программных средств, предназначенных для создания и использования базы данных прикладными программами, а также непосредственно пользователями-непрограммистами.

Применение банков данных позволяет решить следующие проблемы организации и ведения больших массивов

информации: 1) сокращение избыточности; 2) обеспечение целостности; 3) разграничение доступа; 4) обеспечение независимости представления данных.

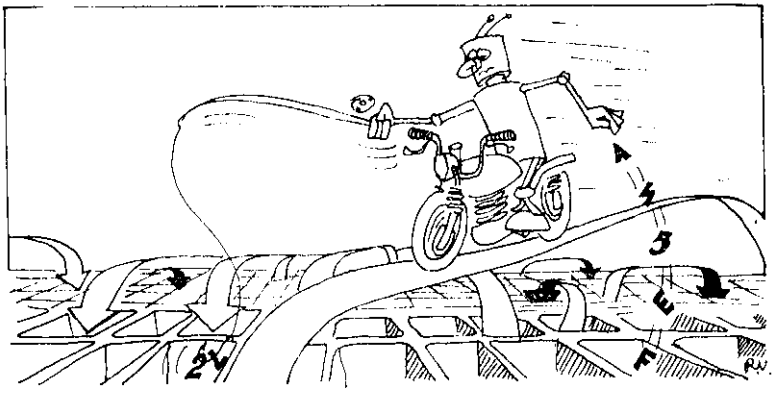
Избыточность вызывается наличием разных форм представления одних и тех же данных, размножением части данных для дальнейшего использования прикладными программами, повторными записями одинаковых данных на различных физических носителях информации.

Целостностью называется свойство БД в любой момент времени содержать лишь достоверные данные. Наличие избыточных, противоречивых и неверно составленных данных нарушает целостность БД.

Для сокращения избыточности производится объединение одинаковых по смыслу, но имеющих различный тип данных в единую БД с приведением к общему, стандартизованному виду. Процесс объединения данных, используемых различными пользователями, в одну общую БД называется *интеграцией базы данных*.

Однако каждый конкретный пользователь должен получить *доступ* лишь к некоторому подмножеству данных из БД, необходимых для выполнения своих прикладных программ. Одновременно с этим обеспечивается режим секретности и повышается степень защищенности данных от несанкционированного доступа.

Одним из важнейших преимуществ применения БД является возможность *обеспечения независимости представления данных* в прикладных программах от типов запоминающих устройств и способов их физической организации. В основном это достигается построением двух



уровней представления данных: логического и физического.

На *логическом уровне* данные представляются в виде, удобном для использования в прикладных программах или непосредственно проектировщиками.

Физический уровень представления данных отражает способ хранения и структуру данных с учетом их расположения на носителях информации в запоминающих устройствах ЭВМ.

Важнейшим понятием в БНД является *модель данных* — формализованное описание, отражающее состав и типы данных, а также взаимосвязи между ними. Модели данных классифицируются по ряду признаков.

В зависимости от объема описываемой информации на логическом уровне различают внешнюю и внутреннюю модели данных. *Внешняя модель данных (логическая подсхема)* описывает структуру информации, относящейся к некоторой конкретной процедуре или к группе родственных проектных процедур.

Внутренняя логическая модель данных (логическая схема) объединяет все подсхемы БД.

По способам отражения связей между данными на логическом уровне различают модели — иерархическую, сетевую и реляционную. Модель называют *сетевой,* если данные и их связи имеют структуру графа. Если структура отражаемых связей представляется в виде дерева, то модель называют *иерархической.* Представление данных в форме таблиц соответствует *реляционной* модели данных.

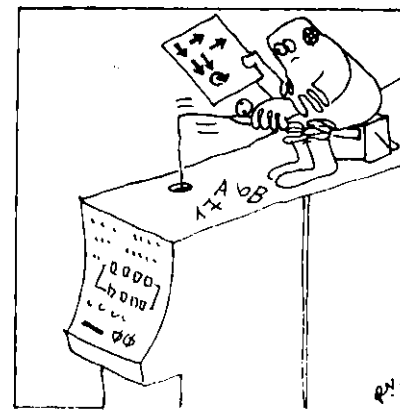
Задание модели данных в БД осуществляется на специальном *языке описания данных* (ЯОД). Иногда в ЯОД выделяют языки описания данных для подсхемы (ЯОД—ПС) и для схемы (ЯОД—С).

Язык описания данных представляет собой совокупность директив, построенных в соответствии с выбранной моделью данных.

■ Примечание. Несмотря на то что ЯОД ориентирован на логический уровень, в него, как правило, включаются директивы, позволяющие управлять расположением данных на внешних носителях.

Прикладные программы, использующие БНД, записываются на некотором алгоритмическом языке (например, ФОРТРАН, ПЛ/1, ПАСКАЛЬ), называемом *включающим языком.* Для обеспечения взаимодействия с БНД

в эти программы должны быть введены операторы обращения к СУБД. Совокупность операторов обращения к СУБД из прикладной программы составляет *язык манипулирования данными* (ЯМД). Основные операции с данными, выполняемые средствами ЯМД, следующие:



1) поиск информации по заданным поисковым признакам в БД;

2) включение в БД новых записей;

3) удаление из БД лишних или ненужных в дальнейшем записей;

4) изменение значений элементов данных в записях.

Как правило, операторы ЯМД реализуются во включающем языке с помощью оператора CALL и выполняют обращения к необходимым подпрограммам СУБД. Непосредственная запись операторов ЯМД во включающем языке используется реже. При этом необходима предтрансляция, в ходе которой операторы ЯМД заменяются эквивалентными группами операторов включающего языка. Затем выполняется окончательная трансляция прикладной программы. В результате появляется возможность осуществить как синтаксический, так и семантический контроль правильности обращения к системе управления базами данных.

Банк данных — сложная информационно-программная система, функционирование которой невозможно выполнить полностью в автоматическом режиме. Контроль за ее состоянием и управление режимами осуществляется человеком либо группой лиц, называемых администратором банка данных. Администратор прежде всего составляет внешние и внутреннюю модели данных, управляет размещением информации на физических носителях. Второй важнейшей его обязанностью является поддержание целостности БИД. Для этого администратор выполняет восстановление БД после сбоев аппаратуры, запись и хранение копий, ведение системного журнала, где фик-

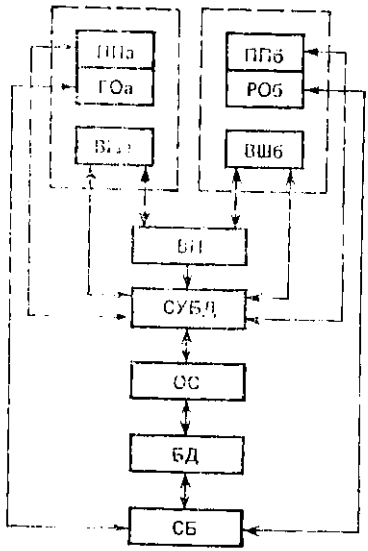


Рис. 2.1. Схема взаимосвязи СУБД с прикладными программами:

ППа, ППб — прикладные программы пользователей *а* и *б*; *РОа, РОб* — рабочие области пользователей *а* и *б*; *ВШа, ВШб* — внешние модели данных пользователей *а* и *б*; *ВП* — внутренняя логическая модель данных; *СБ* — системный буфер

сируются все изменения, вносимые в БД, устранение избыточности данных и др.

Взаимосвязь БД с прикладными программами представлена на рис. 2.1. Прикладные программы *ППа* и *ППб* пользователей *а* и *б* обращаются с запросами к СУБД, которая, пользуясь информацией о конкретной внешней модели и основываясь на описании логической схемы БД, формирует обращение к программным средствам того или иного метода доступа в составе ОС. Полученные данные поступают вначале в системный буфер, а затем помещаются в доступную пользователю рабочую область. Таким образом, СУБД можно рассматривать как некоторую надстройку над ОС, через которую происходит общение прикладного пользователя с массивами информации.

Совокупность модели данных и операций, определенных над данными, называется подходом. В соответствии с моделями данных различают реляционный, сетевой и иерархический подходы. Так как подход лежит в основе построения СУБД, различают реляционные, сетевые и иерархические СУБД. В настоящее время наибольшее распространение получили иерархические и сетевые СУБД (это объясняется возможностью обеспечить быстрый доступ к данным). Однако реляционные СУБД, несмотря на трудность их программной реализации, позволяют более удобно для пользователя описать структуру данных и манипулирование ими.

Тип организации СУБД определяется также степенью структурированности записей в составе БД. *Сильнострук-*

турированная запись — запись, построенная в соответствии с фиксированным, заранее определенным форматом всех элементов описания. К таким данным относятся, например, сведения о микросхемах (см. табл. 2.2).

Фактографические СУБД — СУБД, предназначенные для хранения сильноструктурированных записей. Однако не все данные могут быть сильноструктурированными. Например, в тексте ТЗ на проектирование могут быть выделены лишь элементы, соответствующие заголовку, году издания, организации-разработчику и всему остальному тексту, содержащему информацию символьного типа переменной длины. Такие записи называют *слабоструктурированными,* а соответствующие СУБД — *документальными* или *информационно-поисковыми* (ИПС). В САПР находят применение СУБД обоих типов.

Организация технического обеспечения САПР оказывает влияние на структуру информационного обеспечения и в первую очередь баз данных. Если БД сконцентрирована в одном узле вычислительной сети, то она называется *сосредоточенной*, в противном случае — *распределенной.* Если информационное обслуживание с помощью БД относится ко всей САПР, то БД называют *общей* (*интегрированной* или *центральной*), а если к отдельной проектирующей подсистеме САПР или к отдельному пакету прикладных программ, то *локальной БД.*

Основные проблемы в организации распределенных и локальных БД заключаются в разработке мероприятий по обеспечению целостности данных, своевременному обновлению информации и организации оперативного обмена между ними.

§ 2.2. Реляционный подход

Реляционная модель данных. Реляционные модели данных в последнее время получили широкое распространение вследствие простой формы представления данных, а также благодаря развитому теоретическому аппарату, позволяющему описывать различные преобразования реляционных данных. Основу реляционной модели данных составляет совокупность данных, сформированных в виде таблицы. Такая форма представления данных привычна для специалиста, пользующегося различной справочной литературой.

Из теории множеств известно, что формальным аналогом таблицы выступает отношение. Пусть дана совокуп-

ность множеств D_1, D_2, ..., D_n. *Отношением* R называется некоторое подмножество декартова произведения этих множеств:

$$R \subseteq D_1 \times D_2 \times \ldots \times D_n$$

Декартово произведение $D_1 \times D_2 \times \ldots \times D_n$ — множество всех возможных *кортежей* $(d_1, d_2, \ldots, d_n)$ таких, что $d_i \in D_i$, $i = 1, 2, \ldots, n$. Множества D называют *доменами*, а n — степенью отношения R.

Совокупность кортежей, записанных друг под другом, образует таблицу, строки которой соответствуют кортежам, а столбцы — атрибутам. *Атрибут* A представляет собой некоторое подмножество домена D.

■ Примечание. В общем случае допустимо, чтобы различные домены имели общие элементы, а атрибуты представляли подмножества одного и того же домена.

С другой стороны, атрибут A_i можно рассматривать как проекцию отношения R на i-ю координату. Например, $D_1 = \{a_1, a_2, a_3, a_4\}$, $D_2 = \{b_1, b_2, b_3\}$. Тогда

$$D_1 \times D_2 = \{<a_1, b_1>, <a_1, b_2>, <a_1, b_3>, <a_2, b_1>, <a_2, b_2>, <a_2, b_3>, <a_3, b_1>, <a_3, b_2>, <a_3, b_3>, <a_4, b_1>, <a_4, b_2>, <a_4, b_3>\}.$$

Построим отношение $Q = \{<a_1, b_3>, <a_2, b_1>, <a_2, b_3>, <a_3, b_1>, <a_3, b_2>\}$. Очевидно, что $Q \subset D_1 \times D_2$. Атрибутами являются множества $A_1 = \{a_1, a_2, a_3\}$, $A_2 = \{b_1, b_3\}$. При этом $A_1 \subset D_1$, $A_2 \subset D_2$.

В форме таблицы отношение Q выглядит как табл. 2.1.

Таблица 2.1

Q	A_1	A_2
	a_1	b_3
	a_2	b_1
	a_2	b_3
	a_3	b_1
	a_3	b_3

Степень отношения Q равна 2.

Запись вида $Q(A_1, A_2)$ называется *схемой отношения* Q и содержит наряду с названием отношения имена атрибутов.

Совокупность схем отношений составляет схему *реляционной* БД.

■ Примечание. Распространен случай, когда БД, имеющая многие десятки тысяч кортежей, содержит сравнительно небольшое число схем отношений.

В реляционной модели предполагается, что все отношения должны быть *нормализованы*, т. е. каждый кортеж должен содержать лишь атомарные (неделимые)

элементы. Это означает, что отношения не могут быть элементами отношений.

Операции над отношениями выполняются методами реляционного исчисления и реляционной алгебры.

Реляционное исчисление. Оно базируется на теоретических основах исчисления предикатов. Использование реляционного исчисления дает возможность манипулировать с данными на уровне выходного документа, что позволяет строить удобные для пользователя ЯМД непроцедурного типа. Пользователь имеет возможность производить описание необходимого ему отношения независимо от процедур поиска и порядка действий над данными.

■ Примечание. Напомним, что предикатом $P(x_1, x_2, \ldots, x_n)$ называется функция, принимающая значения «истина» или «ложь», от аргументов, определенных в конкретных областях $\mathbf{D}_1$, $\mathbf{D}_2, \ldots, \mathbf{D}_n$. При построении высказываний используются логические связки $\wedge$, $\vee$, $\neg$, $\cdot$, $\leftrightarrow$, называемые соответственно конъюнкцией, дизъюнкцией, отрицанием, импликацией и эквивалентностью. Кроме того, применяются термы сравнения, имеющие вид $x_i * x_j$, где $*$ – символ операции сравнения, в качестве которого могут применяться символы $=$, $\neq$, $>$, $<$, $\geqslant$, $\leqslant$. Кванторы существования $\exists$ и всеобщности $\forall$ позволяют отнести высказывание ко всему рассматриваемому множеству. Так, выражение $\exists x \in \mathbf{X}(f(x) > a)$ означает, что среди элементов множества $\mathbf{X}$ найдется по крайней мере один, при котором оказывается истинным неравенство, заключенное в скобках. Если использовать квантор всеобщности $\forall x \in \mathbf{X}(f(x) > a)$, то получим высказывание: для всех элементов множества $\mathbf{X}$ некоторая функция $f(x)$ больше заданного значения a. Неравенство $(f(x) > a)$ представляет собой предикат: «функция от x больше константы a». Предикат принимает значение «истина» (1) или «ложь» (0). Областью определения аргумента x предиката является множество $\mathbf{X}$. Если указанный предикат обозначить $P(x)$ и опустить явное указание области определения $\mathbf{X}$, то получим более принятую в исчислении предикатов запись: $\exists x P(x)$ и $\forall x P(x)$.

Таким образом, по определению кванторов существования и всеобщности имеем следующие соотношения:

$$\exists x P(x) \leftrightarrow P(b_1) \vee P(b_2) \vee \ldots \vee P(b_m);$$

$$\forall x P(x) \leftrightarrow P(b_1) \wedge P(b_2) \wedge \ldots \wedge P(b_m),$$

где $\{b_1, b_2, \ldots, b_m\} = \mathbf{X}$.

В реляционном исчислении принято связывать с отношением $\mathbf{R}(\mathbf{A}_1, \ldots, \mathbf{A}_n)$ некоторый предикат $P(x_1, \ldots, x_n)$, аргументы которых имеют одинаковые области определения, таким образом, что если $P(a_1, a_2, \ldots, a_n) = 1$, то кортеж $\langle a_1, a_2, \ldots, a_n \rangle$ принадлежит отношению $\mathbf{R}$ $a_i \in \mathbf{A}_i$,

$i=\overline{1, n}$. В противном случае кортеж не входит в состав указанного отношения. Отсюда следует, что посредством задания некоторого предиката может быть задано и соответствующее ему отношение.

Так как до сих пор задавались отношения лишь перечислением кортежей, то реляционное исчисление представляет новый способ задания отношений.

● **Пример построения отношения.** Задано отношение $R_1(A_1, A_2) = \{<5,1>, <10,4>, <7,2>, <9,8>\}$. Поставим отношению R_1 в соответствие предикат P_1:

$$R_1(A_1, A_2) \leftrightarrow P_1(x_1, x_2),$$

где переменные x_1 и x_2 имеют области определения A_1 и A_2. Тогда предикат $P_2(x_1) \leftrightarrow \forall x_2 P_1(x_1, x_2) \wedge (x_2 > 2)$ формирует новое отношение $R_2(A_3)$, в которое войдут лишь те значения x_1, для которых соответствующие значения x_2 в отношении R_1 оказались больше 2: $R_2(A_3) = \{<10>, <9>\}$.
Ясно, что предикат P_2 однозначно определяет отношение R_2.

В [19] приводится описание псевдоязыка реляционного исчисления. Кратко рассмотрим его использование для операций поиска.

Оператор построения нового отношения

СОЗДАТЬ R (...) : L,
где R (...),

означает наименование искомого отношения (в скобках указываются имена атрибутов; символ «:» имеет смысл выражения «такое, что»; L — некоторое высказывание относительно используемых отношений, значений атрибутов, их взаимозависимости). Имена атрибутов обычно задаются совместно с именем отношения, разделителем выступает точка. Например, аргумент A_2 из отношения R_1 будем обозначать $R_1 \cdot A_2$.

Оператор принадлежности

ПУСТЬ В ОТНОШЕНИИ P КОРТЕЖ X

означает, что переменная X принимает значения, равные кортежам отношения P. Он используется в случаях, когда переменная X связывается квантором и описывает область определения связанной переменной.

● **Примеры запросов.** Пусть БД задана в виде совокупности трех отношений, представленных в табл. 2.2 ... 2.4.
1. Получить коды всех микросхем

СОЗДАТЬ W (ОМС.КМ)

Таблица 2.2

ОМС

Код типа микросхемы (КМ)	Логическая функция микросхемы (ЛФМ)	Коэффициент разветвления (КР)	Мощность потребления, мВт, (ПМ)	Задержка, нс (ЗД)
155ЛА3	4×2 И—НЕ	10	110	19
155ЛА6	2×4 И—НЕ	30	86	25
155ТМ2	2 D—Т	10	150	55
155ТВ1	JK—Т	10	100	55
155ИР1	4 РЕГ	10	410	35

Таблица 2.3

ОУЗ

Код узла (КУ)	Логическая функция узла (ЛФУ)	Применение (П)	Разрядность (Р)
Р2	Регистр	АК10	16
ДШ10	Дешифратор	АК20	10
Сч50	Счетчик	АК10	32
Сч46	»	АК30	10
Р3	Регистр	АК20	32
СМ5	Сумматор	АК10	16
СМ15	»	АК30	8
Р1	Регистр	АК20	16

Таблица 2.4

ОСУ

Код узла (КУ)	Код типа микросхемы (КМ)	Количество (К)
Р2	155ЛА3	16
Р2	155ИР1	4
ДШ10	155ЛА3	2
ДШ10	155ЛА6	3
Сч50	155ТМ2	16
Сч50	155ЛА6	16
Сч46	155ТВ1	10
Сч46	155ЛА3	4
Р3	155ИР1	8
Р3	155ЛА3	8
СМ5	155ТМ2	8
СМ5	155ЛА6	10
СМ15	155ТМ3	8
СМ15	155ЛА3	6
Р1	155ТВ1	16

Результат

W

КМ
155ЛА3
155ЛА6
155ТМ2
155ТВ1
155ИР1

Тот же результат может быть получен и с помощью оператора

СОЗДАТЬ W (ОСУ.КМ)

Все повторяющиеся значения кодов микросхем встречаются в отношении W лишь однажды

2. Получить коды микросхем, для которых мощность потребления не превосходит 130 мВт.

СОЗДАТЬ W (ОМС.КМ): ОМС.ПМ<130

Результат

W

КМ
155ЛА3
155ЛА6
155ТВ1

3. Получить коды узлов, в составе которых присутствуют микросхемы 155ЛА3 в количестве не менее 5 шт.

СОЗДАТЬ W (ОСУ.КУ): ОСУ.КМ='155ЛА3'∧ОСУ.К⩾5

W

КУ
Р2
Р3
СМ15

■ Примечание. Приведенные выше примеры основывались на использовании одного и того же отношения. Поиск, требующий более одного отношения, описывается с помощью кванторов. При этом для каждой переменной, связанной квантором, должна быть описана область определения с помощью оператора принадлежности.

4. Получить логические функции микросхем, используемых в узле СМ5.

ПУСТЬ В ОТНОШЕНИИ ОСУ КОРТЕЖ X

СОЗДАТЬ W (ОМС.ЛФМ): ∃X(X.КУ='СМ5'∧X.КМ=

=ОМС.КМ)

Результат

W

ЛФМ
2D — T 2×4 И—НЕ

Переменная **X** принимает последовательно значения кортежей отношения ОСУ.

5. Получить коды узлов, содержащих в своем составе микросхемы с мощностью потребления менее 90 мВт.

ПУСТЬ В ОТНОШЕНИИ ОМС КОРТЕЖ **X**

СОЗДАТЬ **W** (ОСУ.КУ): ∃ **X**(**X**.КМ) = ОСУ.КМ ∧ **X**.ПМ < 90)

Результат

W

КУ
ДШ 10 Сч 50 СМ 5

6. Получить логические функции узлов, содержащих в своем составе микросхемы с мощностью потребления менее 90 мВт.

ПУСТЬ В ОТНОШЕНИИ ОМС КОРТЕЖ **X**

ПУСТЬ В ОТНОШЕНИИ ОСУ КОРТЕЖ **Y**

СОЗДАТЬ **W** (ОУЗ.ЛФУ): ∃ (**Y**.КУ = ОУЗ.КУ) ∧

∧ (∃ **X**(**X**.КМ = **Y**.КМ ∧ **X**.ПС < 90)).

Результат

W

ЛФУ
Дешифратор Счетчик Сумматор

Здесь используются все три отношения: одно из них описывается в искомом отношении **W**, а два других определяются операторами принадлежности.

7. Получить коды узлов, не содержащих микросхемы с номером 155ЛА6.

ПУСТЬ В ОТНОШЕНИИ ОСУ КОРТЕЖ **X**

СОЗДАТЬ **W** (ОУЗ.КУ): ∀ **X**(**X**.КУ ≠ ОУЗ.КУ ∨ **X**.КМ ≠

≠ '135ЛА6')

Результат

W

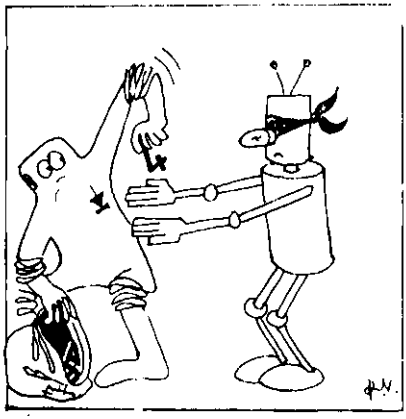
КУ
P2
Сч46
P3
СМ15
P1

В условии, устраивающем нас, при поиске нужной записи ОИЗ.КУ указывается перебор всех X, для которых либо названия узлов не совпадают, либо, если названия равны, номер микросхемы не совпадает с ЛА6.

Приведенные примеры показывают, что реляционное исчисление позволяет описать самые разнообразные виды искомых отношений. Однако отсутствие процедурности существенно затрудняет реализацию языков, основанных на реляционном исчислении, поскольку все процедуры поиска, построения и обработки отношений приходится строить полностью автоматически, скрытно от пользователя.

Определенная степень процедурности появляется в реляционной алгебре.

Реляционная алгебра. Алгебраический подход всегда предполагает наличие операндов и совокупности операций над ними. В реляционной алгебре в качестве операндов выступают отношения. Основными операциями, выполняемыми над отношениями, являются объединение, пересечение, вычитание, декартово произведение, проекция, ограничение, соединение, деление.



Операции объединения, пересечения, вычитания выполнимы лишь над парами совместимых отношений, т. е. таких отношений, у которых степени равны, а соответствующие атрибуты принадлежат одинаковым доменам. Пример совместимых отно-

Таблица 2.5

СОСТАВ1

Код узла (КУ)	Код типа микросхемы (КМ)	Количество (К)
Р2	155ЛА3	16
Р2	155ИР1	4
ДШ10	155ЛА3	2
Сч 50	155ТМ2	16
СМ5	155ТМ3	8

Таблица 2.6

СОСТАВ2

Код узла (КУ)	Код типа микросхемы (КМ)	Количество (К)
Сч 50	155ТМ2	16
Р3	155ИР1	8
СМ5	155ТМ2	8

Операцией объединения отношений **А** и **В** строится множество кортежей, принадлежащих либо отношению **А**, либо отношению **В**:

ПОЛУЧИТЬ **Р** ОБЪЕДИНЕНИЕМ **А** И **В**

Например,

ПОЛУЧИТЬ СОСТАВ3 ОБЪЕДИНЕНИЕМ СОСТАВ1 И СОСТАВ2

Результирующее отношение приведено в табл. 2.7.

Таблица 2.7

СОСТАВ3

Код узла (КУ)	Код типа микросхемы (КМ)	Количество (К)
Р2	155ЛА3	16
Р2	155ИР1	4
ДШ10	155ЛА3	2
Сч50	155ТМ2	16
СМ5	155ТМ2	8
Р3	155ИР1	8

Операцией пересечения отношений **А** и **В** строится множество кортежей, принадлежащих как отношению **А**, так и отношению **В**:

ПОЛУЧИТЬ **Р** ПЕРЕСЕЧЕНИЕМ **А** И **В**

Например,

ПОЛУЧИТЬ СОСТАВ4 ПЕРЕСЕЧЕНИЕМ СОСТАВ1 И СОСТАВ2

Результат операции представлен в табл. 2.8.

Таблица 2.8

СОСТАВ4

Код узла (КУ)	Код типа микросхемы (КМ)	Количество (К)
Сч50	155ТМ2	16
СМ5	155ТМ2	8

Операция вычитания отношений **А** и **В** строит множество кортежей, принадлежащих отношению **А**, но не принадлежащих отношению **В**:

ПОЛУЧИТЬ Р ВЫЧИТАНИЕМ В ИЗ А

Например,

ПОЛУЧИТЬ СОСТАВ5 ВЫЧИТАНИЕМ СОСТАВ2 ИЗ СОСТАВ1

Результат операции приведен в табл. 2.9.

Таблица 2.9

СОСТАВ5

Код узла (КУ)	Код типа микросхемы (КМ)	Количество (К)
Р2	155ЛА3	16
Р2	155ИР1	4
ДШ10	155ЛА3	2

Операция декартово произведение отношений **А** и **В** строит множество кортежей, полученных конкатенацией каждого кортежа из отношения **А** с каждым кортежем из отношения **В**.

■ Примечание. *Конкатенация* двух кортежей представляет собой один кортеж, составленный из двух исходных простым приписыванием второго вслед за первым.

