МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ИНСТИТУТ ЭЛЕКТРОННОЙ ТЕХНИКИ
(технический университет)

На правах рукописи
Экз № ____

УДК 681.3.06

ВАРАНКИН НИКОЛАЙ ВАСИЛЬЕВИЧ

РАЗРАБОТКА АДАПТИВНОЙ ИНТЕРАКТИВНОЙ ОБОЛОЧКИ ДЛЯ ЗАКАЗНЫХ САПР

Диссертация на соискание ученой степени
кандидата технических наук

Специальность 05.13.12 - системы автоматизации проектирования

Научный руководитель
доктор технических наук
профессор А. Г. Соколов

Москва - 1993 г.

СОДЕРЖАНИЕ

СОКРАЩЕНИЯ

БД - база данных

БК - битовая карта

ДП - динамическая память

ЗакСАПР - заказная система автоматизированного проектирования

ЛВС - локальная вычислительная сеть

ОДЗ - область допустимых значений

ОС - операционная система

ПГП - пакет графических программ

ППП - пакет прикладных программ

ПП - прикладная программа

ПС - пространственная сортировка

САПР - система автоматизированного проектирования

СБИС - сверхбольшая интегральная схема

СМ - системная магистраль

СУБД - система управления базой данных

УСД - управляющие структуры данных

MSC - компилятор языка Си фирмы Microsoft

MSW - Microsoft Windows

XWS - X Window System

ВВЕДЕНИЕ

Разработка программных систем автоматизации по индивидуальным требованиям заказчиков является стержнем научной, технической и коммерческой деятельности независимых фирм, не привязанных к какому-либо одному промышленному предприятию. Накопление и обобще-ние опыта внедрения и эксплуатации различных вариантов САПР создает реальные предпосылки для создания более качественного и функционально наполненного программного обеспечения. Ярким приме-ром тому служит успех таких фирм, как Cadence, Mentor Graphics, Personal CAD Systems и др., чьи системы проектирования доминируют на рынке несмотря на их высокую стоимость.

Разработка заказных САПР (ЗакСАПР) требует гибкого и глубоко продуманного подхода при формировании принципов проектирования та-ких систем. Различия аппаратуры и операционных систем подразуме-вают написание индивидуального программного кода для каждого зака-за. В связи с этим сокращение трудоемкости разработки нового ва-рианта САПР приобретает актуальность, особенно для небольших фирм. Разработка и последовательное применение концепции адаптивного к вычислительной среде программного обеспечения позволяет гибко учи-тывать динамику рыночного спроса при экономии сил и средств на модификацию программ.

К концу 70-х годов в основном были сформулированы основные принципы и методика построения САПР для электроники, предназна-ченных для использования на больших ЭВМ коллективного пользования [1]. Следует отметить труды советских ученых: Норенкова И.П., Казеннова Г.Г., Ильина В.И., Петренко А.И., Баталова Б.В., Аниси-мова В.И. и др. [2-7]. Однако быстрое развитие вычислительной тех-ники в совокупности с растущими потребностями промышленности по-служило

предпосылкой для развития современной концепции САПР элек-троники, ключевыми элементами которой является модульность при-кладных программ и обеспечение максимальной гибкости как в при-кладной области, так и к техническим и системным программным средствам [1,8-10].

Известные методы построения переносимых программ не учитывают специфику современных программных комплексов, интенсивно использу-ющих графические и сетевые возможности современных ЭВМ. Кроме того, требует пристального внимания интенсивно развивающаяся кон-цепция оболочек САПР электроники, устанавливающая новые стандарты в автоматизированном проектировании. В настоящей работе представ-лены результаты исследования и разработки способов обеспечения адаптивности ПО заказных САПР, учитывающие как современные тен-денции в САПР, так и специфику российского рынка программных про-дуктов и вычислительной техники. Особое внимание уделено разработ-ке системно-независимых программных интерфейсов, реализующих на практике концепцию адаптивности.

Диссертация состоит из введения, 5 глав, заключения, списка использованной литературы и приложений.

В главе 1 анализируются способы интеграции прикладных программ САПР в составе оболочек. Показано место заказной САПР и проана-лизированы различные варианты ее включения в такую оболочку. Обос-нован метод использования буферного программного обеспечения, фор-мирующего стабильную системно-независимую среду для прикладных за-дач и математически сформулированы критерии проектирования такого ПО.

В главе 2 исследуются проблемы разработки интерактивно-графи-ческого интерфейса - наиболее сложной компоненты оболочки ЗакСАПР. Представлен краткий обзор наиболее значительных решений в области стандартизации машинной графики и приведено сравнение функциональ-ных возможностей трех наиболее распростаненных графических сред: X

Window System, Microsoft Windows и графики для MS-DOS. Показано, что применение минимального обоснованного набора функций позволяет минимизировать затраты на разработку новых вариантов интерфейса. Приводятся результаты разработки инвариантных внутренних и внешних структур данных и методика взаимодействия с базовыми графическими платформами. Описан программный интерфейс для прикладных программ. Особое внимание уделено методу реализации графического интерфейса для среды Microsoft Windows.

Также в главе 2 рассмотрены состав, алгоритмы функционирования и методы использования простого пакета подпрограмм, предназначен-ного для создания мультиоконной диалоговой среды прикладных прог-рамм. Реализованный на основе объектно-ориентированного подхода, этот пакет позволяет упростить разработку интерфейсов "пользова-тель-САПР". Применение адаптивного графического интерфейса во мно-гом решает задачу обеспечения адаптивности к различным вычисли-тельным платформам.

В главе 3 исследуются проблемы управления динамической памятью в системах, отличных от ОС UNIX. Показана необходимость исполь-зования обобщенной модели памяти, на основе которой возможно построение базового набора программ управления. Приводятся специ-фикации программного интерфейса для различных вычислительных плат-форм, методика его применения и практические примеры программ.

В главе 4 на основе анализа известных структур баз данных графических объектов поставлена задача разработки структуры быст-родействующей БД, ориентированной на иерархическую организацию данных. Приведены оценки скорости локальной и глобальной выборки данных. Представлен перечень функций, выполняемых СУБД, и ее прог-раммный интерфейс. Особое внимание уделено методам ускорения выборки данных из иерархии, а также проанализированы методы вычис-ления трансформации координат и показана эффективность применения дробных вычислений для некоторых ЭВМ.

В качестве иллюстрации прикладного использования БД в главе 4 также рассматривается метод решения одной из трудоемких задач - контроля проектных норм топологии СБИС. Показано, что метод пространственной сортировки в сочетании с "прозрачной" выборкой конту-ров позволяет получить максимально возможную скорость проверки.

Глава 5 посвящена описанию структуры и функциональных возможностей графического редактора топологии Layout Windows, в котором использованы представленные в диссертации концепции, методы, алгоритмы и программные интерфейсы. Дополнительно рассматривается пример организации сервисных средств доступа к данным. Приводится перечень реализованных функций и пример их использования в прикладной программе. Таким образом, показано и практически подтверждено, что метод адаптивной интерактивной оболочки ЗакСАПР позво-ляет с минимальными затратами реализовать программы САПР для раз-личных вычислительных платформ.

В приложении приведены акты внедрения в промышленности результатов диссертационной работы, а также: текст программы для демонстрации возможностей адаптивного графического интерфейса, текст программы определения скорости вычисления трансформации координат. Приводится формат описания правил контроля топологии графического редактора Layout Windows.

К новым научным результатам, полученным в данной работе, можно отнести следующие:

1. Обоснована эффективность структуры заказной САПР на базе открытых адаптивных системно-независимых программных интерфейсов, обеспечивающая высокую мобильность прикладных программ. Выделены компоненты оболочки заказной САПР и приведены методы их реали-зации.

2. Предложены инвариантные к вычислительным средам структуры данных интерактивно-графического интерфейса, уменьшающие количество функций управления интерфейсом.

3. Предложены алгоритмы и схема передачи управления в прило-жении Windows, которые реализуют интерактивно-графический интер-фейс и позволяют использовать стандартное построение прикладных программ.

4. Предложена обобщенная модель динамической памяти, описы-вающая отношения управления между прикладной программой и различ-ными вычислительными средами.

5. Предложена оригинальная структура компактной графической базы данных, ориентированная на обработку иерархически организо-ванных данных.

6. Предложены методы аккумулирования и использования обобщен-ной информации о графических объектах, ускоряющие выборку данных из иерархической БД.

7. Разработаны оригинальные алгоритмы контроля проектных норм топологии на основе разработанной СУБД, не требующие развертывания иерархии в одноуровневое представление.

Приведенные методы, алгоритмы и интерфейсы применены в гра-фическом редакторе топологии Layout Windows, используемом в сос-таве САПР СБИС, микросборок и печатных плат.

БЛАГОДАРНОСТИ

Данная работа была доведена до практического внедрения благо-даря усилиям многих людей. Автор выражает благодарность коллегам научно-исследовательской группы КИПАРИС кафедры ПКИМС за поддержку и критические замечания при выработке концепции адаптивной графи-ческой оболочки САПР и в ходе ее реализации. Особая признатель-ность научному руководителю группы проф. д.т.н. Соколову А.Г., энтузиазм и энергия которого обеспечили практическое знакомство с передовой зарубежной вычислительной техникой и программным обеспе-чением.

Автор благодарен СП "БСД-Силикон", любезно предоставившему вычислительную технику, а также НИИ "Научный Центр", МНПВП "Силикон", в/ч 11135 и Российскому НИИ технологий микроэлек-троники, финансировавшим исследования и практические разработки в сложных условиях перестройки и введения рыночных отношений.

Особые слова благодарности моей жене Татьяне за терпение и поддержку в течение всех лет работы над диссертацией.

ГЛАВА 1

ОСОБЕННОСТИ ИНТЕГРАЦИИ СОВРЕМЕННЫХ САПР

1.1.Оболочки САПР.

На структуру современных САПР сильное влияние оказывает необ-ходимость обеспечить открытость системы. Это связано с тем, что предлагаемый обычно фиксированный набор программ автоматизации не может служить универсальным инструментарием для всех проектиров-щиков и для всех возникающих задач. Для решения отдельных проблем, как показывает мировой опыт, необходимо использовать дополнитель-ное программное обеспечение, разработанное третьими фирмами. При интеграции этих программ в среду существующих САПР возникают зада-чи согласования методики проектирования и информационных связей.

Кроме того, к настоящему времени стало очевидным, что для поддержки проектирования сложных объектов типа СБИС необходимы специальные средства обеспечения целостности данных. Решение всех этих задач привело к функциональному выделению части программного обеспечения САПР, обеспечивающего сервис и всю инфраструктуру для инструментальных средств проектирования [13-17]. Так как эти сред-ства охватывают обслуживание со всех "сторон", они получили очень удачное название "оболочки" САПР (в иностранной литературе - CAD framework).

Рассмотрим структуру и выполняемые функции ряда зарубежных оболочек САПР. В работе [16] описывается интегрированная среда проектирования ультрабольших интегральных микросхем GARDEN. Глав-ная стратегия ее функционирования заключается "в обеспечении очень

специализированных интерфейсов, которые служат изолирующим слоем между приложениями и вычислительной средой". Основные компоненты этой среды составляют (рис.1-1): 1) графический интерфейс (ИГ-интерфейс) пользователя, 2) интерфейс данных проектирования (БДИ) и 3) системный и сервисный интерфейс (ССИ).

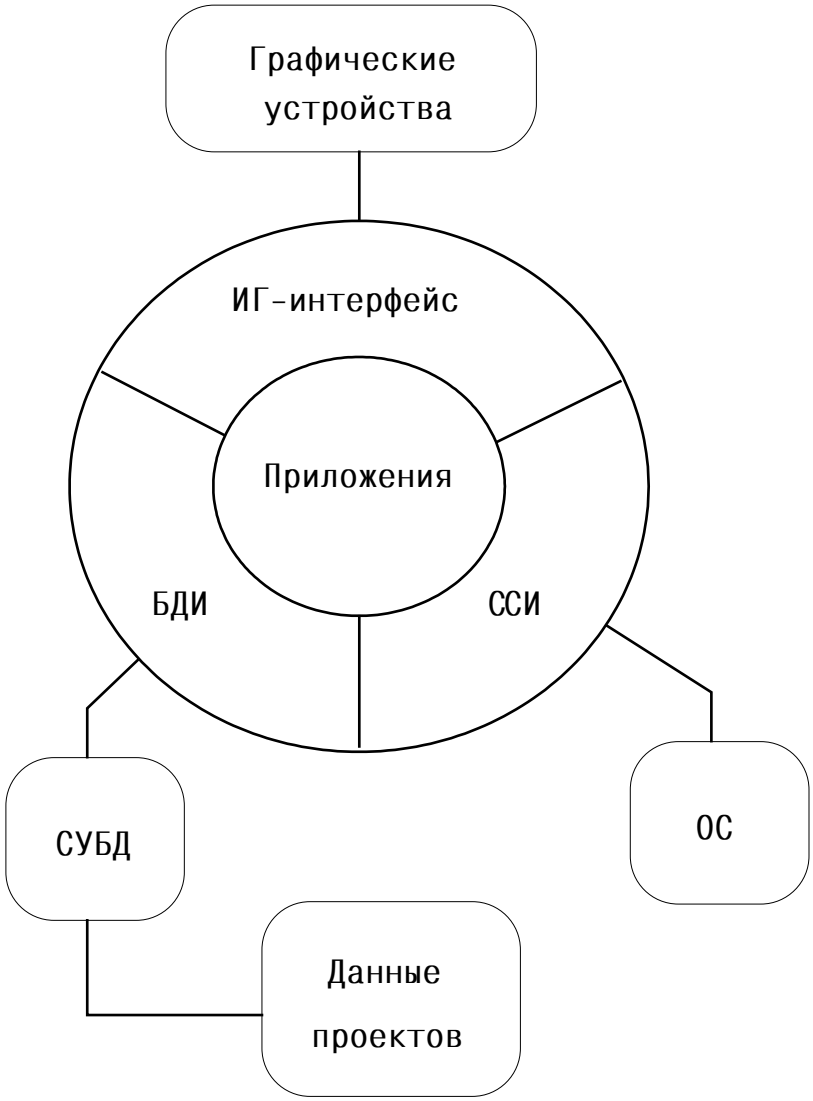


Рис. 1-1 [16]

Структура интегрированной среды САПР ультраБИС GARDEN

В GARDEN упор делается именно на формальное определение интерфейсов в противоположность адаптации случайного набора сервисных средств какой-либо одной ОС. Подчеркивается, что эти интерфейсы рассматриваются как независимые программные единицы. В результате разработчики инструментальных приложений САПР могут полностью со-

средоточится на решении прикладных задач, не занимаясь вопросами переносимости на различные ЭВМ.

Кроме целей создания стабильной прикладной среды, в GARDEN решаются и другие задачи. Это 1) контроль версий и целостности проекта, 2) задание методологии проектирования, 3) обратная трассировка (откат) проектных операций, 4) иерархическое представление проекта, 5) документирование и 6) поддержка национальных языков.

Как подчеркивается в [14], оболочки САПР базируются на единст-ве концепций технического проектирования и управления данными про-екта. Также указывается на важность человеческого фактора в проектировании: конструктор должен иметь возможности не только разрабатывать электронное устройство, но и управлять методикой этого проектирования. Структура оболочки TeamNet, представленная на рис.1-2, акцентирует внимание на следующих возможностях: 1) использо-вание ПО третьих фирм и 2) выделение в отдельную подсистему объектно-ориентированного файлового сервиса САПР на базе вычисли-тельных сетей.

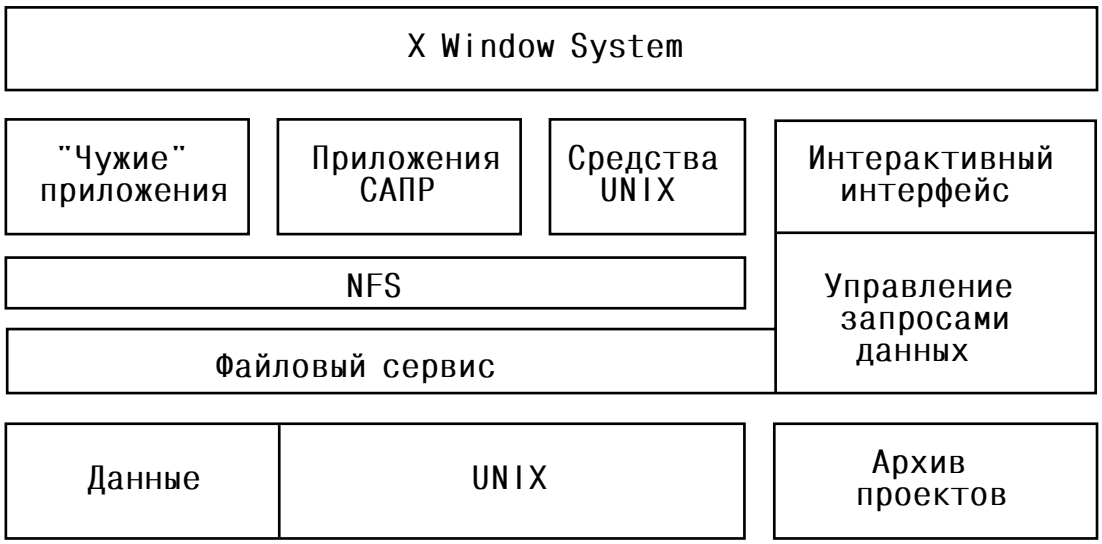


Рис. 1-2 [14]

Структура оболочки САПР TeamNet

В той же работе [14] приводится блок-схема оболочки САПР фирмы Atheron Technology, в которой упор делается на горизонтальное раз-

деление сервисных процедур и вертикальное расположение инструментальных программ. Тем самым подчеркивается интегрирующая роль обслуживающих программ оболочки (рис.1-3).

Рис. 1-3 [14]

Блок-схема оболочки САПР фирмы Atheron Technology

1..N - инструментальные программы САПР

В работе [18] предлагается определить оболочку САПР в терминах выполняемых ею функций, а именно:

1) единый, или точнее, графический интерфейс пользователя;

2) интерфейс прямого взаимодействия программ, входящих в оболочку;

3) управление последовательностью операций проектирования;

4) управление базой данных;

5) управление данными проекта;

6) авторизация использования программ в вычислительной сети;

7) управление многовариантностью проектирования;

8) язык программирования оболочки.

Там же [18] подчеркивается, что оболочка САПР должна выполнять функции "операционной системы" для программ проектирования, освобождая их от множества рутинных задач.

В подтверждение этого тезиса можно привести решения, использованные в оболочках наиболее современных САПР проектирования электроники. Фирма Cadence использует оболочку, ориентированную на модели представления данных проекта. Для облегчения ориентировки пользователя в иерархии моделей конкретного проекта используется специальное графическое средство, наглядно отображающее структуру проекта. Для вызова прикладной программы достаточно двойного нажатия кнопки "мыши" над выбранным представлением объекта.

САПР фирмы Mentor Graphics (начиная с версии 8) использует оболочку Falcon для интеграции собственных продуктов и программ третьих фирм. Аналогично фирме Cadance используется специальный менеджер проектных данных и программных средств. Существенное отличие заключается в использовании наглядного представления как для данных, так и для программ в виде "иконок". Это облегчает воспри-ятие пользователем списка имен файлов (имен проектов), а также открывает богатые возможности по использованию технологии "тащи и бросай" (drag and drop). Все операции можно выполнять с помощью "мыши".

Следует упомянуть оболочку Nelsis [20], в которой реализованы результаты интересных исследований по созданию оболочки систем автоматизации, не ограничивающейся только электроникой, а базирующейся на общей объектно-ориентированной модели обработки данных OTO-D [22]. В этой оболочке пользователь работает не с файлами, а с абстрактными объектами, имеющими множественное представление.

В настоящее время есть все основания говорить о формирующемся стандарте на открытые системы проектирования в области электро-ники. Начиная с 1988 г. координирующие усилия некоммерческой орга-низации под названием CFI (CAD Framework Initiative - инициатива по оболочкам САПР), объединившей пользователей и разработчиков САПР, привели к

выработке первой версии промышленного стандарта открытой САПР, который в настоящее время уже опубликован. Основная линия инициативы была выражена как "создание свободной рыночной модели для инструментов автоматизации в электронике и создание сред-оболочек для их поддержки путем разработки эффективных про-мышленных рекомендаций, которые устранят барьеры к интеграции" [18,19]. Масштабность предпринятых усилий, успех полученных ре-зультатов и объявленная ведущими фирмами-разработчиками САПР под-держка стандарта делают его заслуживающим пристального внимания.

Важнейшие компоненты интегрирующей оболочки и их основные взаимосвязи, согласно концепции CFI, изображены на рис.1-4. В та-кой схеме операционная система выполняет свои обычные задачи уп-равления процессами, файловой системой и предоставляет средства коммуникации в локальной вычислительной сети (ЛВС).

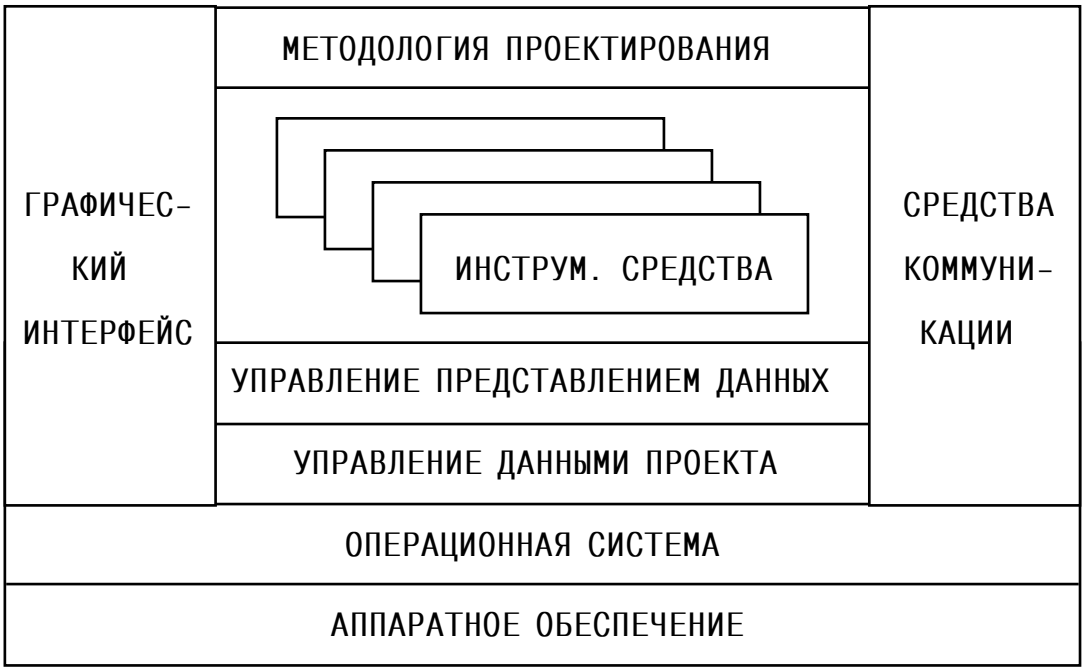


Рис.1-4

Компоненты интегрирующей оболочки САПР по концепции CFI

В качестве инструмента визуализации информации компонентами оболочки выступает обобщенный интерфейс пользователя, который в силу важности графических возможностей чаще называется как графи-ческий

интерфейс пользователя. Главная задача этого интерфейса - обеспечить единую форму представления информации на экране, одина-ковые средства управления изображением и единые способы организа-ции диалога для уменьшения нагрузки на пользователя и облегчения работы с системой. В качестве такого интерфейса для рабочих станций на платформе UNIX / VMS служит сетевая графическая система под названием X Window System, расширенная пакетами программ типа OSF/Motif или OPEN LOOK. Для персональных компьютеров ряда IBM PC/AT аналогичными возможностями обладает среда Windows (WindowsNT) фирмы Microsoft.

Средства коммуникации инструментальных программ позволяют организовать прямой бездисковый обмен данными и оперативной / управляющей информацией. Это позволяет скоординировать выполнение программ как на одном рабочем месте, так и в ЛВС.

Управление данными и их представлением может рассматриваться как интеллектуальная система управления базой данных (СУБД), предоставляющая единые средства доступа к информации, независимо от их физической / логической организации и фактического местоположе-ния в вычислительной сети [20]. Наряду с широко известными базами данных реляционного типа [21], для оболочек САПР наиболее прог-рессивной технологией сегодня считаются объектно-ориентированные базы данных [18, 22]. Их главное преимущество в поддержке струк-турного иерархического представления данных.

Управление методологией проектирования достаточно ново для оболочек САПР, но жизненно необходимо для обеспечения целостности больших сложных проектов. Эти средства позволяют описать этапы (маршрут) проектирования, их взаимозависимость и типы используемой и получаемой информации, что позволяет управлять выполнением работ при участии коллектива разработчиков. Для описания графа могут ис-пользоваться модели, формализующие отношения между объектами.

1.2. Концепция заказных САПР в современных условиях.

Заказные САПР [1,8-10] могут выступать не только в роли самостоятельных инструментов автоматизации проектирования, но и вхо-дить в качестве компоненты в комплексную САПР, расширяя и дополняя ее штатные возможности. Внедрение готовых программ в среду проектирования предполагает экономию времени и средств по сравнению с разработкой новых САПР и позволяет учесть динамику рыночного спроса.

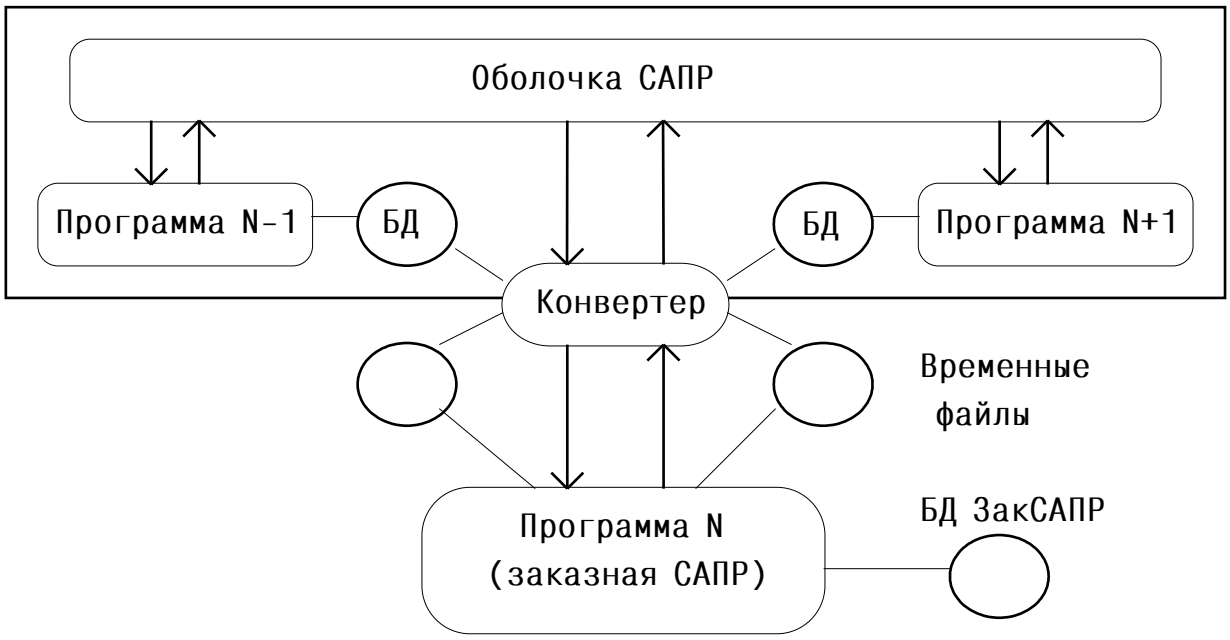


Рис. 1-5 [18]

Схема подключения "инородных" программ в оболочку САПР

Однако разнообразие целевых САПР, в качестве которых могут выступать и оболочки САПР, различия в методиках проектирования, форматах данных и их смысловом содержании, равно как и особенности ЭВМ и операционных систем, не позволяют осуществлять интеграцию программных средств без существенных затрат. Возникающие проблемы и пути их решения рассматривались в работах [1-12], актуальных и сегодня.