ПОЛУЧИТЬ Р ПРОИЗВЕДЕНИЕМ А НА В

Например (табл. 2.10, 2.11, 2.12),

ПОЛУЧИТЬ ИМ ПРОИЗВЕДЕНИЕМ МС НА УЗ

Таблица 2.10

МС

Код типа микросхемы (КМ)	Логическая функция (ЛФМ)
155ТМ2	2D—Т
155ТВ1	УК—Т

Таблица 2.11

УЗ

Код узла (КУ)	Логическая функция (ЛФУ)
Р2	Регистр
Сч50	Счетчик

Таблица 2.12

НМ

Код типа микросхемы (КМ)	Логическая функция (ЛФМ)	Код узла (КУ)	Логическая функция (ЛФУ)
155ТМ2	2D—Т	Р2	Регистр
155ТМ2	2D—Т	Сч50	Счетчик
155ТВ1	УК—Т	Р2	Регистр
155ТВ2	УК—Т	Сч50	Счетчик

В этом примере применение операции произведения позволяет получить отношение НМ, содержащее все возможные варианты номенклатуры «микросхема—узел».

Операция проекция является унарной (определенной лишь для одного отношения). Для этой операции должны быть заданы отношение **А** и список атрибутов **С**. При этом сначала строится **В** из кортежей отношения **А**, содержащих лишь атрибуты из списка **С**. Затем в отношении **В** вычеркиваются повторяющиеся строки. Оставшееся отношение и будет результатом выполнения операции проекции:

ПОЛУЧИТЬ **Р** ПРОЕКЦИЮ **А** НА **С**

Например,

ПОЛУЧИТЬ КУСТ ПРОЕКЦИЮ ОУЗ НА **П**

В этом примере отношение ОУЗ взято из табл. 2.3. С помощью операции проекции получаем перечень устройств, содержащих узлы из отношения ОУЗ. Результат операции приведен в табл. 2.13.

Таблица 2.13

КУСТ

Применение (П)
АК10
АК20
АК30

Операция ограничение также является унарной и дает возможность построения «горизонтального» подмножества исходного отношения, кортежи которого удовлетворяют заданному логическому условию:

ПОЛУЧИТЬ Р ОГРАНИЧЕНИЕМ А ПО УСЛОВИЮ L

Условие L содержит простые сравнения, соединенные знаками логических операций.

Например,

ПОЛУЧИТЬ ОМС1 ОГРАНИЧЕНИЕМ ОМС ПО УСЛОВИЮ (ПМ<105)

Отношение ОМС взято из табл. 2.2. Результат операции представлен в табл. 2.14:

ПОЛУЧИТЬ НУ1 ОГРАНИЧЕНИЕМ ОУЗ ПО УСЛОВИЮ (П=АК20')∧(Р<20)

Таблица 2.14

ОМС1

Код типа микросхемы (КМ)	Логическая функция (АФМ)	Коэффициент разветвления (КР)	Мощность потребления (ПМ)	Задержка (ЗД)
155ЛА6	2×4 И—НЕ	30	86	25
155ТВ1	УК—Т	10	100	55

Отношение ОУЗ взято из табл. 2.3.

В результате операции получаем новое отношение НУ1, содержащее лишь те строки из отношения ОУЗ, для которых разрядность меньше 20, а применение соответствует устройству АК20 (табл. 2.15).

Операция соединения позволяет строить новое отношение посредством конкатенации кортежей двух исходных отношений. Однако конкатенация производится лишь при выполнении заданного логического условия. Операция соединения может быть представлена в виде

последовательности операций декартова произведения и ограничения:

ПОЛУЧИТЬ Р СОЕДИНЕНИЕМ А И В ПРИ УСЛОВИИ L

Таблица 2.15

НУ1

Код узла (КУ)	Логическая функция (ЛФУ)	Применение (П)	Разрядность (Р)
ДШ10	Дешифратор	АК20	10
Р1	Регистр	АК20	16

Тот же результат может быть получен посредством выполнения последовательности операций:

ПОЛУЧИТЬ Р1 ПРОИЗВЕДЕНИЕМ А НА В
ПОЛУЧИТЬ Р ОГРАНИЧЕНИЕМ Р1 ПО УСЛОВИЮ L

Например,

ПОЛУЧИТЬ СП СОЕДИНЕНИЕМ СМ1 И ОСУ ПРИ УСЛОВИИ (СМ1.КМ=ОСУ.КМ)

В этом примере отношение ОСУ взято из табл. 2.4, а отношение СМ1 и результат СП приведены в табл. 2.16, 2.17.

Таблица 2.16

СМ1

Код типа микросхемы (КМ)	Мощность потребления (ПМ)
155ЛА3	110
155ТВ1	100

Таблица 2.17

СП

Код типа микросхемы (КМ)	Мощность потребления (ПМ)	Код узла (КУ)	Код типа микросхемы (КМ)	Количество (К)
155ЛА3	110	Р2	155ЛА3	16
155ЛА3	110	ДШ10	155ЛА3	2
155ЛА3	110	Сч46	155ЛА3	4
155ЛА3	110	Р3	155ЛА3	8
155ЛА3	110	СМ15	155ЛА3	6
155ТВ1	100	Сч46	155ТВ1	10
155ТВ1	100	Р1	155ТВ1	16

Операция деление в определенном смысле обратна операции декартова произведения:

ПОЛУЧИТЬ Р ДЕЛЕНИЕМ А НА В

Для изложения общих принципов построения реляционных алгебраических языков достаточно пояснить операцию деления для частного случая, когда делимое есть отношение второй степени, а делитель — отношение первой степени. Пусть делимое есть отношение **А** с атрибутами A_1 и A_2, а делитель — отношение **В** с атрибутом B_1, определенным на том же домене, что и атрибут A_2. Тогда результатом **Р** операции деления отношения **А** на отношение **В** будет множество значений атрибута A_1 таких, что соответствующие им значения атрибута A_2 совпадают со всеми значениями атрибута B_1:

Например,

ПОЛУЧИТЬ НМС1 ДЕЛЕНИЕМ ТАБС НА СПУ1

Исходные отношения и результаты применения операции деления для делимого ТАБС и делителей СПУ1, СПУ2, СПУ3 представлены в табл. 2.18—2.24 соответственно в виде отношений НМС1, НМС2 и НМС3. Как видно из примеров, операция деления позволила выявить номера тех микросхем, которые входят во все узлы, перечисленные в списке делителя.

Таблица 2.18

ТАБС

Код типа микросхемы (КМ)	Код узла (КУ)
155ЛА3	Р2
155ЛА3	ДШ10
155ЛА3	С446
155ЛА3	Р3
155ЛА3	СМ15
155ЛА6	ДШ10
155ЛА6	Сч50
155ЛА6	Р2
155ТМ2	Сч50
155ТМ2	ДШ10
155ТВ1	Р2
155ТВ1	СМ15

Таблица 2.19

СПУ1

Код узла (КУ)
Р2

Таблица 2.20

СПУ2

Код узла (КУ)
Р2
ДШ10

Преобразования отношений, описанные с помощью реляционного исчисления, могут быть выполнены посредством операций реляционной алгебры. Так, условие примера 3 можно реализовать в виде последовательности операций

Таблица 2.21

СПУЗ

Код узла (КУ)
Р2
ДШ10
СМ15

Таблица 2.22

НМС1

Код типа микросхемы (КМ)
155ЛА3
155ЛА6
155ТВ1

Таблица 2.23

НМС2

Код типа микросхемы (КМ)
155ЛА3
155ЛА6

Таблица 2.24

НМС3

Код типа микросхемы (КМ)
155ЛА3

ПОЛУЧИТЬ Р1 ОГРАНИЧЕНИЕМ ОСУ ПО УСЛОВИЮ (КУ='СМ5')

ПОЛУЧИТЬ Р2 СОЕДИНЕНИЕМ ОМС и Р1 ПРИ УСЛОВИИ (ОМС.КМ=Р1.КМ)

ПОЛУЧИТЬ W ПРОЕКЦИЕЙ Р2 НА ЛФМ

§ 2.3. Иерархический и сетевой подходы

В реляционном исчислении и алгебре полностью отсутствуют указания на то, каким образом производить поиск необходимых данных. Оперирование отношениями (таблицами) предполагает просмотр всех записей. Когда БД велика, а этот случай типичен для САПР, то невозможно производить полный просмотр всех ее записей. Поэтому необходимо предварительное упорядочивание и объединение в группы записей по признакам поиска.

Если реляционное исчисление позволяет описать вид выходного документа, а реляционная алгебра — последовательность операций над отношениями, то для организации поиска нужных записей используются понятия ключа и связи.

Ключ — уникальное имя записи, в качестве которого может выступать как элемент какого-либо атрибута в записи — *простой ключ*, так и совокупность элементов нескольких атрибутов — *составной ключ*. С помощью ключа производится идентификация каждой конкретной

записи, а также упорядочение записей в файле. Например, в качестве ключа БД (см. табл. 2.2, 2.3) могут выступать код типа микросхемы в отношении ОМС и код узла в отношении ОУЗ. Упорядочение по ключу может быть либо прямым, либо выполнено с помощью хеш-функции.

Прямое упорядочение предполагает лексикографическое расположение записей: записи могут быть записаны либо в порядке увеличения значения ключа (при этом можно рассматривать простой ключ как некоторое число), либо в алфавитном порядке. Для второго случая ключи записей отношения ОМС (см. табл. 2.2) будут записаны в следующем порядке: 155ИР1, 155ЛА3, 155ЛА6, 155ТВ1, 155ТМ2.

Хеш-функция производит пересчет ключа в адрес записи на файле. Эта операция выполняется СУБД всякий раз при поиске нужной записи по ключу. Связи позволяют осуществить группирование записей в множества, а также указывать взаимоотношения между этими множествами. На практике связь реализуется, как правило, в виде указателя.

Так, если необходимо указать связь между записью 155ТМ2 в отношении ОМС с записью Сч50 отношения ОУЗ, то в состав записи 155ТМ2 необходимо включить указатель на запись Сч50. В случае присутствия и обратной связи запись Сч50 должна также содержать указатель на запись 155ТМ2. Однотипные записи группируются в *сегменты (типы записей)*.

Графически при описании базы данных разным типам связей соответствуют обозначения в виде различных стрелок: «→» — связь с одной записью; «→→» — связь с несколькими записями. Взаимные связи имеют следующие названия: «←→» — «один к одному», «←→→» — «один ко многим», «←→→» — «многие ко многим». На рис. 2.2 записи изображены в виде прямоугольников, а сегменты представлены кружками. Например, обозначение, приведенное на рис. 2.2, *в*, следует понимать так: каждая запись сегмента А связана с некоторой группой записей сегмента В, а каждая запись из сегмента В связана лишь с одной записью сегмента А.

С помощью указанных обозначений строится *граф логической схемы*, вершины которого — сегменты, а дуги — обозначения типов связей между сегментами. На рис. 2.3 представлен пример логической схемы некоторой БД.

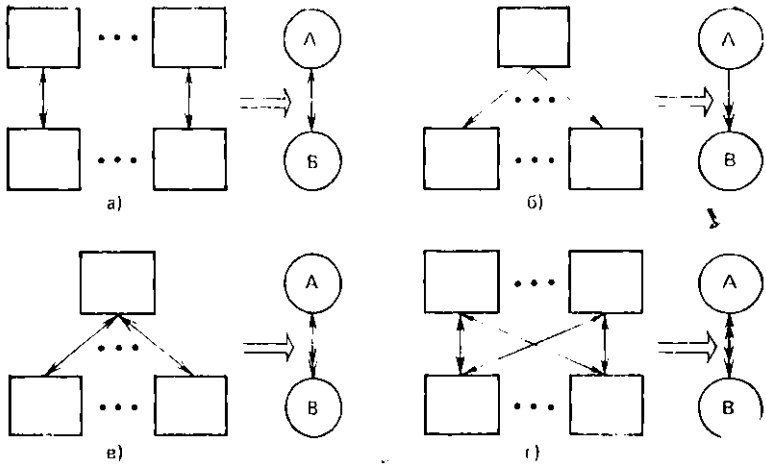


Рис. 2.2. Типы и обозначения связей:

а — «один — к одному»; *б*, *в* — «один — ко многим»; *г* — «многие — ко многим»

Иерархический подход. Иерархическая БД имеет граф логической схемы в виде дерева, а тип связей соответствует рис. 2.2, *б*. Пример логической схемы иерархической БД приведен на рис. 2.4. В иерархической БД связи направлены только от верхних сегментов к нижним, обратные указатели отсутствуют. Это объясняется принципиальным свойством иерархического представления данных: каждая запись приобретает смысл лишь тогда, когда она рассматривается в своем контексте, т. е. любая запись не может существовать без предшествующей ей записи по иерархии. При поиске в иерархической БД необходимо указывать значение ключа на каждом уровне иерархии. Так, для доступа к записи из множества **G** (рис. 2.4) должны быть последовательно указаны ключи записей из множеств **A**, **C** и **G**.

Реляционная БД всегда может быть преобразована в иерархическую. Однако конкретный вид логической схемы зависит от типа запросов. В табл. 2.2 ... 2.4 приве-

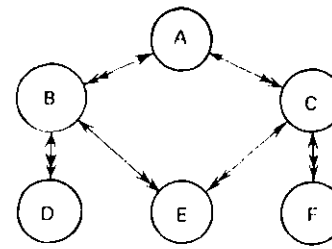
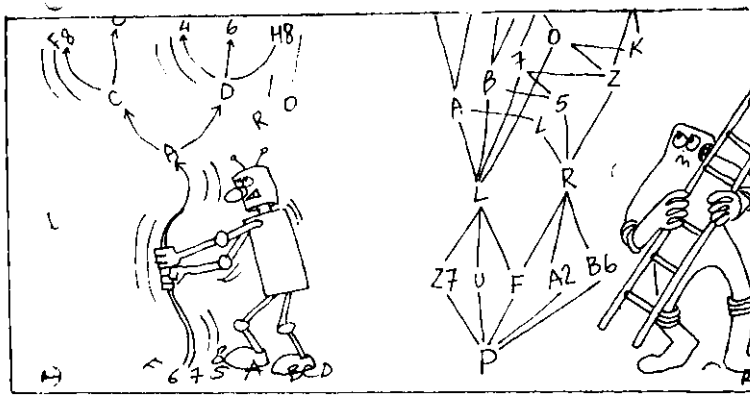


Рис. 2.3. Пример логической схемы базы данных



ден пример фрагмента реляционной БД, содержащего запись о характеристиках микросхем и узлов. Дополним базу данных, представленную в табл. 2.2...2.4, также записями о характеристиках устройств, состоящих из совокупности узлов, образовав сегмент *АК*. Тогда для запросов типа «Найти узлы для заданных устройств», «найти микросхемы для заданных узлов» можно построить иерархическую БД (рис. 2.5). Поэкземплярная реализация связей первых двух сегментов приведена на рис. 2.6. Каждая запись сегмента *АК* связана с группой записей сегмента *ОУЗ*. В данной конкретной реализации запись из *АК* ссылается на первую в группе запись из ОУЗ, сами же записи сегмента ОУЗ объединены с помощью отдельных указателей в цепные списки. Очевидно, что поиск в такой БД легко осуществляется по составному ключу. Например, составной ключ АК20. Р3 позволяет найти все сведения об узле Р3, входящем в состав устройства АК20.

При организации связей между уровнями ОУЗ—ОМС возникает проблема избыточности, заключающаяся в необходимости повторения записей микросхем. Так, микросхема 155ЛА3 должна быть указана во всех записях сегмента ОМС, связанных с записями ДШ10, Сч46, Р3 и СМ15 сегмента ОСУ. В конкретной реализации возможно избежать многократного повторения описания микросхемы 155ЛА3 введением указателя на запись 155ЛА3, хранящуюся отдельно. Однако значение указателя на запись 155ЛА3 должно быть размножено. Это не вызывает существенных затрат памяти, однако усложняет решение проблемы целостности базы данных.

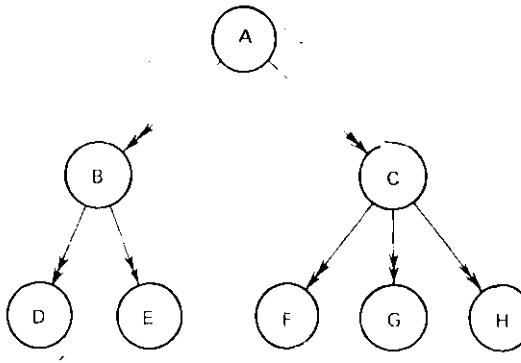


Рис. 2.4. Пример логической схемы иерархической базы данных

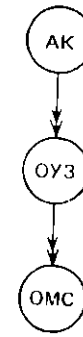


Рис. 2.5. Логическая схема для заданного типа запроса

Трудности в использовании иерархической базы данных возникают при изменении типа запроса. Так, если в рассматриваемом примере БД появится запрос «Найти узлы, включающие заданную микросхему», то пропадают преимущества предыдущего иерархического упорядочения. Действительно, для выполнения этого запроса необходимо просмотреть все записи узлов и все связанные с ними записи микросхем. Для реализации такого запроса было бы целесообразно переупорядочить БД и построить новую иерархию: от записей о микросхемах к записям об узлах.

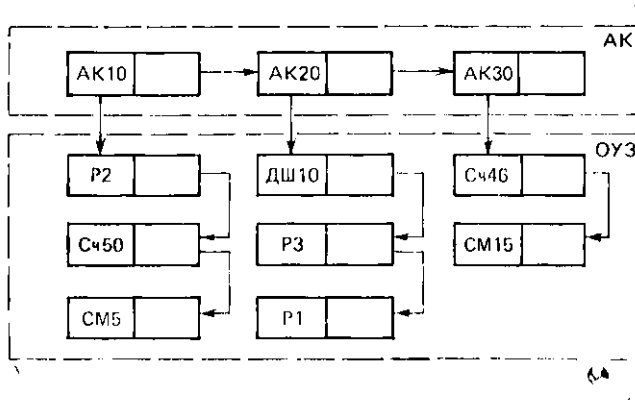


Рис. 2.6. Схема реализации связей АК—ОУЗ

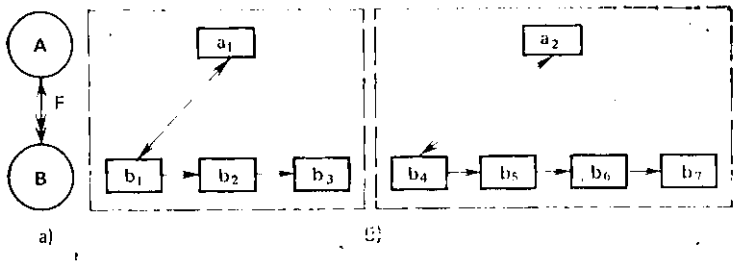


Рис. 2.7. Пример набора (*а*) и экземпляров набора (*б*):

A, B — сегменты; a_1, a_2 — записи сегмента A; b_1, b_2, ..., b_7 — записи сегмента B; F — имя набора

Сетевой подход. Необходимость в организации различного упорядочения записей в БД с целью удовлетворения разных типов запросов привела к разработке *сетевых баз данных*. В сетевой модели данных в принципе разрешены любые группирования записей и организация произвольных связей между ними. Однако на практике целесообразно введение некоторых ограничений. Рассмотрим представление сетевых моделей данных в соответствии с концепцией КОДАСИЛ.

Набор — основная конструкция сетевых моделей, представляющая собой поименное двухуровневое дерево. С помощью двухуровневых деревьев могут быть построены многоуровневые деревья и большинство сетевых структур. Если рассматривать логическую схему, то набор может интерпретироваться как имя связи между двумя сегментами записей типа «один ко многим». *Экземпляр набора* — конкретная реализация такой связи. На рис. 2.7 приведен пример набора и его двух экземпля-

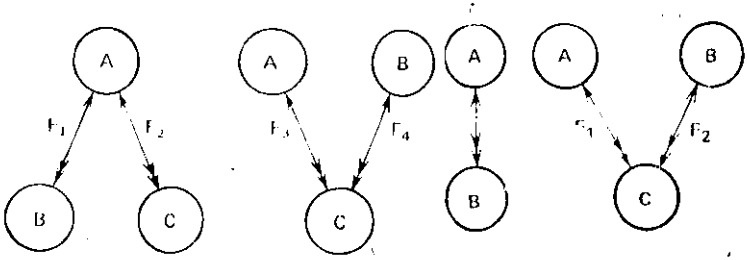


Рис. 2.8. Допустимые варианты построения наборов

Рис. 2.9. Представление отношения «многие — ко многим»

ров. Каждый экземпляр набора содержит одного *владельца* и нескольких *членов набора*. Например, на рис. 2.7 владельцем 1-го экземпляра набора является запись a_1, а членами набора — записи b_1, ..., b_3.

Допустимо, чтобы записи одного сегмента были владельцами нескольких наборов, и, наоборот, записи некоторого набора могут быть членами различных наборов. На рис. 2.8 показаны соответствующие конструкции логической схемы.

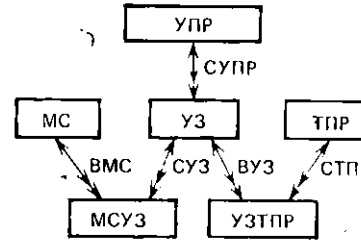


Рис. 2.10. Пример сетевой логической схемы:

УПР — сегмент записей об уникальных проектных решениях; *ТПР* — сегмент записей о типовых проектных решениях; *МС* — сегмент записей о микросхемах; *УЗ* — сегмент записей об узлах; *МСУЗ* — сегмент связи между записями сегментов *МС* и *УЗ*; *УЗТПР* — сегмент связи между записями сегментов *УЗ* и *ТПР*; *СУПР, ВМС, СУЗ, ВУЗ, СТПР* — имена соответствующих наборов

В сетевых СУБД не разрешены связи в логической схеме типа «многие — ко многим». Однако с помощью вышеуказанных свойств набора можно описать такую связь введением дополнительного сегмента записей, задающих взаимоотношение между исходными сегментами, и определить на них два набора так, как это показано на рис. 2.9. Сегмент **С** содержит записи в форме адресных таблиц (аналогично отношению ОСУ). Записи сегмента **С** входят в различные наборы для сегментов **А** и **В**. Если необходимо найти записи в **В**, связанные с данной записью из **А**, то с помощью набора F_1 находят указатели на записи в сегменте **С**, а затем по набору F_2 находят искомые записи в **В**. Если же требуется выполнить обратный поиск из **В** в **А**, то необходимо воспользоваться сначала набором F_2, а затем набором F_1.

Пример сетевой логической схемы приведен на рис. 2.10. Из него видно, что поиск необходимых записей можно производить начиная с сегментов *УПР, МС, УЗ, ТПР*.

§ 2.4. Инвертированные базы данных

Ознакомление с различными моделями данных показало, что поиск необходимой информации требует значительных затрат времени даже

для иерархических СУБД, особенно при больших объемах баз данных. Однако если удается выделить совокупность признаков, по которым формируется запрос, то можно предложить способ организации баз данных, значительно сокращающий время поиска затребованной информации. В основе такого способа лежит понятие инвертированного списка.

Инвертированный список представляет собой таблицу, в левом столбце которой помещены значения данного признака, а в правом — указатели на соответствующие записи. Допустим, в примере базы данных, приведенной в табл. 2.2...2.4, в записях об узлах требуется выявить узлы, имеющие одинаковую разрядность. Тогда все записи об узлах могут быть скомпонованы плотно в памяти друг за другом и иметь порядковые номера, соответствующие их последовательности в таблице ОУЗ. Наряду с этим создается инвертированный список в таком виде:

Разрядность	8	10	16	32
Список указателей . . .	У7	У2, У4	У1, У6, У8	У3, У5

Обозначение УN представляет собой изображение указателя на N-ю запись. Таким образом, инвертированный список как бы заранее хранит ответ на запрос «Назвать характеристики узлов, имеющих заданную разрядность». В примере разрядность выступает в качестве признака в запросе. Конечно, можно выделить и другие признаки, например в каком устройстве применяется узел, каков тип узла. Для каждого признака должен быть построен свой инвертированный список.

Наличие нескольких инвертированных списков позволяет строить запрос в виде некоторой логической функции от совокупности признаков. Наиболее распространен случай построения запроса, когда используются операции дизъюнкции и конъюнкции.

Для поиска необходимой записи должны быть выполнены следующие действия:

1) выделить в запросе признаки поиска;

2) определить вид логической функции между запрашиваемыми признаками;

3) для каждого признака в нужном инвертированном списке найти множество указателей на записи;

4) в соответствии с видом логической функции произвести операции над множествами указателей.

Соответствие операций над множествами указателей виду логической функции должно быть принято из разделов по реляционной модели данных. Так, логической функции дизъюнкции ставится в соответствие операция объединения множеств, конъюнкции — операция пересечения множеств, отрицанию — операция дополнения и т. д. Для запроса «Какие узлы имеют разрядность 16 и используются в устройстве АК10?» в ходе поиска будут построены следующие результаты:

1) в запросе два признака: разрядность и объект применения;

2) вид логической функции в запросе — конъюнкция;

3) для признака «разрядность» в инвертированном списке значению 16 будет соответствовать множество указателей {У1, У6, У8}; признаку «применение» в соответствующем инвертированном списке для значения АК10 — множество указателей {У1, У3, У6};

4) над найденными двумя множествами указателей производится операция пересечения. В итоге получается искомый результат: {У1, У6}. По этим номерам в файле записей будут найдены записи об узлах Р1 и СМ5.

Если для всех записей, хранящихся в базе данных, созданы инвертированные списки для возможных вариантов запросов, то такая база данных называется *инвертированной*. Инвертированные БД широко используются в информационно-поисковых системах (ИПС), предназначенных в основном для хранения текстовых документов. Признаки, по которым отыскивается необходимый документ в ИПС, называются *дескрипторами*. Для каждого дескриптора в ИПС строится ивертированный список, содержащий все возможные значения дескриптора и соответствующие им множества указателей на документы. Запрос в ИПС имеет вид логического высказывания относительно значений дескрипторов и их взаимосвязи. Так, если в качестве документов выступают ГОСТы по проектированию, то дескрипторами могут быть такие понятия, как год издания, объект проектирования, вид обеспечения, объем печатного материала и др. Запрос мог бы выглядеть, например, так: «Найти ГОСТы, изданные не позднее 1981 г., посвященные проектированию АСУ, для информационного и программного обеспечений».

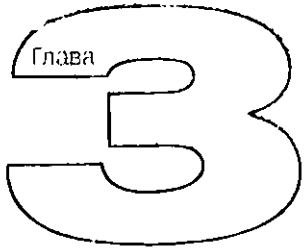
■ Примечание. Вид логической функции, соответствующий этому запросу, предоставляем определить самому читателю на основе предыдущих примеров.

Проектная документация, как уже указывалось, слабоструктурирована. Однако любой проектный документ может быть охарактеризован некоторой совокупностью признаков. Так, техническое задание на проектирование абонентской сети на базе персональных ЭВМ описывается дескрипторами: техзадание, абонентская сеть, проектная документация, персональные ЭВМ. Для хранения текущих проектных документов необходимо заранее заготовить совокупность дескрипторов, которые наряду с содержанием отражали бы и поэтапность составления соответствующих документов. Например, могут быть указаны этапы, стадии, шаги процесса проектирования, а также номера вариантов.

КРАТКИЕ ВЫВОДЫ

Хранение и преобразование данных в современных системах обработки информации выполняются с помощью банков данных (БНД), представляющих собой совокупность базы данных (БД) и системы управления базами данных (СУБД). Одно из важнейших преимуществ БНД — отделение описания данных для пользователя (логический уровень описания данных) от системы физического хранения (физический уровень описания данных), что обеспечивает независимость представления данных в прикладных программах от типов ЗУ и способов их физической организации. В основе построения БД лежит понятие модели данных. Различают реляционную, иерархическую и сетевую модели данных. Для каждой модели определены операции над данными, в совокупности образующие подход. В реляционном подходе выделяют операции, построенные на основе реляционного исчисления и реляционной алгебры. Пользовательские непроцедурные языки высокого уровня строятся на основе реляционного исчисления. Реляционная алгебра позволяет построить более эффективные пользовательские языки за счет снижения степени непроцедурности и явного указания последовательности преобразований сегментов записей. Иерархический подход позволяет существенно сократить время поиска данных в БД, однако он применим лишь к запросам фиксированного вида. Сетевые СУБД обладают преимуществами иерархических и в то же время значительно расширяют допустимые типы запросов на поиск необходимых данных. Любая БД может быть представлена любой моделью данных, однако иерархическая модель, как правило, наиболее избыточна.