Концепция модульных заказных САПР [1,8-10] наиболее удачно подходит для интеграции эффективных прикладных программ как в отечественные, так и в зарубежные системы проектирования и оболочки. Для

последних практически повсеместно используется схема подключе-ния "инородных" программ, изображенная на рис.1-5.

Использование приведенной выше схемы устанавливает минимальные обязательные требования к заказной САПР:

1) ЗакСАПР должна функционировать в рамках ЭВМ и ОС, в среде которых работает оболочка;

2) форматы входных-выходных данных должны согласовываться с БД оболочки.

Эти требования согласуются с концепцией ЗакСАПР. Для инте-грации заказной САПР в открытую оболочку комплексной системы можно выделить три уровня :

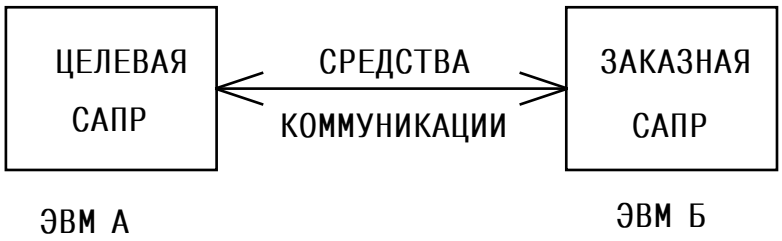


Рис. 1-6

Интеграция в целевую САПР на уровне потоков данных

1) Объединение на уровне организации потоков данных между оболочкой и заказной САПР в ее исходном виде. При этом включаемая САПР используется в той вычислительной среде, для которой она была разработана ранее (рис.1-6). Таким образом могут быть объединены специфические программы расчетов или редактирования данных, рабо-тающие на ЭВМ IBM PC/AT, с комплексными САПР для рабочих станций. Перекодирование и передача данных могут осуществляться любым дос-тупным способом, начиная от переноса на магнитных носителях и кончая использованием ЛВС. Преимущество такого способа интеграции в быстром комплексировании программ в действующий цикл проектиро-вания. Главный недостаток - в сильно затрудненном управлении дан-ными в

оболочке САПР, поскольку импортируемая система фактически от нее изолирована.

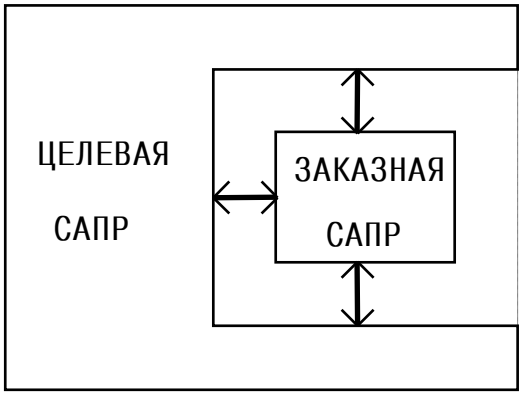


Рис. 1-7

Интеграция в целевую САПР в рамках одной ЭВМ и ОС

2) Объединение в составе оболочки требует (рис.1-7), помимо согласования потоков данных и управления, адаптации заказной САПР к вычислительной системе, на которой функционирует целевая САПР. В первую очередь это относится к интерактивным и графическим средствам. Этот вариант сопряжен с изменением программного кода и более продолжителен. Его основное преимущество в том, что выполнение задач организуется в пределах одного рабочего места, что упрощает обслуживание и сопровождение системы. Кроме того, для пользователей, ранее работавших с данной САПР на других вычислительных машинах, полностью сохраняется сценарий взаимодействия при выпол-нении проектных процедур.

3) Полное включение заказной САПР в оболочку (рис.1-4) предполагает существенную переделку текста программ по следующим причинам: а) необходимо обеспечить единство стиля подачи визуальной информации и способов ведения диалога; б) оболочка и программа должны взаимодействовать строго определенным образом; в) потоки данных должны обеспечиваться и контролироваться средствами обо-лочки САПР. Преимущество такого варианта заключается в предостав-лении пользователю всех возможностей оболочки по управлению проектом и в

удобстве работы, т.к. единство стиля не требует спе-циального обучения и создает комфорт в работе. Однако, данный ва-риант по сути является разработкой новой программы на основе имею-щегося прототипа.

Обобщая представленные уровни интеграции, можно отметить, что каждый из них соответствует определенной степени использования заказной САПР в составе комплекса. Если первый вариант удовлетво-рительно решает задачи при единичном использовании, то последний предполагает массовое применение программного продукта. Только в таком случае усилия, затраченные на интеграцию, будут адекватно окупаться.

Наиболее перспективным вариантом для заказной САПР является соблюдение стандартов CFI для открытой оболочки. Однако исполь-зуемые в настоящее время промышленные системы проектирования прак-тически не удовлетворяют этим стандартам. С другой стороны, сущес-твующие оболочки САПР в стандарте CFI еще очень дороги и слабо распространены у отечественных разработчиков электроники. Поэтому в качестве промежуточного варианта можно ограничиться интеграцией на уровне 2 (см. выше), для чего выработать некоторый временный стандарт для заказных САПР, главная цель которого заключается в минимизации изменений текста программ при их интеграции в различ-ные отечественные и зарубежные системы проектирования.

В основе предложенных решений лежат результаты, ранее достиг-нутые в программировании для обеспечения переносимости программно-го обеспечения [23-27]. Кратко их можно сформулировать в виде следующих рекомендаций:

1) Программы следует строить по модульному принципу, выделяя в отдельные подпрограммы машинно-зависимые алгоритмы и код. Для каждой вычислительной системы пишется свой вариант этих модулей, удовлетворяющий фиксированной спецификации интерфейса.

2) В тех случаях, когда наблюдаются только синтаксические различия в текстах исходной и целевой программ, целесообразно использовать препроцессоры трансляторов языков программирования для автоматической модификации фрагментов программ. Этот способ наиболее предпочтителен в силу строгого формализма, но не всегда реализуем на практике.

3) Необходимо использовать только те базовые типы данных, которые стабильны при переходе от одной ЭВМ к другой. В первую очередь это относится к целым числам и числам с плавающей точкой. В случае ввода / вывода данных в машинном представлении алгоритмы упаковки / распаковки данных должны отслеживать действительный фо-рмат данных.

4) Разработка программ должна вестись с применением контроля корректности используемых данных, ибо, как справедливо замечено в [24], "ошибки, некорректность исходных данных и особенности вычислительных систем являются неотъемлемыми спутниками программ". Это глобальное свойство обусловлено самим характером реализации недетерминированной спецификации задачи в условиях недетерминирован-ного потока исходных данных. В наибольшей степени это относится к интерактивным программам.

5) Для программирования задачи следует использовать языки высокого уровня; машинные языки (язык ассемблера) принципиально непереносимы. Наиболее предпочтительны языки, стандартизованные международными или общеизвестными национальными институтами стандартов. Немаловажную роль играет распространенность языка.

Применяя концепцию оболочек САПР к самой заказной САПР, можно выделить следующие сервисные компоненты (подсистемы) прикладных программ:

1) Графические средства для формирования изображения;

2) Интерактивное взаимодействие с пользователем;

3) Поддержка национального языка;

4) Работа с файловой системой ОС;

5) Управление динамической памятью;

6) Быстродействующая графическая СУБД.

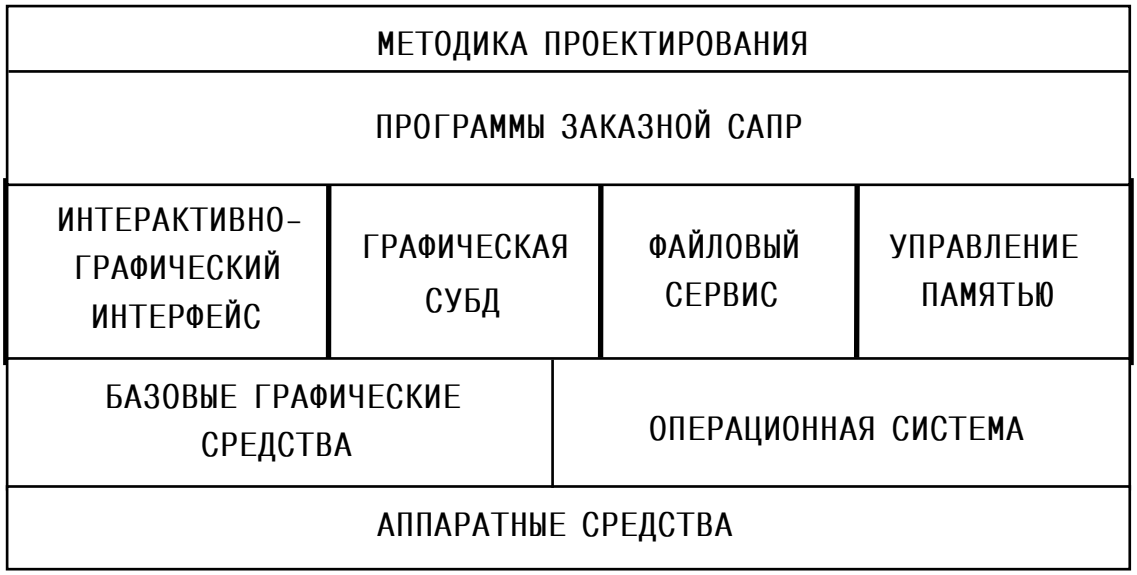


Рис. 1-8

Структура оболочки заказной САПР

Место заказной САПР и ее интерфейсов в вычислительной системе представлено на рис.1-8. Первые три функции были объединены в интерактивно-графическом интерфейсе из-за сильной зависимости друг от друга. В силу того, что используемые сервисные средства практически изолируют программу от внешнего окружения, предоставляя ей некоторую стабильную среду, можно говорить о предложенных стандартах как об оболочке заказной САПР. Модульный принцип построения и возможности настройки придают ей свойства адаптивности к различ-ным вычислительным системам, в том числе и к открытым оболочкам в стандарте CFI.

Комплекс программных интерфейсов. локальных стандартов и соответствующих пакетов инструментальных программ, обеспечивающих наиболее быструю адаптацию заказной САПР к выбранной целевой системе, далее будет рассмотрен подробно. Как будет показано, дос-тигнутые практические результаты позволяют говорить о том, что в базисе использованных интерфейсов возможна интеграция прикладных программ

практически без изменения их исходного текста. В качестве примера на рис.1-9 приведен "ареал" распространения графического редактора топологии "Layout Windows", разработанного на основе данных стандартов.

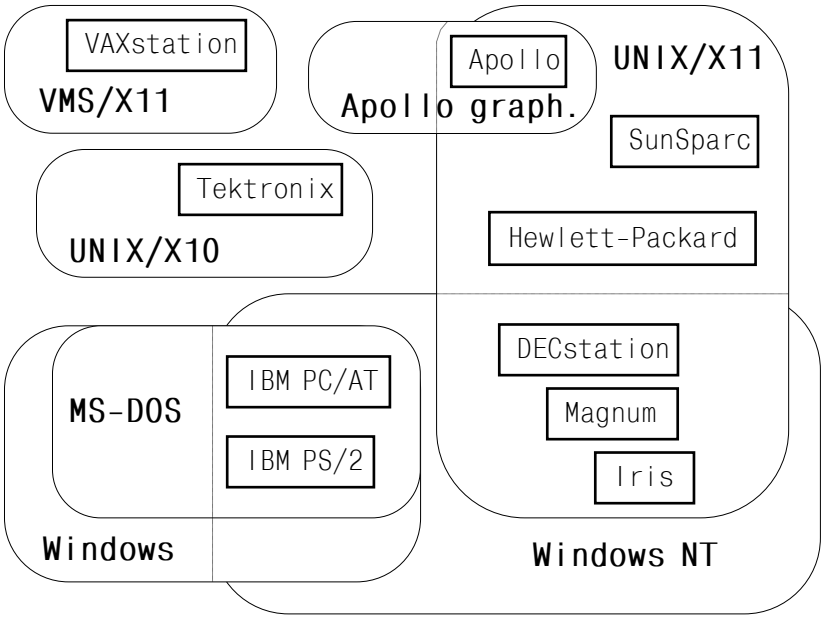


Рис. 1-9

Поддержка ЭВМ и ОС графическим редактором топологии "Layout Windows"

1.3.Постановка задачи разработки адаптивной интерактивно-графической оболочки заказных САПР

Анализ структуры ПО САПР удобно провести на основе системного подхода. Любую систему автоматизации, реализованную в виде программ(ы), можно представить в виде конечной совокупности составляющих ее подсистем, связанных между собой некоторыми отношениями:

$$САПР = \{S_i, \{F_{jk}\}\}, \quad i, j, k = 1..N \qquad (1\text{-}1)$$

где S_i - подсистема, N - число подсистем,

F_{jk} - связь (отношение) подсистемы S_j с подсистемой S_k.

В качестве подсистем могут выступать модули (подпрограммы, функции) языка программирования, на котором написана САПР. Могут использоваться любые уровни детализации, но в данной работе будем использовать наименее абстрактный уровень. Заметим, что из рассмотрения исключаются уровни системного и сервисного ПО, отчасти скрытого от прикладного программиста.

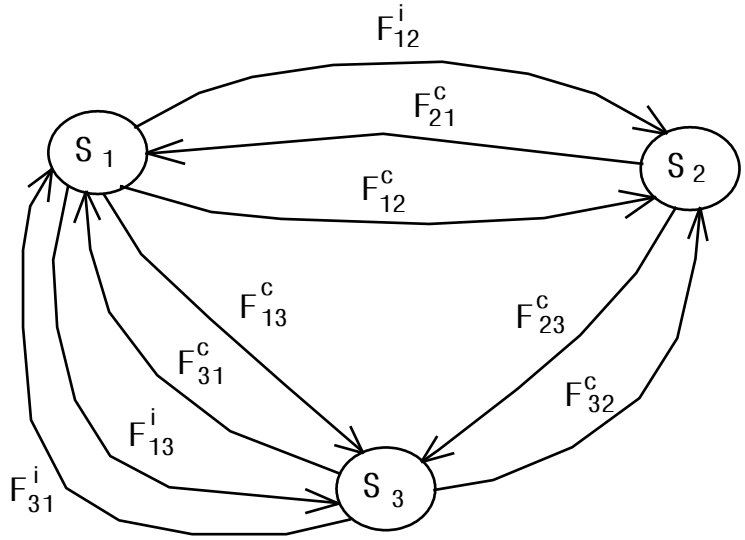


Рис. 1-10

Граф G={s,f} - модель программы САПР

В качестве математической модели системы (1-1) может использоваться представление в виде направленного графа G={s,f}, в котором узлы s соответствуют подсистемам S_i, а ребра f - связям F_{jk} (рис.1-10). В основе такого представления лежит аппарат конечных автоматов или машин Тьюринга. Отметим существенные свойства данно-го графа. Отношение любых двух программных модулей можно подраз-делить на два типа: передача управления F^c_{jk} и передача информации F^i_{jk}. Вообще говоря, передачу управления можно рассматривать и как передачу информации, но используемое разделение более соответ-ствует специфике программирования.

Современная методология программирования [11,12] требует, что-бы каждый модуль имел только один вход и только один выход. Поэтому из

существования F^c_{jk} должно следовать существование F^c_{kj}. Использование механизма подпрограмм (функций) это гарантирует. Передача информации не обладает таким свойством. Даже если сущес-твует F^i_{jk}, из этого не следует существование F^i_{kj}. Однако передача информации может осуществляться и без передачи управления, что происходит при использовании глобальных данных, доступных ряду подсистем. Таким образом, представление в виде гиперграфа позволя-ет описать основные зависимости между модулями системы.

Представление в виде (1-1) не оговаривает единственность существования решения. С практической точки зрения представляет интерес оценка (прогнозирование оценки) реализации САПР по ряду критериев. В настоящей работе рассматриваются только оценки трудоемкости и времени реализации САПР.

Основываясь на системном подходе, с некоторой долей вероятнос-ти можно утверждать, что трудоемкость определяется как сумма трудоемкостей разработки модулей и трудоемкостью взаимной увязки модулей. Последнее определяется наличием соответствующих ребер графа G.

$$Q = \sum_{i=1}^{N} (q_i + \sum_{j=1, j\neq i}^{N} r_{ij}) = \sum_{j=1}^{N} r_{ij} \qquad (1\text{-}2)$$

где $q_i = r_{ii}$ - трудоемкость разработки модуля,

r_{ij} - трудоемкость взаимной увязки модулей i и j.

Точного определения метрик q_i и r_{ij} не найдено. Условимся считать, что метрика r_{ij} принимает значение 0 при отсутствии отношений между модулями i и j; и положительное значение в остальных случаях. Также положим, что метрика q_i положительна.

Если коэффициенты r_{ij} расположить в матрице $R = [r_{ij}]$, то суммарную трудоемкость можно выразить как норму этой матрицы:

$$Q = \|R\| = \sum_{i=1}^{N} \sum_{j=1}^{N} r_{ij}, \quad r_{ij} \geq 0 \qquad (1\text{-}3)$$

В практике структурного программирования [11,12] решение некоторой задачи выполняется как композиция решения нескольких более частных подзадач. Процесс рекурсивно повторяется до полного реше-ния исходной задачи. При этом формируется некоторая иерархия моду-лей, реализующая уровни абстракции в решении общей задачи. Она может быть использована для группирования модулей одного уровня (функционального назначения) и для адекватной перестановки строк и столбцов матрицы $[r_{ij}]$, что позволяет представить ее в блочном виде:

$$[r_{ij}] = \begin{vmatrix} R_{11} & R_{12} & \dots & R_{1B} \\ R_{21} & R_{22} & \dots & R_{2B} \\ \dots & \dots & \dots & \dots \\ R_{B1} & R_{B2} & \dots & R_{BB} \end{vmatrix} \qquad (1\text{-}4)$$

где B - количество образованных блоков.

Блочное представление очень удобно для анализа трудоемкости разработки мобильного ПО на уровне функционально законченных под-систем. Допустим, что требуется программирование Z программ САПР для P=B-Z вычислительных сред. Составим матрицу трудоемкостей $[r_{ij}]$, в которой индексы 1..Z соответствуют программам САПР, а индексы Z+1..B - вычислительным средам. Примем во внимание, что вычислительные среды независимы друг от друга и от программ САПР, что выразится как введение нулевых блочных подматриц.

$$[r_{ij}] = \begin{vmatrix} R_{1,1} & \dots & R_{1,Z} & R_{1,Z+1} & \dots & R_{1,Z+P} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ R_{Z,1} & \dots & R_{Z,Z} & R_{Z,Z+1} & \dots & R_{Z,Z+P} \\ 0 & \dots & 0 & R_{Z+1,Z+1} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & 0 & \dots & R_{Z+P,Z+P} \end{vmatrix} =$$

$$= \begin{vmatrix} R^{Z}_{САПР} & R^{P}_{ад} \\ 0 & R^{P}_{вс} \end{vmatrix} = \begin{vmatrix} R^{Z}_{САПР} & R^{B-Z}_{ад} \\ 0 & R^{B-Z}_{вс} \end{vmatrix} \qquad (1\text{-}5)$$

Норма подматрицы $R^{Z}_{САПР}$ представляет собой трудозатраты на разработку "абстрактного" САПР, т.к. в ней не учитываются особенности вычислительных платформ. Поэтому для дальнейших рассуждений можно положить $\left\| R^{Z}_{САПР} \right\| = const$. Норма диагональной подматрицы $\left\| R^{P}_{вс} \right\| = const$ в силу того, что вычислительные системы берутся "как есть". Подматрица $R^{P}_{ад}$ определяет трудозатраты адаптации "абстрактного" САПР ко всем вычислительным платформам. Минимизация ее нормы может рассматриваться как математическая постановка задачи разработки переносимого адаптивного ПО САПР. Решение задачи можно рассматривать как поиск минимума целевой функции f:

$$f = min \left\| R^{P}_{ад} \right\| \qquad (1\text{-}6)$$

Используя классический подход по построению заказных САПР [1,8-10], требуется учесть P вариантов для Z программ, т.е. в общем случае PZ вариантов. Для типичного САПР из 10 программ для трех различных сред (MS-DOS, UNIX, Windows) потребуется 30 вари-антов интерфейсных модулей, что существенно на фоне затрат на 10 "абстрактных" САПР. Для

минимизации (1-6) в работе [10] было пре-дложено минимизировать функциональный состав системно-зависимых компонент. Это было вполне оправдано при низких требованиях к диа-логовому интерфейсу того периода.

В настоящее время при разработке ПО САПР удобные процедуры взаимодействия с программой рассматриваются как мощный фактор ав-томатизации. Однако большая трудоемкость программирования диалога [28] и его системно-зависимый характер приводят к существенному возрастанию затрат на разработку заказных САПР. Это объясняется тем, что все большее число модулей задействуется в диалоговых про-цедурах, носящих отпечаток вычислительных платформ. В результате теряется основное преимущество подхода ЗакСАПР, когда изменениям или замене подвергается небольшое число модулей.

В качестве главного подхода для минимизации затрат в данной работе предлагается разрабатывать и использовать буферное програм-мное обеспечение, характеризующееся стабильными системно-независи-мыми интерфейсами. В результате ЗакСАПР будет состоять из набора инвариантных к вычислительным средам модулей, обрамленных буферной оболочкой.

На основании изложенного выше сформулируем задачу данной рабо-ты как определение оптимального состава, интерфейсов и выполняемых функций буферных модулей ЗакСАПР, формирующих стабильную среду (оболочку) для остальных проблемных модулей, что позволит миними-зировать затраты на разработку версий ЗакСАПР для новых ЭВМ и/или ОС.

Для математической постановки задачи модифицируем матрицу тру-доемкостей (1-5), в которой учтем влияние интерфейсного ПО введе-нием дополнительного столбца и строки (индекс Z+1 для неблочного вида):

$$[r_{ij}] = \begin{vmatrix} R^Z_{САПР} & R^Z_{инт} & R^P_{зав} \\ R^Z_{ст} & R_и & R^P_{аи} \\ 0 & 0 & R^P_{вс} \end{vmatrix} \qquad (1\text{-}7)$$

Охарактеризуем вновь введенные элементы. $R^Z_{инт}$ отражает затраты на использование интерфейсных подпрограмм и функций для Z программ САПР. $R_и$ есть затраты на разработку методов и алгоритмов, реализующих системно-независимые решения для интерфейса. $R^P_{аи}$ отражает затраты на реализацию интерфейса к P различным средам. И, наконец, $R^Z_{ст}$ выражает затраты на обобщение используемых в Z программах возможностей вычислительных сред и выработку спецификации интерфейса.

Сформулируем задачу разработки адаптивного системно-независимого интерфейса как 1) минимизацию зависимости программ САПР от конкретных вычислительных сред, 2) минимизацию суммарных затрат на разработку интерфейса при условии, что 3) эти затраты будут меньше альтернативной модульной реализации САПР по (1-5):

$$\begin{cases} f_1 = min\left\| R^P_{зав} \right\| \\ f_2 = min\left(\left\| R^Z_{ст} \right\| + \left\| R_и \right\| + \left\| R^P_{аи} \right\| \right) \\ f_2 + \left\| R^Z_{инт} \right\| < \left\| R^P_{ад} \right\| \end{cases} \qquad (1\text{-}8)$$

В идеальном случае можно достичь полной независимости программ САПР на базе буферного интерфейса от особенностей вычислительных сред, т.е. $f_1 = 0$. На практике это вряд ли возможно для произвольно взятого набора программ. Однако для систем автоматизации проектирования это уже может быть реальностью в силу общности

подходов для решения задач и общности пользовательских и системных интерфейсов.

Проведем приблизительную оценку граничных условий в (1-8), исходя из следующих упрощений: 1) трудоемкость разработки прог-раммы на базе интерфейса эквивалентна трудоемкости той же разра-ботки, но "напрямую" к вычислительной среде и равна $r_{инт}$, 2) трудоемкость разработки интерфейсов одинакова для различных сред и равна $r_{аи}$ и 3) интерфейс полностью изолирует программы САПР от особенностей сред. Раскрыв, с учетом приближений, неравенство (1-8.3) до элементов матрицы, получим:

$$\left\| R_{ст}^{Z} \right\| + \left\| R_{и} \right\| + Pr_{аи} + Zr_{исп} < ZPr_{исп}$$

$$r_{аи} < \frac{Z(P-1)r_{исп} - (\left\| R_{ст}^{Z} \right\| + \left\| R_{и} \right\|)}{P} \approx$$

$$\approx Zr_{исп} - \frac{\left\| R_{ст}^{Z} \right\| + \left\| R_{и} \right\|}{P} \quad (1\text{-}9)$$

Таким образом, на разработку дополнительной реализации интер-фейса можно потратить усилий не более затрат на использование ин-терфейса для всего САПР. Для сложных интерфейсов (типа диалого-вого) это вполне достаточный запас.

Последующие главы посвящены исследованию конкретных проблем построения буферных интерфейсов для САПР электроники. Ключевым моментом их разработки является поиск оптимального решения по (1-8).

1.4. Выводы.

Новейшие достижения в области интеграции программных средств САПР позволяют эффективно использовать концепцию заказных систем проектирования. Сложность построения диалоговых систем ставит задачу понижения затрат на разработку высокомобильных ЗакСАПР, для решения

которой в данной работе предложено использовать набор открытых адаптивных системно-независимых инструментальных интер-фейсов - оболочку ЗакСАПР - с помощью которой возможно миними-зировать затраты на реализацию новых вариантов программ как для отчественных, так и для зарубежных САПР. С помощью аппарата матричной алгебры показаны преимущества использования буферного интерфейсного ПО и сформулированы математические критерии для его разработки.