Быстрый и эффективный поиск данных при заданных поисковых признаках осуществляется в инвертированных базах данных.



ОРГАНИЗАЦИЯ ИНФОРМАЦИОННОГО ОБЕСПЕЧЕНИЯ СИСТЕМ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ

§ 3.1. Организация информационного фонда

Проектирование — связанная совокупность процессов преобразования одних данных в другие. При этом данные, являющиеся *результатом* одного процесса преобразования, могут быть *исходными* для другого процесса (*промежуточные данные*).

Система автоматизированного проектирования — сложная и многокомпонентная система, процессы преобразования данных в которой разнообразны. Это приводит к различным трактовкам термина «данные» в САПР. Так, для управляющего монитора САПР в состав данных входит совокупность программных модулей, которые реализуют функции проектирования; для системы диалогового обеспечения САПР данными является множество взаимосвязанных информационных и управляющих кадров экрана дисплея; для функциональных программных модулей к данным относится совокупность исходных и результирующих чисел, необходимых для выполнения конкретной проектной процедуры; пользователю САПР в качестве данных требуется иметь в своем распоряжении исходную проектную документацию, справочные данные, типовые проектные решения и т. д.

Совокупность данных, используемых всеми компонентами САПР, составляет *информационный фонд* САПР.

Назначение информационного обеспечения (ИО) САПР — реализация информационных потребностей всех составных компонентов САПР. Основная функция ИО САПР — ведение информационного фонда, т. е. обеспечение создания, поддержки и организации доступа к данным. Таким образом, *информационное обеспечение* САПР есть совокупность информационного фонда и средств его ведения.

Состав информационного фонда САПР. Программные модули хранятся в виде символических и объектных текстов. Как правило, эти данные мало изменяются в течение жизненного цикла САПР, имеют фиксированные размеры и появляются на этапе создания информационного фонда САПР. Потребителями этих данных являются мониторы различных подсистем САПР.

Исходные и результирующие данные необходимы при выполнении программных модулей в процессе преобразования. Эти данные часто меняются в процессе проектирования, однако их тип постоянен и полностью определяется соответствующим программным модулем. При организации промежуточных данных возможны конфликтные ситуации в процессе согласования между собой данных различных типов.

Нормативно-справочная проектная документация (НСПД) включает в себя справочные данные о материалах, элементах схем, унифицированных узлах и конструкциях. Эти данные, как правило, хорошо структурированы и могут быть отнесены к фактографическим. Кроме того, к НСПД относятся государственные и отраслевые стандарты, руководящие материалы и указания, типовые проектные решения, регламентирующие документы (слабоструктурированные документальные данные).

Содержание экранов дисплеев представляет собой связанную совокупность данных, задающих форму кадра и, следовательно, позволяющих отобразить на экран дисплея информацию с целью организации диалогового взаимодействия в ходе проектирования. Обычно эти данные не изменяются в течение жизненного цикла САПР, имеют фиксированный размер и по своим характеристикам занимают промежуточное место между программными модулями и исходными данными; используются диалоговыми системами САПР в процессе реализации заданого графа диалога.

Текущая проектная документация отражает состояние и ход выполнения проекта. Как правило, эти данные слабоструктурированы, часто изменяются в процессе проектирования и представляются в форме текстовых документов.

● **Пример проектных документов.** При проектировании пакета прикладных программ должны быть построены документы: 1) пояснительная записка; 2) общее описание; 3) руководство программиста; 4) описание языка; 5) описание программ; 6) порядок и методика испытаний.

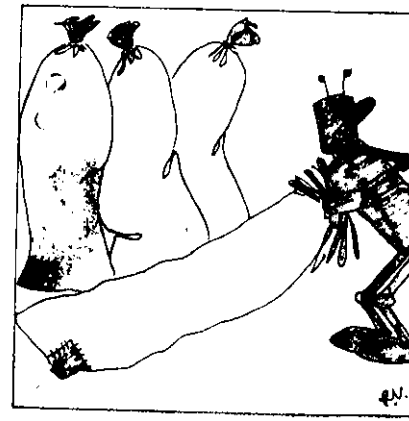
В свою очередь, каждый из указанных документов также состоит из совокупности документов. Так, документ «Порядок и методика испытаний» содержит следующие документы: 1) объект испытаний; 2) цель испытаний; 3) технические требования; 4) процесс проведения испытаний; 5) методика испытаний. Документ «Технические требования» содержит документы: 1) технические характеристики программ; 2) условия эксплуатации; 3) требования к информационной и программной совместимости.

Как видно из примера, совокупность проектных документов может быть представлена иерархической структурой.

Способы ведения информационного фонда САПР. Проблему организации и ведения информационного фонда можно рассматривать в содержательном и организационном аспектах.

С точки зрения содержания интерес представляют причина возникновения необходимости в конкретном данном, получение его значения, достоверность, альтернативы, проанализированные прежде, чем возникло это данное. Содержательный аспект информации, используемой при проектировании, полностью определяется принятой методикой проектирования, разработанными алгоритмами решения частных задач. Эти вопросы носят общеметодологический характер.

С организационной точки зрения важно сформулировать принципы и определить средства ведения информационного фонда, структурирования данных, выбрать способы управления массивами данных. Различают следующие способы ведения информационного фонда САПР: 1) использование файловой системы; 2) построение библиотек; 3) использование банков данных; 4) создание информационных программ адаптеров.



Способы 1 и 2. Использование файловой системы и построение библиотек. Эти способы широко распространены в организации информационного обеспечения вычислитель-

ных систем, поскольку поддерживаются средствами ОС. В приложении к САПР они применяются при хранении программных модулей в символических и объектных кодах, диалоговых сценариев поддержки процесса проектирования, начального ввода крупных массивов исходных данных, хранения текстовых документов. Однако для обеспечения быстрого доступа к справочным данным, хранения меняющихся данных, ведения текущей проектной документации, поиска необходимых текстовых документов, организации взаимодействия между разноязыковыми модулями эти способы малопригодны.

Способ 3. Использование банков данных. Этот способ позволяет: 1) централизовать информационный фонд САПР; 2) произвести структурирование данных в виде, удобном для проектировщика; 3) обеспечить поиск информативно-справочной и проектной документации; 4) упростить организацию межмодульного интерфейса путем унификации промежуточных данных.

Способ 4. Создание информационных программ-адаптеров. Проблема организации межмодульного интерфейса породила специализированные системы и программные технологии, например систему АПРОП, ориентированную на построение крупных программных комплексов из готовых модулей. В этой системе промежуточные данные унифицируются с помощью единого процессора и построения специализированных межмодульных информационных программ-адаптеров.

Поскольку файловые и библиотечные системы описаны во второй книге данной серии, в последующих параграфах настоящей главы более подробно излагаются вопросы использования и конкретного применения банков данных в САПР, анализируется проблема межмодульного интерфейса и излагаются рекомендации по ее решению.

§ 3.2. Применение конкретных СУБД в САПР

К настоящему времени разработано много СУБД для поддержания различных структур данных. Например, СУБД «СЕДАН», «ОКА», «СЕТОР», «БАНК-ОС», «ДИСОД», «ИНЕС», «СЕТЬ» применяют в системах информационно-запросных, АСУ информационно-поисковых. Однако возможно их использование и для обеспечения информационных потребностей САПР. В первую очередь речь идет об отображении нормативно-справочной и другой фактографической информации.

Дадим краткий обзор функциональных возможностей и эксплуатационных характеристик наиболее перспективных СУБД.

Система управления базой данных «ИНЕС» ориентирована на поддержание иерархических структур данных. На физическом уровне используется метод доступа, программно имитирующий механизм виртуальной памяти. При этом данные хранятся в блоках памяти и лексикографически упорядочены, а разным сегментам в логической схеме соответствуют различные блоки. Таким образом, блоки также организуются в иерархическую структуру. Особенность СУБД «ИНЕС» — наличие непроцедурного языка манипулирования данными — *языка запросов*.

Для обеспечения взаимодействия конечного пользователя в состав СУБД «ИНЕС» включен язык описания сценария диалога, основным оператором которого является CALL, позволяющий вывести на экран и получить ответное сообщение (пользователь при этом может указать *директивы*, осуществляющие передачу сообщения на начало диалога, окончания текущего шага работы, печать содержимого экрана дисплея и т. д.). Для обеспечения одновременной работы нескольких пользователей за терминалами СУБД «ИНЕС» имеет в своем составе монитор.

Система управления базой данных «ОКА», так же как и СУБД «ИНЕС», принадлежит к системам иерархического типа. В своем составе она имеет средства для задания связей между иерархическими структурами, что дает возможность описывать сетевые структуры ограниченного вида.

В зависимости от типа базы данных в СУБД «ОКА» предусмотрены различные методы доступа.

Иерархический последовательный (HSAM) и *иерархический индексно-последовательный* (HISAM) обеспечивают экономию внешней памяти, ориентированы на статические малоизменяемые базы данных, использующие лишь операции выборки. Это в основном справочные данные, характеристики устройств, программ, типовых проектных решений.

Иерархический прямой (HDAM) и *иерархический индексно-прямой* (HIDAM) *методы доступа* обеспечивают быстрое время реакции системы на запрос, поддержание динамической (быстроизменяющейся) БД. К такой БД могут относиться характеристики текущих вариантов проекта, промежуточные результаты, данные для обмена между программными системами.

В СУБД «ОКА» имеются средства для организации диалогового взаимодействия с конечным пользователем — *запросная система*. Эта система использует лексику естественного языка и позволяет описывать поисковый запрос к СУБД, выводить необходимые данные, получать сведения о БД или ее части и т. д. Для осуществления взаимодействия с удаленными терминалами используется телемонитор «Кама».

Из иерархических СУБД «ОКА» в настоящее время наиболее развитая система хранения и обработки данных. Она обеспечена сервисными программами, средствами взаимодействия с пользователем. Объем ОП, занимаемой резидентной частью, составляет 300—350 К байт.

К сетевым СУБД относятся системы «СЕТЬ», «БАНК-ОС», «СЕДАН», «СЕТОР», «ДИСОД» и др. Эти СУБД поддерживают сетевые структуры широкой конфигурации, однако имеют различные функциональные и сервисные возможности.

Систему управления базой данных «СЕДАН» и ее дальнейшее развитие «СЕТОР» отличают простота организации,

незначительный объем резидентной части (10—30 К байт), широкое использование файловой системы.

■ Примечание. Более подробные сведения о СУБД «СЕТОР» можно почерпнуть в § 3.3.

Система управления базой данных «СЕТЬ» построена на концепциях комитета «КОДАСИЛ». В этой системе доступ возможен к любой из вершин логической схемы по значению ключа записи с помощью процедур хеширования. Записи, относящиеся к какой-либо вершине логической схемы, могут быть рассортированы в зависимости от числа поисковых ключей. Для *прямого поиска* записей по значению поискового ключа предусмотрены средства ведения индексов, которые целесообразно использовать для разделения больших массивов записей на отдельные группы с последовательным просмотром. Язык манипулирования данными включается непосредственно в текст включающего языка. Препроцессор ЯМД заменяет операторы ЯМД на соответствующие операторы включающего языка.

Оперативная обработка запросов обеспечивается с помощью *языка запросов*, ориентированного в основном на специалиста в области обработки данных. Этот язык позволяет осуществить выборку физическую, по ключу, найти текущую запись в наборе, определить запись владельца набора, задать условия поиска необходимой записи в наборе с помощью логических операций И, ИЛИ, НЕ над результатами сравнений значений для элементов данных в записях. Для пользователей-непрограммистов язык запросов позволяет обратиться к каталогизированным процедурам с варьированием значений их параметров.

Система оперативного доступа используется либо в среде специализированного телемонитора «СЕТЬ», либо универсального телемонитора «КАМА».

Система управления базой данных «ДИСОД» — одна из наиболее развитых отечественных СУБД. Это многофункциональная система хранения и обработки данных, рассчитанная на широкий класс практических применений в области решения информационных задач. Однотипные записи базы данных СУБД «ДИСОД» организованы в файлы, между которыми могут быть установлены связи, позволяющие создавать как иерархические, так и сетевые структуры. Возможность установления межфайловых связей основана на равенстве значений пары атрибутов, объявленных как *атрибуты связи* в записях каждого из связываемых файлов. Эти связи могут быть либо установлены, либо ликвидированы без перезагрузки базы данных, это обеспечивает хранение *динамических структур данных*.

Кроме механизма межфайловых связей некоторые атрибуты записи в логической схеме могут быть определены как *поисковые атрибуты*. По значению этих атрибутов формируются инвертированные списки, значительно ускоряющие процесс поиска затребованных записей. Все файлы в СУБД «ДИСОД» доступны для вхождения и поиска небходимой информации.

Язык манипулирования данными выполнен с помощью оператора CALL по правилам включающего языка (языка ассемблера, КОБОЛ, ФОРТРАН, ПЛ/1), имя точки входа в систему — DISOD. Кроме ЯМД, СУБД «ДИСОД» располагает следующими языками общения с БД для пользователя-непрограммиста:

язык диалоговой подготовки справок (содержит команды формирования справок и команды управления диалогом, позволяет пользователю в диалоговом режиме вводить запросы и получать на терминале ответы в виде сообщений или таблиц с результатами обработки данных);

язык диалоговой обработки данных (дает возможность формировать различные виды отчетов на основе имеющейся в БД информации. Программы формирования отчета подготавливаются администратором и содержат сценарии диалога в виде последовательности информационных экранов дисплея).

Кроме перечисленных языков в распоряжении администратора БНД находятся дополнительные языковые средства, позволяющие управлять функционированием системы, регламентировать действия пользователя, вести информационный фонд с экрана дисплея и др.

Все диалоговые компоненты СУБД «ДИСОД» функционируют под управлением телемонитора «КАМА». Она требует больших объемов ОП [минимальный объем ОП — 450 К байт; дополнительно должен выделяться объем памяти при работе с диалоговыми средствами (до 350 К байт)].

СУБД «ДИСОД» — универсальная система для хранения и обработки всех видов нормативно-справочной информации и проектных документов в крупных САПР.

§ 3.3. СУБД «СЕТОР»

В качестве примера использования БНД для хранения фактографических данных в САПР рассмотрим СУБД «СЕТОР».

Из важных прикладных особенностей СУБД «СЕТОР» следует отметить малый объем занимаемой ОП и возможность работы в режиме мультидоступа к единой интегрированной БД. Доступ к данным осуществляется через прикладные программы, написанные на одном из стандартных языков программирования (ПЛ/1, РПГ, КОБОЛ, ФОРТРАН, ассемблера), расширенном языком манипулирования данных. Выполнение запроса к БД требует вызова одной из прикладных программ, причем вызов возможен как через общий входной поток операционной системы (вариант пакетной обработки), так и с использованием телемонитора (расширение СУБД «СЕТОР» средствами телеобработки).

Поставляются версии СУБД, предназначенные для установки на ЭВМ серии ЕС, на мини-ЭВМ типа СМ-4, СМ 1420, а также на микро-ЭВМ семейства «Электроника», причем возможен перенос БД под управлением СУБД «СЕТОР» с микро- и мини-ЭВМ на основные модели машин серии ЕС ЭВМ. Поэтому СУБД «СЕТОР» использовать удобно в САПР различного назначения.

СУБД «СЕТОР» не изолированный пакет прикладных программ (ППП) и, следовательно, может предо-

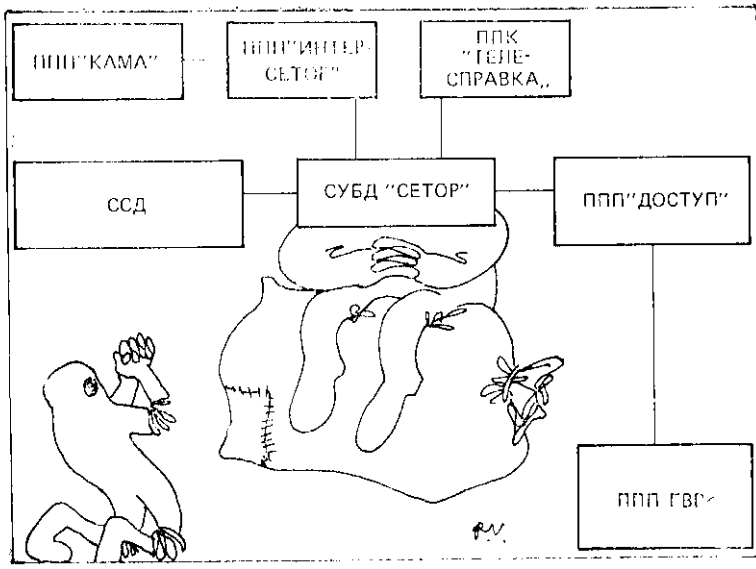


Рис. 3.1.

ставлять пользователю дополнительные возможности в результате работы совместно с другими ППП. Возможное окружение СУБД «СЕТОР» представлено на рис. 3.1, где *ССД* — словарь-справочник данных; *ГВВ* — генератор ввода — вывода.

Пакет прикладных программ «Генератор ввода — вывода» через ППП «Доступ» обеспечивает автоматизированный ввод информации в БД с указанием, каким образом вводимая с документа информация распределяется в структуре БД.

Пакет прикладных программ «ИНТЕРСЕТОР» совместно с ППП «Кама» позволяет работать с БД в режиме теледоступа.

Пакет прикладных программ «Телесправка» предназначен для взаимодействия с БД пользователей, практически не знакомых с программированием. Запросы в БД состоят из фраз, построенных на ограниченном естественном языке, и могут вводиться как непосредственно с перфоносителей, так и через экран дисплея с использованием ППП «Кама» и «ИНТЕРСЕТОР». Результат запроса может выдаваться либо на экран дисплея, либо на печатающее устройство.

К средствам обеспечения целостности данных относится «Словарь-справочник данных», являющийся дополнением к встроенным в СУБД «СЕТОР» средствам поддержки целостности БД, таким, как программа ведения системного журнала и программа восстановления БД.

Система управления базой данных «СЕТОР» обеспечивает независимость данных от программ не только на уровне записей, но и на уровне элементов. Это означает, что без изменения имеющихся ПИП можно модифицировать структуру БД вплоть до отдельных элементов записей. Все это позволяет в определенных пределах менять логическую схему БД.

Важная особенность СУБД «СЕТОР» — возможность работать в многозадачном режиме. Этот режим возможен как при пакетной обработке, так и при использовании средств теледоступа и означает, что с одними и теми же данными возможна работа нескольких программ. Возникающая при этом проблема синхронизации решается в СУБД «СЕТОР» следующим образом: на время обновления записи данной программой блокируется доступ из любой другой проблемной программы. По окончании обновления блокировка снимается.

Возможны следующие основные режимы работы СУБД «СЕТОР»:

1) *однозадачный* [в разделе памяти каждой программы находятся копия ядра СУБД и блок описания схемы. Синхронизация при обращении к БД из разных разделов в этом случае не обеспечивается (рис. 3.2)];

2) *мультизадачный* в одном разделе [все прикладные программы являются для СУБД подпрограммами и находятся в одном разделе ОП; в этом же разделе помещаются копия ядра системы и блока описания схемы (рис. 3.3)];

3) *мультизадачный* в разных разделах [ядро системы и блок описания схемы находятся в отдельном разделе, каждая прикладная программа вместе со своей подсхемой также находится в отдельном разделе ОП (рис. 3.4)].

Система управления базой данных «СЕТОР» ориентирована на поддержку сетевых структур данных, создаваемых из двух типов записей: 1) основных; 2) зависимых. Каждой записи основного типа может соответствовать совокупность записей зависимого типа, а каждая запись зависимого типа может находиться в отношении подчинения к разным записям основного типа. По данным правилам можно строить сетевые структуры любой

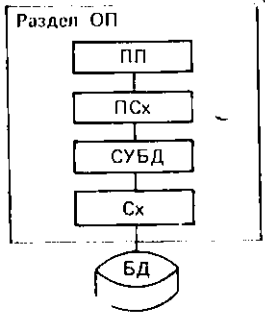


Рис. 3.2. Схема связи СУБД «СЕТОР» с пользователем при однозадачном режиме:

ПП — прикладная программа; *ПСх* — подсхема; *Сх* — схема

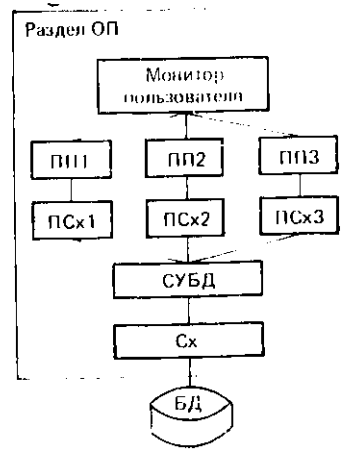


Рис. 3.3. Схема связи СУБД «СЕТОР» с пользователем при мультизадачном режиме в одном разделе ОП:

ПП1, ПП2, ПП3 — прикладные программы; *ПСх1, ПСх2, ПСх3* — подсхемы прикладных программ; *Сх* — схема *БД*

сложности, но при этом нужно учитывать следующие ограничения:

1) доступ к записи возможен только через данные основного типа;

2) данные зависимого типа должны быть связаны хотя бы с одним данным основным;

3) данные одинакового типа не могут быть связаны между собой непосредственно.

Так как сетевые структуры более универсальны, чем иерархические, то иерархические структуры специально СУБД «СЕТОР» не поддерживаются, а имитируются с помощью сетевых структур.

Языковые средства СУБД «СЕТОР» включают в себя язык описания данных (ЯОД) и язык манипулирования данными (ЯМД).

Язык описания данных СУБД «СЕТОР» представляет собой совокупность директив, предназначенных для описания файлов, записей, полей, определения ключей и связей между файлами, задания объемов файлов, определения типов устройств ввода — вывода, размеров буферов в ОП для обмена с БД.

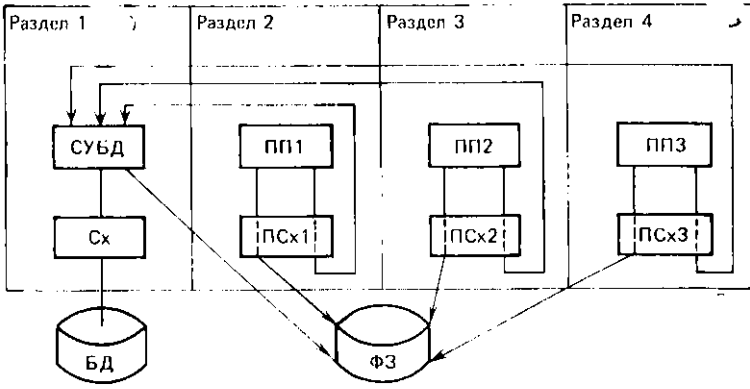


Рис. 3.4. Схема связи СУБД «СЕТОР» с пользователем при мультизадачном режиме в разных разделах оперативной памяти:

ПСх1, ..., ПСх4 — подсхемы прикладных программ *ПП1, ..., ПП4*; *Сх* — схема *БД*; *ФЗ* — файл задач синхронизации

Язык манипулирования данными СУБД «СЕТОР» реализован по способу включающего языка программирования. Этот язык предоставляет все стандартные операции манипулирования данными: включение, удаление, изменение, извлечение записей БД. Имеются и специфические для СУБД «СЕТОР» дополнительные команды манипулирования: поиск в физической последовательности, установка указателя текущей записи на начало файла, модификация связей записи и др. Функции манипулирования данными активизируются операторами CALL — включающего языка программирования.

● **Пример простой БД, содержащей информацию о имеющихся в наличии типовых элементах замены (ТЭЗ) и отдельных микросхемах.** Связи от ТЭЗов к микросхемам участвуют в ответе на запрос: «Какие микросхемы включены в состав данного ТЭЗ?», связи же от микросхем к ТЭЗам соответствуют запросу: «В какие ТЭЗ входит данная микросхема?». Логическая схема базы данных приведена на рис. 3.5. Здесь же показана структура записей каждого файла (связь с файлом дана штриховыми линиями) с указанием имени каждого элемента данных и его длины в байтах. В скобках приводятся наименования всех компонентов схемы. Описание указанной базы данных на ЯОД «СЕТОР» приведено ниже:

```
DBD=BASE 00          } имя схемы
MAIN—FILE=UNIT       } БД (группа 1)
LINKS:               }

DELK (LKUD)          } описание связей фай-
                     } ла ТЭЗ (группа 2)
```

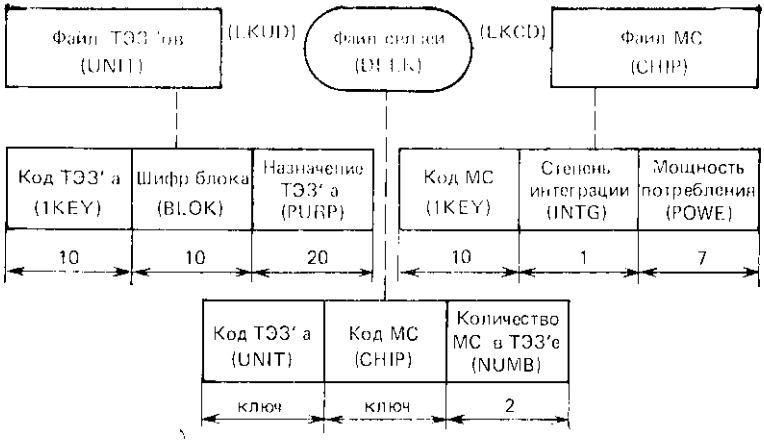


Рис. 3.5. Пример логической схемы базы данных для СУБД «СЕТОР»

```
DATA:
UNIT1 KEY=10
UNITBLOK=10
UNITPUKP=20
```
описание полей записи файла ТЭЗ (группа 3)

```
MEDIA:
UNIT=5061
FILE—SIZE=RECORDS=500
RECORDS—IN—BLOCK=100
```
операторы описания размещения файла ТЭЗ в памяти (группа 4)

```
MAIN—FILE=CHIP
LINKS:
DFLK (LKCD)
DATA:
CHIP1KEY=10
CHIPINTG=1
CHIP POWE=7
```
логическое описание файлов микросхем (группа 5)

```
MEDIA:
FILE—SIZE=RECORDS=500
RECORDS—IN—BLOCK=150
```
физическое описание файла микросхем (группа 6)

```
DEPENDENT—FILE=DFLK
LINKS:
UNIT (LKUD)
CHIP (LKCD)
DATA:
DFLKUNIT=10=KEY
DFLKCHIP=10=KEY
DFLKNUMB=2
MEDIA:
FILE—SIZE=RECORDS=2000
LOAD—LIMIT=90
END DBD
```
операторы описания зависимого файла связей (группа 7)

Команды группы 1 открывают описание данных.
Команды группы 2 описывают связи основного файла с именем UNIT. В примере файл UNIT соединен связью LKUD лишь с зависимым файлом DFLK.
Команды группы 3 описывают структуру записи основного, файла UNIT. В этом описании перед именем поля ставится имя файла. Первое поле, кроме того, является ключом при поиске.
Команды группы 4 и 6 содержат указание по размещению записей на физических носителях: указывается объем буфера (FILE — SIZE) и число записей в блоке (RECORDS — IN — BLOCK).
Команды группы 5 описывают связи и структуру записи основного файла CHIP аналогично тому, как это было сделано для файла UNIT.
Команды группы 7 описывают зависимый файл по тем же правилам, что и основной файл. Однако в записях этого файла присутствуют два ключа, что необходимо для обеспечения ответа на запросы обоих типов.
Пусть обращение к описанной выше БД выполняется из программы, составленной на языке ПЛ/1. В качестве ЯМД «СЕТОР» выступает оператор CALL SETOR (...), записываемый по правилам языка ПЛ/1. Тогда фрагмент программы, содержащей обращение к БД, будет выглядеть следующим образом:

```
   . . .
DECLARE                                          ┐
FUNCT CHAR(5),                                   │
DLIST CHAR (30) VAR,                             │
DAREA CHAR (40) VAR,                             │ группа 1
QUAL1 CHAR (15),                                 │
QUAL2 CHAR (9),                                  │
END CHAR (4),                                    │
REFER CHAR (4);                                  ┘
   . . .
   . . .
FUNCT='OPENM';                                   ┐
QUAL1='UNIT';                                    │ группа 2
CALL SETOR (FUNCT, QUAL1, END);                  ┘
   . . .
FUNCT='OBTNM';                                   ┐
QUAL1='CHIP*155ЛАЗ';                             │
DLIST='CHIP1 KEYCHIP1                            │
NIGCHIPPOWEEND';                                 │ группа 3
CALL SETOR (FUNCT, DLIST, DAREA,                 │
QUAL1,,END);                                     │
   . . .                                         ┘
FUNCT='OBTNF';                                   ┐
QUAL1='CHIP*K133РУ1';                            │
REFER='LKCD'                                     │
QUAL2='DFLK*LKCD';                               │
DLIST='DFLKUNITDFLKNUMBEND.';                    │ группа 4
CALL SETOR (FUNCT, DLIST, DAREA,                 │
QUAL1, QUAL2, END).                              │
   . . .                                         ┘
```

Группа 1 операторов содержит описание переменных, используемых при обращении к БД. В примере все переменные — символьные строки. В операторе CALL SETOR первый параметр определяет команду ЯМД. Так, значение первого параметра OPENM соответствует команде открытия файла, OBTNM — команде поиска в основном файле, OBTNF — команде поиска в зависимом файле и т. д. Остальные параметры соответствуют семантике каждой команды ЯМД. Так, операторы группы 2 открывают файл UNIT (аналогично должны быть открыты и оставшиеся два файла). Группа 3 операторов осуществляет поиск микросхемы К155ЛА3 в файле CHIP и размещает сведения о ней в поле DAREA. Перечень имен полей записи, содержимое которых необходимо выбрать, указывается в параметре DLIST. Группа 4 операторов предназначена обеспечить ответ на запрос: «В какой ТЭЗ и в каком количестве входит микросхема К133РУ1?» Запрос строится от файла CHIP через связь LKCD к файлу DFLK, что и указывается в параметре QUAL2. Семантика остальных параметров такая же, как и в предыдущей группе.