ГЛАВА 2

АДАПТИВНАЯ ОТКРЫТАЯ ГРАФИЧЕСКАЯ ПЛАТФОРМА ЗАКАЗНЫХ САПР

Интерактивно-графические средства оболочки САПР являются одной из самых важных ее частей. Отсутствие единого графического стан-дарта для всех ЭВМ требует исследования линии развития машинной графики в последнее десятилетие и анализа наиболее распространен-ных интерактивно-графических сред с целью выработки подхода к раз-работке соответствующих средств для оболочки ЗакСАПР.

2.1. БАЗОВЫЕ ГРАФИЧЕСКИЕ СРЕДСТВА: КРАТКИЙ ИСТОРИЧЕСКИЙ ОБЗОР

Современные программы для ЭВМ широко используют возможности графики для отображения различной информации и для удобного взаи-модействия с пользователем. Затраты на создание такого математи-ческого обеспечения примерно на 70% [28,29] состоят из затрат на разработку пользовательского интерфейса, который в сильной степени зависит от используемых программных и аппаратных средств машинной графики. Привязка к конкретным графическим пакетам обусловлена требованиями заказчика, индивидуальными особенностями аппаратуры, алгоритмами и интерфейсами графического ПО и т.д.. Это основные причины, по которым разработка современного программного обеспече-ния ведется на базе какого-либо одного графического пакета. Как показывает практика [27], перенос таких программ в другую среду (например, при смене ЭВМ) требует значительной переделки исходного текста.

Существующие графические пакеты и оболочки не обеспечивают полного охвата вычислительной техники и операционных систем. Это

относится даже к такой известной и широко распростаненной плат-форме, как X Window System [29,30], которая требует для себя многозадачной ОС и не может выполняться в рамках однозадачных ОС типа MS-DOS. В связи с этим становится актуальной разработка объединяющей графической платформы, что позволит исключить затраты на модификацию графических приложений при смене вычислительных средств. Прежде чем приступить к изложению разработанной концепции универсальной графической и интерактивной платформы, рассмотрим наиболее важные моменты исторического развития средств машинной графики.

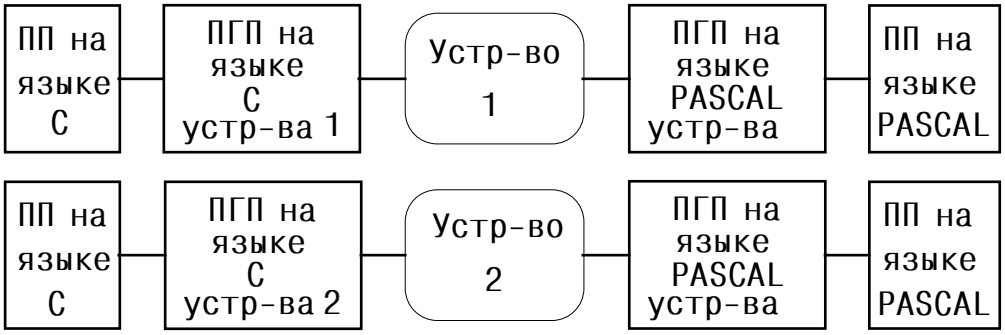


Рис. 2-1

Структура прикладной программы на основе ранних ПГП

По мере появления и распространения устройств графического отображения (дисплеи, графо- и фотопостроители и т.д.) стали разрабатываться различные инструментальные средства, или пакеты графических программ (ПГП), для обеспечения работы с этими устрой-ствами. Первые версии были жестко настроены на конкретную аппара-туру и язык программирования. Для каждой конкретной прикладной области создавался, как правило, свой специфический пакет. Исполь-зование другого устройства, равно как и смена языка программи-рования, приводят к существенной переработке ПП и ПГП. Обобщенная структура прикладной программы (ПП) приведена на рис.2-1. Без по-тери общности ограничимся примером для двух языков программи-рования: C и PASCAL.

В 80..90-е годы рядом фирм - разработчиков компиляторов совместно с национальными институтами стандартов были предприняты успешные

попытки по согласованию сред различных языков програм-мирования, что в особенной мере касалось увязки языка FORTRAN, на котором было написано много прикладных и инструментальных программ, с библиотеками и подпрограммами на современных языках, таких как C, PASCAL, Ada и др. В результате в настоящее время имеется возможность использования ПГП, написанного на языке, отличном от языка прикладной программы. Для согласования программ программистом используются дополнительные декларации, фиксирующие способы передачи параметров в подпрограммы. Такое решение пред-лагается, к примеру, практически во всех современных компиляторах языка C [32-35].

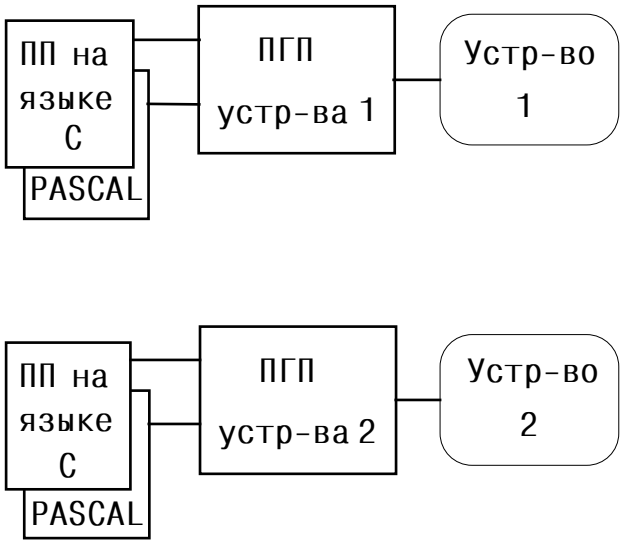


Рис. 2-2

Структура программы на основе универсальных ПГП

Более мощный подход был использован фирмой MIPS при разработке семейства компиляторов для компьютеров на базе архитектуры RISC [36]. Этой фирмой был зафиксирован некоторый общий для большинства современных языков программирования промежуточный уровень, на ко-торый преобразуются программы на всех языках. Дальнейшие преобра-зования программы выполняются над унифицированным промежуточным кодом. Подобные решения позволяют утверждать о языковой независи-мости ПГП в настоящее время. В результате структура обобщенной

программы с универсальным ПГП может быть представлена в виде, изображенном на рис.2-2.

По мере расширения парка графического оборудования начались попытки его унификации на уровне электрического и, что более важно, командного интерфейса. Наиболее ярким примером такого процесса является широкое распространение языка терминалов фирмы Tektronix [37], что на практике означает введение стандарта de facto. Все дисплеи этой фирмы и совместимые с ними, независимо от использующихся способов формирования изображения, обеспечивают выполнение некоторого достаточно мощного общего набора команд. Дисплеи старших моделей выполняют дополнительные команды, расширя-ющие стандартный базис. Как показала практика, прикладные програм-мы, разработанные для дисплеев Tektronix, являются легко переноси-мым и адаптируемым продуктом, т.к. программное обеспечение, бази-рующееся на основных графических командах, будет одинаково работать с любыми дисплеями, и в этом смысле является аппаратно-независимым. Аналогичная ситуация наблюдается для графопострои-телей, где фактическим стандартом является формат HP-GL фирмы Hewlett-Packard [38], и для матричных принтеров, где общепризнан стандарт фирм IBM и Seiko EPSON [39].

Однако, следует заметить, что полной унификации оборудования не произошло ни по одному из типов устройств. Можно говорить о формировании классов устройств, базирующихся на одинаковом наборе команд или эмулирующих их. На практике различные ПГП ограничи-ваются одним классом устройств для достижения наивысшей произ-водительности или функциональной наполненности. В виду того, что в пределах класса могут наблюдаться вариации набора выполняемых команд, ПГП может ориентироваться на минимальный, максимальный или некоторый промежуточный набор команд. В последних двух случаях ПГП должен включать средства эмуляции команд, отсутствующих в кон-кретном типе устройства.

Нельзя не отметить и тот факт, что дисплеи и прочие графичес-кие устройства могут комплектоваться независимо от типа ЭВМ в силу устоявшихся стандартов для аппаратного интерфейса к компьютеру (обычно используется недорогой интерфейс типа RS-232 для терми-налов и интерфейс Centronics для принтеров).

Как дополнение к аппаратной совместимости устройств разви-валась концепция адаптивного программного обеспечения. Применялось включение программного кода для работы с разными классами уст-ройств, либо для уменьшения объема программы при недостатке ресур-сов ЭВМ формировались перекрывающиеся (оверлейные) сегменты кода, или использовались библиотеки динамически загружаемых модулей.

Среди прочих похожих решений выделяется подход, реализованный в ПГП, получившем название MFB (Model Frame Buffer) [40]. Его принципиальное отличие заключается в том, что цепочки символов, представляющие команды для графического оборудования, формируются в процессе интерпретации некоторого языка описания команд. В качестве основы в пакете MFB был использован язык описания команд и ресурсов текстовых устройств, применяемый в операционной системе UNIX в файле termcap для описания терминалов и в файле printcap для описания печатающих устройств [41].

Ключевым элементом такого описания является аппарат поиме-нованных ресурсов, реализованный в виде текстовых строк, чисел и булевых констант. Строки служат для хранения форматов формирования выходного потока на устройство и для определения форматов синтак-сического и семантического разбора сообщений, поступающих от устройства. Численные параметры задают технические параметры типа: количество точек на экране, количество возможных цветов и т.д. Булевые константы используются для задания факта наличия или отсутствия ресурсов или функций, например для информирования о присутствии устройства ввода графических координат. Идентификатор ресурса позволяет установить простую связь между информацией, за-

ключенной в описании, с конкретной программой. Отсутствие необходимой строки или некоторой функции аппаратуры, заданной булевой константой, включает режим эмуляции путем использования других доступных ресурсов.

Для кодирования форматов преобразования данных в строках используются расширенные конструкции функции printf() языка программирования Си. Символ процента и следующая за ним цепочка литер задают преобразование данных в цепочку байтов или обратно. В пакете MFB были использованы дополнительные конструкции преоб-разования данных, в частности, перекодировка в форматы Tektronix, Hewlett-Packard, операции циклического сдвига, маскирования, со-хранения промежуточных результатов в регистрах и многие другие, что делает этот язык достаточно мощным. Конструкции языка MFB позволяют формировать иерархическое описание путем кодирования только исключительных, или особенных, команд и ресурсов данного типа аппаратуры с добавлением ссылки на описание базового набора команд или на младшую модель. Пример описания для векторного дисп-лея УВВК-51-018 приведен ниже:

```
    t1¦T1¦UWWK-51-018 with digityzer,
     MXC#4095, MYC#3071, OFFMX#0, OFFMY#3072,
     OFFDX#4096, OFFDY#1024, TTY,
     GIS=\E^M\E^N\E;\E^]\E,
     GFS=\E^M\E^O\E;$<1000>\E^L$<#1500>\EA\EB\EC\E^_^M,
     BELL=\E^_^G\E^],
     MCL#128, SCS=\E^]\E%X%-#1%&#15%+#96%c,
     DLS=\E^]%t1%t2, DLT=%t1, MPS=\E^]%t1, WPX=\E^]%t1%t1,
     GTS=\E^]%t1\E^_, GTE=\E^],
     GCW#28, GCH#40, GTW#4, GTH#16, APT, GTMR, GTFCH,
     GCS=\E^L$<#1500>\E^],
     KYBRD, KYS=\E^]%t1\E^_\E^O, KYE=\E^N\E^],
     KYB=\E^H, KYX#0, KYY#12,
     LFON=\E^O, LFOFF=\E^N,
     POD, PDB, NPB#4, BU1#65, BU2#68, BU3#73, BU4#81,
     PDS=\EB\EA\E^], PDR=\EB\EE\E^Z, PDE=\EA\EF\E^],
     PDF=%c%Z%T%c%&#31%<<#5%R%c%&#31%¦%R%*#4%X%c%&#31%<<#5%R%c
%&#31%¦%R%*#4%Y,
```

Рис. 2-3

Пример описания дисплея УВВК-51-018 на языке пакета MFB

Возможности пакета MFB позволяют отделить свойства графи-ческого оборудования от исполнительного кода программы, что поз-воляет выполнять настройку программы без перекомпиляции исходного текста. Единственным недостатком данного пакета является его невы-сокое быстродействие, что обусловлено дополнительными затратами времени на синтаксический разбор форматов команд и на преобра-зование данных по формату, которые выполняются тем же процессором ЭВМ. Но на практике такие потери были не всегда заметны по причине невысокой скорости передачи данных по линиям связи. Обобщенная схема прикладной программы на базе концепции MFB изображена на рис.2-4.

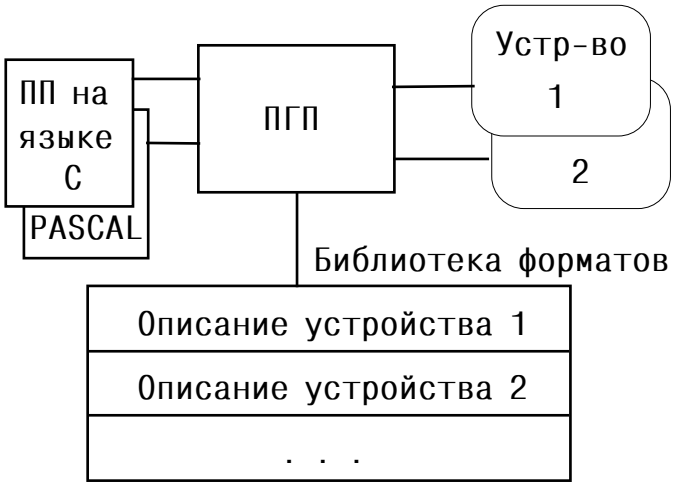


Рис. 2-4

Схема программы на базе графического пакета MFB

В начале 80-х гг. международным комитетом по стандартизации были предприняты усилия по выработке унифицированного стандарта для графических систем, в результате был разработан проект графи-ческой платформы, получивший название GKS (Graphics Kernel System - Ядро графической системы) [42-44]. В ходе выработки стандарта предполагалось выработать концепцию развития машинной графики на 80-е годы путем анализа и объединения потребностей различных групп

пользователей в расчете на массовое применение интерфейса. Это привело к включению в пакет большого количества функций. Следует отметить, что была тщательно проработана структура пакета. В частности, использован механизм драйверов для адаптации к различной аппаратуре, благодаря чему стандарт GKS получил широкое распространение и поддерживается сегодня.

Но кроме очевидных преимуществ унификации, данному пакету программ присущи и серьезные недостатки. В целях унификации координатной системы пакет использует координаты в нормированном (вещественном) виде. С точки зрения быстродействия это не самое лучшее решение, особенно если учесть, что подавляющее большинство графических устройств работает с целочисленными координатами. Огромное количество функций затрудняют программирование пакета для недостаточно мощных ЭВМ. Несмотря на имеющиеся реализации GKS для персональной ЭВМ [43], следует признать, что затраты ресурсов в программе на машинную графику существенны.

В настоящее время взгляд на принципы построения ПГП пре-терпевает существенные изменения, что связано с широким распро-странением персональных компьютеров, рабочих станций, вычисли-тельных сетей, многозадачных операционных систем и удешевлением графического оборудования (дисплеев, графо-построителей и т.п.). Современная персональная ЭВМ или рабочая станция оснащается графи-ческим дисплеем с интерфейсом непосредственно к системной магист-рали (СМ), как это изображено на рис.2-4. При этом исключаются традиционные составляющие времени передачи данных по информаци-онным каналам. Скорость передачи в ПЭВМ или рабочей станции в основном определяется пропускной способностью СМ и быстродействием графического контроллера.

К сожалению, в такой схеме концепция пакета MFB, ориентированного на генерацию цепочек символов и их передачу средствами ОС, утрачивает свои преимущества из-за сложности описания алгоритмов и

форматов команд, работающих с регистрами контроллера (псевдо-ячейками памяти или портами на СМ), а также из-за недопустимо большого времени преобразования данных по сравнению со временем передачи данных по магистрали. Следует также отметить, что в настоящее вре-мя вероятность замены контроллера и/или дисплея в течение жизнен-ного цикла компьютера и/или программы достаточно низка, поэтому обеспечение переносимости прикладной программы, работающей на персональной ЭВМ или отдельной рабочей станции, для пользователя не так актуально. Задача построения адаптивного ПГП остается важной для пользователя программы, работающей в вычислительной сети и для разработчика программного обеспечения, которое должно функционировать на разном оборудовании.

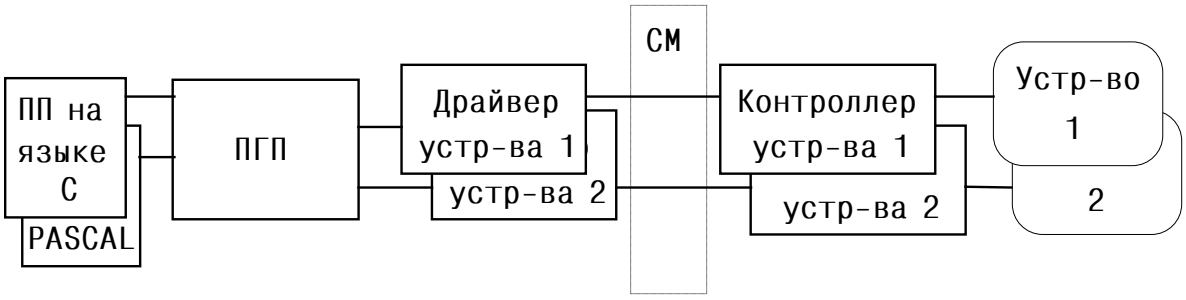


Рис. 2-5

Элементы современной диалоговой системы

Структура современного адаптивного ПГП включает в себя дополнительный элемент - драйвер (сервер) устройства (см. рис.2-5), который служит промежуточным звеном между собственно ПГП и устройством. Его необходимость обусловлена различием способов передачи данных. Графический пакет взаимодействует с прикладной программой на процедурном уровне, драйвер устройства передает данные путем прерываний (ОС типа UNIX) или путем прямого обращения к регистрам устройства (ОС типа MS-DOS). Для передачи данных между ПГП и драйвером фиксируется некоторый протокол и механизм обмена.

Замена устройства обычно сопряжена с заменой драйвера. Для персональных ЭВМ ведущие фирмы-разработчики прикладного программного обеспечения предлагают драйверы для наиболее распространенных контроллеров, стандартный набор которых включает в себя карты CGA, EGA, VGA и SuperVGA. С другой стороны, разработчики нестандартного оборудования предлагают свои версии драйверов для наиболее распространенных пакетов программ. Для поддержки другого контроллера или программы программист должен написать отдельный ПГП или драй-вер.

Для рабочих станций под управлением многозадачных ОС (класса UNIX, VAX/VMS и др.) и объединеных в вычислительную сеть, Массачусетским технологическим институтом, США, была предложена и успешно реализована концепция сетевого графического интерфейса, получившая название X Window System [30]. В качестве сетевого интерфейса для передачи данных между ЭВМ используются стандартные протоколы TCP/IP, а для локального обмена данными - средства меж-процессного обмена ОС. В качестве физического уровня сети наиболее распространены интефейсы типа Ethernet или Token Ring с высокой скоростью передачи (порядка 10-15 Мбит/с для коаксиального кабеля или экранированной витой пары). X Window System является открытой системой и обеспечивает на сегодняшний день наиболее удобную плат-форму для разработки графических программ для рабочих станций.

X Window System являет собой пример успешного глобального проекта в области ПО. В разработке последних версий принимало участие значительное количество программистов во всем мире. Коор-динация версий и поддержка стандарта осуществляется специальным международным органом - X-консорциумом. Мощность и многофункциональность пакета обеспечивают ему положение стандарта de facto для разработки графического ПО широкого применения. Однако большое количество возможностей (~400 функций) приводят к тому, что программы для протокола X11 имеют значительный объем исполнительного кода и требуют больших объемов оперативной памяти на компьютере. Это

условие выполняется для большинства современных рабочих стан-ций и файл-серверов вычислительных сетей, но полноценная реали-зация платформы X Window System на персональных ЭВМ в среде MS-DOS практически невозможна. Имеющиеся X-серверы для MS-DOS и Windows лишь превращают компьютер в графический терминал.

В заключение обзора можно сказать, что на сегодня нет ни одного графического пакета, который был бы повсеместно распростра-нен на любом типе вычислительных машин и поэтому принят за основу для отечественных САПР. Во многом это определяется динамическим развитием средств машинной графики, а также обусловлено различием целей, которые ставились разработчиками пакетов графических прог-рамм.

2.2. Сравнительный анализ программных интерфейсов машинной графики.

Как было показано в предыдущем параграфе, в условиях большого количества типов устройств и графических пакетов разработка пере-носимого программного обеспечения становится трудоемкой задачей. Специфика использования средств графики в том, что они существенно определяют структуры данных и алгоритмы прикладной программы. Рассмотрим принципиальные отличия графических пакетов и возника-ющие проблемы на примере сравнения базовых средств системы X Window System (далее XWS) версии 11 [31], средств среды Microsoft Windows (далее MSW) [45-47] и пакета графики компилятора Си для MS-DOS фирмы Microsoft (далее MSC) [48]. В качестве последнего можно было бы взять графический пакет любого другого компилятора для ЭВМ IBM PC, например фирм Borland, Zortech или Watcom. Сделанный выбор лишь подчеркивает тот факт, что программные про-дукты даже одной фирмы могут иметь сильно различающиеся возможнос-ти и программные интерфейсы.

Старт графического сеанса. XWS предполагает последовательное выполнение процедур установки связи с сервером и создание корневого

окна. Для упрощения пользовательских программ левый верхний угол окна система устанавливает в (0,0). Пакет MSC требует установки графического режима для контроллера, т.к. его обычное состояние - эмуляция текстового терминала. Особенность операции в том, что программа не может точно определить номер видеорежима, что особенно касается карты SuperVGA. Динамический перебор режимов приводит к быстрой смене параметров развертки электронного луча дисплея, а это может вывести монитор из строя. Успешная установка видеорежима приводит к полной очистке видеопамяти дисплея и сбросу всех переменных состояния в исходные значения. Среда MSW требует отказа от использования функции main(). Взамен ее предлагается выполнить инициализацию программы и создание окна в функции с фик-сированным именем WinMain(), в конце которой обязателен цикл обработки сообщений Windows. Программа должна быть перестроена для обработки событий в окне, что представляет существенные трудности при наличии собственного монитора событий. Аналогично XWS координата левого верхнего угла окна (0,0).

Окончание графического сеанса. В XWS желательно выполнить функцию прекращения сеанса, что приведет к уничтожению выделенного окна и перерисовке оставшихся окон. В MSC следует обязательно установить видеорежим, предшествующий запуску программы, чтобы не нарушить функционирование других программ. В MSW окончание работы программы совмещается с уничтожением окна путем посылки специаль-ного сообщения.

Рисование линии (отрезка) в XWS выполняется одной функцией, в MSC и MSW следует выполнить две: одну для установки текущих координат в один конец отрезка и вторую для непосредственного рисования.

Рисование прямоугольника со сторонами, параллельными осям координат. В XWS для закрашенного прямоугольника задается коор-дината левого верхнего угла, ширина и высота прямоугольника. Рисование по контуру выполняется другой функцией, которой следует

задать все точки контура. Для совпадения закрашенного и контурного прямоугольника для первого следует увеличивать ширину и высоту на единицу. В MSC и MSW рисование выполняется заданием координат точек любой из диагоналей. Дополнительный параметр в MSC устанав-ливает режим рисования по контуру или заливку.

Рисование многоугольника в XWS предполагает заполнение специальной структуры данных, в которой, кроме координат точек излома контура, устанавливается метод отсчета координат - относительный или абсолютный. Рисование по контуру или с заливкой выполняется разными функциями. В MSC и MSW координаты точек заносятся в простой массив, дополнительный параметр функции устанавливает режим рисования по контуру или заливку. При рисовании в мелком масштабе в среде MSC зафиксированы фатальные ошибки алгоритма, приводящие к зависанию программы, что требует предварительной фильтрации повторяющихся точек в описании контура.

Рисование текста во всех пакетах представляет собой проблему. В большинстве случаев это связано с отсутствием шрифтов с кирил-лицей в ПГП и с отсутствием документации на формат файлов с описанием шрифтов. Также следует заметить, что многие шрифты защищены авторским правом, поэтому их использование или модифика-ция требуют получения лицензии. В XWS описание формата файла со шрифтом открыто, в MSC и MSW - нет.

Задание цвета рисования в пакетах XWS и MSC выполняется путем указания индекса таблицы, хранящей действительные значения интенсивностей красного, зеленого и синего лучей. Возможно изменение этих значений, но в XWS используются 16-разрядные двоичные числа (0-~64000), в то время как MSC применяет упакованный формат, в котором на каждый цвет отводится 6 двоичных разрядов (0-63), причем ряд контроллеров игнорирует часть этих битов, огрубляя значение. Таблица цветов персонального компьютера получает начальные значения при установке видеорежима и эти значения одинаковы для контроллеров

разных фирм. В XWS формирование таблицы цветов выполняется сервером и, как правило, эта таблица различна для разных ЭВМ, что порождает проблему формирования одинаковой цветовой палитры для прикладной программы. В MSW для изменения цвета следует изменить цвет так называемых "кисти" (brush) и "карандаша" (pen). Цвет задается в упакованном виде, диапазон значений интенсивностей базовых цветов - от 0 до 255.

Задание типа штриховки линий. В XWS специальная функция использует описание в виде массива чисел - поочередных длин штриха и пробела. В MSC функция установки требует передачи 16-разрядной двоичной маски с образцом штриховки. В MSW образцы штриховки ограничены набором из 6 образцов и, кроме того, штриховка входит в описание "карандаша", что требует его переопределения как при изменении штриховки, так и при изменении текущего цвета.

Задание типа заливки в пакетах XWS и MSC выполняется одинаково путем использования двумерной битовой маски и вызовом соответствующей функции. Для MSC размер маски фиксирован - 16x16. В MSW заливка произвольного вида входит в виде двумерного массива пикселов в описание "кисти", поэтому кроме переопределения "кисти" следует изменять и этот массив всякий раз, когда изменяется цвет.

Задание локальной зоны рисования (зоны обрезки) в XWS выполняется различными способами, начиная от битовой двумерной маски и кончая контурным описанием многоугольника. В MSC исполь-зуется только прямоугольная область, при обрезке фигур на границе области формируется не всегда желательная линия. В MSW возможно задание произвольной многоугольной области.

Установка графического контекста используется в XWS и MSW, причем дополнительный аргумент - указатель контекста присутствует во всех функциях рисования. Установка цвета, штриховки, заливки и многих других параметров действует только на заданный контекст. Пакет MSC работает в режиме одного контекста.

Вид курсора в XWS задается путем указания "горячей" точки и двух битовых двумерных массивов для формирования изображения. Размеры массивов произвольны в некоторых пределах. Курсор изоб-ражается постоянно, без гашения. В MSC аналогичные возможности отсутствуют. Формирование курсора обычно выполняется программным путем или с помощью сервисных функций драйвера "мыши" или цифрового преобразователя, параметры которых аналогичны параметрам для XWS. Современные драйверы редко поддерживают режимы SuperVGA графических контроллеров, в связи с чем изображение курсора должно формироваться программой. Кроме того, следует синхронизировать появление изображения курсора с рисованием на экране, чтобы избе-жать ошибок. В MSW предложен набор из нескольких встроенных кур-соров, но также есть возможность программирования драйвера мыши.

Ввод текста с клавиатуры в XWS и MSW выполняется путем про-смотра очереди событий на предмет поиска нажатий клавиш. Получае-мое число - так называемый скан-код - может быть переведен в код ASCII. Эхо-печать символов выполняется программным путем. Кодиров-ка кириллицы и переключение на национальный регистр теоретически присутствуют. В пакете MSC специальные средства для ввода текста отсутствуют, поэтому следует использовать функции ввода стандарт-ной библиотеки языка Си и организовать, при необходимости, эхо-печать. Переключение на национальный регистр зависит от наличия дополнительного драйвера клавиатуры.

Ввод графических координат в XWS и MSW выполняется путем про-смотра очереди событий на предмет поиска интересующих событий. В MSC специальные средства отсутствуют, поэтому следует использовать функции ввода стандартной библиотеки языка C или использовать сервисные функции драйвера "мыши" или цифрового преобразователя. Возможны проблемы с перемещением курсора для ряда драйверов при работе в режимах SuperVGA.

Средства синхронизации. XWS базируется на функциях обмена данными между процессами при локальном обмене или на функциях сетевого протокола при работе в вычислительной сети. Для ускорения передачи информации используется буферизация, что приводит к задержке появления графики на экране. Для принудительной передачи незаполненного буфера используется отдельная функция. Функции рисования, как и многие другие, передают управление вызывающей программе сразу после занесения данных в буфер, не дожидаясь его передачи. В пакете MSC все функции работают непосредственно с контроллером и передают управление вызывающей программе только после полного выполнения операции. В MSW в промежутках между обработкой очереди событий активная программа монопольно владеет процессором, поэтому синхронизация не требуется.

Приведенное сравнение доказывает различия не только в синтаксисе используемых процедур, но и в их функциональном составе и возможностях, что обусловлено различиями в алгоритмах функционирования и отличиями в семантике внутренних структур данных базовых графических пакетов. По этой причине адаптация вызовов графических процедур средствами препроцессора затруднительна или в ряде случа-ев просто невозможна.

2.3.Постановка задачи разработки открытого адаптивного интерактивно-графического интерфейса

Как показывет проведенный выше анализ, необходима разработка специального системно-независимого интерактивно-графического интерфейса. Учитывая опыт аналогичных проектов, можно утверждать, что разработка еще одного набора графических программ на известных принципах может дать аналогичный результат: большой труднопереносимый пакет графических программ. Поэтому необходимо измененить саму концепцию построения графического интерфейса как пакета программ.

Обратимся к выводам параграфа 1.3. Если рассматривать графический интерфейс как средство изоляции прикладных программ от особенностей вычислительных сред, то при его успешной реализации в качестве прямых накладных расходов на создание САПР выступают только затраты на реализацию самого интерфейса (критерии 1-8.2 и 1-8.3). В качестве косвенных затрат выступают затраты на его применение в программах САПР. Если затраты на формирование стан-дарта $R^Z_{ст}$ рассматривать как необходимую, но фиксированную часть общих затрат на интерфейс, то минимизации должна подвергнуться сумма $\left\| R_и \right\| + \left\| R^P_{аи} \right\|$. Заметим, что влияние $R_и$ уменьшается по мере увели-чения количества поддерживаемых платформ P по причине одно-кратности затрат.

Таким образом, одним из факторов уменьшения затрат является упрощение реализации интерфейса для конкретной вычислительной платформы. Если справедливо $\left\| R^P_{аи} \right\| = \sum_{i=1}^{P} \left\| R^i_{аи} \right\|$ в силу свойств нормы матрицы затрат (1-5), то $R^i_{аи}$ имеет размерность n x p, где n - количество модулей в интерфейсе, а p - количество задействованных модулей (средств) вычислительной среды.

Уменьшение числа функций n приводит к пропорциональному уменьшению нормы матрицы $R^P_{аи}$. Также уменьшается размерность матрицы $R^Z_{инт}$, но при этом нельзя с той же уверенностью утверждать об уменьшении ее нормы. Уменьшение числа функций необходимо влечет увеличение количества их вызовов и рост сложности алгоритмов вызовов, когда требуются сложные операции. Увеличение числа функ-ций до некоторого значения облегчает использование, но усложняет реализацию, хотя последние затраты должны относиться на Z программ САПР.

Уменьшение числа функций также уменьшает норму матрицы $R^P_{аи}$ за счет того, что используется меньшее количество модулей (средств) вычислительных сред, т.е. фактор p в используемых обозначениях. Более

того, если использовать готовые пакеты графических программ, то существенно уменьшаются затраты на программирование машинной графики, т.к. можно использовать имеющиеся процедуры и функции, выполняя только информационное согласование. Этот подход следует признать основным для разработки заказных САПР в кратчайшие сроки. Заметим, что такое решение в корне отличается от традиционной стратегии проектирования ПГП, ориентированной на обмен данными непосредственно с устройством или сервером.

Подводя итог, можно сказать, что определение оптимального количества функций для комплексной минимизации затрат есть многокритериальная задача, в сильной степени зависящая от специфики прикладной области автоматизации. Математически ее можно сформулировать как поиск минимума функции:

$$min\ Q(n) = K_{исп} \left\| R_{исп}^{zn} \right\| + K_{аи} \left\| R_{аи}^{pn} \right\| \qquad (2\text{-}1)$$

где коэффициенты $K_{исп}$ и $K_{аи}$ устанавливают значимость затрат на использование и адаптацию соответственно.

Вторая задача обусловлена современными требованиями к разработке программного обеспечения и заключается в обеспечении открытости, адаптивности и системной независимости. Специфическим для отечественных условий требованием выступает поддержка национальных языков для прикладных программ.

2.4. Методы реализации и внутренние структуры данных интерактивно-графического интерфейса.

2.4.1. Определение оптимального функционального состава.

Как отмечено в [13], оптимальное решение - это лучшее в том или ином смысле проектное решение, допускаемое обстоятельствами. Это определение отражает суть поиска оптимума при множестве кри-териев, что имеет место для сложных технических систем, примером которой

является САПР. Поэтому реальной задачей является поиск некоторого локального оптимума на основе известных математических методов. Рассмотрим некоторые из них, выбрав в качестве критерия функцию трудозатрат на разработку интерфейса $Q(Z,n,p)$ и функ-цию длительности разработки $T(Z,n,p)$.

1. Использование частных критериев возможно при установлении точной функциональной зависимости уравнений математической модели от исследуемой переменной, что позволяет использовать аналитичес-кие вычисления для поиска минимума (максимума). Для анализа прог-раммного обеспечения установление таких зависимостей чрезвычайно сложно и поэтому метод носит сильный субъективный характер.

Заметим, что можно определить нижнюю границу числа функций согласно следующим рассуждениям. Если ориентироваться на современ-ные технические средства визуализации информации - растровые дисп-леи на электронно-лучевых трубках или жидкокристаллических индика-торах - то можно определить теоретически минимальный набор функций (модулей). В этих устройствах в основе формирования изображения лежит принцип отображения двумерного массива чисел-точек на плос-кость экрана. В общем случае каждый элемент массива задает "цвет" соответствующей точки. Для черно-белых экранов значение 0 обычно задает гашение точки, а 1 - ее свечение. Для остальных видеосистем элементы массива служат номером цвета или индексом таблицы с палитрой цветов (градациями серого цвета).

Для "раскрашивания" такого экрана достаточно одной функции установки цвета произвольной точки. Дополнительная функция может определять операцию установки цвета как функцию двух аргументов - задаваемого цвета и текущего цвета. Далее, если принять во внима-ние, что для быстрого отображения текста используется специальный неграфический режим, необходимо ввести функцию установки режима (или две функции: для установки и сброса графического режима). Еще две

функции необходимы для взаимодействия с клавиатурой и устрой-ством графического ввода координат типа "мыши" или планшета.

Использование приведенного минимального набора функций не все-гда удобно по причине его слабых возможностей. Функциональное усиление может выполняться двумя способами: на базе самого интер-фейса или с помощью функций базовых пакетов программ. Первый спо-соб не требует программирования для каждой новой платформы, но мо-жет уступать по быстродействию из-за невозможности использования аппаратуры. Второй способ, наоборот, позволяет получить большую скорость, но требует программирования при смене базового ПГП.

2. Для дальнейшего уточнения числа функций можно воспользо-ваться аддитивными (2-2) или мультипликативными методами (2-3), для чего компоненты целевой функции f должны нормироваться. Для учета степени важности дополнительно используют весовые коэффици-енты:

$$f = c_Q \frac{Q(Z,n,p)}{Q^{(0)}(Z,n,p)} + c_T \frac{T(Z,n,p)}{T^{(0)}(Z,n,p)} \qquad (2\text{-}2)$$

$$c_Q \frac{\Delta Q(Z,n,p)}{Q^{(0)}(Z,n,p)} + c_T \frac{\Delta T(Z,n,p)}{T^{(0)}(Z,n,p)} = 0 \qquad (2\text{-}3)$$

где $Q(Z,n,p)$ и $T(Z,n,p)$ - функции трудоемкости и дли-

тельности проектирования,

$Q^{(0)}(Z,n,p)$ и $T^{(0)}(Z,n,p)$ - нормирующие значения

c_Q и c_T - весовые коэффициенты

Эти методы также труднопримени мы из-за неточности метрик прог-раммного обеспечения и поэтому их использование носит сильный субъективный характер.

3. Максиминный (минимаксный) критерий позволяет получить ло-кальный оптимум путем оптимизации "узких" мест. Применительно к интерактивно-графическому интерфейсу он может использоваться для оптимизации функционального состава, если для оценки использовать

методы экспертных оценок. При разработке локального стандарта, каким является рассматриваемый интерфейс, получение этих оценок достаточно просто и они вполне обоснованы. Математически получение оптимального решения выражается как равенство нормированных значений составляющих целевой функции:

$$c_Q \frac{Q(Z,n,p)}{Q^{(0)}(Z,n,p)} = c_T \frac{T(Z,n,p)}{T^{(0)}(Z,n,p)} = const \quad (2\text{-}4)$$

В качестве метода обнаружения "узких" мест можно использовать данные статистического исследования. Суть подхода в подсчете относительного числа вызовов каждой функции в программах. Для прогнозирования можно использовать прототипы программ, для анализа - существующие варианты программ. Можно оценивать как суммарное количество используемых функций, так и частоту их применения с уче-том вложенности программ.

Используя данные частотного анализа, можно определить функции, для реализации которых необходимо использовать базовые графические средства с целью повышения быстродействия. Это, в первую очередь, функции рисования прямоугольника, линии и текста. Остальные функции, если есть такая возможность и не предполагается ухудшения скоростных характеристик, следует реализовывать на базе самого интерфейса, чтобы исключить затраты на адаптацию к вычислительным средам. При этом обнулятся соответствующие строки матрицы $R^P_{аи}$.

4. Метод последовательного конструирования [13], по мнению автора, может быть наиболее успешно применен для рассматриваемого интерфейса. Метод представляет собой обобщение принципа оптимальности динамического программирования для решения многокритериаль-ных задач. Кратко он может быть сформулирован как определение в множестве (возможных) проектных решений W такого подмножества $W^* \subseteq W$, которое инвариантно для любого опыта (эксперимента, реализации и

т.п.) из множества опытов $O = \{O_i\}$. Путем последо-вательного применения метода можно добиться отбрасывания бесперс-пективных вариантов до получения оптимального решения.

Два последних метода использовались для формирования функцио-нального состава и структур данных рассматриваемого интерфейса. В качестве метрики применялись экспертные оценки. Следует заметить, что специфика эксперимента заключается в достаточно продолжитель-ном времени для разработки пробных вариантов интерфейсов и при-кладных программ.

В качестве первого набора вариантов реализации (начало 1991 г.) рассматривались готовые ПГП: 1) пакет MFB для графических терминалов, управляемый потоком символов, 2) пакет X Window System версии 10, 3) графика QuickC фирмы Microsoft и 4) пакет GKS. В качестве опытов проводилось программирование различных ЗакСАПР на базе указанных пакетов. По результатам разработки анализировались экспертные оценки трудоемкости, времени программирования и требу-емые функции. Было замечено, что 1) ряд функций в ПГП не исполь-зуется совсем, 2) некоторое подмножество использованных функций обладает свойством инвариантности, т.е. требует примерно одина-ковых затрат и времени на программирование задач и 3) с целью при-ведения к единому виду следует зафиксировать спецификацию графи-ческого интерфейса.

Вторая серия опытов (реализаций программ) с использованием первого варианта вновь сформированного ПГП преследовала цель опре-делить влияние типов и количества параметров графических функций и влияние внутренних структур данных. Дополнительно анализировались реализации интерфейса для сред X Window System версии 10, Micro-soft Windows и графики для MS-DOS (пакет профессионального прог-раммиста версии 6.0 фирмы Microsoft). В результате были сформи-рованы устойчивые инвариантные представления внутренних структур данных интерфейса и уточнена его спецификация.

Третья серия испытаний проводилась над всеми реализациями интерфейса для выявления его инвариантных частей. В результате были выявлены фрагменты текстов, различающиеся только синтаксисом. С помощью аппарата макроопределений эти различия были убраны, а идентичные части текстов оформлены как отдельные файлы. Что очень важно, пять из пятнадцати основных функций попали в разряд инвари-ант.

2.4.2. Метод инвариантного индексирования ресурсов графического контекста.

Как было отмечено в параграфе 2.2, различие структур данных, которые служат для описания (задания) графических ресурсов, является основным препятствием для приведения спецификаций функций различных ПГП к одному виду. Чтобы эта проблема не возникала в дальнейшем при реализации варианта интерфейса для очередной платформы, следует использовать такие структуры данных, которые могли бы служить инвариантой. Но, с другой стороны, усложнение таких структур усложнит и алгоритмы их обслуживания, что не всегда жела-тельно.

Для решения этой проблемы в данной работе предложено осущест-вить двойную буферизацию описания для всех ресурсов, а в качестве инварианты использовать целые числа - индекс (класс) ресурса и индекс варианта ресурса по таблице с системно-зависимым представлением (рис.2-6). Использование индексов, в отличие от других мето-дов, предоставляет следующие преимущества:

1) Появляется возможность отделить установку (активизацию) ресурса от его определения, а формирование ресурса - от исполнительного кода программы.

2) Образы (структуры данных) ресурсов определяются (формируются) способом, специфическим для каждой реализации интерфейса, но заносятся в "невидимую" для прикладного программиста таблицу.

3) Функции запроса и установки предельных и текущих значений индексов становятся во многом инвариантными к реализации.

4) Индексирование минимизирует затраты на организацию графического контекста, в качестве которого выступает таблица индексов, а не таблица системно-зависимых представлений.

5) Использование дополнительного контекста, состоящего из предельных значений индексов, позволяет привести к единому виду операцию проверки устанавливаемых индексов на соответствие диапазону допустимых значений.

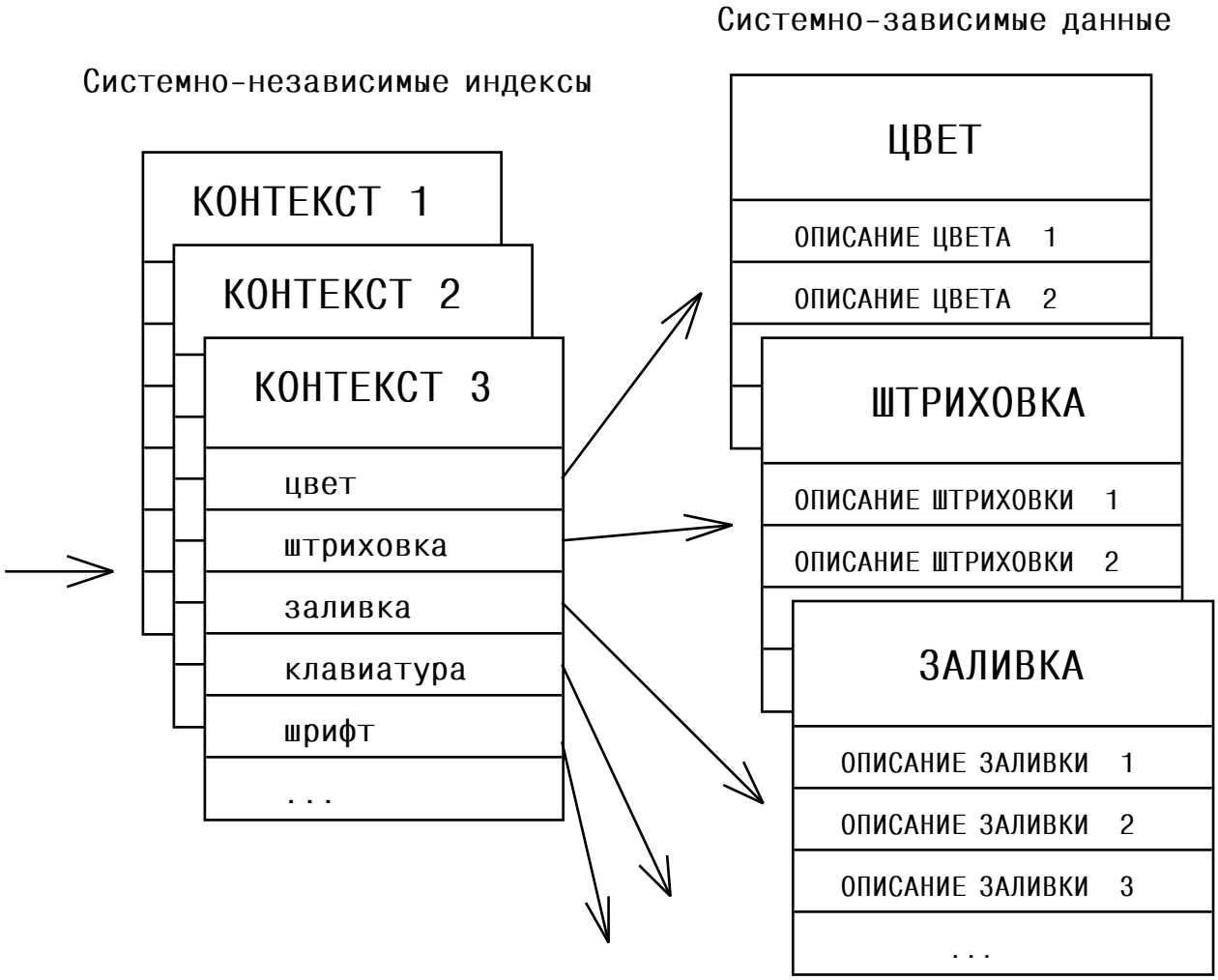


Рис. 2-6

Схема двойной буферизации описания ресурсов

Понятие графического контекста наиболее плодотворно для современного развития графических оболочек, поэтому ему придается особое значение. Использование контекста позволяет сократить коли-чество задаваемых параметров за счет извлечения необходимых данных из

внутренней памяти интерфейса. В явном или неявном виде понятие контекста используется практически во всех ПГП, однако способы задания параметров контекста обычно различаются от параметра к параметру.

В настоящем интерфейсе сделана успешная попытка приведения операций управления ресурсами и самим контекстом к единому виду, для чего используется двойная буферизация параметров контекста (рис.2-6). Число - номер параметра контекста, или класс ресурса - служит индексом для доступа к таблице инвариантных значений пара-метра, а значение из этой таблицы, служащее номером-вариантом ре-сурса, содержит индекс для второй таблицы с описанием системно-зависимых данных. Значения последних используются в качестве пара-метров при вызове функций базовой графической платформы.

Структуры данных второй таблицы весьма различаются от плат-формы к платформе. Заметим, что если задание палитры цветов аппа-ратно невозможно, для установки цвета вторая таблица может не использоваться совсем. Или, если образец заливки может храниться в невидимой части дисплейной памяти, во второй буфер заносится только его идентификатор (ссылка). Для систем со слабо развитой графической системой (например, стандартные контроллеры персональ-ных ЭВМ) приходится хранить битовые карты образов для многих параметров.

2.4.3. Метод представления ресурсов в виде обобщенной битовой карты.

Используемые интерфейсом ресурсы имеют различное назначение и поэтому им соответствуют различные структуры данных. Соответствен-но должны различаться и функции обработки этих ресурсов. Для сокращения способов описания ресурсов в данной работе предложено разделить логическую организацию и метод хранения ресурсов.

Рассмотрим наиболее близкие по характеру ресурсы - образец штриховки линии, образец закраски фигур, образец курсора и шрифт - на примере рис.2-7.

Главная общая деталь описания изображенных ресурсов - это матрица точек, или, по принятой в машинной графике терминологии, битовая карта (bitmap в английской нотации). Матрица может быть одномерной (в случае штриховки), двумерной (в случае заливки) и трехмерной (в случае курсора и шрифта). Размер матриц по каждому из измерений есть параметры, характеризующие соответствующий ре-сурс. Эти параметры битовой карты (БК) положены в основу обобщен-ного описания. Само описание точек может храниться любым из известных методов. Для сокращения занимаемой памяти может исполь-зоваться упаковка точек в байт (цепочку байтов), когда каждой точке БК может быть поставлен в соответствие один бит.

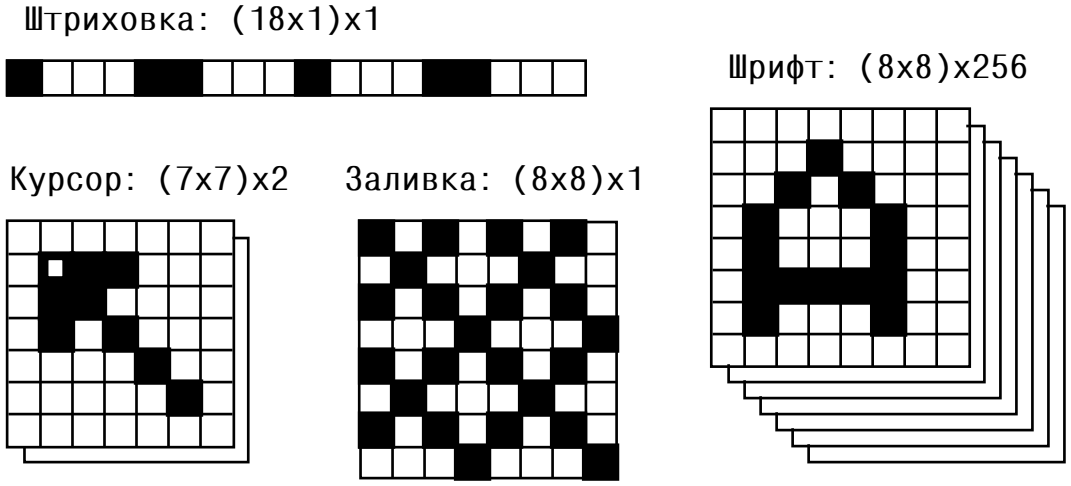


Рис. 2-7

Примеры графических ресурсов

Заметим, что в случае описания курсора появляется как минимум один дополнительный параметр - координаты т.н. "горячей" точки, которая соответствует указываемой точке при пользовании курсором. Для других ресурсов могут понадобиться другие параметры. Обобщая эти

требования, установим, что в описании ресурсов входит как БК, так и список параметров.

Для единства описания необходимо, чтобы оставшиеся ресурсы (таблицы перекодировки клавиш и цвет) также описывались данным методом. Это вполне возможно, если принять во внимание, что при любом методе хранения БК данные в ЭВМ проецируются на последо-вательную цепочку байт или машинных слов. Для всех ресурсов это осуществимо. Таблица перекодировки клавиш может быть представлена как БК размерности n*1*k, где k - количество клавиш и n - коли-чество чисел для описания одной клавиши. Для цвета аналогичная матрица может иметь размерность 3*1*1, если использовать трехцвет-ную кодировку или систему цвет-яркость-насыщенность. Более эко-номичный вариант для кодировки цвета заключается в хранении трех характеризующих чисел в поле параметров. На рис.2-8 приведено полное описание ресурсов интерфейса.

Таким образом, при различии логических описаний ресурсов в качестве единой структуры данных для хранения ресурсов выступает битовая карта в совокупности с характеризующими параметрами. Для удобства эти параметры объединены в заголовок фиксированного фор-мата. Заголовок является текстовой строкой из трех-девяти десятичных целых чисел, заключенной в квадратные скобки. Выбор именно этих символов для скобок не имеет принципиального значения, совершенно свободно могут использоваться другие литеры.

Смысловое значение первых трех чисел заголовка фиксировано: это формат чисел, формирующих БК (десятичные, шестнадцатеричные числа или упакованный двоичный формат), длина в байтах двумерного среза БК и их количество. Эти параметры позволяют быстро определить размеры памяти, требуемой для хранения БК, и формат, в котором хранится сам образ. Смысловое значение остальных чисел зависит от назначения образа (рис.2-8). Для ресурсов типа битовой карты (заливка, штриховка, шрифт и курсор) в поле данных заносятся побайтно линии

сканирования образа, младший бит в байте соот-ветствует правой из восьми точек.