§ 3.4. Информационно-поисковая СУБД «ПОИСК»

Для хранения данных документального типа в САПР используют СУБД типа ИПС. Характерным представителем СУБД такого типа является СУБД «ПОИСК».

Система управления базой данных «ПОИСК» предоставляет пользователям следующие возможности:

1) ввод данных с экрана дисплея или в пакетном режиме;

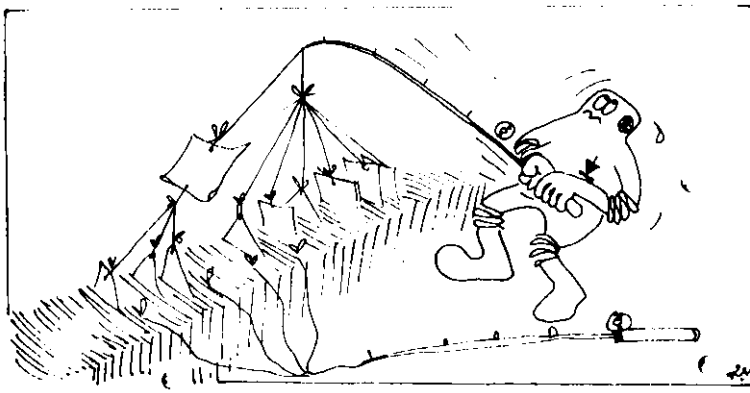
2) корректировку и добавление новых данных;

3) доступ к БД в интерактивном режиме посредством языка запросов;

4) обеспечение секретности информации с помощью паролей пользователей.

Организация базы данных. База данных для СУБД «ПОИСК» есть упорядоченное множество информационных записей. Каждая из записей состоит из полей, соответствующих частям документа, и представляет собой символьные строки.

Например, для приложений САПР записям будут соответствовать тексты проектов, ТЗ, руководящих материалов, стандартов и т. д. Поля же будут соответствовать разделам документов, таким, как название метода, дата разработки, программная реализация, состав ППП и др. В СУБД «ПОИСК» может существовать неограниченное количество БД «ПОИСК», каждая включает в себя однородные записи.



Каждая запись состоит из пронумерованных *полей* фиксированной или переменной длины. Поля фиксированной длины применяются для хранения данных, содержащих заранее определенное и постоянное количество символов, например год издания, код ГОСТа, порядковый номер документа, тип ЭВМ. Поля переменной длины предназначены для хранения таких данных, как название документа, краткая аннотация, организация-разработчик, текст самого документа.

■ Примечание. В СУБД «ПОИСК» термин «поле» соответствует введенному ранее термину «элемент данных».

Поля нумеруются по порядку, каждому полю ставится в соответствие двухбайтовая метка. Пользователь может обращаться к содержимому поля, указывая его порядковый номер.

Допускается выделение *подполей*. Подполя внутри поля выделяются с помощью специальных двухбайтовых разделителей: состоящих из знака логического отрицания «¬» и буквы или цифры.

● **Примеры обращения к полям и подполям записей.** Примерами таких обращений могут служить:

TEXT 16 — к полю 16
TEXT 19 (¬5) — к подполю 5 поля 19
TEXT 19 (¬A) — к подполю А поля 19

Каждая база данных состоит из нескольких машинных файлов на устройствах прямого доступа (НМД). Эти файлы логически связаны между собой. Основной файл MAIN FILE является файлом прямого доступа и

содержит полный набор записей в специальном внутреннем формате хранения. Доступ к записям в подобных файлах осуществляется передачей процедуры доступа физического адреса записи на НМД. Для экономии места на диске записи располагаются на нем одна за другой. Поскольку записи могут иметь разную длину (из-за наличия полей переменной длины), вычислить каким-либо образом физический адрес записи, исходя, например, из ее порядкового номер, нельзя. Поэтому адрес каждой записи хранится в специальном файле, получившем название *файла перекрестных ссылок* CROSS — REFFERENCE. Файл обеспечивает связь порядкового номера записи с физическим адресом этой записи на диске. Использование этого файла позволяет перейти к доступу к записям по их порядковым номерам.

Файл транзакций TRANSACTION используется для хранения вновь поступивших записей перед их включением в основной файл. Новые записи хранятся в этом файле до тех пор, пока специальные программные средства не проверят соответствие новой записи существующим описаниям. Кроме того, в файле транзакций копируются записи, подлежащие модификации. Модифицированная запись также подвергается проверке на соответствие описанию. В случае отсутствия ошибок новая или модифицированная запись включается в основной файл. Подобная организация обеспечивает более высокую степень целостности БД.

Описание структуры каждой записи хранится в таблице FDT определения полей, где указаны номер каждого поля, его длина и кратность. Для поля могут быть определены также правила проверки соответствующего данного при вводе и формат для вывода на печать или дисплей.

Поиск документов. В состав записей могут входить ключевые слова или фразы, используемые для поиска соответствующего документа. Для выделения в тексте этих элементов-дескрипторов используют соответствующие управляющие символы. Поиск документов в СУБД «ПОИСК» осуществляется указанием дескрипторов или их связанной совокупности.

По значениям дескрипторов в СУБД «ПОИСК» строятся инвертированные файлы. Запись инвертированного файла состоит из значения дескриптора и списка номеров записей, соответствующих этому значению. Пусть в записях основного файла с порядковыми номерами 18,

204, 766 и 1039 содержится ключевая фраза «ИО САПР». Тогда запись инвертированного файла содержит фразу «ИО САПР» и цепочку из указанных номеров документов. Поскольку для другого слова цепочка номеров может оказаться более длинной или более короткой, записи инвертированного файла имеют переменную длину.

В СУБД «ПОИСК» существует возможность автоматического контроля вводимых в инвертированный файл дескрипторов с целью исключения из него слов, бессмысленных с точки зрения доступа, например предлогов. Программа, осуществляющая подобный контроль, может также анализировать формы одного и того же слова, приводя их к одному виду с целью исключения дублирования записей в инвертированном списке. В некоторых случаях возникает необходимость группирования отдельных дескрипторов. Например, понятие «ЕСПД» объединяет совокупность ГОСТов по программной документации. Для дескриптора ЕСПД в инвертированный файл новая запись не будет вводиться, чтобы избежать дублирования. Для хранения обозначений подобного рода ведется файл группировки ANY — FILE. Если в этом файле точка входа найдена, тогда соответствующие элементам данной группы списки номеров объединяются.

Язык запросов СУБД «ПОИСК» основан на булевой алгебре, используемой для описания логических операций над множествами записей, определяемыми значениями дескрипторов. Ниже приведен состав логических операций:

OR — логическое ИЛИ. Операция определяет объединение нескольких классов, обозначается символом «*». Если двум дескрипторам A и B соответствуют два класса документов, то результатом операции A*B будет класс, состоящий из документов, содержащих дескриптор A, или дескриптор B, или оба дескриптора. Данная операция расширяет область поиска и соответственно число документов, получаемых на выходе.

AND — логическое И. Операция осуществляет пересечения двух классов; обозначается символом «+». Так, результатом операции A+B будет класс документов, одновременно содержащих оба дескриптора. Например, запрос на поиск документов об операционных системах коллективного доступа можно написать так:

ОПЕРАЦИОННЫЕ СИСТЕМЫ + КОЛЛЕКТИВНЫЙ ДОСТУП

NOT — логическое отрицание. Операция обозначается символом «¬ ». В результате выполнения операции A ¬ B будет получен класс документов, включающих дескриптор A, но не включающих дескриптор B.

Для описания сложных последовательностей логических операций используются скобки по правилам алгебры.

В результате исполнения запроса пользователь первоначально получает в виде ответа последовательность номеров записей основного файла, удовлетворяющих его поисковому предписанию. Если число номеров слишком мало или велико с точки зрения пользователя, то он, модифицируя свой запрос, может увеличить или соответственно уменьшить область поиска. По окончании всех итераций пользователь имеет возможность считать с экрана дисплея или выдать на печать интересующие его записи.

В дополнение к существующим конструкциям языка запросов в СУБД «ПОИСК» существуют возможности для анализа текста документов, уже отобранных в результате выполнения запроса.

Существуют три оператора анализа текста:

1) *логический оператор* (позволяет отобрать документы, содержащие заданный текст в заданном поле), например

TEXT 24 «АССОЦИАТИВНЫЙ».

В результате выполнения оператора будут отобраны документы, содержащие слово АССОЦИАТИВНЫЙ в 24-м поле;

2) *арифметический оператор* (позволяет отобрать документы, в которых содержимое определенных позиций удовлетворяет заданным арифметическим условиям), например

TEXT 00 + 6>«1975»

Будут отобраны документы, в которых данное, расположенное начиная с седьмой позиции поля 00, удовлетворяет условию «данное> 1975»;

3) *оператор наличия* или *отсутствия поля,* например:

TEXT 20P

(отбираются документы, где есть данное в поле 20; P — PRESENCE — присутствие),

TEXT 10A

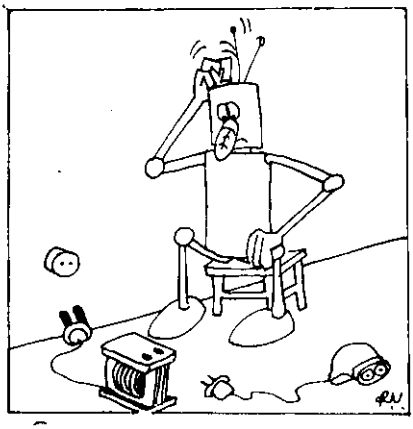
(отбираются документы, где отсутствует поле 10; A — ABSENCE — отсутствие).

При работе с СУБД «ПОИСК» пользователь формирует запрос с помощью вышеописанных операторов. В общем случае запрос состоит из двух частей: в первой части перечисляются значения дескрипторов, соединенных знаками операций булевой алгебры; во второй части содержатся спецификации, определяющие порядок и форму отображения записей на дисплей или печать. СУБД осуществляет поиск каждого из дескрипторов в инвертированном файле с целью определения множеств номеров записей. В соответствии с запросом выполняются операции над множествами записей, в результате которых формируется результирующий список номеров записей, удовлетворяющих запросу. Поиск физических адресов происходит по номерам записей в файле перекрестных ссылок. Затем информация, считанная из основного файла, преобразуется в соответствии со спецификациями второй части запроса в вид, необходимый для выдачи либо на экран дисплея, либо на печать.

§ 3.5. Особенности взаимодействия разноязыковых модулей

Проблемы комплексирования. Рассмотрим комплексирование модулей, составленных на различных языках программирования. Для САПР наиболее употребимы универсальные языки ассемблера, ФОРТРАН, ПЛ/1. Особенности операций с разноязыковыми модулями можно свести в основном к двум группам: 1) особенности установки программной среды и вызова модуля; 2) различия в реализации типов и структур данных, проявляющиеся при обмене информацией между модулями.

Установка среды. При передаче управления из модуля, составленного на одном языке, в модуль, составленный на другом языке, требуется устанавливать *программную среду,* представляющую собой совокупность программ обработки прерываний и аварийных завершений, установки регистров, содержащих адреса областей, использующихся на протяжении выполнения задачи, и др. Поэтому при организации вызова модулей, составленных на различных языках, необходимо всякий раз устанавливать среду вызываемого модуля. Рассмотрим процесс установления среды в языках ассемблера, ФОРТРАН, ПЛ/1.



В алгоритмическом языке ФОРТРАН среда устанавливается модулем IBCOM, находящимся в библиотеке компилятора. Помимо установки среды этот модуль выполняет операции ввода — вывода и др. Для программ, не содержащих операций ввода — вывода и прерываний, среда может не создаваться. Вызов модуля, составленного на языке ФОРТРАН, и установка среды этого языка представляют собой различные, самостоятельные действия.

В алгоритмическом языке ПЛ/1 установка среды обеспечивается совокупностью модулей библиотечных, сгенерированных компилятором ПЛ/1, а также построенных пользователем. Место расположения программной среды определяется вызовом модуля. Таким образом, установка среды в языке ПЛ/1 и вызов самого модуля представляют собой неразрывное целое и считаются одной операцией.

Для модулей, написанных на языке ассемблера, компилятор среды не создает. Чтобы обеспечить нормальное функционирование программы для обработки программных прерываний и аварийных ситуаций, пользователь должен сам включить в свою программу соответствующие средства, например макрокоманды SPIE, STAE, ABEND и др. Механизм задания среды определяется также пользователем, который располагается в любом месте исходного модуля макрокоманды задания среды.

Способ обращения к модулю определяется языковыми средствами организации связи по управлению, которое реализуется через активизацию модуля и возврат управления.

■ Примечание. В ЕС ЭВМ для активизации модуля необходимо присутствие копии модуля в оперативной памяти. Копия заносится в память с помощью загрузки.

Средства передачи управления алгоритмических языков проиллюстрированы табл. 3.1.

Таблица 3.1

Алгоритмические языки	Характеристика средств передачи управления		
	средство	этап соединения	тип загрузки
ПЛ/1	CALL	Компиляция	Предварительная
		Редактирование	По запросу
Ассемблера	ATACH		
	LINK	Выполнение	»
	FETCH		
	XCTL	»	»
	CALL	Компиляция	Предварительная
		Редактирование	По запросу
ФОРТРАН	CALL	Компиляция	Предварительная
	Вызов функции	Редактирование	По запросу

Согласование типов. Различные языки программирования обладают разными наборами типов данных. Типы данных языков программирования приведены в табл. 3.2.

Возможные отношения между типами данных приведены ниже.

К эквивалентным типам данных относятся типы данных, для которых внутреннее представление, сгенерированное компиляторами, идентично.

К косвенно-эквивалентным типам данных относятся типы данных, для которых нет эквивалентного описания, но с помощью имеющихся языковых средств их можно совместить.

К неэквивалентным типам данных относят данные, которые невозможно свести друг к другу с помощью имеющихся языковых средств.

■ Примечание. Описание типов данных для вышеприведенных языков программирования см. в § 1.1.

Рассматриваемые алгоритмические языки различаются не только типами, а также заданиями длины отдельных элементов данных. Длина элементов данных может указываться явно или неявно (по умолчанию). Кроме того, необходимо отметить следующие организационные особенности каждого языка:

Таблица 3.2

Типы данных языков			Внутреннее представление
ПЛ/1	ФОРТРАН	ассемблера	
REAL FIXED	INTEGER	H	Число с фиксированной точкой длиной 2 байт
BINARY		F	Число с фиксированной точкой длиной 4 байт
REAL FLOAT	REAL	E	Число с плавающей точкой длиной 2 байт
BINARY		D	Число с плавающей точкой длиной 8 байт
REAL DICEMAL FIXED	—	P	Последовательность десятичных цифр со знаком длиной 1 ... 16 байт
REAL DICEMAL FLOAT	REAL	E D	—
COMPLEX FLOAT	COMPLEX	—	Два числа с плавающей точкой длиной 4 байт
DECTMAL		—	Два числа с плавающей точкой длиной 8 байт
COMPLEX FIXED DICIMAL	—	—	Два десятичных числа
COMPLEX FLOAT	COMPLEX	—	—
BINARY		—	
COMPLEX FIXED	—	—	Два целых числа длиной 4 байта
BINARY			Два целых числа длиной 2 байта

Продолжение табл. 3.2

Типы данных языков			Внутреннее представление
ПЛ/1	ФОРТРАН	ассемблера	
CHARACTER			Символьная строка длиной 1 ... 32 767 байт
BIT	LOGICAL	B	Битовая строка длиной 1 ... 32 767 байт
		X	Последовательность шестнадцатиричных цифр

а) обратное расположение массивов (в языке ФОРТРАН массив располагается в памяти по столбцам, а в языке ПЛ/1 — по строкам);

б) наличие информационного вектора для переменных в языке ПЛ/1 [поскольку в языке ПЛ/1 память под данные может выделяться динамически, компилятор генерирует информационный вектор, который содержит сведения о переменных, строках и массивах (адрес, длина, количество элементов и пр.)] и имеет различный вид для разных структур данных;

в) выравнивание полей, которое определяется тем, что компилятор располагает данные в соответствии с описаниями, выравнивая их, если необходимо, на границы слов и полуслов.

Обмен данными. Наиболее распространен способ обмена данных с помощью операторов CALL и LINK. При этом осуществляется формирование списка передаваемых данных и списка их адресов. Адрес списка адресов передается вызываемой программе через регистр 1.

Другим распространенным способом обмена данными является использование *общих областей* — статически распределенных участков памяти, к которым может обращаться любой модуль независимо от того, на каком языке он описан. Память под общую область отводит редактор связей во время создания загрузочной программы из совокупности общих областей отдельных модулей. Каждый из рассматриваемых языков имеет средства для описания общих областей: в языке ФОРТРАН — опера-

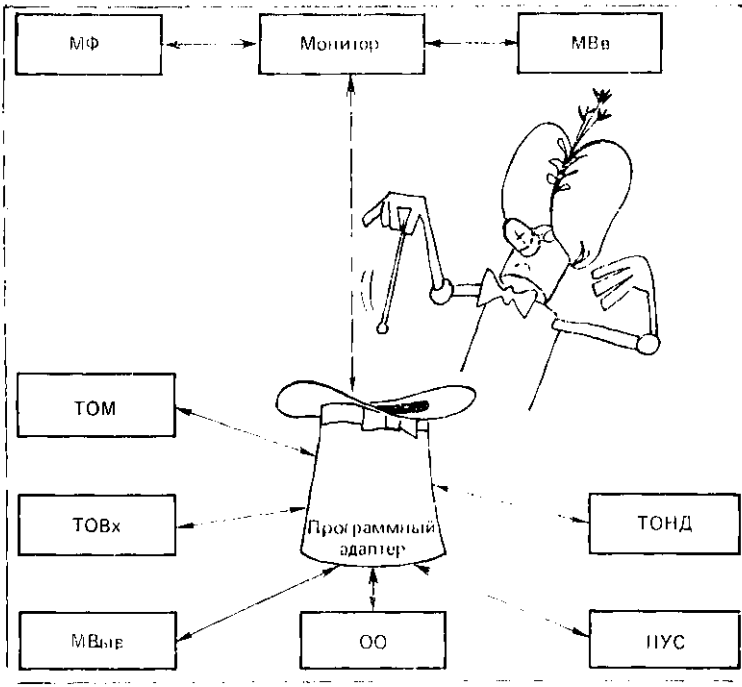


Рис. 3.6.

тор COMMON, в языке ПЛ/1 — оператор STATIC EXTERNAL, в языке ассемблера — оператор COM.

Информационный адаптер. С учетом вышеизложенного для осуществления универсального информационного интерфейса необходимо:

1) произвести контроль наличия исходных данных для каждого отдельного модуля;

2) задать недостающие исходные данные;

3) проверить соответствие типов, структур и последовательности данных аналогичным характеристикам данных, принятым в вызываемом модуле;

4) преобразовать данные в случае несоответствия типов;

5) обеспечить передачу данных вызываемому модулю в соответствии с типом обмена;

6) организовать среду, определяемую языком программирования модуля;

7) проверить результаты;

8) выполнить обратное преобразование данных в вид, принятый для хранения промежуточных результатов;

9) сохранить результаты работы модуля для дальнейшего использования.

Реализация описанных функций выполняется *программой-адаптером*. Информационное обеспечение адаптера (рис. 3.6) включает в себя таблицы описателей модулей, входов модулей, наборов данных, область обмена. На рисунке *МФ* — модули функционирования; *МВв* — модули ввода; *ТОНД* — таблица описателей наборов данных; *ТОМ* — таблица описателей модулей; *ПУС* — программы установки среды; *ТОВх* таблицы описателей входов; *МВыв* — модули вывода; *ОО* — области обмена.

Таблица описателей модуля содержит: имя модуля; идентификатор языка программирования; признак типа обмена, принятый в модуле (параметры, общие области, наборы данных); количество параметров; имя описателя входов модуля; имя области обмена; имя описателя набора данных.

Таблица описателей входов модуля содержит: имя параметра локальное; имя параметра глобальное; характеристику параметра (входной, выходной, модифицируемый); вид структуры (переменная строка, массив арифметический, массив строк, структура, массив структур и т. д.); размерность (для массива); длину (для строк); основание системы счисления (для переменной или элемента массива); форму представления; точность.

Таблица описателей наборов данных содержит: имя набора; имя DD-предложения; тип организации; метод доступа; формат блока; длину блока; атрибуты записи.

Для обеспечения преемственности данных следует предусмотреть область памяти, куда заносились бы те результаты работы каждого модуля, которые принимают участие в дальнейших расчетах. Этой цели служит *область обмена* — память, выделенная для промежуточных результатов, передаваемых от модуля к модулю. В нее заносятся исходные данные для первого модуля, результаты вычислений, недостающие данные для промежуточных модулей.

Для организации области обмена необходимо:

1) осуществить анализ данных, которые будут циркулировать в среде программного комплекса в составе САПР с целью определения одинаковых по смыслу переменных;

2) всем одинаковым по смыслу переменным присвоить одно и то же имя, отражающее семантику этих переменных (глобальные переменные);

3) каждой переменной, встречающейся только один раз, присвоить уникальное имя, отражающее семантику величины (локальные переменные);

4) каждой глобальной и локальной переменной поставить в соответствие тип и структуру, являющиеся наиболее информативными.

Указанная совокупность переменных составляет область обмена, в которой хранится текущее значение каждой из переменных в наиболее информативном виде.

В крупных САПР, программы которых оперируют с большим числом входных, промежуточных и результирующих переменных, области обмена удобно организовать в виде некоторого банка данных. Это позволяет возложить часть функций, выполняемых адаптером, на СУБД, что в конечном итоге сокращает время на разработку информационного и программного обеспечения САПР.

Таким образом, адаптер выполняет всю совокупность операций по организации информационного взаимодействия между программными модулями. В случае разноязыковых моделей адаптер практически берет на себя выполнение соответствующих функций операционной системы. Достаточно сложной является также задача построения области обмена, поскольку ее решение связано со структурированием всех переменных, участвующих в информационном обмене. В крупных САПР, программные модули которых оперируют с большим числом входных, промежуточных и результирующих переменных, функции адаптера по организации и взаимодействию с обменными областями целесообразно переложить на типовые СУБД.

Банки данных в настоящее время находят все более широкое применение для организации межмодульного интерфейса. Их использование наиболее эффективно, когда совокупность модулей программного обеспечения зафиксирована и не подлежит изменениям в дальнейшем. В этом случае необходимо составить логическую схему для всей области обмена, в которой были бы указаны наименования переменных, их взаимосвязи, тип представления. Обращение из программных модулей для получения значений необходимых переменных должно выполняться с помощью операторов взаимодействия с СУБД. Применение банков данных для целей организации информацион-

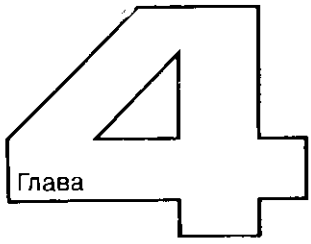
ного межмодульного обмена сокращает сроки разработки информационного и программного обеспечений с САПР.

КРАТКИЕ ВЫВОДЫ

Совокупность данных, используемых САПР, составляет ее информационный фонд. Основное назначение ИО САПР — ведение информационного фонда, т. е. создание, поддержка и организация доступа к данным. Многокомпонентность состава САПР порождает разнообразие типов данных в информационном фонде (программы модулей, массивы чисел, подготовленные заранее кадры экрана дисплея, нормативно-справочная проектная информация, текущая проектная документация и т. д.). Для хранения и обработки этих данных используются файловая и библиотечная системы в составе ОС, банки данных, информационные программы-адаптеры. Из перечисленных средств наибольшая информационная нагрузка в современных САПР приходится на долю банков данных.

Для поддержания локальных баз данных следует использовать сравнительно простые и занимающие небольшой объем оперативной памяти СУБД (например, «СЕТОР»). В интегрированных БД со сложными, динамически изменяющимися в процессе проектирования внешними моделями, должны использоваться СУБД «ДИСОД». Хранение проектной документации, ГОСТов, руководящих и методических проектных материалов, описания типовых проектных решений необходимо осуществлять средствами ИПС (например, ИПС «ПОИСК»).

Организация информационного взаимодействия между разноязыковыми модулями ставит перед разработчиками САПР задачи восстановления программной среды, согласования данных разного типа, учета особенностей представления одинаковых структур данных в различных алгоритмических языках. Наиболее универсальный способ решения перечисленных задач — построение программного адаптера, полностью регламентирующего информационный обмен между модулями в составе специального программного обеспечения САПР. Включение в состав программного адаптера промышленных СУБД позволяет упростить его алгоритм и сократить сроки разработки.