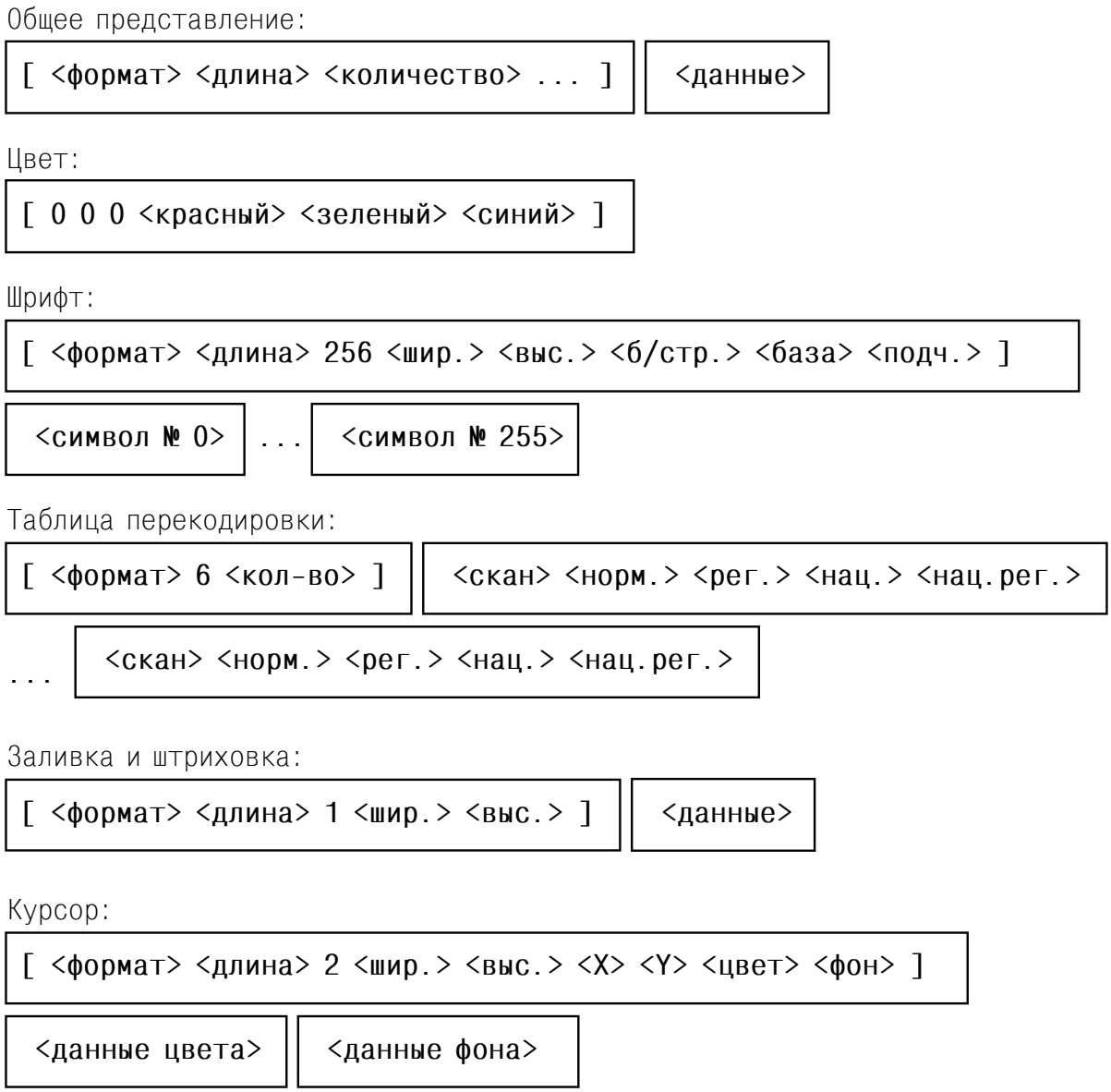


Рис. 2-8

Отображение логического представления ресурсов на обобщенную битовую карту

Для задания палитры цвета образ не используется совсем, три числа заголовка задают интенсивность красного, зеленого и синего лучей дисплея числами в диапазоне 0..255. Для определения шрифта (гарнитуры), образцов заливки и штриховки, описания курсора два числа задают ширину и высоту образа в точках экрана. Для шрифта дополнительно указывается количество байт на одну строку литеры,

высота базовой линии и линии подчеркивания. Для курсора задаются координаты "горячей точки" и цвет рисунка и фона. Образ таблиц перекодировки фиксирован: каждая логическая единица состоит из шести полей - первые два задают код, возвращаемый графической пла-тформой и используемый для поиска в таблице образа, а следующие четыре определяют действительное значение кода для четырех состо-яний: нормальное, верхний регистр, национальный регистр, верхний национальный регистр.

Таким образом, ввиду общности описания, для задания ресурсов в прикладной программе можно ограничиться только двумя функциями: 1) системно-независимой функцией чтения и загрузки в память обобщен-ной БК и 2) функцией формирования таблицы с описанием ресурсов (рис.2-7) на основе обобщенной БК.

2.4.4.Функциональный состав и спецификации интерактивно-графического интерфейса.

Разработанный на основе изложенных методов набор программ на языке Си получил название Н-интерфейса. Как показала практика, наиболее эффективный вариант реализации интерфейса - отдельный программный код для каждой графической платформы. Необходимые модули компилируются и подключаются к прикладной программе обычным для систем программирования образом. Объединение модулей разных платформ, функционирующих в рамках одной ОС, нецелесообразно по причине кардинальных различий во внутренних структурах данных и увеличении времени компиляции при отсутствии видимых преимуществ. Особенно это относится к ситуации разработки программного обеспечения на заказ, когда спецификация вычислительной среды известна точно.

В интерфейсе применен ряд известных методов [23-27,44] защиты внутреннего состояния от возможных ошибок прикладной программы. Во-первых, все алгоритмы и операции, для которых есть риск возни-

кновения ошибок или неоднозначностей, выполняют проверку корректности заданных параметров и возвращают код ошибки в случае неудачи. Во-вторых, все важные структуры данных интерфейса явля-ются внутренними (локальными) данными без права доступа извне. Их изменение выполняется только посредством интерфейса. Для получения истинных характеристик используемого оборудования или графической платформы применятеся механизм запроса максимальных значений для каждого параметра контекста путем вызова функций интерфейса.

Для исключения перекодировки координат, вызванной произволь-ным направлением осей координат в оборудовании или базовом ПГП, интерфейс не меняет направление осей имеющейся системы координат. Для информирования прикладной программы о действительных направле-ниях осей по отношению к правой декартовой системе выполняется за-прос к интерфейсу. Для ряда подпрограмм, для которых не важно направление осей, например рисование прямоугольника, выполняется автоматическая корректировка параметров без установки кода ошибки. При установке локальной области аналогичная проверка и корректи-ровка позволяет прикладной программе получить истинные координаты границ в системе левый-правый / верх-низ, естественной для пользо-вателя программы.

Ниже приведено краткое описание функций H-интерфейса:

Начало графической сессии. Передаются стандартные параметры функции main() для извлечения параметров командной строки запуска прикладной программы, name задает строку-заголовок для передачи программам управления окнами в многозадачных ОС. Для первона-чального определения параметров контекста используется информация внешних инвариантных файлов или буферов прикладных подпрограмм с описанием, которое преобразуется в вид, приемлемый для исполь-зуемой графической платформы. Данные, необходимые для последующих установок, после выполнения действий, специфических для конкретной реализации интерфейса, заносятся во внутреннюю таблицу интерфейса.

```
int Hopen( int argc, char * argv[], char * name )
```

Окончание графической сессии не требует никаких параметров.

```
void Hclose()
```

Ввод описания ресурсов из файла. Передается спецификация файла с описанием ресурсов. Формат файла и алгоритмы этой функции по-зволяют делать ссылку на другой файл с дополнительным (основным) описанием.

```
int Hcap( char * file_spec )
```

Определение ресурсов. Передаются индексы ресурса (цвет, закраска и т.д.), источник получения данных (имя файла, дескриптор файла или буфер в памяти) и указатель на данные.

```
int Hdefine( int resource, int source, void * info )
```

Установка значений контекста или номера контекста. Передаются индекс ресурса (цвет рисования, цвет фона, цвет курсора, штрихов-ка, закраска, шрифт, таблица клавиатуры, таблица "мыши", привязка по x или по y, функция рисования, режим вывода текста, локальная область или номер контекста) и устанавливаемое значение. Для установки локальной области передаются указатели на координаты левого, нижнего, правого и верхнего угла области, которые после выполнения операции сортируются соответствующим графической плат-форме образом.

```
int Hset( int resourse, int index )
int Hset( Hviewport, int *left, int *bottom, int *right, int *top)
int Hset( int resourse, ... )
```

Запрос значений контекста или номера контекста. Передается индекс ресурса, возвращается устанавленное значение. Дополнительно к приведенным выше индексам ресурса можно запросить направление осей координат, высоту и ширину знакоместа символа. Для запроса локальной области передаются указатели на координаты левого, нижнего, правого и верхнего угла области, в которые заносятся границы соответственно графической платформе. В запросе можно задать возврат текущих для данного контекста значений или их максимальные значения.

```
int Hquery( int resourse, ... )
```

Рисование. Интерфейс выполняет рисование точки, отрезка, ломаной линии, многоугольника, прямоугольника, текста и области пикселов и включает задание координат, специфических параметров и усло-вие выполнения обрезки примитива clip. Все необходимые для рисова-ния параметры определяются контекстом.

```
int Hdrawpoint( int x, int y, int clip )
int Hdrawline( int x1, int y1, int x2, int y2, int clip )
int Hdrawbrokenline( int xy[], int vertices, int clip )
int Hpolygon( int xy[], int vertices, int clip )
int Hdrawtext( int x, int y, char * text, int clip )
int Hdrawarea( int x, int y, void * area )
```

Запрос событий. Служит для опроса клавиатуры и устройства ввода графических координат на предмет событий, задаваемых фильтром событий filter_mode. Дополнительно filter_mode включает режимы перекодировки кодов по таблицам или возврат аппаратного значения, режим вывода или гашения курсора, флаг позиционирования курсора в заданную точку, режимы "резинки" и др. Возвращаемое значение содержит признак события вместе с необязательным кодом клавиши, кнопки или

исключительного состояния. По переданным ука-зателям координат заносятся координаты "мыши" в момент наступления события.

```
int Hgin( int filter_mode, int * x, int * y )
```

Ввод текста с клавиатуры использует функцию Hgin() для опроса только событий нажатия клавиш. Коды заносятся во внутренний буфер, адрес которого возвращается. Дополнительные режимы задают эхо-печать, примитивное редактирование, ввод ограниченного количества символов или завершение ввода клавишей "ВВОД" ("Enter"). В режиме редактирования и эхо-печати в возвращаемый буфер и на экран заносится содержимое строки source.

```
char * Hkeyboard( int mode, int amount, char * source )
```

В среднем на разработку версии интерфейса для каждой новой платформы было затрачено от 5 до 20 чел.-дней. Правильность принятого подхода подтверждается адекватным функционированием в различных вычислительных средах всех программ, разработанных на базе H-интерфейса. Во всех случаях переноса ПО на другую ЭВМ никакой переделки прикладных программ в части графики и интерактивного интерфейса не потребовалось.

2.4.5.Особенности реализации интерфейса для среды Microsoft Windows (NT).

Среда Microsoft Windows начинает приобретать все большую популярность среди пользователей. Этому способствует рост производительности и дешевизна компьютеров IBM PC, улучшение качества цветных дисплеев и появление все большего числа приложений для Windows. Пользовательский интерфейс среды удовлетворяет рекомендациям организации OSF [55]. Среда Windows обеспечивает режим работы

приложений, близкий к многозадачному режиму ОС UNIX. Один из сильных аргументов в ее пользу - поддержка большинством производи-телей ЭВМ новой версии под названием WindowsNT, выполняющей роль мощной графической операционной системы и предназначенной не только для машин типа IBM PC, но и для ЭВМ других фирм на базе RISC-процессоров. Поэтому на национальном рынке среду Windows и WindowsNT следует рассматривать как мощную альтернативу UNIX-ориентированным системам типа X Window System и OPEN LOOK.

Программирование для Windows требует полного отказа от традиционной концепции программирования с явной передачей управления от одной подпрограммы (функции) к другой. Согласно рекомендациям по разработке приложений [45-47], вводится новый порядок взаимодействия прикладной программы и вычислительной среды, суть которого - прикладная программа строится как обработчик событий в окне (рис.2-9). Точкой входа с точки зрения программиста служит функ-ция WinMain(), которая выполняет две главные задачи:

1) Регистрирует класс окна функцией RegisterClass(), создает и визуализирует окно функциями CreateWindow() и ShowWindow();

2) Организует главный цикл обработки сообщений окна путем запроса сообщения функцией GetMessage() с последующей передачей его диспетчеру Windows функцией DispatchMessage().

Прекращение работы приложения происходит по получении специального сообщения, выданного либо самим приложением, либо средой Windows при закрытии окна. При этом функция WinMain() окончательно возвращает управление в среду Windows и та завершает приложение.

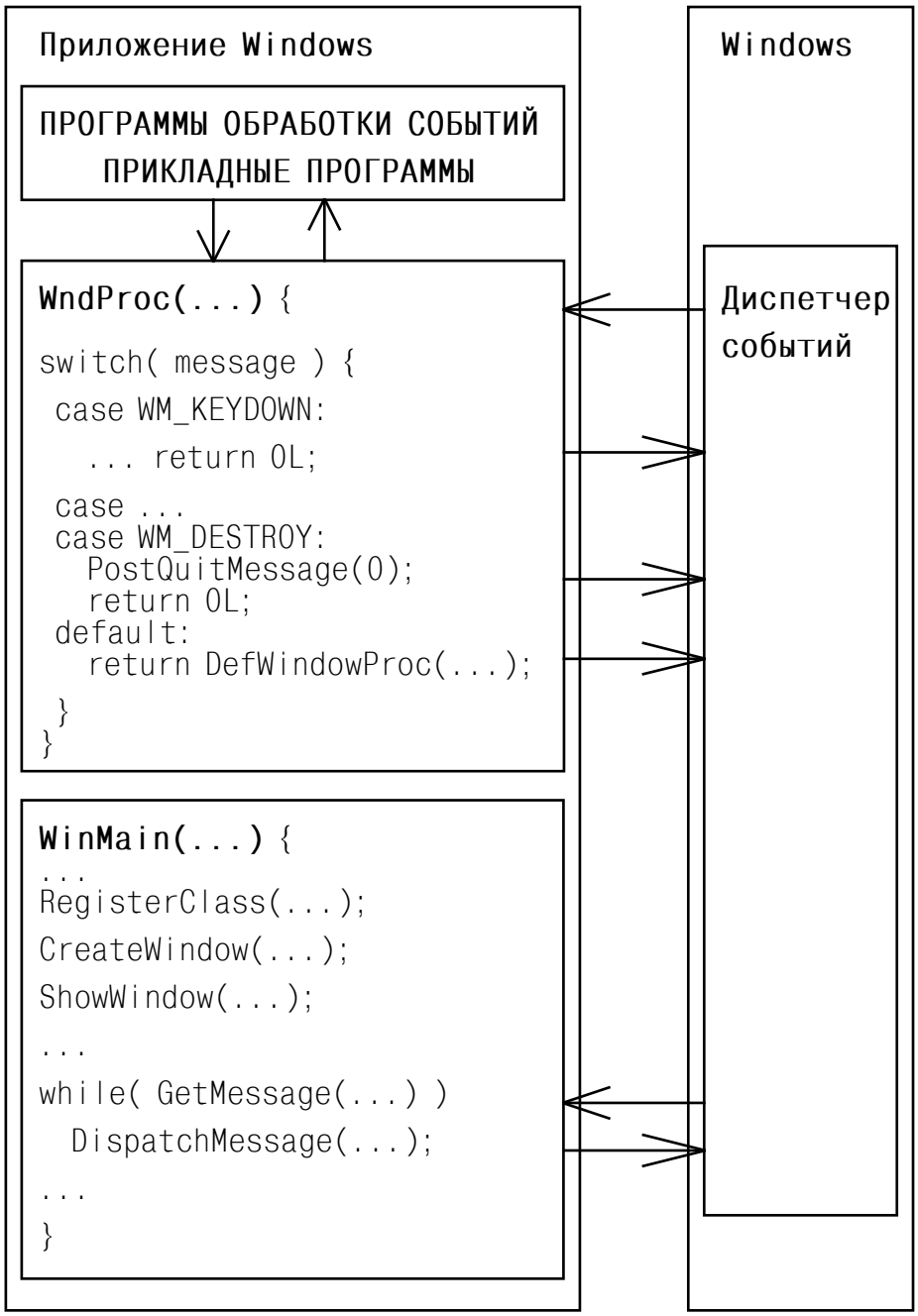


Рис. 2-9

Структура стандартного приложения Windows

Выполнение прикладных алгоритмов инициируется в функции WndProc(), получающей сообщения от диспетчера Windows. В этой функции после анализа типа сообщения и его параметров возможно организовать выполнение прикладных алгоритмов. Такая схема прог-раммы чрезвычайно удобна для разработки интерактивных приложений, но, к сожалению, не имеет эквивалентных возможностей в других графических платформах, а поэтому непереносима. Следует отметить, что при программировании

задач с интенсивными вычислениями необхо-димы "паузы" для запуска обработки сообщений, ибо в противном случае работа с другими приложениями будет заблокирована до окон-чания вычислений.

Подводя итог, можно заключить, что именно нестандартное взаи-модействие прикладных алгоритмов со средой Windows служит главным препятствием для переноса существующих программ на эту платформу, требуя по сути разработки новых программ. В данной работе далее показано, что эта проблема может быть успешно решена.

Автором разработана модифицированная схема передачи управления в приложении Windows (рис.2-10), с помощью которой удалось сохра-нить обычный порядок передачи управления в ранее разработанной или разрабатываемой прикладной программе, как это показано на блок-схеме в предыдущем параграфе. С точки зрения программиста управле-ние по-прежнему передается от вычислительной среды в функцию main() (отсутствует в приложении Windows!), где после инициализа-ции H-интерфейса можно выполнять рисование, запрос событий и дру-гие действия согласно программируемым алгоритмам. Что чрезвычайно важно, такая программа будет одинаково успешно функционировать и в среде Windows, и в среде MS-DOS, и в среде UNIX. При этом H-интерфейс для реализации своих функций пользуется всеми возмож-ностями среды Windows, что выгодно отличает его от аналогичных пакетов, в частности пакета QuickWin [48] самой фирмы Microsoft или пакета WinC фирмы Zortech [35].

В основе модифицированной схемы передачи управления (рис.2-10) лежит разбиение и перенос частей стандартных функций WinMain() и WndProc() по функциям H-интерфейса. Корректность сделанных измене-ний подтверждена практикой.

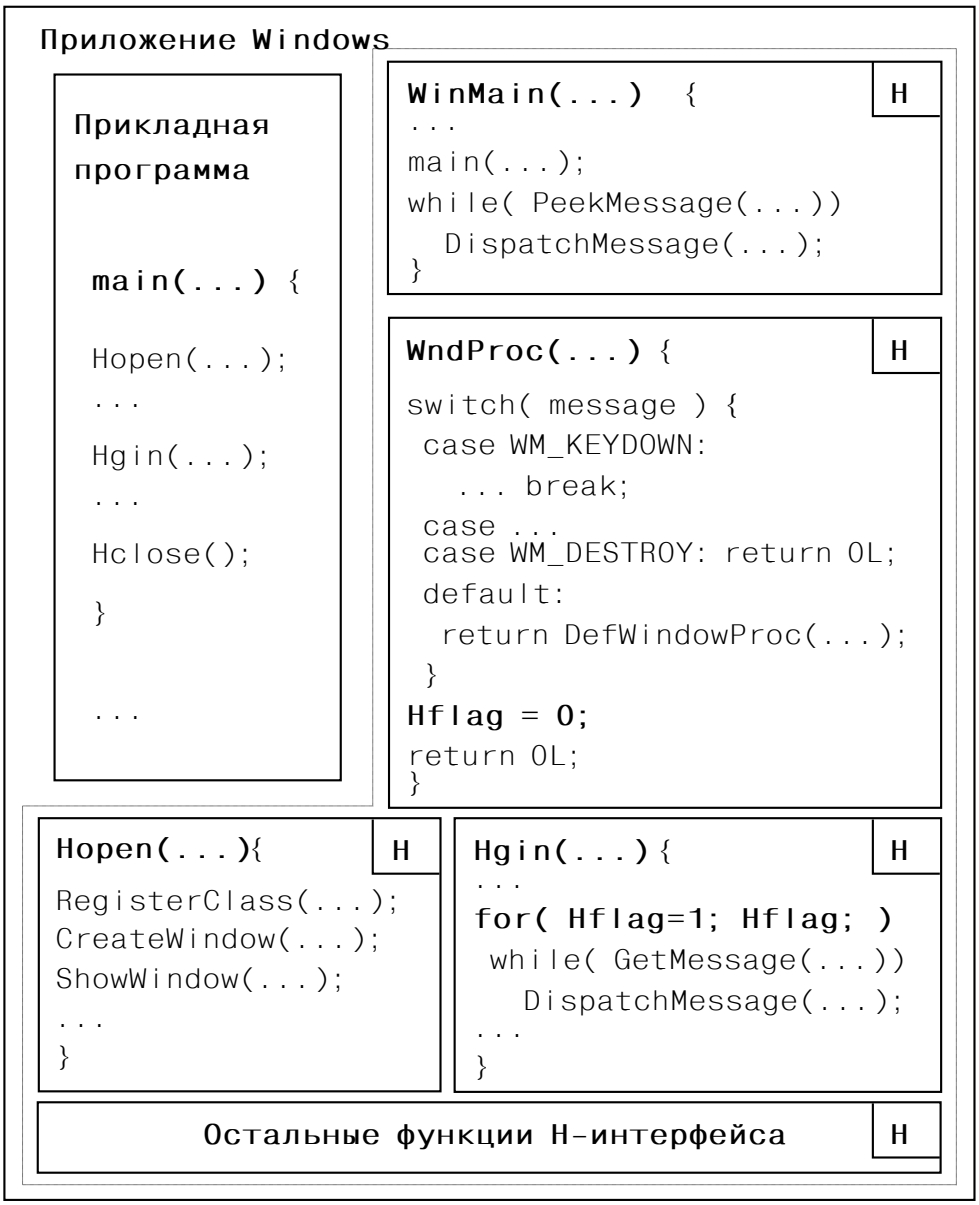


Рис. 2-10

Структура модифицированного приложения Windows - программы на базе H-интерфейса

Точкой входа в программу по-прежнему является функция WinMain(), но теперь она входит в состав H-интерфейса и, как и функция WndProc(), скрыта от пользователя и не требует програм-мирования разработчиком прикладной программы. Измененное назначе-ние функции WinMain() составляет подготовка параметров и вызов настоящей функции main(), имя которой модифицируется файлом hlib.h во избежание конфликта имен при формировании исполнительного моду-ля редактором связей. Окончание работы main() означает естествен-ный конец работы

приложения, поэтому после формального извлечения оставшихся событий окна больше никаких действий не производится и управление окончательно передается в среду Windows.

Выполнение функции main() происходит обычным порядком. После передачи управления в функцию Hopen() выполняются отложенные действия по регистрации класса окна, созданию и визуализации окна, равно как и обычные действия H-интерфейса по настройке своей среды. Таким образом успешно выполняется первая половина стандар-тной функции WinMain().

Дальнейшие передачи управления внутри прикладной программы имеют одну особенность с точки зрения Windows - приложение не обрабатывает события принятым порядком. Эта часть программы пере-несена в функцию Hgin(). Заметим, что в существующих реализациях интерфейса это единственная функция по обработке событий - все остальные используют Hgin(). Исследования Windows показали, что совсем отказаться от предложенного стандартного цикла обработки сообщений нежелательно, поэтому в H-интерфейсе используется трех-этапная обработка события, управляемая флагом. Сначала выполняется обычный цикл запроса и диспетчирования сообщений функциями GetMessage() и DispatchMessage(). После сброса флага в нулевое значение, что говорит об обнаружении приемлемого события, цикл прерывается. Во время выполнения цикла Windows вызывает стандарт-ную функцию обработки сообщений окна WndProc(), где происходит предварительная фильтрация, обработка события для H-интерфейса и сброс флага. Окончательная обработка события выполняется в функции Hgin(), после чего прикладная программа получает событие, приве-денное к спецификации H-интерфейса.

Отметим важность полученного результата:

1. Программы переносятся в Windows без изменения исходного текста.

2. Программа для среды Windows является обычным приложением, что позволяет использовать все ее возможности, в особенности под-держку новых графических устройств.

3. Программа сохраняет свой привычный внешний вид и способы управления неизменными.

4. Благодаря тому, что сложные алгоритмы взаимодействия при-ложения со средой Windows скрыты от прикладного программиста, раз-работка программ ведется в привычном стиле, гарантирующем перено-симость программы в другие среды.

Успех реализации интерфейса для Windows в очередной раз подт-вердил жизнеспособность концепции адаптивного графического интер-фейса.

2.5. Простая мультиоконная среда для прикладных программ.

Программирование диалога с пользователем для различных задач имеет много общего. Поэтому многие пакеты графических программ содержат дополнительный набор функций для построения диалоговых процедур. Современный интерфейс учитывает особенности восприятия информации человеком, накопленные методы удобной и неутомительной работы с информационными системами [29,55-62]. В основе организа-ции диалога лежит модель мультиоконной среды, управляемой потоком событий. Информация, выводимая в разграниченные области экрана, позволяет легче ориентироваться во время работы, а гибкое управ-ление конфигурацией окон способствует организации комфортной обстановки на "рабочем столе".

Мультиоконная среда как инструментальное средство программиста служит для быстрой разработки эффективных пользовательских интер-фейсов. Несмотря на то, что существует ряд аналогичных программных продуктов (OSF/Motif, OPEN LOOK, Microsoft Windows и др.), которые похожи на 70-80% [58], главное препятствие для использования этих оболочек для построения переносимых адаптивных программных средств

заключается в том, что они базируются на графических пакетах, имеющих ограниченное распространение. Motif является раширением библиотек Xlib и XToolKit системы X Window System, функционирующей только в многозадачной ОС. Программы для OPEN LOOK также базиру-ются на XWS, но для их выполнения под управлением ОС, отличной от Sun/os, необходимы дополнительные условия. Среда Windows ограни-чена машинами класса IBM PC/AT, WindowsNT расширяет ее для ряда современных ЭВМ, в частности, RISC-компьютеров. Это ограничение относится и к общеизвестному стандарту фирмы IBM, под индексом SAA (System Application Architecture - архитектура прикладных систем) [57], не говоря уже об менее известных платформах.

Второе препятствие для разработки единой графической среды прикладной программы на сегодня заключается в противоречии между сложностью мультиоконного интерфейса и его функциональностью. Сов-ременнные требования к диалоговой прикладной программе достаточно строги, т.к. подразумевают разработку интерфейса, удобного для неподготовленного пользователя и профессионала. При конструирова-нии программы приходится использовать самые разнообразные инстру-ментальные подпрограммы и функции. Их количество (~400 для Xlib/Motif и ~600 для Windows!), а также взаимосвязи с прикладной программой и между собой приводят к тому, что размер программы становится существенным для вычислительных машин с ограниченными ресурсами, что в первую очередь относится к персональным ЭВМ под управлением MS-DOS. Это также приводит к тому, что ни один из перечисленных пакетов программ не может быть выбран как основа для построения диалога с пользователем, если исходить из условия пере-носимости ПО на ЭВМ, используемые отечественными предприятиями и НИИ.

Разработанная мультиоконная среда была спроектирована и реали-зована, исходя из требования компактности и достаточной функцио-нальности. Основная цель разработки состояла в экспериментальном подтверждении предположения о том, что вполне возможно построить

мультиоконную оболочку небольшого размера, приемлемого для MS-DOS. С другой стороны, следовало очертить минимальные функциональные возможности пакета, обеспечивающие разработку интеллектуальных программ САПР электроники. Как основное свойство такой мульти-оконной среды закладывалось использование переносимого Н-интер-фейса для выполнения интерактивных и графических операций. В ка-честве примера программы - полигона для испытаний - был выбран графический редактор топологии "Layout Windows" [63]. В силу но-визны проводимых работ разработанный пакет носил экспериментальный характер и был в достаточной мере ориентирован на поддержку гра-фической базы данных (БД), используемой в редакторе.

На основе анализа существующих пакетов для разработки мульти-оконных программ, а также после анализа программ-аналогов редак-тора [40,55-62] и технического задания на разработку были выделены основные функции, выполняемые оконным интерфейсом:

1) мультиоконное отображение данных, меню, контрольной и управляющей информации прикладной программы;

2) создание, изменение, сохранение и автоматическая загрузка конфигурации окон;

3) обработка событий (нажатие клавиш, движение "мыши" и т.п.).