ДИАЛОГ В СИСТЕМАХ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ

§ 4.1. Организация диалога

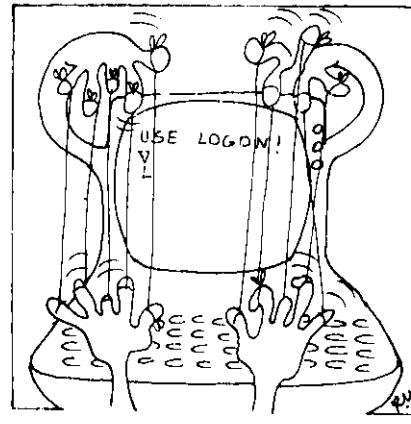
Функционально законченная программа, выполняемая ЭВМ без вмешательства проектировщика, называется *машинной процедурой*.

Пользуясь исключительно машинными процедурами, невозможно выполнить проектирование объекта. Сложность и многообразие проектных вариантов, многокритериальность процесса принятия решения, трудности полной автоматизации процедур синтеза вынуждают непосредственно включать в процесс проектирования человека-проектировщика. Как правило, при проектировании человек выполняет функцию принятия решения, как наиболее трудно поддающуюся алгоритмизации, а ЭВМ производит вычисления.

Взаимодействие между человеком и ЭВМ происходит посредством *сообщений* — совокупности данных, достаточной для выполнения определенных действий. Сообщение, поступающее от ЭВМ к человеку, называется *выходным*; сообщение от человека к ЭВМ называется *входным*. Обычно сообщение размещается на экране дисплея и называется *кадром*. *Обмен* - последовательность, включающая сообщение от человека к ЭВМ, реакцию ЭВМ (*машинную процедуру*), сообщение от ЭВМ к человеку.

Диалог - последовательность обменов, выполнение которой приводит к решению поставленной задачи, т. е. диалог служит методом решения задачи, где пользователь знает задачу, а ЭВМ используется для решения подзадач. Диалог в САПР используется для: а) обеспечения доступа к базе данных САПР; б) ввода данных для выполнения машинной процедуры; в) просмотра на экране дисплея результатов; г) контроля за ходом выполнения

машинной процедуры. В диалоге в общем случае допустима ситуация, когда входное сообщение не может немедленно инициировать машинную процедуру в силу либо неточности, либо недостаточности содержащейся в сообщении информации. В таком случае возникает последовательность обменов до полного определения смысла исходного сообщения. Такая последовательность обменов называется *метадиалогом*.



Типы диалога. Диалог подразумевает наличие двух участников: человека и ЭВМ. Каждый из них может находиться либо в активном, либо в пассивном состоянии. Участник будет находиться в *активном состоянии*, если он выполняет действие по анализу полученного сообщения и формированию нового, и в *пассивном состоянии*, если не предпринимает никаких действий в ожидании сообщения. В диалоговом взаимодействии ситуация, когда оба участника диалога находятся в пассивном состоянии, является *тупиковой*, поскольку из нее невозможно выйти, опираясь лишь на средства ведения диалога.

Если оба участника диалога поочередно меняют свои состояния, диалог называют *синхронным* (в синхронном диалоге участники как бы поочередно включают и выключают друг друга из разговора).

Если оба участника диалога одновременно находятся в активном состоянии, то такой диалог называют *асинхронным* (в асинхронном диалоге человек имеет возможность в любой момент времени вмешаться в ход выполнения машинной процедуры с целью ее приостановления или внесения изменений). Асинхронный диалог распространен в приложении к имитационным моделям, оптимизационным процедурам, организации вычислительного процесса. В этом случае человеку со стороны ЭВМ постоянно поставляются на экран дисплея сообщения о текущем состоянии машинной процедуры. Человек, как и ЭВМ, находится в активном состоянии и при необхо-

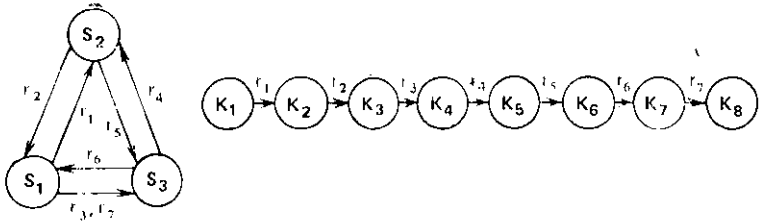


Рис. 4.1. Пример последовательности кадров и графа состояний экрана:

K_i — i-й кадр; S_j — j-е состояние экрана; r_l — l-я реакция пользователя;
{K_1, K_3, K_7}→S_1;
{K_2, K_5}→S_2;
{K_4, K_6, K_8}→S_3

димости прерывает активность ЭВМ, переводя ее в пассивное состояние.

Граф состояний экрана. Если определен диалог, то может быть построена последовательность выходных кадров. Идентичные кадры в такой последовательности называются *эквивалентными* и соответствуют одному *состоянию экрана* дисплея. Итак, диалог — последовательность изменяющихся состояний экрана. Если свернуть последовательность кадров, объединяя их по эквивалентным состояниям экрана, то получим *граф состояний экрана дисплея.* Различным вершинам графа соответствуют различные состояния экрана, а дугам — возможные переходы (каждой дуге ставится в соответствие реакция человека). Пример последовательности кадров и полученного из него графа состояний экрана приведен на рис. 4.1. Граф состояний экрана позволяет в наглядной форме представить возможности диалога, произвести его перекомпоновку, сгруппировав отдельные состояния.

Классификация сообщений. Сообщение несет в себе совокупность сведений, передаваемых одним участником диалога другому, с тем чтобы вызвать определенную реакцию у партнера. Реакция партнера определяется содержанием сообщения. По содержанию сообщение может быть *информационным, запросом* или *ответом.* Если сообщение предназначено для передачи вопроса партнеру и предполагает получение от него обязательного ответа на этот вопрос, то первое сообщение называется запросом, а второе — ответом. Таким образом, запрос всегда предполагает последующий ответ, а любой ответ формируется как реакция на предшествующий запрос.

Если же сообщение содержит сведения, не предполагающие немедленного действия и не являющиеся ответом на запрос, то оно называется информационным.

Взаимодействие человека с ЭВМ должно производиться в форме, понятной и удобной для восприятия человеком. Различают формы сообщений: табличную, директивную и с использованием ограниченного естественного языка (ОЕЯ).

Табличная форма сообщения представляет собой форматизированную таблицу, размещаемую на экране дисплея, с поименованными полями, где и располагается необходимая информация в символическом или числовом виде.

Директива (команда) есть некоторый оператор, выбираемый из заданного множества операторов и формируемый в соответствии с определенными для него синтаксисом и семантикой. Директивная форма сообщений предоставляет участнику диалога большую свободу действий по сравнению с табличной.

Сообщение в форме некоторой фразы на ОЕЯ наиболее удобно в применении к проектировщику, не знакомому с алгоритмическими языками, но желающему иметь широкий диапазон возможных действий. Как правило, использование ОЕЯ влечет введение метадиалога с целью устранения неопределенностей, возникающих при построении фраз. Несмотря на сложность реализации, эта форма сообщений все шире используется в САПР, освобождая проектировщика от изучения специализированных языков общения с ЭВМ.

В рамках приведенной классификации в настоящее время сложились некоторые типовые соотношения. Так, информационные сообщения являются выходными, а директивные — входными сообщениями. Выходные запросы табличной формы, предназначенные для задания проектировщиком исходных данных последующей машинной процедуре, называются *шаблонами*. Аналогичные запросы, содержащие перечень возможных альтернатив продолжения процесса проектирования с указанием их шифров (номер либо имя), называются «меню». Выходные информационные сообщения, как правило, имеют смысл подсказок. Если форма входных и выходных сообщений — ОЕЯ, то диалог называется свободным.

Для взаимодействия с проектировщиком, не знающим структуры диалога, следует рекомендовать использование шаблонов и «меню», а также свободный диалог. Для

проектировщика, хорошо знакомого с графом состояний экрана, целесообразно использование директив, позволяющих исключить метадиалог и сократить время на поиск необходимых проектных решений.

§ 4.2. Программное обеспечение диалога

Требования к программному обеспечению диалога. Программное обеспечение, применяемое для организации диалога в САПР, представлено в виде следующих групп средств:

1) диалоговые системы коллективного пользования;

2) системные средства ОС для программирования диалога (методы доступа);

3) подпрограммы для организации диалога.

Для АП могут использоваться как диалоговые системы (ДС) общего назначения, так и специальные диалоговые системы. Первые ориентированы на решение общих задач автоматизированной обработки данных, а вторые — на конкретные предметные области проектирования.

При создании специализированных ДС можно удовлетворить всем требованиям, предъявляемым к организации человеко-машинного взаимодействия в САПР, но это сопряжено, как правило, со значительными затратами времени и материальных средств. Использование ДС общего назначения позволяет сократить сроки разработки диалоговой системы САПР и уменьшить затраты на программирование и отладку, однако это достигается за счет снижения требований к системе и одновременного повышения требований к квалификации пользователей.

Рекомендуется применять ДС общего назначения на начальных этапах создания и эксплуатации САПР для отработки и проверки методологии проектирования, технологии обработки данных и прикладных программ. В дальнейшем возможна модификация ДС общего назначения с учетом специфических требований по организации диалога в САПР.

При разработке специализированной или выборе общецелевой диалоговой системы для САПР необходимо учитывать:

1) наличие диалогового и пакетного режимов обработки запросов;

2) ориентацию системы на пользователя-непрограммиста, не знающего язык управления заданиями ОС;

3) возможность расширения системы путем включения диалоговых прикладных программ на языках высокого уровня;

4) отсутствие жестких ограничений на прикладные программы (по структуре, объему памяти, использованию стандартных средств ввода — вывода и т. п.);

5) возможность управления диалогом с помощью «меню» и директив;

6) наличие средств для асинхронного диалогового взаимодействия (прерывание счета и возможность оперативного изменения данных);

7) желательность общения на русском языке (сокращение команд, сообщения и т. п.);

8) формат выходного сообщения, который должен позволять вводить информацию по кадрам — «шаблонам».

Обзор диалоговых систем. Рассмотрим характеристики широко распространенных диалоговых систем общего назначения для ЕС ЭВМ (табл. 4.1).

Система диалогового удаленного ввода заданий (ДУВЗ) обеспечивает работу с локальными дисплеями и удаленными пишущими машинками при диалоговой подготовке пакетных заданий. Рассчитана на программистов, возможности расширения ограничены.

Режим разделения времени (РРВ) в ОС ЕС покрывает широкий спектр применений, основным из которых является автоматизация программирования и отладки программ в диалоге. Это очень мощная система, обслуживает различные типы терминалов, но требует большого объема памяти и эффективно эксплуатируется лишь на старших моделях ЕС ЭВМ. Основное ее достоинство — простота адаптации пакетных программ для работы в диалоге. Однако система не обеспечивает ввода по шаблонам кадров информации.

Система телеобработки данных «КАМА» служит для построения диалоговых информационных систем и систем передачи данных. Имеет все возможности для организации удобного диалога пользователей-непрограммистов с системой, однако на прикладные программы накладывают существенные ограничения по использованию стандартных средств ввода — вывода и по объему памяти программ. Диалоги программируются специальными макрокомандами.

Таблица 4.1

Характеристики диалоговых систем общего назначения	Диалоговые системы					
	ДУВЗ ОС ЕС	РРВ ОС ЕС	КАМА	JEC	CJE	PRIMUS
Основное функциональное назначение	Подготовка и обработка пакетных заданий	Автоматизация программирования и отладка	Телеобработка данных в информационных системах	Подготовка и обработка пакетных заданий	Подготовка и обработка пакетных заданий	Автоматизация программирования и обработки данных
Режимы обработки данных прикладной программой	Пакетный	Диалоговый и пакетный	Диалоговый	Пакетный	Пакетный	Пакетный и диалоговый
Категории пользователей системы	Программист	Программист и непрограммист	Непрограммист	Программист	Программист	Программист и непрограммист
Возможно включение диалоговых прикладных программ	Нет	Да	Да	Нет	Нет	Да
Языки написания диалоговых прикладных программ	—	Любой язык программирования	Ассемблер ПЛ/1 КОБОЛ	—	—	Ассемблер ПЛ/1 ФОРТРАН
Средства программирования диалога	—	Операторы ввода — вывода языков высокого уровня	Специальные макрооператоры	—	—	Макрокоманды и подпрограммы диалога

Продолжение табл. 4.1

Характеристики диалоговых систем общего назначения	Диалоговые системы					
	ДУВЗ ОС ЕС	РРВ ОС ЕС	КАМА	JEC	CJE	PRIMUS
Тип взаимодействия	Синхронный	Синхронный и асинхронный	Синхронный	Синхронный	Синхронный	Синхронный
Ограничения на составление прикладных программ	Есть	Нет	Есть	Есть	—	Есть
Язык общения	Английский	Английский	Русский	Английский, русский	Русский	Английский, русский
Формат входного сообщения	Строка	Строка	Кадр	Кадр	Кадр	Строка, кадр
Необходимо знание командного языка или языка управления заданиями	Да	Да	Нет	Да	Да	Да
Методы доступа к терминалу	ГМД, БТМД	ОТМД	БТМД, ОТМД	ГМД, БТМД	Физический	ГМД, БТМД

Диалоговый редактор текстов JEC кроме своего основного назначения позволяет использовать специальным образом оформленные прикладные программы. Пользователь осуществляет с ними связь с помощью команд получения и выдачи сообщений. Запускаются эти программы, как пакетные задания. JEC — технологическая система для программистов.

Система диалогового ввода заданий CJE предназначена для диалоговой подготовки пакетных заданий. Отличительной чертой данной системы является управление диалогом преимущественно с помощью «меню» и подсказок. Все обозначения запросов и их параметров задаются на русском языке. Такая форма общения обеспечивает быстрое обучение пользователей и простоту эксплуатации ДС. Однако система не имеет возможности включения диалоговых прикладных программ.

Диалоговая система коллективного пользования PRIMUS используется для автоматизации обработки данных. Может расширяться за счет дополнения прикладных программ. Однако в системе внешний диалог по подготовке данных и запуску прикладных программ осуществляется в директивной форме на английском языке.

■ Примечание. Ни одна из рассмотренных выше систем не удовлетворяет в полной мере перечисленным ранее требованиям.

§ 4.3. Методы доступа для программного обеспечения диалога

При создании специализированных ДС необходимо использовать системные программные средства диалога. Они входят в состав ОС и оформлены в виде методов доступа. *Метод доступа* определяет набор операций для работы с терминалами данного типа, структуру сообщений и правила их обработки. Программисту предоставляются макрокоманды на языке ассемблера для программирования диалога.

В составе ОС ЕС имеются методы доступа: 1) графический (ГМД); 2) базисный телекоммуникационный (БТМД); 3) общий телекоммуникационный (ОТМД).

Графический метод доступа обеспечивает работу локальных дисплеев комплекса ЕС7906, графических дисплеев и связь мини-ЭВМ (типа СМ ЭВМ) с ЕС ЭВМ.

Базисный телекоммуникационный метод доступа обслуживает как локальные, так и удаленные терминалы, в частности широко распространенные дисплеи комплекса ЕС7920. Имеет гибкие и мощные средства для построения систем телеобработки.

Общий телекоммуникационный метод доступа имеет более ограниченные средства по организации связи с терминалами, однако макрокоманды метода доступа представляют язык высокого уровня для обработки сообщений. Особенность ОТМД заключается в наличии программы управления сообщениями (ПУС), которая обеспечивает централизованную обработку сообщений. Прикладные программы пользователей могут быть сделаны практически независимыми от ПУС и используемых терминалов. Терминальный ввод — вывод программируется с использованием стандартных средств языков высокого уровня для работы с перфокартами и устройствами печати, а при выполнении программы потоки данных связываются с соответствующим терминалом.

Подготовка сообщения к обмену. При любом методе доступа на физическом уровне можно выделить две фазы ввода сообщений с терминала:

1) определение готовности терминала к передаче входного сообщения пользователя;

2) прием ЭВМ входного сообщения.

Готовность терминала к передаче часто оформляется в виде управляющего сигнала «Внимание». Этот сигнал может быть обнаружен ОС через систему прерываний или с помощью периодического опроса (например, по времени). Пока пользователь не сформирует входное сообщение и не нажмет клавишу, вызывающую сигнал «Внимание», система будет обрабатывать другие диалоговые или пакетные запросы. Наличие сигнала «Внимание» позволяет в принципе организовать асинхронное диалоговое взаимодействие пользователя с САПР.

Кроме обработки сигнала «Внимание» и непосредственного ввода — вывода методы доступа предусматривают промежуточное хранение сообщений в оперативной или внешней памяти ЭВМ (буферы и очереди), сборку и разборку сообщений при передаче по частям, добавление или удаление управляющей информации и пр.

На рис. 4.2 показан алгоритм прохождения односегментного сообщения через систему, управляемую ОТМД. Рассмотрим шаги обработки сообщения в программе управления сообщениями.

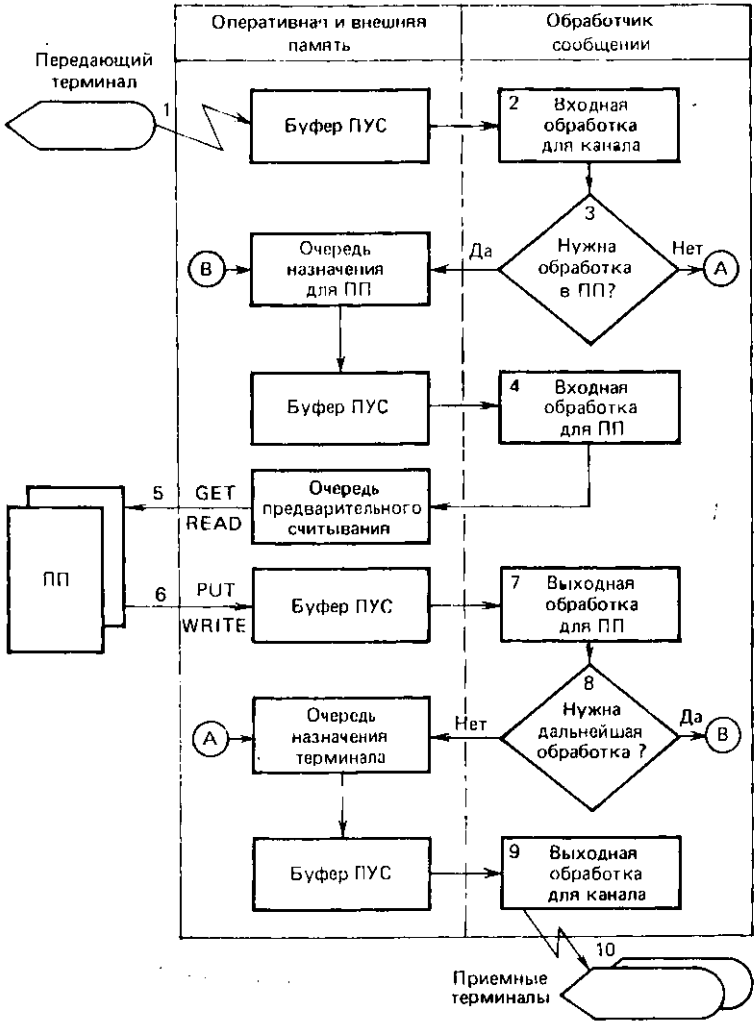


Рис. 4.2. Схема алгоритма ПУС

Шаг 1. Обнаружение ОС готовности терминала к передаче и принятие входного сообщения в ЭВМ.

Шаг 2. Накопление данных в буфере (как только сформирован очередной сегмент, выполняется входная обработка сообщения: контроль, преобразование кодов, выдача сообщений об ошибках и т. п.).

Шаг 3. Передача сообщения в очередь к ПП (если требуется обработка в ПП), иначе передача сообщения в очередь терминала.

Шаг 4. Прием сообщения из очереди к ПП в буфер и передача после входной обработки в очередь предварительного считывания.

Шаг 5. Обработка прикладной программой по операторам GET или READ данных из очереди считывания.

Шаг 6, 7. Передача результатов работы прикладной программы по операторам PUT или WRITE в буфер ПУС и далее выходная обработка для ПП.

Шаг 8. Поступление сообщения в очередь к другой программе и в очередь терминала.

Шаг 9. Выходная обработка сообщения для канала связи.

Шаг 10. Передача сообщения на один или несколько терминалов.

Диалог с помощью локального дисплея. Чтобы организовать синхронный диалог с локальными дисплеями одного типа в рамках небольших САПР, нет необходимости строить сложную программу управления сообщениями. Можно децентрализовать обработку сообщений в специальных подпрограммах организации диалога, которые вызываются из прикладных программ. В простейшем случае достаточно разработать с помощью макрокоманд базисного телекоммуникационного доступа на языке ассемблера подпрограмму, осуществляющую: 1) выход выходного сообщения; 2) ожидание сигнала внимания; 3) ввод входного сообщения.

Обращение к этой подпрограмме имеет вид

```
CALL DSPLY (NC, DATA, TERM),
```

где NC — номер строки для установки курсора; DATA — область сообщений (при входе в подпрограмму должна содержать выходное сообщение, при выходе — ответ пользователя); TERM — имя описания терминала (задается с помощью языка управления заданиями).

При использовании подобных подпрограмм для связи прикладной программы с терминалом возникает задача преобразования прочитанной числовой информации во внутримашинное представление и обратного преобразования при выводе чисел. В языке ПЛ/1 имеются средства внутреннего преобразования данных (GET/PUT STRING). В языке ФОРТРАН таких возможностей нет, однако можно применить промежуточный файл сообще-

ний, с которым прикладная программа осуществляет ввод — вывод по формату.

С помощью аналогичных вышерассмотренной подпрограмм можно создавать простые монопольные (на одного пользователя) ДС. Для создания ДС коллективного использования необходим монитор управления вычислительными процессами пользователей, средства для управления данными в режиме диалога (редактор текстов, библиотекарь, архиварнус и др.) и средства пакетирования заданий. При этом к защите ресурсов пользователей от взаимного влияния и к надежности программ предъявляются повышенные требования.

В прикладных программах необходимо обеспечить контроль вводимой информации, обработку программных ситуаций (деление на нуль, переполнение и т. п.) и санкционирование доступа к информации с помощью парольной защиты. Никакие действия пользователя за терминалом не должны приводить к аварийному завершению его диалогового процесса и тем более системы в целом.

● **Пример программы на языке ПЛ/1.** Программа предназначена для выбора диалоговой подпрограммы по меню. Экран в программе представлен двояко: непрерывной областью и массивом строк. В начале программы очищается область экрана, формируются строки меню и выводятся на дисплей с помощью подпрограммы терминального ввода — вывода. Ответом на меню служит номер функции, вводимый пользователем в первой позиции последней строки. Допустимый ответ преобразуется в индекс перехода, по которому осуществляется обращение к выбранной подпрограмме. После исполнения подпрограммы происходит возврат на выдачу меню. В случае неверного ответа меню выдается пользователю для повторного выбора функции.

```
SELECT: PROCEDURE;
    DECLARE SCREEN CHAR (192Ø),
    LINE (Ø:23) CHAR (8Ø) DEF SCREEN,
    TERM CHAR (8) INIT ('TERM1'),
    MSG CHAR (8Ø) INIT ('␣'),
    REPLY CHAR(1),
    (IND, NL) BIN FIXED;
MENUE: SCREEN='␣';
    LINE(3)='ПЕРЕЧЕНЬ ДОСТУПНЫХ ФУНКЦИЙ';
    LINE(5)='1. ВВОД И КОРРЕКЦИЯ ДАННЫХ';
    LINE(6)='2. РЕШЕНИЕ ЗАДАЧИ';
    LINE(7)='3. ПРОСМОТР И ПЕЧАТЬ РЕЗУЛЬТАТОВ';
    LINE(8)='4. КОНЕЦ РАБОТЫ';
    LINE(21)=MSG;
    LINE(22)='ВВЕДИТЕ НОМЕР ФУНКЦИИ';
    NL=23;
    CALL DSPLY (NL, SCREEN, TERM);
    REPLY=SUBSTR (LINE(23),1,1);
    IND=Ø; MSG='␣';
    IF REPLY>'Ø' & REPLY<'5' THEN IND=REPLY;
```

```
    GO TO M (IND);
M(Ø): MSG='НЕВЕРНЫЙ ОТВЕТ'; GO TO MENUE;
M(1) : CALL EDIT; GO TO MENUE;
M(2): CALL SOLVE; GO TO MENUE;
M(3): CALL REPORT; GO TO MENUE;
M(4): RETURN;
END SELECT
```

§ 4.4. Система CJE диалогового ввода заданий в ОС ЕС

Система CJE рассматривается как пример одной из универсальных редактирующих систем, которые широко используются в САПР в процессе подготовки и решения отдельных задач проектирования.

При работе с системой CJE пользователю предлагаются средства подготовки заданий и занесения их во входной поток ОС ЕС, получения информации о состоянии заданий, выводы результатов выполнения заданий на экран дисплея и печатающее устройство. Она располагает средствами для корректировки текста задания, записи его в раздел библиотечного набора данных, считывания раздела из библиотеки, обмена сообщениями с оператором ЭВМ.

Граф состояний экрана дисплея в системе CJE приведен на рис. 4.3.

Система CJE предлагает для использования около 30 различных команд. Для удобства работы эти команды объединены в группы, называемые режимами: 1) формирования задания; 2) выполнения задания; 3) просмотра печати результатов или раздела; 4) корректировки задания; 5) записи задания; 6) выполнения библиотечных функций; 7) выполнения специальных функций. Сразу после подключения пользователя к системе на его экран выдается «меню» с указанием номера и названия каждого режима.

Это состояние дисплея пользователя в дальнейшем будем называть исходным состоянием. Выбрав требуемый режим, пользователь указывает его номер, после чего система выдает на экран очередное «меню» с указанием команд, входящих в этот режим.

Из графа состояний экрана видно, что по завершении работы в любом из режимов система автоматически переводит дисплей в исходное состояние.