В качестве методологии построения мультиоконной среды был использован объектно-ориентированный подход - самая прогрессивная современная технология программирования [64]. В качестве основного объекта выступает окно - прямоугольная очерченная область экрана. Для выполнения функций программы используется базовый набор, включающий простое графическое окно и следующие текстовые окна: меню, консоль (окно диалога), окно информации о точке съема и окно менеджера окон. Дополнительно для поддержки графической БД были разработаны: окно отображения топологии базы данных и окно управления таблицей слоев. Небольшое количество классов окон сделали реализацию среды

достаточно простой, а исключительное использование H-интерфейса гарантирует переносимость на различные вычислительные системы.

Еще одной характерной особенностью мультиоконной среды явля-ется отказ от использования многоугольной границы обрезки в случае наложения окон. Такое решение является спорным с современной точки зрения, однако в его пользу говорят следующие аргументы:

1) Упрощение алгоритма обрезки при рисовании в окнах, т.к. всегда предполагается, что границы обрезки совпадают с границами окна. При учете наложения окон необходимо определить контур(ы) обрезки, которые не обязательно будут прямоугольниками. Из литера-туры [50-54] известно, насколько сложнее алгоритмы обрезки многоугольной границей алгоритмов обрезки прямоугольной областью.

2) Уменьшение количества функций менеджера окон упрощает его реализацию и уменьшает размер модуля.

3) Данное решение локализовано внутри мультиоконной среды и в любой момент может быть изменено.

Представленный выше набор классов окон предполагает наличие соответствующих структур данных для их описания. В отличие от сред типа OSF/Motif и OPEN LOOK в разработанном программном обеспечении все структуры данных открыты для прикладного программиста. Это, по мнению автора, способствует лучшему пониманию деталей функциони-рования программы. Основной тип данных описывает абстрактное окно, не вдаваясь в детали прикладной области и состоит из компонент, общих для всех классов окон: класс и статус (состояние) окна, координаты углов рамки окна, основные цвета изображения и т.д.

Гибкость описания заключается в том, что в дескрипторе окна задаются адреса функций для рисования изображения и отработки со-бытий. Конкретные функции подставляются в структуру при создании окна, когда известна принадлежность окна к определенному классу. В процессе функционирования мультиоконной среды прикладные программы могут изменять эти функции (устанавливать адреса на другие функции),

что изменит метод рисования и/или способ обработки событий. При функционировании программы менеджер окон использует эти ссылки на функции для их вызова, абсолютно не "догадываясь" о действительном характере выполняемых действий. Этот метод позво-ляет отделить наиболее общие операции управления окнами от при-кладной программы.

Соответственно разделяется и вся структура программы. Если при традиционном подходе порядок выполнения операторов языка всегда можно было проследить по тексту программы (рис.2-11), то использу-емая концепция не позволяет этого сделать. Это происходит потому, что вызов функций обработки событий и функций рисования выполня-ется в зависимости от действий пользователя программы, порядок которых неизвестен априори. Кроме того, вызов функций происходит не явно по имени, а по ссылке (указателю), как это показано на рис.2-12.

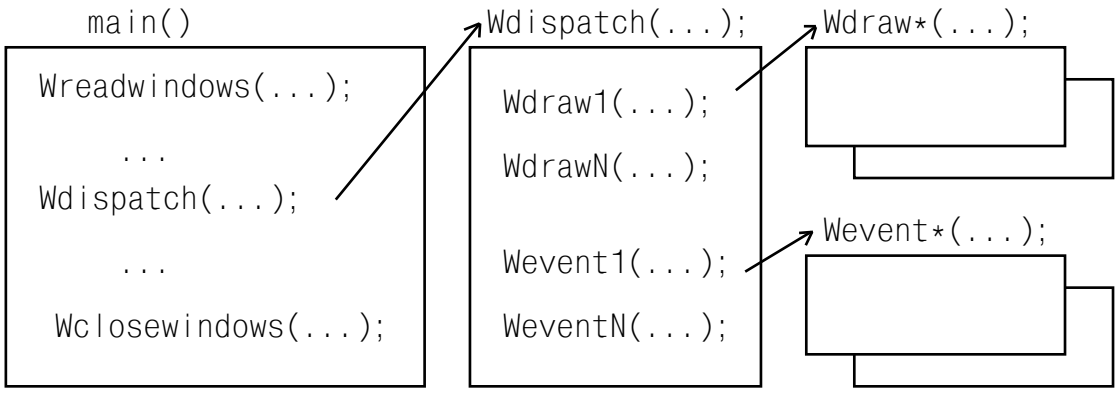


Рис. 2-11

Явная передача управления в диалоговых программах

Предложенный механизм накладывает ряд требований на функции рисования и обработки событий. Во-первых, оговаривается порядок и типы аргументов, что гарантирует успешность их вызова менеджером окон. Но самое главное заключается в том, что при выполнении операций, требующих продолжительного взаимодействия с оператором (например, при рисовании ломаной линии), программа обработки событий должна отслеживать фазу выполнения операции. Соответствен-но появляются статические данные, хранящие информацию об этом. Как

показала практика разработки и реализации алгоритмов, отслеживание состояния функции удобно лишь при коротких операциях, состоящих из одного-трех шагов. Большее количество фаз алгоритма усложняет программу, затрудняя ее отладку и понимание функционирования.

Поэтому для многошаговых операций лучше использовать метод монопольного "владения" устройствами коммуникации, когда управление не возвращается в программу поиска событий (Wrun() на рис.2-12) до достижения определенного состояния, число которых значительно сокращено. В качестве таких точек возврата удобно выбрать заверше-ние операции или отказ от нее. Очевидно, что такая функция должна сама выполнять элементарные операции менеджера окон, например, проверку событий на принадлежность окну.

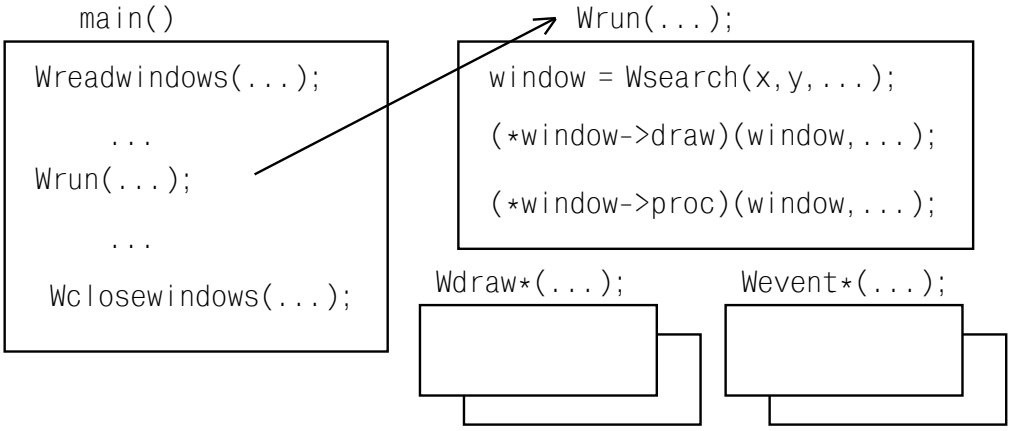


Рис. 2-12

Неявная передача управления в диалоговых программах

Как было замечено в ходе исследования различных вариантов оконных систем, большинство классов окон устанавливает взаимно-однозначное соответствие между координатной системой экрана и при-кладной системой координат. Это преобразование может использо-ваться менеджером окон для пересчета координат положения курсора, габаритов окна и других аналогичных вычислений и поэтому позволяет выполнять независимое от типа окна масштабирование, позициониро-вание, вращение и другие операции путем изменения коэффициентов

трансформации с последующим вызовом функции рисования. Что также важно, функции обработки событий могут использовать готовые пересчитанные координаты, в результате чего исключаются одинаковые фрагменты кода для многих программ.

Расширение набора окон, или введение нового класса окна, не требует объявления новых типов данных для окна. Используются те же структуры данных. Необходимо только разработать алгоритмы и написать код для двух функций:

1) Прорисовка изображения окна, использующая в качестве дополнительных аргументов трансформацию координат и габариты окна в экранной и прикладной системе координат. Функция должна удовлетворять зафиксированному порядку передачи параметров, чтобы имелась возможность ее запуска как из обычных подпрограмм, так и менед-жером окон в произвольные моменты времени.

2) Обработка событий, как результат действий пользователя или менеджера окон. Как было показано, возможны два сценария взаимодействия: реакция на произвольное событие и отслеживание внутреннего состояния или захват очереди событий для монопольной обработки цепочки предполагаемых событий.

Одним из важных направлений иследования и экспериментальной проработки было установление возможности автоматической конфигурации окон, изолированной от скомпилированной программы в дисковом файле конфигурации. Все рассмотренные пакеты программ [45-47, 55,56] не акцентируют разработчиков приложений на использование гибкой оконной среды. Напротив, специальные программы автоматиза-ции формирования имиджа программы генерируют фиксированный код, соответствующий фиксированному положению экранных форм. В результате пользователь программы, особенно массового применения, оказывается в плену решений разработчика программы, не имея возможности поменять что-либо.

В разработанном пакете поддержки мультиоконного пользовательского интерфейса эта задача была успешно решена. В противополож-ность большинству программ на основе Motif прикладная программа не занимается детальным созданием и расположением окон, а всего лишь передает специальной функции спецификацию файла с конфигурацией окон. Эта конфигурация может произвольным образом изменяться каждым пользователем программы. Как результат, теряется фиксиро-ванное "лицо" программы, но зато появляется возможность удовлетво-рения пожеланий каждого пользователя в отдельности!

Разработанный пакет программ, несмотря на положительные решения, не лишен и недостатков. К ним следует отнести простой внешний вид большинства окон и нестандартное обращение к менеджеру окон. Следует признать важность оформления и способа управления окнами, рекомендованного OSF [55], и принципиально одинакового для альтернативных мультиоконных систем. Сказанное лишь подчеркивает необходимость дальнейших исследований и экспериментальных разработок.

2.6. Методика использования интерфейса в прикладных программах.

2.6.1. Структура прикладной программы на базе интерфейса.

Блок-схема обобщенной прикладной программы на базе интерактивно-графического интерфейса приведена на рис.2-13. Отметим, что большинство ПГП (кроме Microsoft Windows) предлагают похожую схему для написания графических программ, что облегчает программирование для специалистов, ранее пользовавшихся другими пакетами. Небольшое количество функций и раздельное управление ресурсами способствуют легкому пониманию процедур интерфейса. Пример программы на базе H-интерфейса для демонстрации возможностей машинной графики приведен в приложении.

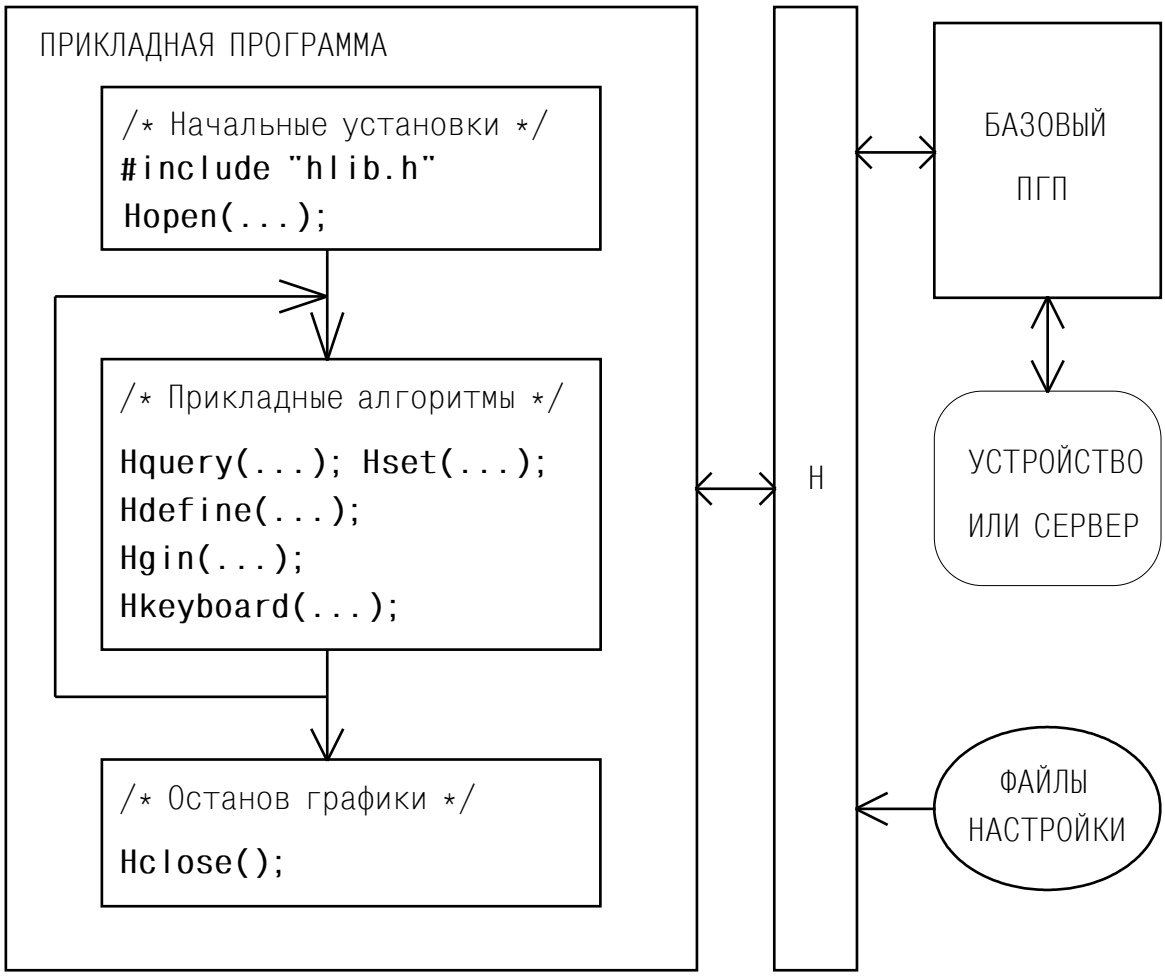


Рис. 2-13

Структура прикладной программы на базе H-интерфейса

Рассмотрим основные моменты программирования с использованием H-интерфейса. Обязательные элементы программы изображены на рис.2-13 и включают:

1) директиву вставки файла с определением констант и прототи-пов функций в каждом файле, использующем интерфейс:

#include "hlib.h"

2) функцию Hopen(), которая должна быть вызвана перед первой функцией, использующей графику или выполняющей интерактивные опе-рации.

3) функцию Hclose(), после вызова которой запрещается вызов всех функций интерфейса, кроме Hopen().

Между вызовами функций Hopen() и Hclose() допускается произвольное обращение к остальным функциям интерфейса, что изображено на рис.2-13 в виде цикла. Отметим ряд важных моментов:

1. В связи с тем, что априори не всегда известен точный размер экрана (окна в мультиоконных средах типа Windows), рекомендуется запросить точные границы вызовом функции Hquery() с такими параметрами:

Hquery(Hviewport ¦ Hmax, &left, &bottom, &right, &top);

Первый комбинированный параметр определяет тип запроса - максимальные границы окна, а четыре последующих передают указатели на переменные, в которые функция занесет координаты левой, нижней, правой и верхней границ экрана (окна). Этот запрос также следует выполнять всякий раз после получения события "изменение границ окна", чтобы отследить действия пользователя в мультиоконной среде.

2. Перед выполнением функций рисования следует установить необходимые параметры графического контекста, если текущие значения отличны от требуемых. Для каждого ресурса выполняется отдельный вызов функции, порядок вызовов несущественен. Пример ниже демонст-рирует установку сплошной линии цвета 5:

Hset(Hstyle, Hsolid); Hset(Hcolor, 5);

3. Если набор возможных параметров рисования меняется достаточно часто, рекомендуется использовать различные контексты. Тогда вместо цепочки функций Hset() для каждого ресурса выполняется однократная установка контекста. В примере ниже устанавливается контекст номер 3:

Hset(Hcontext, 3);

4. Для ускорения рисования следует указывать значение парамет-ра обрезки "Hnoclip", если есть возможность быстро определить, что фигура полностью входит в прямоугольную область рисования. В противном случае интерфейс выполняет проверку и обрезку фигуры, а эти операции требуют значительного времени.

2.6.2. Методика обеспечения лингвистической инвариантности прикладных программ.

Разработка программного обеспечения на заказ в ряде случаев требует использования языка, на котором пользователь общается с программой, отличающегося от языка операционной системы или не поддерживаемого ей. В первую очередь это относится к отечественным САПР на базе импортной вычислительной техники. В рамках рассматриваемого H-интерфейса это достаточно легкая задача, т.к. концепция интерфейса не привязана к какому-либо одному языку. Рассмотрим этапы решения.

1. В первую очередь следует отделить все используемые програм-мой сообщения, подсказки, меню и прочие тексты от исходного кода самой программы и поместить их в отдельные файлы. Для вывода текс-тов на экран дисплея следует сначала вводить их из файла во вре-менные или постоянные буфера, а всем функциям передавать указатели на буфера взамен передачи текстовых констант. Этот метод требует дополнительного комментирования программы, т.к. убранные текстовые константы могли нести некоторую смысловую нагрузку.

2. Следует сформировать кодовую таблицу. Для основы рекомен-дуется брать стандартную 7-битную таблицу ASCII, в которой варьи-ровать коды 128-255. Этой таблицей устанавливается соответствие изображения символов их коду. В качестве русской таблицы в насто-ящее время используется модифицированная альтернативная кодировка, но при необходимости может использоваться другая кодировка. На основе модифицированной таблицы выполняются все дальнейшие дейст-вия.

3. Используя таблицу кодов, следует создать или откорректи-ровать файл(ы) с описанием битовых карт шрифтов таким образом, чтобы каждому коду из модифицированного диапазона соответствовало свое начертание символа. Для автоматизации процесса корректировки автором использовалась специальная программа (также разработанная на базе H-интерфейса).

4. На основе таблицы кодов выполняется создание или корректировка таблицы перекодировки кодов клавиш, в которую вносятся коды символов национального языка согласно раскладке клавиш на клавиатуре.

5. Используя таблицу кодов, формируются файлы с текстами, переведенными на требуемый язык.

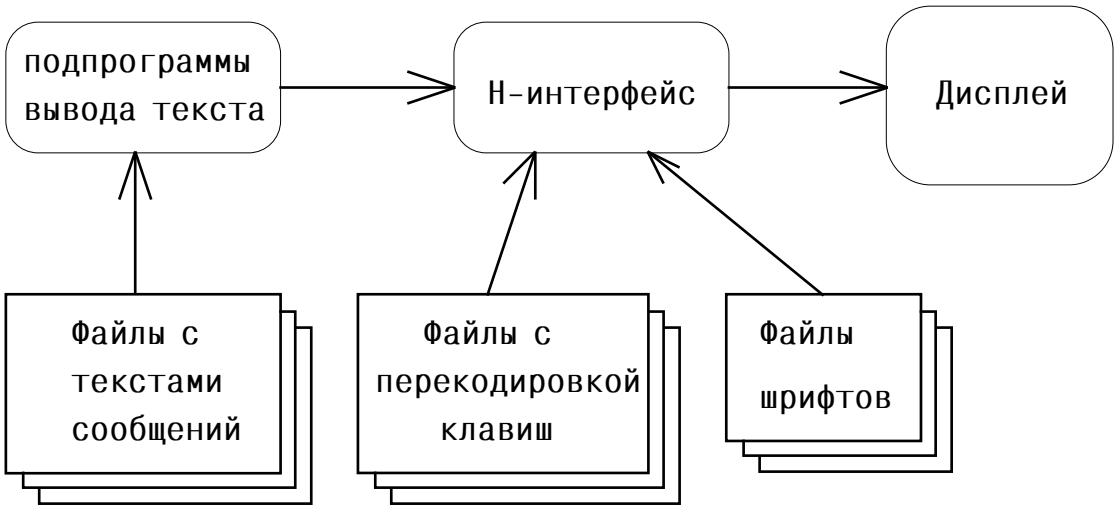


Рис. 2-14

Потоки текстовых данных и текстовых ресурсов

В результате последовательного выполнения указанных выше шагов будет сформирован набор файлов для прикладной программы. Замена этих файлов автоматически меняет язык общения пользователя с программой. Заметим, что файлы с кодами клавиш и шрифтами могут использоваться другими программами. На рис.2-14 иллюстрируются потоки данных, а на рис.2-15 кодовые зависимости. В связи с тем, что все файлы отделены от скомпилированного кода программ, их мо-дификация доступна конечному пользователю программы, что особенно важно для создания комфортной среды.

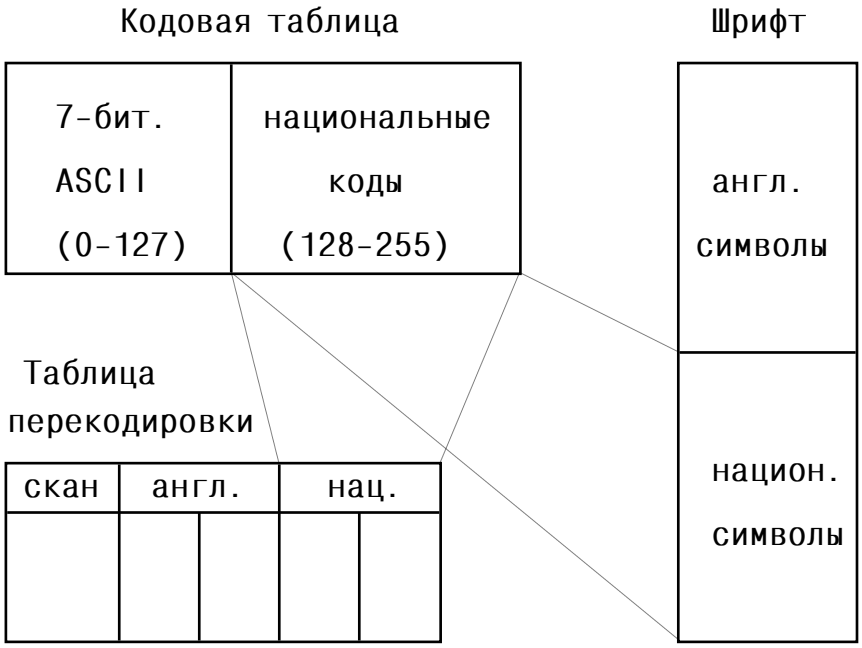


Рис. 2-15

Схема формирования поддержки национального языка

Для переключения современных клавиатур интерфейсом анализируется нажатие клавиш типа "Alt", "Compose" и аналогичных. Для старых и специфических клавиатур используется комбинация нажатий редко используемых клавиш, как это реализовано для рабочих станций Apollo 35xx. В связи с тем, что индикация национального режима затруднена, используется способ звукового подтверждения фиксации национальной или стандартной клавиатуры. Временное переключение (без фиксации) выполняется при нажатой, но не отпущенной клавише типа "Alt", а фиксация - при одновременном их нажатии.

2.7.Выводы.

В условиях быстро меняющихся графических стандартов необходима разработка локального устойчивого интерактивно-графического интерфейса. На основе анализа использования графических функций в ЗакСАПР электроники с помощью метода последовательного конструиро-вания сформирован функциональный состав адаптивного графического интерфейса. Разработана спецификация и внутренние структуры дан-ных, в основе которых лежит двойная буферизация ресурсов и единый формат

для их хранения. Показано, что таким путем можно добиться минимизации числа системно-зависимых функций и достичь высокой мобильности ЗакСАПР.

Для подтверждения концепции выполнены реализации интерфейса для всех открытых графических платформ - промышленных стандартов: X Window System, Microsoft Windows(NT), Apollo и MS-DOS-платформ (компиляторы Microsoft, Borland, Zortech). Практически доказано, что метод адаптивного открытого графического интерфейса позволяет минимизировать время и затраты на разработку ЗакСАПР. Показано, что на базе интерфейса можно строить системно-независимые пользовательские мультиоконные среды. Разработана методика обеспечения языковой независимости программ.

ГЛАВА 3

УПРАВЛЕНИЕ ДИНАМИЧЕСКОЙ ПАМЯТЬЮ

3.1. Методы управления динамической памятью в системах программирования. Постановка задачи.

Стандартная схема управления динамической памятью (ДП) в языке программирования Си состоит из вызова пары функций. Первая начи-нает, а вторая завершает жизненый цикл выделенной для процесса (программы) памяти:

1) функция malloc() возвращает указатель на выделенную память запрошенного размера. Тип указателя стандартный для языка С. После вызова функции можно оперировать с выделенной памятью обычным для языка С способом.

2) функция free() служит для сообщения операционной системе о том, что данная память больше не будет использована в программе и передается ОС. Возвращенный ранее функцией malloc() указатель становится недействительным после вызова free().

В отличие от ОС UNIX, в которой используется приведенная выше схема, в операционной системе MS-DOS для доступа к памяти выше 640 кбайт используются разнообразные средства. В частности, фирмой Microsoft предлагаются драйверы (программы, постоянно находящиеся в памяти и обеспечивающие протокол управления) himem.sys и emm386.sys [65]. Соответственно, транслятор языка Си той же фирмы содержит пакет программ для работы с этой памятью, реализующий процедуры управления сегментами по более сложной схеме [48]. Стандартные функции malloc()

и free() обеспечивают обычный меха-низм выделения и освобождения памяти, но в пределах 640 кбайт.

Среда MSW содержит свой собственный механизм управления стра-ничной памятью [45,47], что исключает использование стандартных функций malloc() и free(). Предлагается пользоваться исключительно средствами MSW, реализующими страничный (сегментный) механизм.

Подводя итог, можно сказать, что для разработки сложных пере-носимых программ приходится использовать нестандартные системы уп-равления памятью более низкого уровня, чем схема для ОС UNIX, а это требует дополнительных усилий по программированию. Для увязки этих (и других аналогичных) систем управления вполне возможно осуществить эмуляцию стандартных функций malloc() и free(), но это может приводить к нежелательным эффектам, из-за которых разработ-чики нестандартных средств предлагают свои уникальные протоколы и наборы функций. Поэтому более предпочтителен вариант разработки системно-независимого набора средств управления памятью. Для ус-пешного решения проблемы необходимо решить следующие задачи:

1. Сформировать обобщенную модель управления ДП. Внутреннее единство и непротиворечивость модели гарантируют стабильность про-граммного интерфейса.

2. Разработать протокол управления памятью на основе модели и соответствующий набор программных средств.

3. Разработать методику использования для различных вычисли-тельных сред.

3.2. Обобщенная модель управления динамической памятью.