Режим формирования задания. Он объединяет три функции: 1, ЭК — формирование задания с экрана; 1, ВД — формирование

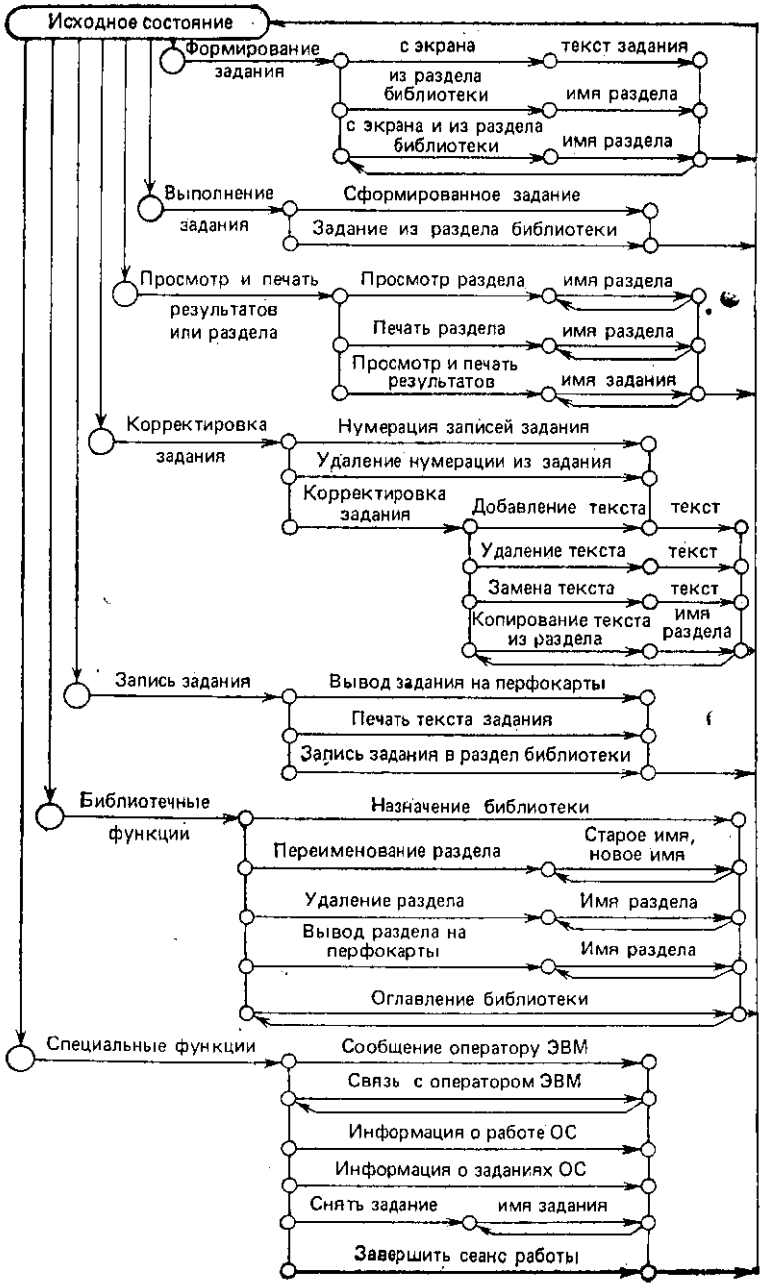


Рис. 4.3. Граф состояний экрана системы CJE

задания с экрана и из библиотеки; 1, (MEMBER) — формирование задания из раздела библиотеки.

Функция формирования задания производится на рабочем файле, который создается в начале сеанса работы пользователя с системой. В начале работы с каждой функцией режима на экран выдается подсказка о правилах работы при использовании этой функции. Ознакомившись с ними, пользователь может приступить непосредственно к формированию задания. В том случае, когда пользователь при работе с системой допускает ошибку, на экран дисплея выдается предупреждающее сообщение. По завершении работы каждой функции на экран пользователя выдаются информационное сообщение и подсказка, позволяющая продолжить диалог.

Режим выполнения задания. Он объединяет две функции: 2, ЗД — выполнение сформированного задания; 2, (MEMBER) — выполнение задания из раздела библиотеки, где MEMBER — имя раздела библиотечного набора данных, содержащего текст задания.

Функции этого режима позволяют включить задания пользователя во входной поток ОС ЕС ЭВМ.

Задание должно быть либо заранее помещено в раздел библиотечного набора данных, либо сформировано на рабочем файле с помощью одной из функций режима «Формирование задания».

После постановки ОС задания в очередь на экран дисплея пользователя выдается подтверждающее сообщение и система переходит в исходное состояние.

Режим просмотра и печати результатов или раздела. Режим объединяет три функции:

3, РЗ — просмотр раздела;

3, ПЧ — печать раздела;

3, ЗД — просмотр результатов выполнения задания.

Первые две функции позволяют, не изменяя содержимого рабочего файла, просмотреть на экране дисплея или вывести на печать содержимое одного или нескольких разделов библиотечного набора данных. Третья функция дает возможность просмотреть на экране содержимое набора данных, принадлежащего пользователю и находящегося в любой из выходных очередей системы ОС ЕС. На экран выводятся 80 символов каждой записи набора данных начиная с позиции, указанной пользователем. После просмотра всех записей выходного набора данных пользователю предлагается: 1) перейти в исходное состояние; 2) повторить просмотр набора данных: 3) удалить набор данных из выходной очереди; 4) вывести набор данных на печать и удалить его из выходной очереди. В двух последних случаях по завершении работы система переходит в исходное состояние.

Режим корректировки задания. Этот режим работы системы обеспечивает внесение изменений в сформированное на рабочем файле задание в соответствии с функциями: 4, УД — удалить нумерацию из текста задания; 4, НМ — пронумеровать записи задания; 4 ЗД — корректировка задания.

Функция удаления нумерации обеспечивает заполнение пробелами позиций 73 ... 80 каждой записи задания пользователя.

Функция нумерации задания заносит в те же позиции каждой записи ее порядковый номер от 00000001 до 99999999.

Функция корректировки задания позволяет вносить изменения в содержимое рабочего файла пользователя как непосредственно с помощью клавиатуры дисплея, так и с помощью подкоманд функций, список которых выдается на экран в начале работы.

Подкоманды функции корректировки задания позволяют уда-

лять, добавлять и заменять записи в рабочем файле. Добавляемые записи могут или вводиться непосредственно с экрана, или копироваться из раздела библиотечного набора данных. В процессе корректировки рабочий файл можно «листать» как вперед, так и назад на произвольное число записей.

Исправленное задание пользователя может быть в дальнейшем вновь представлено на выполнение или сохранено в разделе библиотечного набора данных.

Режим записи задания. Функции режима позволяют переписывать создаваемый в процессе работы текст в разделы библиотек, а также выводить его на печать.

Режим объединяет три функции: 5, ПК — вывод задания на перфокарты; 5, ПЧ — печать текста задания; 5, (MEMBER) — запись в раздел библиотеки, где MEMBER — имя раздела, в который будет записан текст задания.

Эти функции не изменяют содержимого рабочего файла и могут быть использованы неоднократно в процессе работы. По завершении выполнения каждой из функций система переходит в исходное состояние.

Режим выполнения библиотечных функций. В системе СЛЕ предусмотрено пять функций, использующих при своей работе личные библиотеки пользователя: 6, НЗ — назначение библиотеки; 6, ПР — переименование раздела; 6, УД — удаление раздела; 6, ПК — вывод раздела на перфокарты; 6, ОГ — просмотр оглавления библиотеки.

После нормального завершения работы с каждой из этих функций система переходит в исходное состояние.

Режим выполнения специальных функций. Режим объединяет все оставшиеся функции системы: 7, СБ — сообщение оператору ЭВМ; 7, СВ — связь с оператором ЭВМ. Эти функции обеспечивают передачу на главную консоль ОС ЕС сообщения пользователя. После получения ответа оператора пользователь может выйти в исходное состояние или продолжить диалог; 7, СТ — информация о работе ОС. Эта функция выдает на экран дисплея информацию о заданиях, выполняемых в данный момент операционной системой. О каждом задании сообщается: 1) имя задания; 2) имя шага задания (шага процедуры), выполняемого в данный момент; 3) границы оперативной памяти, в которых он выполняется; 4) время, оставшееся до завершения пункта задания; 5) количество подключенных подзадач; 7, ЗД — информация о заданиях ОС. Эта функция выдает на экран дисплея информацию о заданиях во входных и выходных очередях ОС. На экран выдаются также имена классов ОС и имена заданий, находящихся в этих классах; 7, (REN) — снять задание с именем REN (пользователь может снять задание, имя которого составлено таким образом, что в его начале находится идентификатор пользователя. При попытке снять чужое задание на экран выдается предупреждающее сообщение и никаких действий с этим заданием не производится); 7, КС — завершить сеанс работы (эта команда используется для окончания работы с системой СЛЕ. После ее выдачи на диске сохраняются информация и характеристика сеанса работы, уничтожается рабочий файл пользователя и посылается сообщение оператору ЭВМ об окончании работы данного пользователя с системой СЛЕ).

КРАТКИЕ ВЫВОДЫ

Проектирование сложного объекта невозможно выполнить полностью автоматически без участия проектировщика. Диалоговые системы, обеспечивающие взаимодействие проектировщика с ЭВМ, являются обязательной составной частью современных САПР. Диалог есть последовательность обменов сообщениями между ЭВМ и человеком. Сообщения могут быть входными и выходными, информационными, запросами и ответами. Диалог может иметь формы сценарную, таблицы, директивы и на ограниченном естественном языке. Важным понятием диалогового взаимодействия является граф состояний экрана дисплея.

Программное обеспечение диалога представлено диалоговыми системами коллективного пользования, диалоговыми методами доступа, подпрограммами обращения к терминалу. Диалоговые методы доступа в составе ОС ЕС — графический, базисный телекоммуникационный, общий телекоммуникационный — предоставляют разработчику САПР различные программные средства для организации диалога. В развитых САПР, построенных на базе ЕС ЭВМ, рекомендуется использовать режим разделения времени (РРВ), а также систему телеобработки данных «КАМА». Для подготовки заданий широко используются системы диалогового ввода заданий типа JEC, CJE, DУВЗ.

Глава 5

ПАКЕТЫ ФУНКЦИОНАЛЬНОГО ПРОЕКТИРОВАНИЯ

§ 5.1. Структура пакета функционального проектирования на макроуровне

Большой класс задач АП составляют анализ во временно́й области и параметрическая оптимизация объектов при функциональном проектировании на макроуровне. Большинство таких объектов описывается системами обыкновенных дифференциальных уравнений (ОДУ). Для анализа этих объектов широко используются методы:

1) узловых потенциалов для формирования математической модели системы;

2) неявные численного интегрирования ОДУ;

3) Ньютона для итерационного решения системы нелинейных алгебраических уравнений;

4) Гаусса (или его модификации) для решения систем линейных алгебраических уравнений.

Пакеты функционального проектирования, реализующие данное математическое обеспечение и его современные модификации, — одни из наиболее сложных в САПР. Самыми яркими представителями таких пакетов служат пакеты схемотехнического проектирования, воплощающие в себе, как правило, все передовые достижения в области математического обеспечения анализа и оптимизации и предназначенные для решения задач большой размерности.

■ Примечание. Далее изложение материала будет ориентировано в основном на такие пакеты. Однако это ни в коей мере не сказывается на его общности.

К основным требованиям, предъявляемым к пакетам функционального проектирования, наряду с рассмотренными выше (см. Введение) также относятся:

приемлемые затраты ресурсов ЭВМ для анализа объектов большой размерности (описываемых жесткими системами ОДУ, содержащих тысячи переменных);

возможность анализа проектируемых технических систем с учетом разнообразия и взаимовлияния протекающих физических процессов, т. е. одновременный анализ процессов различной физической природы;

живучесть пакета, подразумевающая длительный срок его эффективной эксплуатации.

В составе пакета функционального проектирования выделяют три основные части (подсистемы): 1) языковую; 2) обрабатывающую; 3) монитор.

■ Примечание. Функции монитора пакета функционального проектирования схожи с функциями монитора САПР и описаны в § 1.2.

В пакете может также присутствовать подсистема управления локальными базами данных и диалогового взаимодействия с пользователем. Локальные базы данных организуются наиболее рациональным для данного пакета способом и используются для хранения информации, не подлежащей обобществлению с другими пакетами проектирования. Они могут применяться также в качестве буферов между пакетом проектирования и основной базой данных САПР, если характеристики последней не удовлетворяют данный пакет по какой-либо причине. Наличие в пакете проектирования собственных средств диалога может потребоваться для организации к нему множественного доступа.

Языковая подсистема. Она организует общение пользователя с пакетом функционального проектирования посредством проблемно-ориентированных входных языков, т. е. языков, близких к профессиональным языкам пользователей. При разработке пакета проектирования на макроуровне целесообразно предусмотреть возможность его использования в различных организациях и предметных областях, это способствует широкому внедрению автоматизированных средств в практику проектирования и быстрой окупаемости затрат на разработку программного обеспечения. Такая возможность реализуется, в частности, за счет создания пакета проектирования, открытого по отношению к входным языкам.

Лингвистическая открытость пакета функционального проектирования основывается на концепции использования языков двух уровней. *Верхний уровень* занимают **входные языки, ориентированные на определенные груп-**

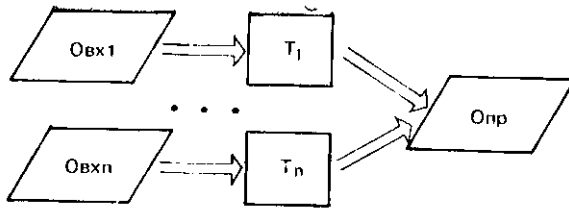


Рис. 5.1. Схема трансляции на промежуточный язык:

$O_{вхi}$ — описание объекта проектирования на i-м входном языке; T_i — i-й транслятор; $O_{пр}$ — описание на промежуточном языке

пы пользователей и дающие средства лаконичного и удобного описания ограниченных классов объектов. *Нижний уровень* составляет единый промежуточный язык, более универсальный, но менее удобный для пользователя.

Для перевода описаний на входных языках в описание на промежуточном языке используются соответствующие трансляторы, как это показано на рис. 5.1.

Входные трансляторы САПР согласно существующей теории машинного перевода формальных языков принято рассматривать состоящими из трех основных блоков: 1) лексического; 2) синтаксического; 3) генератора кода. Все блоки имеют доступ к общему набору массивов и таблиц.

Лексический блок осуществляет распознавание базовых элементов предложений входного языка, т. е. разбивает входные фразы на отдельные слова, из которых они состоят. Каждому слову в описании на входном языке ставится в соответствие *лексема,* размещаемая в одном из массивов транслятора. Лексемы состоят из двух частей, называемых классом и значением. Например, имя математической модели элемента проектируемого объекта, задаваемое на входном языке как последовательность русских или латинских букв, преобразуется лексическим блоком в лексему, имеющую класс «имя модели»; значением этой лексемы служит указатель на строку таблицы паспортов математических моделей элементов.

В задачу *синтаксического блока* входит перевод последовательности лексем, сформированной лексическим блоком, в последовательность атомов. Атомы также

состоят из класса и значения и являются, по сути, лексемами слов того промежуточного языка, на который осуществляется перевод. Одновременно с трансляцией производится проверка правильности построения предложений входного языка (синтаксический контроль). Класс и значение атомов, порядок их следования соответствуют *лексике* (набору слов) и *синтаксису* промежуточного языка.

Генератор кода преобразует атомы в слова и фразы промежуточного языка.

Взаимодействие блоков в трансляторе зависит от особенностей входного и промежуточного языков. Так, если эти языки синтаксически близки, то можно построить транслятор по схеме с одним проходом. В таких трансляторах вызов блоков осуществляется циклически: лексический блок вырабатывает лексему; она передается синтаксическому блоку, который порождает один или несколько атомов; генератор кода «развертывает» эти атомы в слова промежуточного языка; затем управление вновь передается лексическому блоку и т. д., пока не будет исчерпано описание на входном языке. Если языки синтаксически различны, то возникает необходимость в организации иной схемы взаимодействия блоков транслятора, при которой они включаются в работу однократно: сначала лексический блок выдает полную цепочку лексем, затем синтаксический блок переводит ее в последовательность атомов, преобразуемую генератором кода на последнем этапе в предложения промежуточного языка. Такой транслятор называется трехпроходным. Многопроходные трансляторы характеризуются большими затратами памяти, необходимой для хранения всей цепочки лексем и (или) атомов.

Внедрение нового, более совершенного программного обеспечения в промышленность обычно связано с необходимостью освоения пользователями новых входных языков, что требует достаточно больших затрат времени. Пакеты функционального проектирования, использующие данную двухуровневую языковую концепцию, могут быть внедрены на предприятия совершенно безболезненно, если будут оснащены трансляторами с входных языков тех программ, которые эксплуатировались раньше.

■ Примечание. Адаптация такого ПО к объектам иной физической природы требует лишь замены библиотеки подпрограмм математических моделей элементов и создания транслятора с

нового входного языка, разработанного в соответствии с терминологией, сокращениями, ГОСТами, соглашениями, принятыми в данной предметной области.

Возможны два подхода к построению пакетов проектирования с двумя языковыми уровнями, различающиеся сложностью разработки трансляторов. *Первый подход* реализован в программном комплексе КРОСС; в нем универсальность промежуточного языка обеспечивается его построением как языка описания математических моделей в виде систем уравнений. *Второй подход* в качестве промежуточного языка использует язык описания линейных графов, отображающих структуру динамических объектов с сосредоточенными параметрами, в сочетании со средствами описания произвольных функциональных зависимостей. Перевод описания структуры проектируемого объекта на язык графов проще, чем в систему уравнений, поэтому второй подход следует считать в общем случае предпочтительным при создании проблемно-ориентированных пакетов проектирования, открытых к входным языкам.

Трансляторы с входных языков должны обеспечивать возможность интерактивного режима работы, предоставлять пользователю широкие возможности по редактированию входных данных, оперативному обнаружению и исправлению синтаксических ошибок в конструкциях языка. Работы по созданию таких трансляторов могут быть в высокой степени автоматизированы при использовании специальных генераторов трансляторов, включаемых в инструментальную подсистему САПР.

Промежуточный язык пакета проектирования, как и его входные языки, состоит из двух подмножеств: языка описания объекта (ЯОО) и языка описания задания на проектирование (ЯОЗ). Первый является непроцедурным и служит для предоставления пакету сведений о структуре и элементах проектируемой технической системы. Второй необходим для указания пакету задания на расчет и должен быть процедурным.

При исследовании физического объекта во временно́й области ЯОЗ должен содержать по крайней мере четыре основных оператора: 1) расчета статического состояния; 2) расчета переходных процессов; 3) многовариантного анализа; 4) оптимизации. Для реализации режима интерактивного взаимодействия пользователя с пакетом не только на этапе подготовки исходного описания, но и на этапе расчета по одному из этих операторов

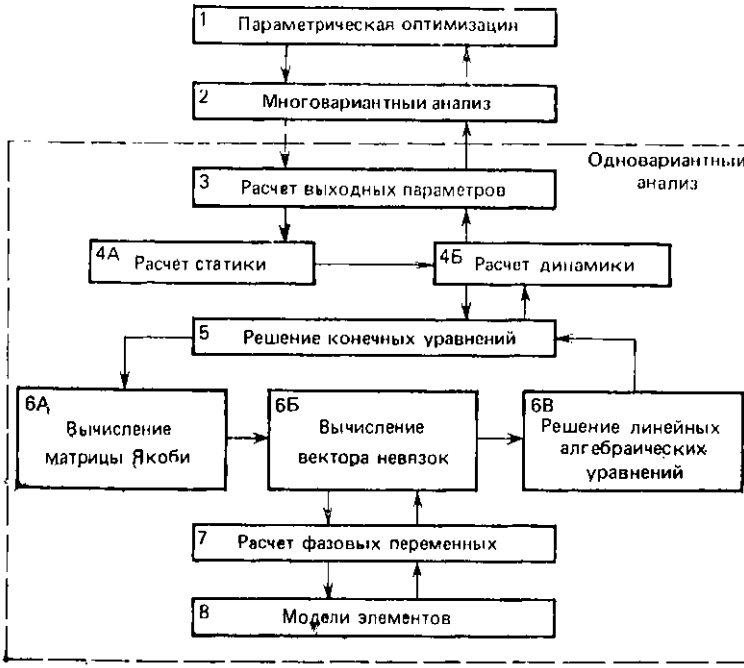


Рис. 5.2. Схема маршрута взаимодействия подпрограмм пакета функционального проектирования

в ЯОЗ должны быть включены дополнительные операторы прерывания и изменения хода вычислительного процесса, корректировки исходных данных, индикации любых переменных, массивов и трасс, расстановки точек останова и т. п.

■ Примечание. В § 5.3 дано краткое описание промежуточного языка программного комплекса ПА-6.

Обрабатывающая подсистема. Она осуществляет непосредственное решение проектных задач. Именно эта подсистема реализует заложенное в пакет математическое обеспечение. На рис. 5.2 схематично представлена структура обрабатывающей подсистемы, отражающая маршрут взаимодействия подпрограмм при решении задачи параметрической оптимизации непрерывного объекта по временно́й области. Она характеризуется ярко выраженной вложенностью модулей и, следовательно, разной частотой их использования: чем ниже уровень, занимаемый модулем, тем чаще он используется. Так, ти-

новые значения частот использования модулей уровня 6 относительно уровня 4 имеют порядок 10^3.

Пакеты функционального проектирования как программы, обрабатывающие предложения и директивы входного языка, являются *языковыми процессорами*. Существует два типа языковых процессоров: интерпретаторы и трансляторы. Структура пакета проектирования, построенного по принципу интерпретации, укрупненно показана на рис. 5.3. Его языковая подсистема *ЯП* воспринимает описание проектируемого объекта и задания на его расчет на входном (или промежуточном) языке и порождает (обычно в ОП) структуры данных, содержащих описание объекта и решаемой задачи в удобной для машинной обработки форме *ВПД*. Эта информация в дальнейшем интерпретируется обрабатывающей подсистемой (*ОБрП*).

■ Примечание. Важно, что еще до начала работы пакета в нем имеется полностью скомпонованная обрабатывающая подсистема. В ее состав входят универсальные модули, ориентированные на решение всего круга задач, допускаемых пакетом. Вызов необходимых модулей и их настройка на конкретные данные осуществляются непосредственно на этапе счета. Делается это обычно многократно (тем чаще, чем ниже уровень, занимаемый модулем). В дальнейшем такой пакет для краткости будем называть *пакетом интерпретатором*.

Пакет функционального проектирования, построенный с использованием принципа трансляции, до начала расчета не имеет полной и скомпонованной обрабатывающей подсистемы. Языковая подсистема такого пакета воспринимает описание на промежуточном (входном) языке и в качестве выхода выдает на *объектном языке* программу(ы) вычислений в соответствии с входным описанием. Объектным языком может быть любой алгоритмический язык высокого уровня (ПАСКАЛЬ, ФОРТРАН, ПЛ/1 и др.) или язык машинных команд. Если в качестве объектного используется машинный язык, то полученные программы будут сразу готовы к выполнению. Если же объектный язык – язык высокого уровня, то дополнительно необходим перевод объектной программы в машинные команды. Транслятор, выходом которого являются объектные программы в машинных командах, называют *компилятором*.

Обычно полученные в результате работы транслятора объектные программы реализуют самые важные (наиболее критичные по временным затратам), но не все процедуры и алгоритмы, необходимые для расчета. Тогда

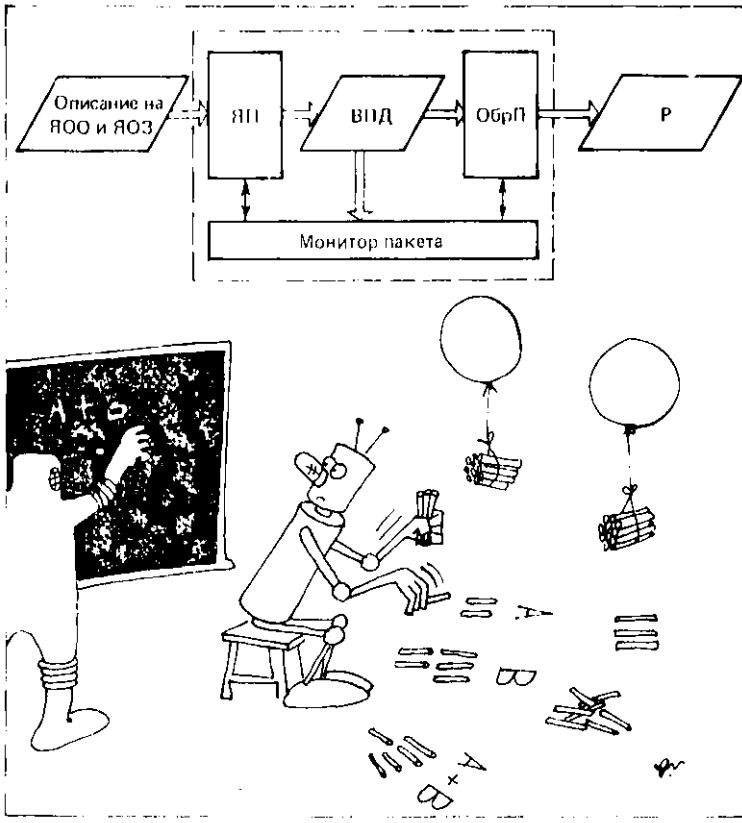


Рис. 5.3.

последние объединяются каким-либо образом с необходимыми для данного конкретного варианта расчета подпрограммами из постоянных библиотек пакета проектирования. Эти подпрограммы реализуют интерпретирующие вычисления и могут быть подпрограммами моделей элементов, методов интегрирования, расчета выходных параметров и т. п. Полученная в результате объединения сгенерированных и библиотечных модулей обрабатывающая подсистема пакета проектирования называется *рабочей программой,* ее выполнение непосредственно служит получению желаемого результата.

Выбор типа языкового процессора. В настоящее время при создании пакетов проектирования находят приме-

нение оба принципа, хотя чаще используется принцип интерпретации, а пакеты-трансляторы сочетают в себе оба этих принципа, причем в разных пакетах в различной степени. Так, в программе многоуровневого моделирования MACRO генерируется на языке ФОРТРАН только подпрограмма, реализующая алгоритм Гаусса для решения системы линейных алгебраических уравнений, в пакете КРОСС в виде объектной программы на языке ПЛ/1 оформляются уравнения математической модели всей проектируемой системы, в программном комплексе ПА-6 компиляции подлежит большинство модулей нижних уровней структуры обрабатывающей подсистемы. Пакеты-трансляторы, а тем более компиляторы, обычно более сложны в разработке, чем интерпретаторы.

Применение для создания ПО того или иного подхода оказывает *ключевое* влияние на вычислительную эффективность пакета. Объектные программы, полученные в результате работы языковой подсистемы пакета-транслятора, в большинстве случаев являются значительно более быстродействующими по сравнению с пакетами-интерпретаторами (при условии, конечно, одинаковости используемого математического обеспечения). Объясняется это тем, что объектная программа представляет собой линейную программу, содержащую минимально необходимое количество команд, адресная часть которых настроена непосредственно на используемые элементы данных. Все операции, нужные для определения оптимальной последовательности команд и их операндов, выполняются однократно на этапе трансляции. Разность в быстродействии иллюстрируется рис. 5.4, *а*, где представлены графики затрат машинного времени T на однократное выполнение процедуры прямого хода Гаусса интерпретирующей и скомпилированной подпрограммами при учете разреженности матрицы коэффициентов системы линейных алгебраических уравнений (в эксперименте использовался прямой способ кодирования разреженных матриц, среднее количество ненулевых элементов справа от диагонали составляло 2,2), N — порядок решаемой системы ЛАУ. Бо́льшие затраты для интерпретирующей подпрограммы обусловливаются накладными расходами, связанными с определением координат элементов разреженной матрицы и преобразованием индексов массивов в действительные адреса. Выбор тех или иных способов кодирования разреженных матриц позво-

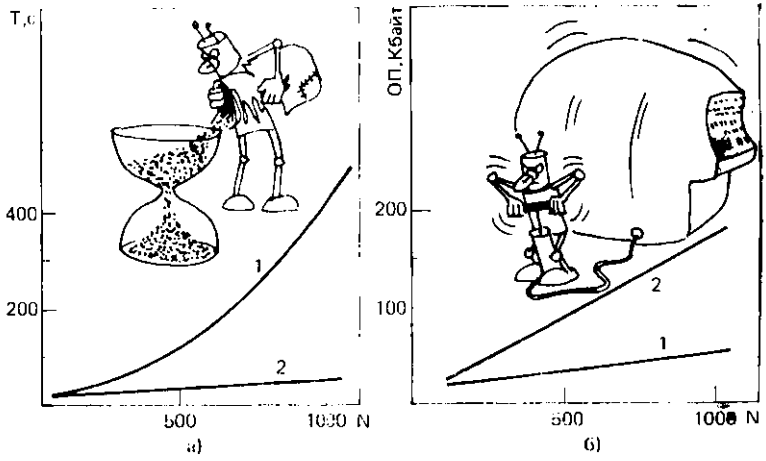


Рис. 5.4.

ляет сократить количество операций, необходимых для идентификации ненулевых элементов, но не настолько, чтобы значительно сблизить графики.

■ Примечание. Процедура решения системы линейных алгебраических уравнений — одна из наиболее «длительных» и часто используемых при анализе как во временной, так и в частотной областях.

В общем случае скомпилированный вариант любой процедуры расчета требует большего объема ОП, чем ее интерпретирующий вариант. На рис. 5.4, *б* даны графики зависимости затрат памяти от порядка решаемой системы уравнений N для скомпилированной и интерпретирующей процедур прямого хода Гаусса. Наклон прямой, характеризующей зависимость затрат памяти от порядка N для скомпилированного варианта, больше наклона прямой, характеризующей ту же зависимость для интерпретирующего варианта. Однако разница в затратах памяти для систем с порядком не более 1000 несущественна для современных ЭВМ.

В задачах большей размерности несложно организовать такую дисциплину обмена с внешней памятью, которая практически не будет сказываться на реальном времени выполнения алгоритма Гаусса. Линейный характер объектной программы способствует ее весьма эффективному выполнению на ЭВМ с виртуальной памятью.

При исследовании затрат ОП необходимо принимать в расчет и архитектурные особенности пакета проектирования, построенного по тому или иному принципу. Пакетом-транслятором в рабочую программу включаются только необходимые для данного конкретного расчета подпрограммы. В пакетах-интерпретаторах обычно все подпрограммы, объединенные в обрабатывающую подсистему, должны находиться в ОП ЭВМ, так как непосредственно до момента обращения к ним неизвестен конкретный набор требуемых подпрограмм. Разнесение модулей по оверлейным сегментам возможно не всегда (случай подпрограмм моделей элементов), а когда это возможно, проблема избыточного объема ОП все равно полностью не решается, поскольку занятым считается объем памяти, требуемый наибольшей из подпрограмм одного уровня оверлейной структуры.

Использование динамической загрузки подпрограмм в ОП не всегда удобно, так как в этом случае исключается возможность использования общих областей и глобальных структур данных. Другой аспект вопроса о затратах ОП — распределение ОП под структуры данных. В пакетах-трансляторах память под структуры данных рабочей программы выделяется статически и строго необходимого размера. В обрабатывающей подсистеме интерпретатора необходимо динамическое распределение памяти, однако его использование связано с увеличением затрат машинного времени.

Рассмотренные факторы объясняют то обстоятельство, что на практике затраты ОП для многофункциональных пакетов-интерпретаторов, состоящих из большого количества модулей, оказываются бóльшими, чем для таких же пакетов-трансляторов.

Важным преимуществом пакетов-трансляторов является следующее. Фаза собственно трансляции описания на промежуточном языке в объектную программу включается однократно и занимает малую долю от всех затрат времени ЭВМ на анализ и параметрическую оптимизацию физического объекта. На этом этапе языковая подсистема пакета, построенного по принципу трансляции, выполняет многие из тех функций и процедур, которые в обрабатывающей подсистеме пакета-интерпретатора выполняются многократно уже на этапе расчета и поэтому являются источником больших «накладных» расходов. Вследствие этого на эффективность пакетов-трансляторов оказывают слабое влияние способы пред-

ставления и обработки структур данных, используемых их языковой подсистемой, т. е. относительно мягкие требования, предъявляемые к языковой подсистеме пакета-транслятора, позволяют в ряде случаев использовать более простые алгоритмы и межмодульные интерфейсы по сравнению с теми, которые реализуются в пакетах-интерпретаторах.

Проведенное сравнение двух подходов к построению пакетов функционального проектирования на макроуровне дает возможность сделать вывод о предпочтительности использования принципа трансляции. Но, как отмечалось ранее, трансляторы различаются объектными языками, на которых представляются сгенерированные ими модули рабочей программы. Трансляторы на алгоритмический язык высокого уровня весьма просты и эффективны. В них удобно обрабатывать произвольные функциональные зависимости, описание которых задается пользователем на входе языковой подсистемы на алгоритмическом языке, совпадающем с объектным. К таким описаниям достаточно добавить операторы объявления переменных, пролога и эпилога, реализующие стандартный интерфейс, и полученный текст можно передавать на выход. Использование широко распространенного алгоритмического языка в качестве объектного обеспечивает машинную независимость пакета проектирования. Необходимость дополнительной обработки объектных программ компиляторами из состава используемой ОС и низкая вычислительная эффективность полученной рабочей программы по сравнению с рабочей программой на машинном языке являются недостатками данного подхода. Поэтому при разработке языковой подсистемы пакета-транслятора целесообразно предусмотреть возможность смены набора модулей, ответственных за генерацию текста объектных программ на том или ином языке. При использовании принципа информационной локализованности сделать это несложно.

Пакеты-компиляторы, выходом языковой подсистемы которых являются программы на машинном языке, могут различаться типом загрузки объектной программы в ОП. Например, в пакете ПАРМ используется схема «компиляция — выполнение», в нем генерируемые машинные команды помещаются в специально отведенную область ОП. По окончании генерации всей объектной программы ей передается управление. Такой подход характеризуется большими затратами ОП, так как языковая подсисте-

ма пакета и вырабатываемая им объектная программа располагаются в памяти одновременно. Другой недостаток этого подхода — необходимость разрешения ссылок между сгенерированными модулями и «постоянными» подпрограммами каким-либо образом в самом пакете проектирования. От перечисленных недостатков свободны пакеты-компиляторы, выходом языковых подсистем которых являются объектные программы в виде, пригодном для обработки редактором связей из состава ОС.

Выбор процедур для генерации. Использование принципа трансляции связано с усложнением организации пакета функционального проектирования, но не всегда приводит к заметному увеличению быстродействия. Поэтому имеет смысл подвергать генерации только те процедуры решения, для которых скомпилированный вариант значительно более эффективен, чем интерпретирующий.

На выбор способа построения программы любой процедуры оказывают влияние следующие критерии: 1) частота исполнения процедуры при расчете; 2) способ доступа к данным, используемый в процедуре; 3) степень инвариантности процедуры к структуре и размерности проектируемого объекта.

Затраты машинного времени подпрограммой, редко выполняемой при расчете проектируемого объекта (десятки раз), слабо сказываются на общих затратах на весь расчет (если, конечно, время однократного выполнения такой подпрограммы соизмеримо с временем выполнения большинства остальных подпрограмм), но ее объем может значительно влиять на объем всей памяти, необходимой для расчета объекта.

Сгенерированный вариант процедуры с последовательным доступом к данным требует почти таких же затрат времени ЭВМ, как и ее интерпретирующий вариант, но затраты памяти для обоих вариантов могут различаться многократно. Например, отношение объема скомпилированной программы для процедуры суммирования векторов, не содержащих ненулевые элементы, к объему такой же, но интерпретирующей программы, пропорционально размерности этих векторов.

■ Примечание. Пример же более высокого быстродействия скомпилированных процедур с прямым доступом к данным приведен выше (прямой ход алгоритма Гаусса при решении системы линейных алгебраических уравнений с сильно разреженной матрицей коэффициентов).

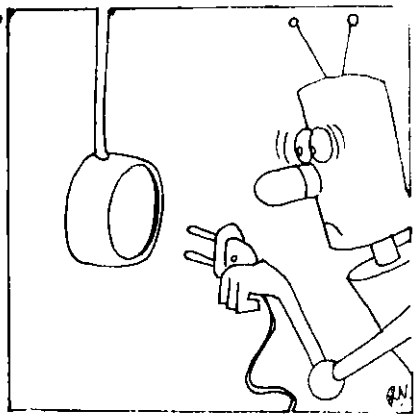
Иногда целесообразно, где это возможно, процедуры с прямым доступом к памяти реализовывать в виде модулей интерпретирующего и генерируемого. Собственно алгоритм процедуры воплощается в интерпретирующем модуле, который использует наиболее подходящее для этого представление данных. Единственной функцией *генерируемого модуля* является создание регулярных структур данных, удобных для *интерпретирующего модуля.*

В анализе технических систем участвует ряд универсальных процедур, инвариантных к виду и размеру системы, вычисления в которых всегда проводятся по одному и тому же алгоритму для постоянного количества данных и постоянных размеров массивов. К таким процедурам относятся, например, математические модели элементов систем. Генерация, выполняемая при каждом расчете новой проектируемой системы, для таких процедур бессмысленна. Процедуры должны быть однократно тщательно запрограммированы и помещены в постоянную библиотеку пакета.

Рассмотренные выше критерии позволяют, например, выделить в иерархической структуре математического обеспечения пакета функционального проектирования (см. рис. 5.2) элементы, подлежащие генерации (алгоритм Гаусса, расчет матрицы Якоби и вектора невязок, обращение к подпрограммам моделей элементов). Все остальные процедуры и алгоритмы, участвующие в анализе и параметрической оптимизации проектируемого объекта, должны быть реализованы в интерпретирующем виде и храниться в постоянных библиотеках пакета проектирования.

Открытость пакетов функционального проектирования. Важной характеристикой пакетов проектирования, в решающей степени влияющей на живучесть пакета и на затраты по его эксплуатации и сопровождению, является их *открытость по отношению к элементам математического обеспечения* (методам интегрирования, моделям элементов, алгоритмам расчета внешних воздействий и выходных параметров, методам многовариантного анализа и оптимизации). Степень открытости пакета проектирования характеризуется степенью сложности (а следовательно, и затратами) включения в него новых элементов математического обеспечения.

В зависимости от степени открытости пакетов проектирования их с точки зрения



пользователя можно разделить на *закрытые, ограниченно-открытые, открытые.*

Закрытый пакет проектирования — пакет, реализация в котором нового элемента математического обеспечения требует перепрограммирования большого числа входящих в его состав модулей. Такая модификация пакета под силу только его разработчику, а затраты на нее могут быть соизмеримы с затратами на создание нового пакета.

В ограниченно открытом пакете проектирования реализация нового метода или алгоритма состоит во включении в его состав нового модуля. Все изменения, связанные с подключением новой подпрограммы, локализованы в одном из «старых» модулей и проводятся по формальным правилам, задаваемым соответствующей инструкцией. Такие изменения могут быть выполнены квалифицированным пользователем-программистом самостоятельно. Включение новых модулей в ограниченно открытый пакет проектирования и их отладка связаны с большими накладными расходами, требуют от пользователя глубокого знания структуры ПО и особенностей используемой ОС. Такие пакеты проектирования не защищены от внесения ошибок пользователя.

Открытый пакет проектирования характеризуется тем, что его расширение не требует каких-либо изменений в уже существующих модулях. Связь новых и старых модулей происходит с помощью паспортов, описывающих интерфейсы модулей. Такие паспорта заполняются для каждого вновь включаемого модуля и располагаются в локальной БД пакета проектирования. Создание открытых пакетов проектирования требует предварительной структуризации математического обеспечения, основанной на анализе процедурного состава известных алгоритмов анализа и оптимизации (см. рис. 5.2). Полученная обобщенная структура вычислительно-

го процесса позволяет планировать модульную структуру обрабатывающей подсистемы пакета и связи модулей по информации. Важный принцип создания открытых пакетов — принцип модульности, означающий согласование структуры математического и программного обеспечений, подразумевающее, в частности, воплощение каждого элемента математического обеспечения в отдельном модуле (ограниченном числе модулей).

Так как связь модулей по управлению и информации в открытых пакетах должна осуществляться с помощью системы паспортов, то в паспорте модуля содержатся сведения об информационных массивах и переменных, используемых в нем, о его объеме, значениях параметров, используемых «по умолчанию» и т. п. В открытых пакетах-интерпретаторах вызов в ОП модулей происходит по командам динамической загрузки, обращению к модулю предшествует интерпретация его паспорта, считанного из локальной БД. Но из загруженных таким образом модулей исключается вызов каких-либо подпрограмм, а к ним самим невозможно обращение по команде на языках высокого уровня. В этом основная трудность создания открытых пакетов-интерпретаторов.

В пакетах-транслляторах подобных трудностей не возникает, так как в них объединение сгенерированных и библиотечных модулей в рабочую программу производится системным редактором связей, разрешающим стандартно все необходимые внешние ссылки. Обращение из скомпилированных подпрограмм к библиотечным строится в полном соответствии с интерфейсом последних, описанным в паспорте. Если библиотечной подпрограмме требуются рабочие массивы, то они выделяются в рабочей программе статически и строго необходимого объема.

Пакетный и диалоговый режимы работы пакетов функционального проектирования. На макроуровне пакеты проектирования должны допускать пакетный и диалоговый режимы работы.

Пакетный режим необходим при проектировании сложных технических систем, одновариантный анализ которых может требовать десятков минут машинного времени. Для реализации такого режима функционирования пакетов-интерпретаторов необходим их запуск автономно от монитора САПР средствами ОС в качестве одной из фоновых задач ЭВМ. Любая ошибка, допущенная пользователем во входном описании, приводит к необходимости перезапуска пакета проектирования. В этом

отношении пакет-транслятор предоставляет пользователю больше возможностей.

Диалоговый режим работы входных трансляторов и языковой подсистемы позволяет пользователю оперативно обнаруживать и устранять синтаксические (и даже некоторые смысловые) ошибки в описании на входном (промежуточном) языке. После генерации объектных подпрограмм и их объединения с библиотечными в единую рабочую программу последняя может быть направлена по желанию пользователя в пакетную обработку, а пользователь будет иметь возможность продолжить работу с данным пакетом или с любой другой проектирующей подсистемой САПР. Об окончании пакетного задания монитор САПР уведомляет пользователя, который может средствами выходного языка пакета проектирования обработать необходимым ему образом результаты расчета.

■ Примечание. При необходимости несложно организовать передачу сгенерированных языковой подсистемой пакета-транслятора объектных программ (на языке высокого уровня) для дальнейшей обработки и расчета на мощном вычислительном комплексе. Построение входных трансляторов и языковой подсистемы реентерабельными обеспечивает множественный доступ пользователей в данном режиме работы пакета.

Диалоговый режим функционирования пакетов проектирования необходим при выполнении проектных операций, формализованных в малой степени.

При реализации диалогового режима в пакетах, построенных по принципу трансляции, некоторое неудобство для пользователей представляет временна́я задержка между этапами ввода исходного описания и началом расчета, связанная с необходимостью двухпроходной трансляции (с входного языка на промежуточный и с промежуточного в объектные подпрограммы) и компоновки рабочей программы. Однако она окупается повышенной скоростью расчета по сравнению с пакетом-интерпретатором.

§ 5.2. Программный комплекс ПА-6 для функционального проектирования динамических объектов

Программный комплекс ПА-6 предназначен для анализа и параметрической оптимизации технических объектов, описываемых системами ОДУ. Основными элементами ма-

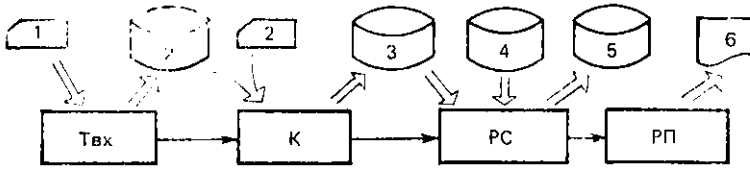


Рис. 5.5. Общая схема функционирования программного комплекса ПА-6:

$T_{вх}$ — входной транслятор; *К* — компилятор; *РС* — редактор связей; *РП* — рабочая программа

тематического обеспечения анализа в ПА-6 являются методы узловых потенциалов, комбинированный неявно — явный интегрирования ОДУ, Ньютона, Гаусса. На основе этих методов в комплексе реализованы современные диакоптические алгоритмы анализа (латентного подхода, раздельного итерирования, «временно́го» анализа), позволяющие эффективно моделировать объекты большой размерности, содержащие сотни и тысячи фазовых переменных. Использование этих методов требует разбиения (декомпозиции) анализируемых объектов на фрагменты. В ПА-6 такое разбиение должен осуществлять пользователь по функциональному признаку. Кроме того, предусмотрена возможность совместного анализа объектов с непрерывными и дискретными моделями.

Комплекс ПА-6 построен по принципу компиляции объектных модулей на машинном языке и ориентирован на использование в рамках операционных систем ЕС ЭВМ.

Рассмотрим принципы функционирования комплекса ПА-6 в пакетном режиме.

Общая схема функционирования комплекса ПА-6. Комплекс ПА-6 представляет собой средство синтеза рабочих программ, реализующих конкретные маршруты проектирования, задаваемые пользователем средствами входных или промежуточного языков. Общая схема функционирования ПА-6 представлена на рис. 5.5. Первым в работу вступает один из входных трансляторов $T_{вх}$, осуществляющих перевод описания технического объекта и задания на его проектирование с входного языка конкретной предметной области *1* в универсальный промежуточный язык *2*. Кроме того, входные трансляторы могут организовывать работу с библиотеками параметров, стандартных фрагментов и макромоделей отдельных предметных областей, осуществлять связь с локальными и общей БД САПР. В качестве $T_{вх}$ может использоваться программа, восстанавливающая эквивалентную схему объекта по результатам его конструкторского проектирования (например, программа, восстанавливающая принципиальную электрическую схему интегральных микросхем по описанию их топологии).

Комплекс ПА-6 допускает работу пользователя непосредственно с промежуточного языка. Обработка описания на промежуточном языке *2* производится компилятором *К*, представляющим собой языковую подсистему ПА-6, снабженную собственным монитором. В результате его работы во внешней памяти ЭВМ создается вре-

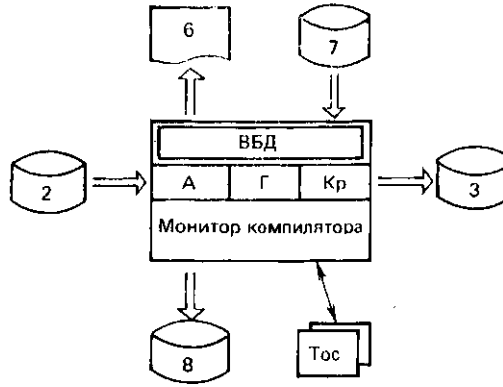


Рис. 5.6. Схема функционирования компилятора программного комплекса ПА-6:

А — анализатор; *Г* — генератор; *Кр* - конструктор; *ВБД* - внутренняя база данных; $Т_{ОС}$ — транслятор из состава *ОС*

менная библиотека *3* объектных модулей, содержащая подпрограммы и управляющие блоки, необходимые для расчета объекта. Далее работает редактор связей *РС* из состава используемой *ОС*, который компонует загрузочный модуль рабочей программы *РП* (обрабатывающей подсистемы ПА-6) из модулей двух типов: сгенерированных компилятором и библиотечных, постоянно хранящихся в библиотеках *4* комплекса. Полученная таким образом рабочая программа загружается в ОП, с этого момента и начинается собственно расчет проектируемого объекта.

Для реализации данной последовательности выполнения программных компонентов в комплексе ПА-6 предусмотрено два способа.

Способ 1. Он предусматривает использование возможностей ОС для перехода от шага к шагу обработки подобно тому, как это принято для процедур трансляции — редактирования — выполнения программ, записанных на стандартных языках программирования.

Способ 2. Этот способ связан с использованием специального монитора, осуществляющего динамический вызов компонентов комплекса ПА-6 в необходимой последовательности.

Способ 1 обеспечивает меньшие затраты ОП, поскольку дает возможность выделить каждому шагу обработки область памяти строго необходимого объема. Однако способ 2 более гибок (его использование для организации диалогового режима работы комплекса ПА-6 описано ниже).

Компилятор (языковая подсистема) комплекса ПА-6. Схема функционирования компилятора дана на рис. 5.6. Собственно компилятор комплекса ПА-6 составляют анализатор *А*, генератор *Г* и конструктор *Кр*, управляемые монитором.

Анализатор *А* считывает входной файл *2*, содержащий предложения промежуточного языка, и обеспечивает лексический и синтаксический анализ описания объекта и задания на расчет. Он

выдает в выходной набор данных *6* дубликат входного описания, диагностические сообщения и по желанию пользователя справочную информацию. Структурированный характер промежуточного языка (см. § 5.3) позволяет для обработки каждой его конструкции использовать отдельный модуль или группу модулей. Это способствует снижению требований к объему ОП, занимаемой анализатором, так как появляется возможность создания оверлейных структур.

Результатом работы анализатора *А* будет набор таблиц, списков, массивов, составляющих *внутреннюю базу данных ВБД компилятора,* располагаемую в ОП. Основные элементы этой БД — упакованное описание структуры проектируемого объекта, таблицы паспортов подпрограмм моделей элементов, подпрограмм расчета выходных параметров и т. п. Операторы языка описания задания преобразуются анализатором в псевдокоманды, содержащие метку и код команды, режимные параметры, имя подпрограммы, реализующей необходимые для выполнения данной команды методы, параметры подпрограммы. Последовательность псевдокоманд описывает программу вычислений, которые должны быть выполнены рабочей программой. Память ЭВМ под внутреннюю БД выделяется только динамически, что определяет ее рациональное использование. При недостатке ОП некоторые наиболее крупные массивы выгружаются во внешнюю память ЭВМ. Во внутренней БД широко используется аппарат перекрестных ссылок между логически связанными элементами данных, что значительно повышает быстродействие компилятора за счет минимизации времени доступа к обрабатываемым данным. Анализатор пополняет внутреннюю БД информацией, считанной из паспортов библиотечных подпрограмм. Эта информация необходима для лексического и синтаксического контроля входного описания. Паспорта сгруппированы в каталоги библиотечных подпрограмм и хранятся во внешней памяти *7* ЭВМ.

Промежуточный язык комплекса ПА-6 допускает описание произвольных функциональных зависимостей пользователя (например, новых моделей элементов) на алгоритмических языках высокого уровня непосредственно во входном файле компилятора *2*. Такие фрагменты текста распознаются анализатором и переписываются им в специальный набор данных *8*. По окончании работы анализатора монитор осуществляет динамический вызов необходимых системных трансляторов $T_{ос}$. Полученные в результате их работы объектные модули помещаются во временную библиотеку *3*.

В задачу генератора Г входит генерация объектных модулей процедур рабочей программы РП: обращения к моделям элементов проектируемого объекта, расчета матрицы Якоби и вектора невязок, прямого и обратного ходов алгоритма Гаусса, расчета данных для печати и др. Непосредственно генерации предшествует оптимальная перенумерация переменных математической модели объекта. Генерация объектных модулей производится в соответствии с делением проектируемого объекта на фрагменты. Такой подход необходим для реализации диакоптических методов анализа и способствует снижению требований к ОП, занимаемой компилятором, так как возникает возможность последовательной обработки фрагментов объекта с сохранением во внутренней БД только необходимого минимума информации о них.

Все управляющие блоки и массивы, необходимые рабочей программе, генерируются в виде заполненных или пустых поименованных программных секций необходимой длины. Этим обеспечивается полное использование ОП (а следовательно, и ее экономия)

рабочей программой при статическом ее распределении. Для обеспечения доступа к произвольным элементам данных рабочей программы, необходимого, например, при интерактивном режиме работы, генератор строит специальный блок указателей, содержащий символические имена и ссылки для всех массивов рабочей программы. Последовательность псевдокоманд, описывающих задание на расчет объекта, преобразуется генератором в табличный вид и оформляется в виде объектного модуля. Информацией о размерах созданных модулей генератор пополняет внутреннюю БД, а сами объектные модули помещает во временную библиотеку *3*.

Генератор — самый сложный блок компилятора. От качества сгенерированных им объектных программ в значительной степени зависит эффективность всего программного комплекса, поэтому в генераторе проводится оптимизация генерируемого кода.

Конструктор комплекса ПА-6 планирует состав и структуру загрузочного модуля рабочей программы РП, используя для этого возможности управляющих предложений и механизм автовызова редактора связей ОС ЕС. Источниками подпрограмм, из которых компонуется рабочая программа, являются временная библиотека объектных модулей *3* и постоянные библиотеки *4* (подпрограмм моделей элементов; подпрограмм методов интегрирования, многовариантного анализа и параметрической оптимизации; подпрограмм внешних воздействий на проектируемый объект; подпрограмм расчета выходных параметров по результатам моделирования; управляющих и сервисных подпрограмм и т. п.).

Используя информацию из внутренней БД, конструктор *Кр* с помощью управляющих предложений добавляет к монитору рабочей программы только те подпрограммы из перечисленных выше библиотек, которые необходимы в данном конкретном маршруте проектирования.

Применяемые в комплексе ПА-6 диакоптические методы позволяют варьировать объемом ОП, требуемым под рабочую программу РП, допуская взаимное перекрытие массивов и подпрограмм различных фрагментов анализируемого объекта. При этом целесообразно перекрывать только подпрограммы — тогда обмен с внешней памятью будет односторонним. Такой подход и реализован в конструкторе комплекса ПА-6, который по определенным правилам сочетания модулей и известным их объемам с учетом размера доступной рабочей программе памяти планирует, если это необходимо, загрузочный модуль оверлейной структуры. При самых жестких требованиях к объему рабочей программы ее оверлейная структура может иметь до четырех точек перекрытия. Структура рабочей программы оптимизируется с целью минимизации количества обращений к внешней памяти ЭВМ. Сгенерированный конструктором набор управляющих предложений помещается в специальный раздел временной библиотеки *3* объектных модулей.

Работой компилятора управляет монитор, который осуществляет вызов в необходимые моменты анализатора, генератора и конструктора, располагаемых в отдельных оверлейных сегментах, фиксирует время их выполнения, организует единообразный доступ к внутренней БД и наборам данных на внешних носителях, обрабатывает режимные параметры (опции) компилятора. Опции позволяют управлять форматом вывода, задавать объем ОП, доступной рабочей программе, выводить в удобной форме информацию из внутренней БД, распечатывать структуру матрицы Якоби, таблицы перенумерации и т. п

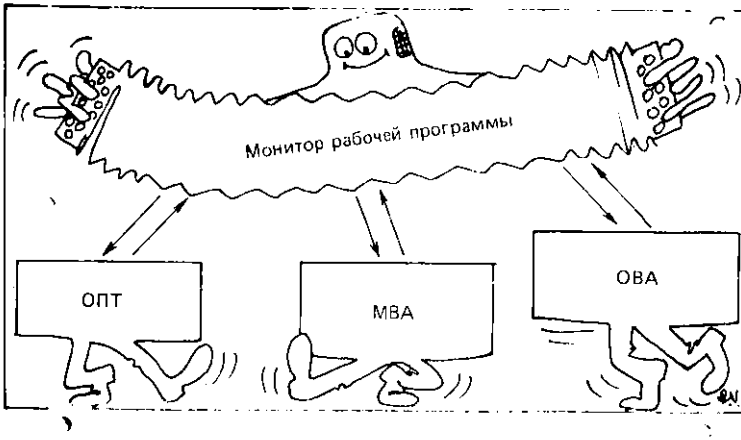


Рис. 5.7.

Вслед за компилятором начинает работать редактор связей ОС ЕС, который считывает набор управляющих предложений, сгенерированный конструктором во временной библиотеке *3*, и создает загрузочный модуль заданной структуры из заданных объектных модулей, расположенных во временной *3* и постоянных *4* библиотеках. Этот модуль помещается в ОП, после чего ему передается управление, т. е. начинается собственно расчет объекта.