Обобщенную модель удобно сформировать, воспользовавшись тео-рией множеств. Абстрагируясь от конкретных деталей построения памяти в вычислительной системе, всю доступную память можно пред-ставить в виде конечного упорядоченного множества элементов, каж-дый из которых соответствует минимально адресуемому элементу памя-ти. В

подавляющем числе современных ЭВМ и ОС такой единицей служит байт. Запишем определение ДП в виде:

$$M = \{m_k, k = 0,..,N-1\} \qquad (3\text{-}1)$$

где N - количество адресуемых элементов (байт) ДП,

m_k - минимально адресуемый элемент памяти.

Заметим, что каждому элементу m_k ставится во взаимно-однозначное соответствие адрес (индекс, указатель и т.п.) этого элемента. Поэтому определение памяти может быть представлено в виде, эквивалентном (3-1):

$$M = \{s_k, k = 0,..N-1\} \qquad (3\text{-}2)$$

где s_k - адрес минимально адресуемого элемента памяти m_k.

Динамический характер использования памяти предполагает, что одна часть ее будет задействована в программе, а другая останется свободной для дальнейшего использования. Поэтому множество M всей памяти можно представить в виде двух непересекающихся подмножеств:

$$M = \{A, F\}: \; A \cap F = \varnothing \; и \; A \cup F = M \qquad (3\text{-}3)$$

где A - занятая память, F - свободная память.

Граница между множествами может изменяться достаточно произвольно. В начальный момент времени вся память свободна, т.е. F=M и A=0. Вполне допустимы ситуации полного использования памяти, когда F=0 и A=M.

Дальнейшее усложнение модели связано с тем, что память запрашивается и выделяется не только отдельными элементами, а и последовательно расположенными блоками. Блоки, как и вся память, явля-ются упорядоченными множествами. В терминах теории множеств опре-деление блока можно записать следующим образом:

$$B(i,j) \subset M, \text{ если } \exists\ i \le j,\ i \in \{0,1,..,N-1\},$$
$$j \in \{0,1,..,N-1\} : \forall k \in \{i, i+1,\ldots, j\} \Rightarrow \quad (3\text{-}4)$$
$$\Rightarrow s_k \in B$$

где B - блок памяти.

Как и отдельные элементы ДП, блоки могут быть свободными или занятыми. Для обозначения типа блока будем использовать верхний индекс "a" для занятых блоков и "f" - для свободных. Так как минимальный элемент памяти тоже может быть представлен как блок, определение памяти может быть записано в виде:

$$M = \{\ \{B_i^a\},\{B_j^f\}\ \},\ B^a \cup B^f = M,\ B^a \cap B^f = \varnothing, \quad (3\text{-}5)$$

причем sup(i)=sup(j)=N.

Определение блока (3-4) использует глобальные индексы i и j в качестве границ блока. Учитывая однозначное соответствие индекса адресу элемента ДП, в качестве чисел, определяющих блок, можно использовать начальный адрес (элемент) блока и его длину, которую выразить через разницу индексов. Это определение удобнее (3-4) тем, что использует характеристики блока памяти, используемые в программировании:

$$B(i,j) \equiv B(s_i, z) \quad (3\text{-}6)$$

где s_i - начальный адрес, $z = j - i + 1$ - длина.

Функционирование системы управления памятью в терминах теории множеств можно охарактеризовать как выполнение операций деления и объединения подмножеств (блоков). Запишем функцию выделения памяти malloc() как операцию поиска блока необходимого размера и возвращение адреса начального элемента блока:

$$malloc(z) = \begin{cases} s_i, \text{ если } \exists B^f(s_i, z_i): z \le z_i \\ 0, \text{ если } \forall B^f(s_i, z_i) \Rightarrow z > z_i \end{cases} \quad (3\text{-}7)$$

Заметим, что при $z < z_i$ выполняется разбиение свободного блока:

$$B^f(s_i, z_i) \xrightarrow{malloc()} B^a(s_i, z) + B^f(s_{i+z}, z_i - z) \qquad (3\text{-}8)$$

Функция освобождения памяти free() заключается в переводе ранее занятого блока в подмножество свободных. При необходимости блок может объединяться со свободными соседними блоками:

$$B^a(s, z) \xrightarrow{free()} B^f(s, z)$$

$$B^f(s, z) + B^f(s_k, z_k) =$$

$$= \begin{cases} B^f(s_k, z_k + z), \ \text{если } s = s_{k+z_k} \\ B^f(s, z_k + z), \ \text{если } s = s_{k-z} \end{cases} \qquad (3\text{-}9)$$

где $B^f(s_k, z_k)$ - соседний свободный блок.

Объединение именно соседних блоков является существенной особенностью множества M, описывающего ДП. Операция объединения позволяет избежать излишней фрагментации области свободного пространства. Однако случайный характер занятия и освобождения блоков может приводить к тому, что занятые блоки будут мешать объединению свободных блоков. При суммарном размере свободного пространства, достаточном для удовлетворения запроса на выделение памяти, свободный блок требуемого размера может не существовать. В данной ситуации для увеличения размера непрерывного свободного блока необходимо передвинуть занятые блоки, но такая операция естественным образом изменит начальный адрес блока, ранее переданный в прог-рамму.

Для решения проблемы необходимо исключить зависимость описания блока памяти от адреса. Для этого передаваемый в программу адрес должен иметь характер инварианты. Из-за того, что многие системы программирования не обеспечивают именно это свойство адреса, его необходимо включить в модель. Т.е., каждой паре (s,z) или (i,j), определяющей блок, ставится в соответствие некоторая инвариантная к перемещению блока величина. Образуются тройки чисел (n,s,z) или (n,i,j), где n - инвариантный индекс. Эти тройки чисел образуют

множество "описателей" блоков памяти. При изменении начального адреса блока инвариантный индекс остается неизменным. Определение памяти (3-5) предстанет в виде:

$$M = \{ \ \{B_i^a\}, \{B_j^f\}, \{(n, s_k, z_k)\} \ \} \qquad (3\text{-}10)$$

где (n, s_k, z_k) - инвариантное описание k-го блока.

Данный метод индексирования не свободен от недостатков. Для получения или изменения адреса необходим просмотр множества индексов. Изменение адреса должно обязательным образом завершаться уведомлением программы о проведенных передвижках. Однако организация всех систем управления памятью подразумевает, что эта система выполняет пассивную роль по отношению к остальной программе. Следовательно, программе необходимо самой "заботиться" об истинности используемых адресов.

Уменьшение количества обращений для верификации адреса может быть достигнуто при разделении множества занятых блоков на два подмножества т.н. фиксированных и перемещаемых блоков. К первым отнесем блоки, перемещение которых запрещено. Для них адрес оста-ется неизменным и его верификация не требуется. Для использования данных в перемещаемых блоках эти блоки необходимо зафиксировать и установить действительный адрес. Для обозначения перемещаемых и фиксированных блоков будем использовать верхние индексы "am" и "ax" соответственно.

$$M = \{ \ \{B_i^{ax}\}, \{B_i^{am}\}, \{B_j^f\}, \{(n, s_k, z_k)\} \ \} \qquad (3\text{-}11)$$

где $B^a = B^{ax} \cup B^{am}, \ B^{ax} \cap B^{am} = \varnothing$

Запишем операции фиксирования (блокирования) и разблокирова-ния, выполняемые функциями lock() и unlock(), соответственно:

$$B^{am}(s,z) \xrightarrow{lock()} B^{ax}(s,z)$$
$$B^{am}(s,z) \xleftarrow{unlock()} B^{ax}(s,z) \qquad (3\text{-}12)$$

Введение двух последних операций позволяет использовать дополнительную память для хранения перемещаемых блоков. В этом случае операции блокирования и разблокирования комбинируются с операциями "перемещения" данных. Реализация механизма перемещения блока не обязательно включает физическое перемещение данных; могут использоваться механизмы расширения основного адресного пространства, переключения сегментов и т.д. Важной особенностью этого механизма является использование памяти большего размера, чем память для фиксированных и свободных блоков. В качестве такой памяти могут использоваться дисковые файловые системы или любая другая специальная аппаратура.

Объявление только трех типов блоков порождает шесть возможных операций по перемещению блоков, изображенных на рис.3-1. Однако такая модель требует дальнейшего уточнения.

Заметим, для MS-DOS с поддержкой виртуальной памяти вся управляемая память (в том числе и свободная) разделяется на два непересекающихся подмножества. Одно подмножество представляет собой буферную область в адресном пространстве программы, а другое - виртуальную память. Фиксированные блоки могут храниться только в буферной области, а перемещаемые - только в виртуальной.

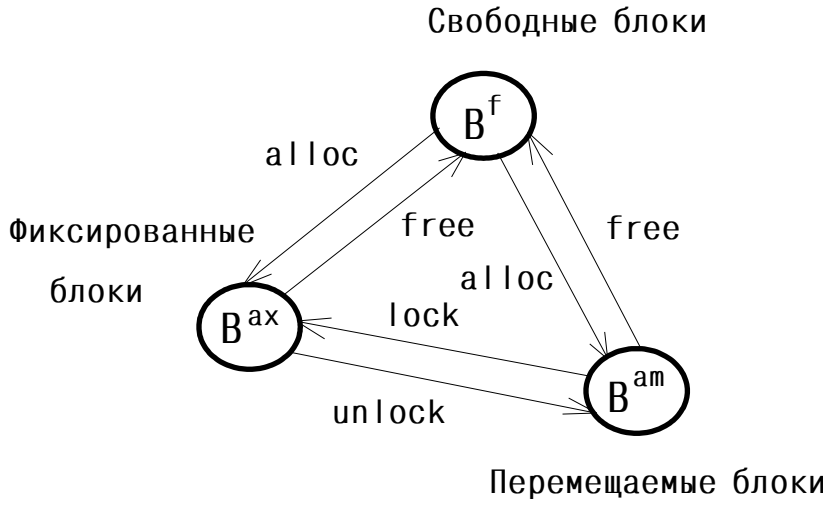


Рис. 3-1

Типы блоков памяти и операции управления

Программы, ориентированные на фиксирование только необходимой информации, могут обрабатывать большие объемы данных при минимальном расходовании дефицитной оперативной памяти вычислительной системы. Это замечание относится к ОС MS-DOS и среде Microsoft Windows, в системах UNIX и VMS использование таких алгоритмов оправдано только, если необходима высокая мобильность ПО.

Для свободных блоков запишем, применяя верхний индекс "fu" для буферной памяти и "fv" для виртуальной:

$$B^{f} = B^{fu} \cup B^{fv}, \; B^{fu} \cap B^{fv} = \varnothing \quad (3\text{-}14)$$

В данном случае для разделение памяти по принципу хранения запишем:

$$M = U \cup V, \; U \cap V = \varnothing$$
$$U = B^{ax} \cup B^{fu}, \; V = B^{am} \cup B^{fv} \quad (3\text{-}15)$$

Ранее сделанное предположение о единой свободной памяти не выполняется и поэтому в модели следует запретить "транзитные" цепочки (3-16), иначе модель будет допускать "перетекание" блоков памяти, что физически невозможно.

$$B_i^{f} \xrightarrow{alloc} B_i^{am} \xrightarrow{lock} B_i^{ax} \xrightarrow{free} B_i^{f}$$
$$B_i^{f} \xrightarrow{alloc} B_i^{ax} \xrightarrow{unlock} B_i^{am} \xrightarrow{free} B_i^{f} \quad (3\text{-}16)$$

Исключение этих цепочек означает, что все остальные выполня-емые с ДП операции носят челночный, или вложенный, характер. Выде-ляемая память в конечном итоге должна возвратиться в то же под-множество свободных блоков, из которого она была выделена. В результате справедливы только операции (3-17), которые наглядно изображены на рис. 3-2.

$$B_i^f \xrightarrow{alloc} B_i^{am} \xrightarrow{free} B_i^f$$

$$B_i^f \xrightarrow{alloc} B_i^{am} \xrightarrow{lock} B_i^{ax} \xrightarrow{unlock} B_i^{am} \xrightarrow{free} B_i^f \qquad (3\text{-}17.1)$$

$$B_i^f \xrightarrow{alloc} B_i^{ax} \xrightarrow{free} B_i^f$$

$$B_i^f \xrightarrow{alloc} B_i^{ax} \xrightarrow{unlock} B_i^{am} \xrightarrow{lock} B_i^{ax} \xrightarrow{free} B_i^f \qquad (3\text{-}17.2)$$

```
ВключитьВиртуальнуюПамять();

индекс = ВыделитьПамять( размер );

адрес = ПолучитьАдрес( индекс );
...
ОсвободитьАдрес( индекс );
...
адрес = ПолучитьАдрес( индекс );
...
ОсвободитьАдрес( индекс );

ОсвободитьПамять( индекс );

...
индекс = ВыделитьПамять( размер );

адрес = ПолучитьАдрес( индекс );
...
ОсвободитьАдрес( индекс );
...
адрес = ПолучитьАдрес( индекс );
...
ОсвободитьАдрес( индекс );

ОсвободитьПамять( индекс );

ВыключитьВиртуальнуюПамять();
```

Рис. 3-2

Трехуровневая обобщенная схема управления динамической памятью

В отличие от известных моделей, используемых для описания управления памятью в рамках аппаратных средств, в данной модели использован уровень абстракции на уровне отношений, возникающих между прикладной программой и ее системным окружением. За счет этого удается сократить число рассматриваемых аспектов и описать операции управления ДП, различные для систем программирования в ОС UNIX, MS-DOS и среде Microsoft Windows, как операции с множествами блоков памяти. Каждая из перечисленных систем реализует только часть из этих операций, остальные могут быть доопределены на основе модели.

3.3. Методы реализации системно-независимого управления памятью на основе обобщенной модели.

Все операции управления ДП свободно выполняются, если есть доступ к таблице (множеству) описателей блоков (см.(3-10)). Созда-ние отдельной таблицы требует дополнительного расхода памяти и поэтому на практике это неэффективно, а доступ к таблицам ОС не всегда возможен. Поэтому в предложенном автором наборе функций реализовано подмножество всех возможных операций. Для обоснования выбора рассмотрим аспекты практической реализации допустимых опе-раций.

Во-первых, для совместимости с ОС UNIX необходимо реализовать стандартные функции malloc() и free(). В рамках модели их особен-ность в том, что они оперируют фиксированными блоками памяти и поэтому могут быть описаны цепочками команд (3-17.2), причем вто-рая цепочка может быть использована для реализации функций в сре-дах, где отсутствуют их прямые аналоги.

Во-вторых, цепочки команд (3-17.1) присутствуют в среде MSW в качестве основных средств управления памятью, а в MS-DOS - для управления расширенной памятью (спецификации XMS и EMS). Для их реализации в ОС UNIX можно положить, что функции lock() и unlock() не выполняют никаких действий.

И, наконец, отметим одну важную деталь: функции alloc() в модели различны, т.к. результатом их вызова являются блоки разного типа. То же самое относится и к функциям free(). Но согласно ранее сделанного ограничения (3-16) освобождение блока памяти одного типа должно выполняться без перевода его в другой тип, что соот-ветствует вложенным вызовам функций.

В данной работе предлагается 2 набора функций, реализующих управление динамической памятью: управление низкого уровня для сложных программ и эмуляция функций malloc() и free() для простого управления небольшими объемами памяти. Использование низко-уровневой схемы при разработке программ в ОС UNIX или в стан-дартной (до 640 кбайт) ОС MS-DOS выглядит достаточно необычно, однако гарантирует переносимость программы. Ниже представлено описание разработанного системно-независимого интерфейса. В связи с тем, что механизм, обеспечивающий выделение и обслуживание памяти, различен для вычислительных платформ, для памяти будем использовать термин "виртуальная".

На основе обобщенной модели и анализа различных систем прог-раммирования [35,45,48] автором была сформирована трехуровневая обобщенная схема управления динамической памятью. Каждый уровень состоит из вызова пары функций, между которыми включается произ-вольное количество парных вызовов функций нижнего уровня. Первый уровень обеспечивает запуск и останов средств обслуживания вирту-альной памяти. Следующий - выделение и освобождение памяти, причем для ссылки на память используется индекс специального типа. Третий уровень служит для обеспечения непосредственного доступа к содер-жимому выделенной памяти. На рис. 3-2 в предыдущем параграфе представлена данная схема на псевдо-коде, в конце главы приведен текст примера программы на языке Си.

Второй набор функций необходим для корректной замены стандартных функций malloc() и free(). Детали реализации пред-ставлены ниже для каждой вычислительной платформы.

Точная спецификация для языка Си основного набора функций определяет один тип данных, константу и 6 новых функций:

1) VMindex - тип данных для описания индекса выделенного сегмента виртуальной памяти.

2) VMNULL - значение индекса, означающее недействительный индекс.

3) void VMopen(void) - процедура инициализации системы виртуальной памяти. Делает возможным выполнение прочих функций.

4) void VMclose(void) - процедура остановки системы виртуальной памяти.

5) VMindex VMalloc(int size) - функция-запрос выделения виртуальной памяти размера "size" байт, возвращаемое значение - индекс выделенного сегмента.

6) void VMfree(VMindex index) - процедура освобождения выделенной памяти, индекс "index" которой передается в качестве параметра.

7) char * VMlock(VMindex index) - функция получения указате-ля на выделенную память по ее индексу "index" и блокировка указанного сегмента для обеспечения корректности указателя до разблокировки.

8) void VMunlock(VMindex index) - процедура разблокировки памяти указанного индекса. Вычислительная среда обеспечивает сохранность данных, но полученный ранее функцией VMlock() указа-тель становится недействительным.

9) char * malloc(int size) - стандартная функция выделения памяти и возвращающая указатель на нее. Отличие заключается в том, что выделенная память считается заблокированной и не может быть перемещена вычислительной системой. Исключительное использование этой функции для выделения больших сегментов памяти не запреща-ется,

но может приводить к перегрузке вычислительных платформ типа Microsoft Windows, и поэтому не рекомендуется.

10) void free(char * pointer) - стандартная функция освобождения памяти. Отличие заключается в том, что сначала выполняется разблокирование памяти.

Рассмотрим реализацию функций для наиболее распространенных систем.

Для ОС типа UNIX функции VMopen(), VMclose() и VMunlock() не имеют смысла и удаляются из исходного текста средствами препроцессора языка Си. Индекс определяется как обычный указатель на выделенную память. Функции malloc() и free() - стандартные для ОС.

```
typedef char * VMindex;
#define VMNULL NULL
#define VMopen()
#define VMclose()
#define VMalloc(size) malloc(size)
#define VMfree(index) free(index)
#define VMlock(index) (index)
#define VMunlock(index)
```

Для среды MS Windows излишними являются функции инициализации и остановки системы виртуальной памяти, т.к. она является неотъемлемой частью MSW и функционирует постоянно. За исключением функций malloc() и free(), для всех функций выполняется переопре-деление имен средствами препроцессора. Выделенная память относится к классу глобальной перемещаемой, но не отбрасываемой (флаги GMEM_MOVEABLE и GMEM_NODISCARD), с целью обеспечить динамическую загрузку и выгрузку, выполняемую средой MSW.

```
typedef HANDLE VMindex;
#define VMNULL NULL
#define VMopen()
#define VMclose()
#define VMalloc(size) \
```

```
  GlobalAlloc(GMEM_MOVEABLE|GMEM_NODISCARD,(DWORD)size)
#define VMfree(index)   GlobalFree(index)
#define VMlock(index)   GlobalLock(index)
#define VMunlock(index) GlobalUnlock(index)
```

Реализация функций malloc() и free() для MSW осложняется тем, что при реализации функции free() необходимо знать не указатель на сегмент памяти, а его индекс. В качестве решения предложено увеличить выделяемую память на размер индекса, который хранить в начале сегмента. Для достаточно больших сегментов это приращение размера становится несущественным. Ввиду простоты подхода ниже приведен текст этих программ на языке Си.

```
char * malloc( int size )
  {
  VMindex segment, * index_pointer;
  char * body;

  /* выделение сегмента */
  segment = VMalloc( size + sizeof(VMindex) );
  if( segment == VMNULL ) return( NULL );

  /* получение указателя на заблокированную память */
  body = VMlock( segment );

  /* сохранение индекса в начале сегмента */
  index_pointer = (VMindex *)body;
  *index_pointer = segment;

  /* возвращение адреса на начало свободного участка */
  return( body + sizeof(VMindex) );
  }

void free( char * pointer )
  {
  VMindex segment, * index_pointer;

  /* получение значения индекса */
  index_pointer = (VMindex *)pointer;
```

```
--index_pointer;
segment = *index_pointer;

/* разблокировка и освобождение памяти */
VMunlock( segment );
VMfree( segment );
}
```

Для среды MS-DOS есть некоторая неоднозначность в реализации интерфейса. При небольшом объеме запрашиваемой динамической памя-ти, которая без всякого риска может быть выделена в пределах млад-ших 640 кбайт, либо при отсутствии дополнительной памяти совсем, использование функций работы с расширенной памятью нежелательно или просто невозможно. В качестве решения предлагается два варианта применения интерфейса:

1) Для работы, возможной в пределах 640 кбайт, использовать модифицированный вариант для ОС UNIX, в котором фактически применяются только функции malloc() и free() как реализация VMalloc() и VMfree(), соответственно. Суть модификации - в исполь-зовании функций, соответствующих модели памяти программы в MS-DOS.

2) Для остальных случаев использовать реализацию низкоуровне-вого управления в приведенном ниже варианте (пример для компиля-тора Microsoft C 7.00), в котором malloc() и free() стандартны, точнее, также определяются в соответствии с заданной моделью памяти программы. Теоретически для двух последних функций возможно использовать их реализацию для MSW, но на практике ограниченность буферной области памяти, а также дополнительные затраты памяти и потери времени привели к отказу от такого варианта. Особенности реализации: 1) используется оперативная память как типа XMS, так и типа EMS, 2) константа 2000, определяющая размер буферной зоны в области 640 кбайт для заблокированных сегментов, была подобрана экспериментально для сложных и больших программ, поэтому допуска-ется, что в случае других программ значение может быть изменено, 3)

память при разблокировке всегда считается измененной (_VM_DIRTY) по сравнению с исходным ее состоянием.

```
typedef _vmhnd_t VMindex;
#define VMNULL        _VM_NULL
#define VMopen()      _vheapinit(0,2000,_VM_EMS¦_VM_XMS)
#define VMclose()     _vheapterm()
#define VMalloc(size)  _vmalloc((unsigned long)size)
#define VMfree(index)  _vfree(index)
#define VMlock(index)  _vlock(index)
#define VMunlock(index) _vunlock(index,_VM_DIRTY)
```

3.4. Методика применения системно-назависимого управления памятью.

Предлагаемый интерфейс позволяет строить достаточно эффектив-ное системно-независимое управление динамической памятью приклад-ной программы. Стратегия использования интерфейса может быть определена следующим образом:

1) Проанализировать частоту использования структур данных, для которых используется ДП. При необходимости модифицировать алгорит-мы, настроив их на динамическое использование небольших объемов данных.

2) Использовать двухуровневую схему доступа (индекс-адрес) для динамически используемых данных.

3) Функции malloc() и free() применять для небольших по общему объему структур данных, особенно при большой частоте обращения к ним. Данные функции подразумевают выделение и фиксирование памяти, поэтому желательно их использовать при старте программы, чтобы уменьшить сегментацию памяти.

В заключение приведем небольшую программу на языке Си для иллюстрации применения функций:

```
/* включаем файл с определением интерфейса памяти */
```

```
#include "vmalloc.h"

main() /* далее тело главной программы */
 {
 /* программа строит динамический список из 10 элементов
   и потом его уничтожает */

 struct list { /* элемент динамического списка */
   VMindex buffer; /* индекс буфера данных */
   char * succ;    /* указатель на следующий элемент */
   }
  * tmp, /* указатель для различных целей */
  * root = NULL; /* указатель начала списка */
 int i;
 char * bufptr;

 VMopen(); /* инициализация интерфейса памяти */

 /* формирование списка */

 for( i=0; i<10; ++i )
  {
  /* выделение памяти для элемента списка */
  tmp = malloc( sizeof(list) );
  if( tmp == NULL ) { VMclose(); exit(1); }

  /* включение элемента в начало общего списка */
  tmp->succ = root;
  root = tmp;

  /* выделение памяти для данных элемента списка */
  tmp->buffer = VMalloc( 512 );
  if( tmp->buffer == VMNULL ) { VMclose(); exit(2); }

  /* блокировка памяти */
  bufptr = VMlock( tmp->buffer );
  bufptr[0] = \0; /* инициализация данных */

  /* разблокировка памяти */
  VMunlock( tmp->buffer );
  }
```

```
 /* здесь может быть выполнена прочая обработка */

 /* удаление списка соответствующими функциями */

while( root != NULL )
  {
  tmp = root->succ;
  VMfree( root->buffer );
  free( root );
  root = tmp;
  }

 VMclose(); /* остановка интерфейса памяти */

 exit(0); /* выход из программы */
 }
```

3.5. Выводы.

Использование обобщенной модели управления динамической па-мятью позволяет абстрагироваться на уровне отношений прикладной программы с системной оболочкой (ОС). Предствление памяти в виде конечных множеств позволяет формализовать выполняемые операции и ограничения. На основе модели реализован гибкий системно-незави-симый интерфейс для ОС UNIX, MS-DOS и среды Microsoft Windows, гарантирующий мобильность программ и скрывающий от программиста принципиальные различия конкретного управления ДП. Разработана ме-тодика применения интерфейса управления памятью.

Практическая проверка изложенного метода на примере графичес-кого редактора топологии "Layout Windows" [63] показала возмож-ность организации системно-независимого управления памятью, не требующего изменения исходного текста программы при ее переносе в различные вычислительные среды.

ГЛАВА 4

БЫСТРОДЕЙСТВУЮЩАЯ ИЕРАРХИЧЕСКАЯ БАЗА ДАННЫХ ГРАФИЧЕСКИХ ОБЪЕКТОВ

4.1. Способы организации баз данных графических объектов.

Выбор адекватной структуры данных для хранения графической информации очень важен для достижения высоких технико-экономических параметров САПР, особенно при обработке больших объемов информа-ции, свойственных сверх- и ультра-большим интегральным схемам, машиностроительному и архитектурному проектированию и т.п. Сложность создания оптимальной базы данных (БД) объясняется противоречивос-тью требования скорости выполнения запросов к БД и объемом дос-тупных ресурсов вычислительной системы. Для достижения высокой скорости обработки современные САПР формируют временные (опе-ративные, виртуальные) графические БД непосредственно в оператив-ной памяти ЭВМ. Основные классы структур безотносительно к графи-ческим приложениям были рассмотрены еще в [21], там же приведены оценки времени поиска произвольного элемента БД. В настоящей главе проводится анализ структур и алгоритмов для БД, наиболее эффек-тивных в графических приложениях. Структуры данных поясняются на примере рис.4-1.

Связанный список объектов является классической формой органи-зации простых БД для небольших объемов информации и представляет собой развитие структуры данных типа файл для дисковых устройств [21]. Несомненными преимуществами списка являются простота обслу-живания (операции добавления, удаления, поиска элементов и т.д.) и

отсутствие дополнительных управляющих структур данных (УСД). Одна-ко при увеличении длины несортированного списка значительно воз-растает время поиска произвольного элемента, которое имеет порядок O(N), где здесь и далее N - количество элементов в БД. В данном случае N - количество элементов в списке, или, для краткости, его длина. Значительное время поиска объясняется тем, что требуется просмотр всех элементов списка для проверки условия поиска.