Рабочая программа (обрабатывающая подсистема) комплекса ПА-6. Выполнение рабочей программы происходит под управлением монитора, в функции которого входят интерпретация псевдокоманд, отражающих операторы промежуточного языка описания задания, передача управления на диспетчеры, контролирующие вычисления по той или иной псевдокоманде, анализ кодов возврата, организация циклов псевдокоманд, ведение службы времени, установка контрольных точек и т. п.

Модульная структура рабочей программы комплекса ПА-6 совпадает со структурой базового математического обеспечения, представленной на рис. 5.2. Однако в комплексе ПА-6 группы модулей «параметрическая оптимизация» ОПТ, «многовариантный анализ» МВА, «одновариантный анализ» ОВА являются равноуровневыми и располагаются в отдельных перекрываемых сегментах оверлейной структуры рабочей программы. Связь между ними по управлению и информации осуществляется через монитор рабочей программы, как это показано на рис. 5.7. Поэтому подпрограммы, составляющие эти группы, должны быть повторновходимыми, это несколько усложняет их программирование, но зато кроме значительной экономии ОП дает возможность организации вложенных циклов операторов языка описания задания промежуточного языка комплекса ПА-6.

Диалоговый режим работы комплекса ПА-6. Этот режим в комплексе ПА-6 реализуется специальным интерактивным монитором, осуществляющим динамический вызов в ОП необходимого входного транслятора, компилятора, редактора связей ОС ЕС и сгенерированной рабочей программы.

В настоящее время существуют две диалоговые версии программного комплекса; одна из них предназначена для работы в режиме разделения времени (TSO) стандартных операционных систем ЕС ЭВМ, а другая — в рамках оригинальной диалоговой системы СИЕ, разработанной в МВТУ им. Н. Э. Баумана. Обе версии допускают работу с комплексом ПА-6 одновременно нескольких пользователей.

В зависимости от характера требуемых от монитора действий команды диалогового режима разбиты на две группы. Первая группа команд используется для «общения» пользователя с рабочей программой на этапе ее выполнения (команды прерывания и запуска рабочей программы, индикации и модификации различных переменных математической модели объекта, управления выдачей результатов, изменения последовательности выполнения псевдокоманд и т. п.). Вторую группу составляют команды корректировки структуры проектируемого объекта. Для выполнения таких команд диалоговый монитор должен выполнить всю цепочку динамических вызовов «входной транслятор — компилятор комплекса ПА-6 — редактор связей — рабочая программа», на что требуется определенное машинное время, обусловливающее задержку реакции комплекса ПА-6 на команду пользователя.

§ 5.3. Промежуточный язык программного комплекса ПА-6

Ниже приводится описание некоторого подмножества универсального промежуточного языка базовой версии программного комплекса ПА-6, которое для краткости назовем *миниязыком*. Включенные в это подмножество конструкции языка позволяют описывать непрерывные объекты различной физической природы и задание на их анализ во временной области.

Для описания объектов должны использоваться элементы из постоянной библиотеки комплекса ПА-6, так как в миниязык не включены средства описания произвольных функциональных зависимостей пользователя.

Подготовка описания объекта. Она проводится пользователем вручную и состоит из следующих этапов.

Этап 1. Объект разделяется на однородные в физическом отношении подсистемы, а подсистемы в свою очередь делятся на дискретные элементы.

Этап 2. Для каждой подсистемы зарисовывается эквивалентная схема, содержащая элементы, имеющиеся в постоянной библиотеке комплекса ПА-6 (двухполюсники и многополюсники).

Этап 3. Узлы эквивалентной схемы нумеруются целыми положительными числами из непрерывного ряда, начинающегося с 1.

Этап 4. Выявляются элементы, обладающие одинаковыми параметрами.

Этап 5. Выбирается согласованная система единиц величин (как правило, СИ). В дальнейшем все числовые величины миниязыка должны задаваться в этой системе без указания их размерности.

■ Примечание. Этапы 1 и 2 подготовки описания объекта подробно описаны в четвертой книге.

Если при составлении эквивалентной схемы объекта возникает необходимость в элементе, отсутствующем в постоянной библиотеке комплекса ПА-6, то пользователь должен сам запрограммировать

математическую модель этого элемента и поместить подпрограмму в библиотеку. Другой, более простой для неподготовленного пользователя способ преодоления этой трудности — использование специальных библиотечных элементов, реализующих простейшие функциональные зависимости (линейные, нелинейные, пороговые). Из таких элементов пользователь может «собрать» любой необходимый ему макроэлемент. После завершения работ рассмотренных этапов пользователь кодирует описание объекта и задание на его проектирование на миниязыке.

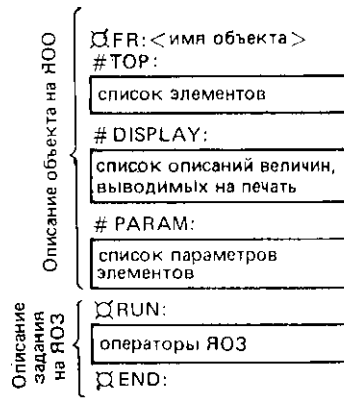


Рис. 5.8. Структура описания объекта проектирования и задания на его расчет на миниязыке программного комплекса НА-6

Структура описания на миниязыке комплекса ПА-6. Структура описания объекта и задания на проектирование представлена на рис. 5.8. Сначала задается описание структуры и элементов объекта с помощью конструкций ЯОО, затем следует описание задания на проектирование. Описание объекта состоит из следующих разделов:

1) описание структуры (список элементов с указанием номеров узлов их подключения);

2) описание величин, выводимых на печать в процессе моделирования объекта;

3) описание параметров элементов эквивалентной схемы.

Каждый раздел описания начинается с ключевого слова (заголовка), первым символом которого является ¤ или ≠

Все предложения должны начинаться с первой позиции строки, пробел в первой позиции служит признаком строки-комментария (из этого правила есть одно исключение, описанное ниже).

При описании конструкций языка будем использовать следующие обозначения <xxx> — символы, заключенные в угловые скобки, являются простым именем сложного элемента языковой конструкции (нетерминалом); [xxx] — элемент языковой конструкции, заключенный в квадратные скобки, не является обязательным.

В качестве примера использования этих обозначений рассмотрим заголовок описания проектируемого объекта. Он должен иметь следующий вид:

¤ FR:[<ИО>]

где ИО — имя объекта, последовательность не более шести букв латинского алфавита и цифр, обязательно начинающаяся с буквы.

Описание структуры (топология) эквивалентной схемы на миниязыке комплекса ПА-6. Описание структуры начинается с заголовка ТОР:, за которым на последующих строках следует список описаний элементов, составляющих схему. Описание одного элемента в списке выглядит следующим образом:

<ИММЭ> [<ДИЭ>] <НУ—1> ... <НУ—N>[,<ИПП>];

где ИММЭ — имя математической модели элемента (оно же имя элемента), последовательность не более трех букв латинского алфавита, не начинающаяся с букв А, В, М, ДИЭ — десятичный идентификатор элемента, произвольная последовательность не более трех цифр (позволяет выделить конкретный элемент среди всех элементов схемы, обладающих одним ИММЭ); НУ—J —номер узла схемы, к которому подключен j-й вывод (полюс) элемента; ИПП — идентификатор признака повторяемости параметров элементов, произвольная последовательность не более шести букв (русских и латинских) и цифр, обязательно начинающаяся с буквы.

Считается, что все элементы, в описании которых присутствует одинаковый ИПП, имеют одинаковые параметры, числовые значения параметров для них задаются однократно (см. ниже).

Имя математической модели элемента фактически представляет собой имя библиотечной подпрограммы, реализующей математическую модель элемента. Оно совместно с десятичным идентификатором составляет идентификатор элемента (ИЭ), который позволяет выделить конкретный элемент среди всех элементов схемы. В описании элемента разделителем между номерами узлов и идентификатором элемента служит один пробел. Например,

С␣11␣, 46, РЫЧАГ	— элемент «масса», <ИЭ>=С, <ИПП>=РЫЧАГ
UP25␣1 68 , L4S;	— элемент «упругость» <ИЭ>=UP25, <ИПП>=L4S
GZ4␣8␣9␣17␣23;	— зависимый источник расхода <ИЭ>=GZ4

В качестве последнего примера использован зависимый источник расхода, применяемый при описании гидравлических и пневматических систем. Этот элемент — четырехполюсник [в нем источник фазовой переменной типа потока включен между первыми двумя полюсами (подсоединенными к узлам *8* и *9* фрагмента)] и зависит от фазовых переменных типа потенциала на двух последних полюсах (подсоединенных к узлам *17* и *23*), т. е. $g=K(\varphi_{17}-\varphi_{23})$, где K — параметр элемента (рис. 5.9).

Указанная функциональная зависимость реализована в заранее составленной подпрограмме, носящей имя GZ. Для простых линейных двухполюсных элементов (в примерах это С и UP25) порядок перечисления номеров узлов подключения на правильность расчета влияния не оказывает, от него зависят только знаки распечатываемых фазовых переменных, связанных с этими элементами (см. ниже).

Миниязык разрешает располагать на каждой строке произвольное (но целое) количество описаний элементов, при этом между ними никаких разделителей (в том числе и пробел) не допускается. Порядок описаний элементов произвольный, комментарии могут

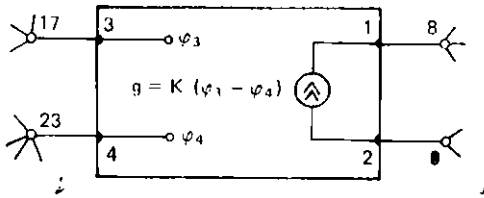


Рис. 5.9. Эквивалентная схема зависимого источника расхода GZ

занимать свободную от описаний элементов часть строки (как это и сделано в примерах), но должны быть отделены от них по крайней мере одним пробелом.

Описание параметров элементов эквивалентной схемы на миниязыке комплекса ПА-6. Вслед за заголовком # **PARAM:**, записываемым на отдельной строке, задается список числовых значений параметров элементов эквивалентной схемы. Параметры элементов описываются строго в том же порядке, в котором сами элементы записаны под заголовком # TOP:. Для элементов, имеющих несколько параметров, числовые значения перечисляются в порядке, принятом для данного элемента (и зафиксированном в документации на подпрограмму элемента). Параметры элементов, обладающих одинаковым идентификатором признака повторяемости (ИПП), указываются только один раз, для первого по порядку элемента с данным ИПП. Числовые значения параметров элементов записываются в виде бесформатных действительных чисел (ДЧ). Общая форма записи действительных чисел в миниязыке следующая:

$$[-][<\text{ЦЧМ}>][.<\text{ДЧМ}>][\text{E}[-]<\text{П}>]$$

где ЦЧМ — целая часть мантиссы (последовательность десятичных цифр); ДЧМ — дробная часть мантиссы (последовательность десятичных цифр); П — порядок (последовательность не более двух десятичных цифр).

Общее число символов в записи ДЧ не должно превышать 16. Разделителем действительных чисел в списке служит символ «;», слева и справа которого находится любое (в том числе нулевое) количество пробелов. При перенесении списка с одной строки на другую символ «;» должен оставаться на первой строке. Комментарий может занимать целую строку или свободную от ДЧ часть строки, но обязательно должен начинаться с символа «/» (косая черта) — это и есть то исключение из правил о комментариях, о котором говорилось выше.

Описание величин, выводимых на печать, на миниязыке комплекса ПА-6. В комплексе ПА-6 используется представление результатов расчета переходных процессов в виде таблиц и графиков значений указанных пользователем величин. Шаги модельного времени по-

строения таблиц и графиков выбираются пользователем (см. ниже). Описание величин, выводимых на печать, начинается с заголовка

#: DISPLAY:, на последующих строках задается список величин, выводимых на печать. Каждая величина описывается одним из трех следующих способов.

Способ 1. Печать переменных двухполюсного элемента

<ПП>:<ИЭ>[=[<ДЧ-1>,<ДЧ-2>]]

где ПП — признак печати, принимающий значение U, I или P; ИЭ — идентификатор двухполюсного элемента.

Если ПП=I, то на печать выводится фазовая переменная типа потока, направленная в элемент с идентификатором ИЭ из узла, указанного первым при описании этого элемента в разделе топологии. Если ПП=U, то на печать выводится разность фазовых переменных типа потенциала узлов, указанных первым и вторым. При ПП=P на печать выводится значение мгновенной мощности, потребляемой двухполюсником (для элементов — источников энергии эта величина, естественно, отрицательна).

Символ «=» означает необходимость построения графика данной величины. Пара действительных чисел ДЧ-1 и ДЧ-2 задает пределы, в которых строится график. Если они опущены, то график будет построен в пределах минимального и максимального значений, принимаемых величиной в процессе расчета. При отсутствии в описании величины печати символа «=» комплекс ПА-6 выдает только числовые значения данной величины, сведенные в столбец таблицы.

Способ 2. Печать разности фазовых переменных типа потенциала произвольных узлов фрагмента

<ИВ>(<НУ-1>, <НУ-2>)[=[<ДЧ-1>, <ДЧ-2>]]

где ИВ — имя величины печати [произвольная последовательность не более шести букв (русских и латинских) и цифр, обязательно начинающаяся с буквы]; НУ-1 и НУ-2 — номера узлов, разность фазовых переменных типа потенциала которых выводится на печать под именем ИВ.

Способ 3. Печать фазовой переменной типа потока произвольного вывода многополюсного элемента

I:<ИЭ>/<НВ>/[=[<ДЧ-1>,<ДЧ-2>]]

где НВ — номер ветви многополюсного элемента с идентификатором ИЭ, фазовая переменная типа потока которой выводится на печать. За положительное принято направление от узла схемы в элемент.

Описания величин печати в списке разделяются друг от друга символом «;». В одной строке можно размещать произвольное целое число описаний, последним символом продолжаемой строки обязательно должен быть разделитель «;». Комментарии могут располагаться на свободной от списка части строки и должны быть отделены от описаний по крайней мере одним пробелом. Например,

DISPLAY:

I:UP25= —1,1E—1; — включить в таблицу результатов фазовую переменную типа потока через двухполюсник UP25, построить для нее график в пределах —1 ... 0,1;

U:C4=; ДАВЛ(28,17); — включить в таблицу результатов разность фазовых переменных типа потенциала на выводах двухполюсника C4 и в узлах *28* и *17*, для разности на C4 дополнительно построить график в пределах минимального и максимального значений;

I:TN111/3/ — включить в таблицу результатов фазовую переменную типа потока, направленную в многополюсник TN111 по его третьему выводу.

Описание задания на расчет объекта на миниязыке комплекса

ПА-6. Вслед за заголовком RUN: записываются операторы (директивы) языка описания задания (каждый с первой позиции отдельной строки). Общий вид оператора

<ОП>:<ИП>[(СПП)][,<СРП>]

где ОП — один из операторов миниязыка; ИП — имя одной из подпрограмм реализации соответствующего оператора, последовательность не более шести букв латинского алфавита и цифр, обязательно начинающаяся с буквы; СПП — список параметров подпрограммы; СРП — список режимных параметров оператора.

Параметры подпрограммы реализации оператора задаются в виде ключевых слов длиной в три латинские буквы, знака равенства «==» и действительного числа, являющегося значением параметра подпрограммы. Записанные таким образом параметры разделяются в СПП символом «,». Некоторые параметры подпрограммы (или все) могут быть опущены, в этом случае они принимают значения «по умолчанию» (эти значения указываются для каждой подпрограммы в ее документации).

Рассмотрим правила записи двух операторов миниязыка совместно с одной из подпрограмм из состава комплекса ПА-6.

Оператор задания нулевых начальных условий

INIT:ZERO

В результате выполнения этого оператора все фазовые переменные объекта принимают нулевые значения. В задании на миниязыке этот оператор всегда должен быть первым.

Оператор расчета переходных процессов комбинированным методом интегрирования

DYNA:COMBYN (ACR=<ДЧ-1>, EPS=

=<ДЧ-2>, SMN=<ДЧ-3>, SMX=<ДЧ-4>),

END=<ДЧ-5>, DTP=<ДЧ-6>, DTG=<ДЧ-7>

где ДЧ-1, ..., ДЧ-4 — параметры подпрограммы комбинированного неявно-явного метода интегрирования [ДЧ-1 — гарантируемая локальная погрешность интегрирования (абсолютная величина, по умолчанию 0,001); ДЧ-2 — константа автоматического выбора шага интегрирования (выбирается как 0,5 от ДЧ-1, по умолчанию 0,0005); ДЧ-3 — минимальный (он же начальный) шаг интегрирования (по умолчанию 10^{-6}); ДЧ-4 — максимальный допустимый шаг интегрирования (по умолчанию 1000)]; ДЧ-5, ..., ДЧ-7 — режимные параметры оператора [ДЧ-5 — отрезок модельного времени расчета переходных процессов; ДЧ-6 — шаг модельного времени построения таблицы величин, выводимых на печать; ДЧ-7 — шаг модельного времени построения графиков].

Режимные параметры DTP и DTG не обязательны. Их отсутствие в операторе означает отказ от вывода таблиц и графиков величин печати.

Рассмотренный миниязык не отражает всего многообразия возможностей программного комплекса ПА-6 (совместный анализ непрерывных и дискретных объектов, многовариантный анализ и параметрическая оптимизация, описание произвольных функциональных зависимостей пользователей и др.), однако его вполне достаточно для проведения цикла лабораторных работ и выполнения упражнений (см. седьмую и восьмую книги).

§ 5.4. Пути совершенствования пакетов функционального проектирования

В настоящее время наблюдаются две основные тенденции в развитии пакетов функционального проектирования. Первая связана с повышением их вычислительной эффективности, т. е. с сокращением затрат ресурсов ЭВМ (в первую очередь машинного времени) на выполнение отдельных проектных операций и процедур, а вторая — с расширением их области применения и функциональных возможностей.

Повышение вычислительной эффективности. Улучшение характеристик экономичности пакетов дает возможность рассмотрения на каждом иерархическом уровне проектирования более крупных (с точки зрения количества составляющих их элементов) объектов проектирования, что является прямой предпосылкой к улучшению качества и ускорению процесса проектирования. Особо остро проблема экономичности подсистем функционального проектирования стоит при создании САПР в такой бурно развивающейся области, как микроэлектроника. Эффективность пакетов функционального проектирования определяется в первую очередь экономичностью входящей в его состав подсистемы анализа, поэтому основные усилия специалистов-разработчиков таких пакетов направлены на поиски путей повышения быстродействия процедур моделирования. В настоящее время эти поиски ведутся в двух направлениях: 1) совершенствование математического обеспечения и организации ПО путем учета специфики ММ объектов проектирования различных предметных областей и технологий производства; 2) использование ЭВМ неклассической архитектуры.

Учет специфики ММ объектов проектирования на макроуровне делает во многих случаях эффективным с точки зрения затрат машинного времени применение декомпозиционных методов

анализа, сводящих решение задачи большой размерности к решению подзадач меньшей размерности. Например, свойство пространственной разреженности ИС позволяет использовать при их электрическом анализе различные методы численного интегрирования дифференциальных уравнений для ММ различных фрагментов ИС, выбирая для каждого фрагмента наиболее подходящий метод. Ряд методов использует свойство временной разреженности ИС, осуществляя обнаружение «неактивных» в текущий момент времени участков схемы и исключение соответствующих им переменных и уравнений из общей ММ системы. Учет однонаправленности ММ МДП-транзисторов позволяет приблизительно на два порядка поднять быстродействие программ анализа путем замены классических методов анализа (см. рис. 5.1) на релаксационные, в основе которых лежат итерационные алгоритмы Гаусса — Якоби и Гаусса — Зейделя.

■ Примечание. Пространственная разреженность системы понимается как отсутствие непосредственной связи между «удаленными» элементами этой системы. Наличие в рассматриваемой системе в какой-либо момент времени элементов, не меняющих свое состояние, называется *временной разреженностью*. Однонаправленной называется ММ, которая не отражает влияние выходных переменных элемента на входные.

Использование ЭВМ нетрадиционной архитектуры совместно со специальными декомпозиционными методами может в ряде случаев обеспечить многократное повышение эффективности пакетов функционального проектирования за счет распараллеливания вычислений на этапе анализа. Такое распараллеливание, например в подсистемах макроуровня, возможно на различных этапах решения ММ системы. Так, в программе CLASSIE, разработанной специально для супер-ЭВМ CRAY-1, распараллеливаются вычисления в моделях элементов и обработка повторяющихся фрагментов моделируемого объекта. Существуют методы распараллеливания процедуры решения систем алгебраических уравнений. Однако наибольшее ускорение достигается при использовании релаксационных методов решения систем ОДУ за счет минимизации времени, необходимо на синхронизацию процессов решения, выполняемых на различных процессорах многопроцессорной вычислительной системы.

В последние годы специально для повышения быстродействия пакетов моделирования на микроуровне разработаны спецпроцессоры (например, матричные), подключаемые к обычным ЭВМ.

Реализация рассмотренных выше путей повышения вычислительной эффективности пакетов функционального проектирования часто требует использования иных по сравнению с описанными в § 5.1 и 5.2 принципов построения ПО.

Расширение области применения. Увеличение масштабов использования пакетов функционального проектирования идет в настоящее время по двум взаимосвязанным направлениям: 1) создание универсальных пакетов, пригодных для использования во многих предметных областях; 2) расширение круга пользователей за счет ориентации пакетов на персональные ЭВМ.

Универсальность пакетов достигается использованием в них математического обеспечения (МО), инвариантного к предметным областям, что часто противоречит требованиям экономичности и минимальной сложности пакетов. Одним из путей повышения универсальности и одновременно эффективности пакетов функционально-

го проектирования является объединение в них на общей организационной основе элементов МО микро-, макро- и метауровней, что позволяет в рамках одного пакета выполнять смешанный многоуровневый анализ сложных систем.

Использование персональных ЭВМ дает возможность приобщиться к средствам автоматизированного проектирования значительно большему, чем в настоящее время, количеству разработчиков технических изделий и, более того, за счет «персонализации» компонентов САПР создать мощный *индивидуальный* (т. е. наиболее удобный для конкретного пользователя) инструмент проектирования, значительно увеличивающий интеллектуальный потенциал разработчика.

Расширение области применения пакета функционального проектирования невозможно без наличия в нем специальных средств модификации и расширения, обеспечивающих его всемерную открытость для включения новых программных компонентов, в первую очередь подпрограмм моделей конкретных технических систем. Синтез моделей, их программирование представляет собой часто довольно сложную научно-техническую задачу, решить которую большинство пользователей самостоятельно не может. Поэтому ожидается, что в ближайшие годы для решения проблемы синтеза моделей технических объектов могут найти широкое применение *системы баз знаний.*

КРАТКИЕ ВЫВОДЫ

Одними из наиболее сложных в ПО САПР являются пакеты функционального проектирования на макроуровне. В их составе выделяют три основные подсистемы: языковую, обрабатывающую и монитор. Языковая подсистема организует общение пользователя с пакетом, используя концепцию языков двух уровней. Обрабатывающая подсистема осуществляет непосредственное решение проектных задач. Обе названные подсистемы функционируют под управлением монитора.

Среди проектирующих пакетов различают пакеты-интерпретаторы и пакеты-трансляторы. В пакетах-трансляторах расчету предшествует этап генерации рабочей программы, реализующей необходимый для расчета алгоритм. В пакете-интерпретаторе все расчеты выполняются с помощью универсальной обрабатывающей подсистемы. Пакеты-трансляторы более быстродействующими по сравнению с пакетами-интерпретаторами, но требуют больших затрат памяти.

Используемая литература

1. *Ильин В. Н., Коган В. Л.* Разработка и применение программ автоматизации схемотехнического проектирования. — М.: Радио и связь, 1984. — 368 с.

2. *Льюис Ф., Розенкранц Д., Стирнз Р.* Теоретические основы проектирования компиляторов. — М.: Мир, 1979. — 654 с.

3. *Брукс Ф. П.* Как проектируются и создаются программные комплексы. — М.: Наука, 1979. 152 с.

4. *Зелковиц М., Шоу А., Гэннон Дж.* Принципы разработки программного обеспечения. — М.: Мир, 1982. — 368 с.

5. *Майерс Г.* Надежность программного обеспечения. — М.: Мир, 1980. — 360 с.

6. Требования и спецификации в разработке программ. — М.: Мир, 1984. — 344 с.

7. *Поренков Н. Н.* Введение в автоматизированное проектирование технических устройств и систем. — М.: Высшая школа, 1980. — 264 с.

8. *Турский В.* Методология программирования. — М.: Мир, 1981. — 264 с.

9. *Хьюз Дж., Мичтом Дж.* Структурный подход к программированию. — М.: Мир, 1980. — 278 с.

10. *Йодан Э.* Структурное проектирование и конструирование программ. — М.: Мир, 1979. — 415 с.

11. *Бездородов Ю. М.* Индивидуальная отладка программ. — М.: Наука, 1982. — 192 с.

12. *Александров А. А., Бойко В. В., Вейнеров О. М.* Системы управления базами данных/Под ред. *В. М. Савенкова.* М.: Финансы и статистика, 1984. — 224 с.

13. *Пурвин Ю. В., Михайлов Н. А., Демидов Н. В.* Система управления базой данных СЕДАН. — М.: Финансы и статистика, 1981. — 102 с.

14. *Денинг В., Эссиг Г., Маас С.* Диалоговая система «человек — ЭВМ». Адаптация и требования пользователя. — М.: Мир, 1984. — 112 с.

15. *Блэкман М.* Проектирование систем реального времени. — М.: Мир, 1977. — 346 с.

16. *Мартин Дж.* Системный анализ передачи данных. Т. I. — М.: Мир, 1975. — 256 с.

17. Новые средства программирования для ЕС ЭВМ. Транслятор с языка АЛГОЛ-68 и диалоговая система/*Г. Ф. Дейкало, Б. А. Новиков* и др. — М.: Финансы и статистика, 1984. — 207 с.

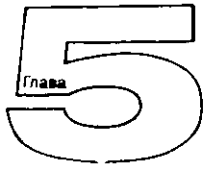
18. *Мартин Дж.* Организация баз данных в вычислительных системах. — М.: Мир, 1979. — 611 с.

19. *Дейт К.* Введение в системы баз данных. — М.: Наука, 1981. — 464 с.

20. *Цикритзис Д., Лоховски Ф.* Модели данных. — М.: Финансы и статистика, 1985. — 317 с.

21. *Кузин Л. Т., Строгановский В. С., Щукин Б. А.* Банки данных в системах автоматизированного проектирования АСУТП. — М.: Машиностроение, 1984. — 48 с.

Оглавление



УЧЕБНОЕ ИЗДАНИЕ

СИСТЕМЫ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ

Владимир Геннадьевич Федорук
Валерий Михайлович Черненький

ИНФОРМАЦИОННОЕ И ПРИКЛАДНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Заведующая редакцией *И. И. Хрусталева.* Редактор *Л. И. Андрианова.* Младший редактор *Т. Ф. Артюхина.* Художники: *Р. Р. Витковский, Ю. Д. Федечкин, В. И. Хомяков.* Художественный редактор *М. И. Чурилов.* Технический редактор *Р. С. Родичева.* Корректор *Г. И. Кострикова*

ИБ № 6008

Изд. № СТД--500. Сдано в набор 27.01.86. Подп. в печать 06.03.86. Т-05983. Формат 84×108$^1/_{32}$. Бум. тип. № 1. Гарнитура литературная. Печать высокая. Объем 8,40 усл. печ. л. 8.61 усл. кр.-отт. 9.10 уч. изд. л. Тираж 40 000 экз. Зак. № 71. Цена 35 коп.

Издательство «Высшая школа», 101430, Москва, ГСП-4, Неглинная ул., д. 29/14. Московская типография № 8 Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли, 101898, Москва, Центр, Хохловский пер., 7.