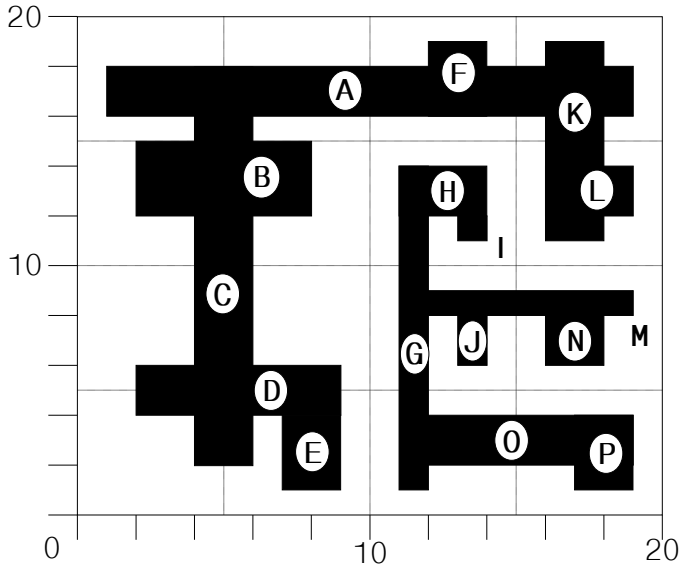


Рис.4-1 [74]

Пример графических данных топологии БИС

Для ускорения времени поиска все рассмотренные ниже схемы хра-нения графических данных используют различные УСД для организации сортировки в двумерном пространстве. В общем случае пространство может быть расширено до n-мерного. В виду общности применяемого подхода эти методы можно классифицировать как "пространственную сортировку" (ПС) объектов.

Одним из ранних способов ПС является метод двумерного массива ячеек (2-D bin structure) [66-68], в котором область допустимых значений (ОДЗ) координат разбивается равномерной сеткой на MxM прямоугольных ячеек. Соответственно в качестве УСД используется квадратная матрица указателей на списки элементов БД, покрываемых

данной ячейкой. В среднем список элементов одной ячейки умень-шается в MxM раз, пропорционально уменьшая время поиска. Вы-полнение запросов к БД включает дополнительную операцию по вычис-лению диапазона просматриваемых индексов матрицы, после чего вы-полняется операция для списков всех выбранных ячеек.

Опыт использования данной структуры выявил ее существенные не-достатки. Во-первых, размер ячейки и количество ячеек являются очень критическими параметрами БД. При увеличении размера ячейки возрастает длина ее списка объектов, тем самым замедляя операции поиска. При уменьшении размера ячейки возрастает количество объек-тов БД, попадающих на линии сетки, что порождает неоднозначность хранения объекта. Во-вторых, эта структура не приспособлена для неравномерно расположенных и / или разногабаритных объектов, ха-рактерных иерархическому проектированию. Для графической информа-ции такого свойства часть ячеек может содержать длинные списки, в то время как другие ячейки будут пустыми. И, наконец, рабочее пространство ограничено размерами M ячеек по координатной оси и есть риск появления объектов, выходящих за эти пределы.

Двумерный массив ячеек выявил одну существенную проблему, присущую всем схемам с жестко детерминированным характером сетки. Для графической информации нерегулярного вида может существовать некоторое подмножество объектов, попадающих на линии сетки. При решении вопроса об их принадлежности к конкретной ячейке возникает неоднозначность. С одной стороны, объект должен быть включен во все ячейки, его покрывающие, ибо в противном случае он может быть пропущен при поиске. С другой стороны, если объект хранится более чем в одном экземпляре, то при внесении изменений в БД следует учитывать все его копии. Если объект хранится только в одном экземпляре, но на него сделана более чем одна ссылка (схемы класса множественного хранения, в английской нотации - multiple storage [66,69,75]), то возникает необходимость исключения повторной обра-ботки уже

"использованного" объекта, что может быть реализовано с помощью флажков [75]. Но, как замечено в [74], выполнение запроса к БД при такой схеме будет включать дополнительный обход дерева УСД, что может удвоить время поиска объекта.

Для массива ячеек было предложено одно интересное решение [67], которое заключается в введении дополнительной "резиновой" ячейки с изменяемыми границами. В нее включаются все объекты, попадающие на границы сетки. При поиске объектов список "рези-новой" ячейки просматривается в обязательном порядке. Однако длина этого списка сильно зависит от главного параметра БД - размера ячейки и может быть очень большой. Пример на рис.4-1 показывает, что для ячейки 10x10 в "резиновую" ячейку попадает 3 прямоуголь-ника (A,C,G), а для ячейки 5x5 - 8 прямоугольников (A,B,C,D,G, K,M,O), что больше 50%.

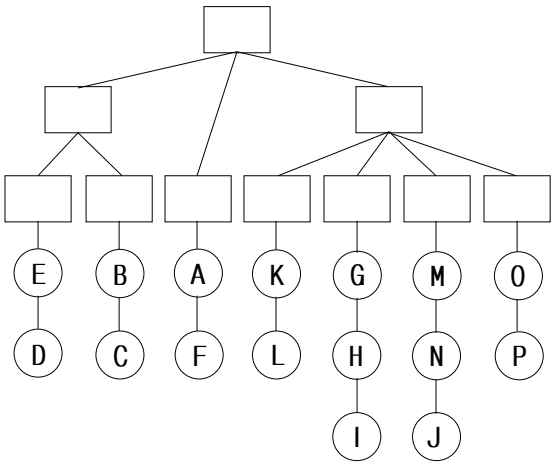


Рис.4-2

Структуры данных R-дерева

В предложенном R-дереве [70] сделана попытка принципиального отхода от дереминированной сетки путем группирования объектов в произвольные "резиновые" ячейки (в оригинале - оболочки). В качес-тве принципа группирования чаще всего применяется критерий мини-мального взаимного расстояния между объектами одной ячейки и / или максимальной длины списка объектов. Гибкость подхода позволяет полностью исключить попадание объектов на границы ячейки путем

растягивания ее сторон при необходимости. Соответственно исключается множественное хранение объекта или множественные ссылки.

Также в [70] было показано, что когда количество ячеек достигает существенной величины, неоправданно возрастают затраты на проверку критерия поиска для самих ячеек. Эта проверка существенно необходима, т.к. границы ячеек неизвестны априори и могут изменяться в процессе модификации БД. Для решения этой проблемы было предложено принципиальное решение, заключающееся в создании иерархии ячеек. Каждая супер-ячейка содержит несколько ячеек низшего уровня, а те, в свою очередь, тоже могут содержать ячейки или списки объектов. Супер-ячейки создаются при превышении длины списка объектов. Таким образом, заложенный в схеме БД механизм позволяет адаптироваться к произвольной графической ин-формации, не зная заранее ее свойств. При этом достигается доста-точно высокая скорость поиска и экономное использование памяти для УСД. Пример возможных структур данных R-дерева для рис.4-1 и длины списка не более 3 приведен на рис.4-2. Прямоугольниками обозначены УСД, кругами - объекты БД.

Предложенные в R-дереве принципиально важные решения не получили широкого распространения в связи с присущими методу серьез-ными недостатками [70]. Недетерминированность структуры БД на практике приводит к созданию перекрывающихся ячеек, что вызывает неоднозначность размещения объекта в конкретной ячейке. Также достаточно сложными являются алгоритмы обслуживания БД.

В [71] было предложено Q-D, или квадро-дерево (в оригинале - quad tree), а в [72] и [73] показана его эффективность для быстрой обработки прямоугольников. Квадро-дерево представляет собой результат рекурсивного деления прямоугольной области, покрывающей все объекты БД. Каждая область, или ячейка по используемой ранее терминологии, делится на 4 части путем проведения одной секущей линии по каждой координате, если длина списка объектов ячейки-родителя превышает некоторый порог - параметр БД. Объекты, вхо-дящие в новые

ячейки, включаются в списки дочерних ячеек, а объекты, попадающие на секущие линии, включаются в два дополни-тельных списка объектов ячейки - "родителя". Пример УСД для фигур рис.4-1 приведен на рис.4-3. В отличие от R-дерева, такая схема БД гораздо проще для обслуживания и она получила широкое применение на практике. Однако, как замечено в [74], метод хранения объектов, попадающих на секущие линии ячеек, очень неэффективен с точки зрения скорости поиска, т.к. в списки может попасть очень много обектов - в наихудшем случае порядка $O(\sqrt{N})$ - и все они должны быть просмотрены.

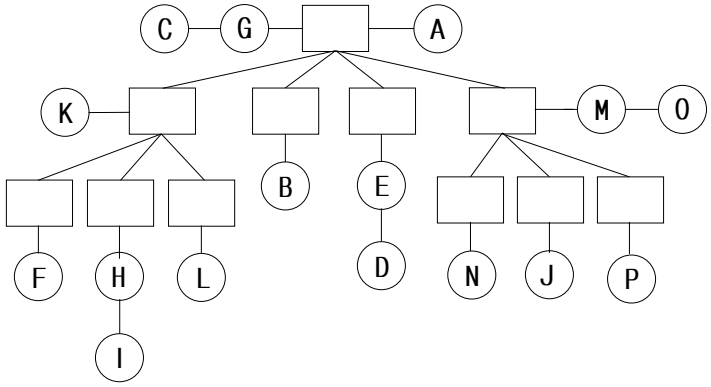


Рис.4-3 [74]

Структуры данных квадро-дерева

В [75] предложен ряд принципиальных усовершенствований для квадро-дерева. Во-первых, для ускорения поиска введен принцип мно-жественного хранения объектов, попадающих на секущие линии ячеек. Во-вторых, для уменьшения глубины дерева и, соответственно, умень-шения времени поиска, понятие квадро-дерева (2x2-дерева) расширено до MxM-дерева, причем коэффициент деления может меняться для раз-ных ячеек, чтобы получить для них список объектов приемлемой дли-ны.

Для достижения наивысшей скорости поиска графических объектов была предложена схема типа бинарного дерева, получившая название k-D дерева [76]. В работах [77] и [78] было представлено более детальное описание k-D дерева и алгоритмы поиска объектов для приложений САПР. Эффективность метода заключается в том, что в листьях дерева хранится

не список объектов, как в R-дереве или в квадро-дереве, а только один объект. Оценка времени поиска произвольного объекта минимальна для всех рассмотренных выше схем БД и составляет $O(\log_2 N)$, но за такую скорость приходится платить большими затратами памяти для УСД.

В наиболее распространенном варианте k-D дерева - 4-D дереве [78,79] - используется циклическое деление пространства координат пополам с помощью четырех ключей, названных дискриминантами: x_1, y_1, x_2, y_2. Каждый дискриминант выбирается таким образом, чтобы результирующее дерево оставалось сбалансированным, в противном случае ухудшается время поиска произвольного объекта. Пример УСД для фигур рис.4-1 приведен на рис.4-4 (ключи x_1, y_1, x_2, y_2).

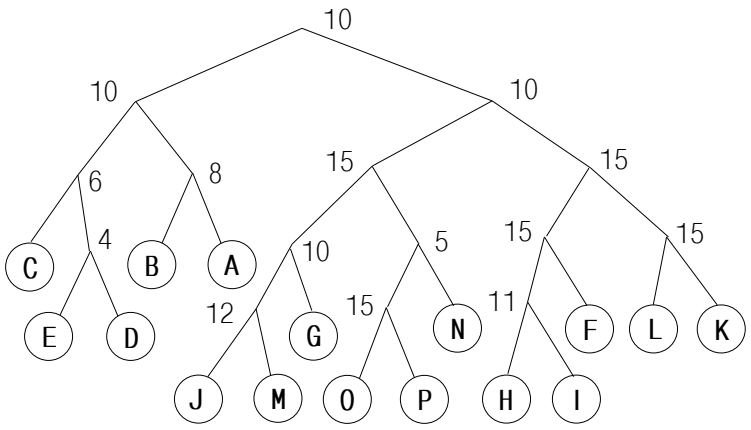


Рис.4-4

Структуры данных 4-D-дерева

В [70] предложен вариант структуры БД, названный k-D-B деревом. Он занимает промежуточное место между k-D деревом и квадро-деревом. УСД, формируемые с помощью дискриминанта, описы-вают некоторую область, покрывающую два дочерних элемента УСД или непосредственно объект БД, тем самым уменьшая потребность в памяти. Само дерево УСД по-прежнему является бинарным.

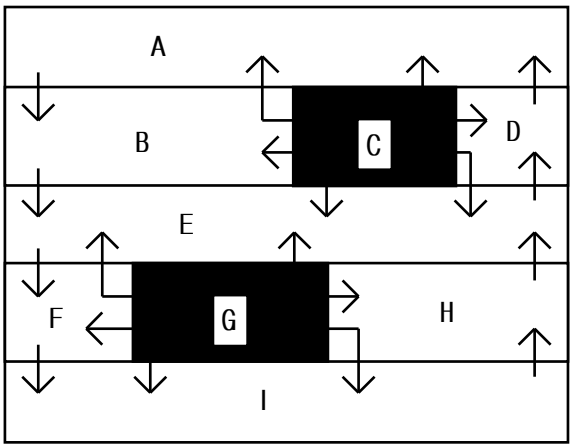


Рис.4-5 [58]

Пример тайловых графических структур

В [82] была предложена тайловая структура БД (в иностранной литературе - corner stiching - сшивание углов), непохожая на все БД, описанные выше. Тайловая структура предполагает однородность графических данных - допускаются только прямоугольники. Это очень сильное ограничение было ослаблено в [83], где БД может включать трапеции. Такое расширение позволяет разбивать все прочие объемные фигуры на набор прямоугольников и трапеций перед их помещением в БД. При построении базы данных от каждого объекта должно быть построено ровно 4 ссылки: на два соседа непосредственно сверху и справа и на два соседа снизу и слева (рис.4-5). Подобная схема идеально подходит для решения задач, связанных с поиском соседа для исследуемого объекта [83-86], т.к. необходимо просмотреть только 4 ссылки на ближайших соседа. Кроме того, облегчается задача поиска свободных участков, являющихся неотъемлемой частью БД. К недостаткам тайловых структур, в первую очередь, относится значительное по сравнению с другими структурами время формирования БД (по данным [85] - до 16 мин. для топологии из 32500 транзис-торов), большое время поиска произвольного объекта - порядка $O(\sqrt{N})$ [70,74] и значительные дополнительные затраты памяти на форми-рование пустых прямоугольников. Нельзя не отметить сложность алгоритмов

обслуживания БД и неоднозначность ситуаций с пересека-ющимися объектами.

4.2. Постановка задачи оптимального построения иерархической базы данных.

Подведем краткий итог проведенного выше анализа:

1. С точки зрения универсальности применения различные вари-анты деревьев УСД предпочтительнее, т.к. они обеспечивают более простую систему пространственных отношений между объектами, а, соответственно, и более простые структуры данных и алгоритмы.

2. Для уменьшения времени глобальной выборки данных следует минимизировать структуру УСД, например, удлинением списка объектов в листьях дерева. В связи с этим k-D-деревья оказываются менее предпочтительными по сравнению с другими вариантами деревьев.

3. Для уменьшения времени локальной выборки данных следует минимизировать последовательные списки объектов в листьях дерева УСД. Кроме того, структуры данных должны обеспечивать наиболее быстрый доступ к этим спискам. По этим критериям наиболее пред-почтителен вариант двумерного массива ячеек, который обладает наи-меньшим временем поиска списка объектов.

Таким образом, построение оптимальной структуры БД представ-ляет собой достаточно сложную задачу из-за противоречивости требо-ваний. Рассмотренные в предыдущем параграфе схемы хранения графи-ческих данных в качестве критериев эффективности используют либо минимум времени поиска объекта, либо минимум УСД при достаточно высокой скорости поиска. Кроме того, они предполагают организацию данных проекта в одноуровневом представлении или предполагают раз-вертывание иерархии проекта в один уровень. Однородность представ-ления позволяет строить эффективные алгоритмы хранения и обработки графических данных.

В настоящее время объемы обрабатываемой информации настолько велики, что организация данных в виде некоторой структурной иерар-хии становится неотъемлемым требованием при проектировании баз данных и систем управления базами данных (СУБД). Специфика данных проекта и множество субъективных и случайных факторов, свойствен-ных "ручному" формированию такой структуры, приводит к ее сильной неоднородности, что ухудшает оценки времени поиска и увеличивает непроизводительные затраты памяти для рассмотренных УСД.

Что наиболее важно, оптимальная иерархическая организация ин-формации САПР, особенно при значительной доле ручного проектиро-вания, предполагает незначительный объем данных в каждом узле иерархии. Этот факт принижает преимущества многих алгоритмов, рас-считанных на большие объемы одноуровневой информации, т.к. абсолютное время выполнения операций различными алгоритмами над небольшим количеством объектов достаточно мало и тем более несущественна разница между ними. Для больших объемов информации при использовании k-D деревьев достигается наивысшая скорость, приближающаяся к $O(\log_2 N)$, а при использовании MxM-деревьев ско-рость определяется как $O(\log_{MxM}(N/N_{списка}) + N_{списка})$. Это рассуждение позволяет выдвинуть тезис о том, что быстродействие алгоритмов (в частности, алгоритмов поиска произвольного объекта) не играет первоочередную роль. На основании сказанного на первый план выдвигается простота и компактность внутренних структур данных БД, а также простота алгоритмов их обработки.

Следует принять во внимание, что рассматриваемые в литературе [21,66-81] БД предполагают выполнение какого-либо одного узкого класса операций над объектами. Такими операциями при проектирова-нии СБИС являются контроль проектных норм, восстановление элект-рической схемы и т.п. При построении комплексных систем автома-тизации проектирования возникает задача разнообразной обработки проектных данных. Внутренняя структура БД, эффективная для одной задачи, может

оказаться неэффективной для другой задачи, а динамическое создание специфических временных БД для каждой проектной операции может приводить к дополнительной трате времени и ресурсов.

Кроме того, если СУБД оперативной БД не обеспечивает возврат неизмененных данных в постоянную БД в том же виде, в каком они были до выполнения операции, то возникает дополнительная задача определения соотношения результатов с предыдущим состоянием проекта. Для ряда приложений, например, для проектирования топологии, следует учитывать необходимость применения послойного хранения информации. При объединении в одну структуру объектов разных слоев экономится память для УСД, но увеличивается время поиска за счет проверки критериев для "чужих" слоев, также находящихся в списках. С другой стороны, построение УСД отдельно для каждого слоя существенно увеличивает затраты памяти при оптимальном времени поиска.

Заметим далее, что в случае "прозрачной" выборки объектов БД, когда объекты нижних уровней проекта (потомки) проецируются на уровень обработки, для выполнения операции требуется копирование объекта во временную область памяти и трансляция координат. Это действительно необходимо, т.к. объект-потомок в БД существует в одном экземпляре, но может использоваться многократно путем ис-пользования параметризованной ссылки. Если суммарное время копиро-вания, выполнения трансформации и проведения операции над объектом (особенно при сложной геометрической форме) становится доминирую-щим по сравнению со временем поиска, то для достижения баланса производительности СУБД в целом не имеет смысла дальнейшее уско-рение алгоритмов, используемых для одноуровневого представления. Ибо это приведет к усложнению структур данных и алгоритмов, что, в свою очередь, увеличит потребности в памяти для кода и данных.

Из представленных выше замечаний следует вывод о необходимости разработки структуры БД, оптимизированной для иерархического при-менения. Сформулируем основные требования - критерии оценки БД:

1. БД должна обеспечивать минимальную компактную структуру УСД для обработки больших проектов (при недостатке ресурсов ЭВМ).

2. Алгоритмы СУБД должны быть адаптивными к параметрам, определяющим структуру БД.

3. БД должна обеспечивать хранение и обработку иерархически организованной информации.

4. СУБД должна выполнять автоматическую трансформацию координат при выборке данных с любого уровня иерархии, а также иметь возможность выполнять специальные операции, не заложенные в СУБД.

4.3. Структура и алгоритмы базы данных для иерархического проектирования.

На основании вышеперечисленных критериев была разработана БД графических объектов [80], ориентированная на хранение и обработку графической информации в иерархическом представлении. При разработке БД был принят во внимание тот факт, что организация БД по принципу квадро- или MxM-дерева обеспечивает наиболее оптимальное сочетание высокой скорости выполнения запросов с небольшими затратами памяти на УСД. Это позволяет строить достаточно эффективные САПР на базе вычислительной техники среднего класса и даже на персональных ЭВМ типа IBM PC/AT. Разработанная БД основывается на принципах MxM дерева, как наиболее современной БД, но имеет ряд принципиальных отличий.

Для стандартного MxM-дерева параметрами выступают коэффициент деления M и максимальная длина списка объектов в листьях дерева. Рассмотрим их влияние на структуру БД и алгоритмы СУБД. Каждый УСД содержит двумерный массив размерности MxM из указателей на УСД нижнего уровня. Если зафиксировать M (например, в квадро-дереве M=2), то алгоритм обхода дерева будет использовать фиксированный способ выбора (перебора) этих указателей. Но если параметр M явля-ется

динамически изменяемым, то алгоритм обхода должен быть адаптивным к тому, что размерности массива указателей "плавающие".

Существует два принципиальных способа определить размерность массива. Тривиальный способ заключается в хранении граничных индексов в дескрипторе дерева, если они одинаковы для всех УСД, либо в самом УСД. Его преимущество в том, что вычисление фиксированных индексов достаточно просто, особенно при детерминированных разме-рах УСД. Недостатком способа выступает необходимость хранения всего массива указателей, независимо от того, насколько он запол-нен ссылками на поддеревья. Для дерева из L уровней это составит (4-1) указателей, что при больших M весьма существенно:

$$M^2 \sum_{l=0}^{L-1} M^{2l} = M^2 \frac{M^{2L} - 1}{M^2 - 1} \approx M^{2L} \qquad (4\text{-}1)$$

Альтернативная возможность хранения массива заключается в упорядочивании УСД нижнего уровня в линейный список. Преимущество метода в том, что требуется минимальный объем памяти, т.к. в списке хранятся только непустые УСД. Соответственно упрощаются алгоритмы перебора: вместо вычисления и/или перебора индексов выполняется простой просмотр всех УСД. Существенным недостатком этого подхода служит тот факт, что всегда необходима проверка всех УСД из списка, даже при локальном поиске, когда критерию поиска может удовлетворять только один УСД из списка.

Если критерием для выбора одного из методов выступает минимум памяти, то следует организовывать УСД в линейный список. Но если критерием служит минимум времени поиска, то следует проанализировать время вычисления критерия для УСД нижнего уровня. При точечном поиске в MxM-дерева из L уровней требуется L-1 раз выпол-нить выбор нижележащего УСД. Если используется фиксированный мас-сив указателей, для детерминированных размеров УСД существуют очень быстрые методы вычисления индексов массива, использующие методы

двоичной арифметики [75]. В случае линейного списка УСД полученное время умножается на длину списка P*M*M, где P - вероятность присутствия УСД в списке. Если дерево содержит много информации, например, все слои фотошаблона СБИС, то эта вероят-ность приближается к 1.

Максимальная длина списка объектов в листьях дерева и коли-чество уровней (глубина) дерева УСД определяют время локального поиска следующим образом:

$$T = T_{УСД}(L - 1) + T_{объекта}N_{списка} \qquad (4\text{-}2)$$

где $T_{УСД}$ - время поиска УСД нижележащего уровня;

$T_{объекта}$ - время проверки критерия поиска для одного объекта;

$L = 1 + [\log_{M^2} \frac{N}{N_{списка}}]$ - количество уровней дерева.

Формула (4-2) дает лишь приблизительную оценку времени локального поиска. Пространственное расположение и свойства реаль-ных фигур в общем случае вносят погрешности, которые сложно фор-мализовать. Но (4-2) может использоваться для оценки влияния пара-метров БД на время поиска. Увеличение коэффициента дробления M позволяет наиболее существенным образом уменьшить время, в силу логарифмической зависимости для количества уровней. Однако для списковой организации УСД при увеличении M время выбора УСД умно-жается на квадрат M (без учета вероятности).

Длина последовательного списка объектов влияет на время поиска двояким образом. Во-первых, подразумевается, что все элементы списка должны быть безусловно просмотрены и поэтому уменьшение списка пропорционально уменьшает время поиска. Во-вторых, увеличе-ние длины списка тоже может привести к уменьшению длины поиска за счет

уменьшения количества уровней дерева УСД, особенно для спис-ковой организации УСД. Найдем локальный минимум для (4-2):

$$\left.\begin{aligned} \frac{dT}{dN_{списка}} &= \frac{-T_{УСД}}{N_{списка}/nM^2} + T_{объекта} = 0 \\ \frac{d^2T}{dN^2_{списка}} &= \frac{T_{УСД}}{N^2_{списка}/nM^2} \rangle 0 \end{aligned}\right\rangle \Rightarrow \qquad (4\text{-}3)$$

$$\Rightarrow N_{списка\ min} = \frac{T_{УСД}}{T_{объекта}/nM^2}$$

Формула (4-3) дает для фиксированного массива указателей оптимальное значение длины списка около 1. Это вызвано тем, что проход последовательного списка в сильной степени влияет на суммарное время. Для альтернативной списковой организации УСД оптимальная длина списка становится больше:

$$N_{списка\ min} = \frac{T_{УСД}M^2}{T_{объекта}/nM^2} \qquad (4\text{-}4)$$

Значения оптимальной длины списка по (4-4) при равных временах проверки приведены в табл.4-1.

Таблица 4-1

M	2	3	4	5	6	7	8
M*M	4	9	16	25	36	49	64
Nсп	3	4	6	8	10	13	15

Рассмотрим возможные способы управления структурой БД посредством ограничения длины последовательного списка объектов в дере-ве. Первый подход заключается в контроле точной длины списка. Для этого

необходимо вести подсчет количества объектов в списке. Уменьшения числа подсчетов можно добиться использованием инкремен-тального метода, при котором в УСД хранится счетчик объектов. При превышении счетчиком некоторого значения выполняется достраивание структуры УСД вниз и разбиение списка объектов данного УСД на списки объектов УСД нижнего уровня. Назовем этот метод детермини-рованным.

Другой подход заключается в использовании вероятностного под-хода. Если исходить из некоторой вероятности расположения объектов в пространстве, то можно определить площадь, покрываемую списком объектов фиксированной длины. Для практических применений в микроэлектронике можно записать:

$$S_{списка} = S \frac{N_{списка}}{N} \qquad (4\text{-}5)$$

где S - суммарная площадь СБИС, рисунка и т.д.

Вероятностный подход не обеспечивает точную длину списка, од-нако для него не требуется перебор и разбиение списка объектов. Вместо этого выполняется только проверка данного УСД на минималь-ность размеров. При длинных списках это может быть эффективнее перебора, более того, исключаются ситуации невозможности разбиения списка близкорасположенных объектов. Другим преимуществом служит возможность построения структуры УСД в случае небольшого коли-чества разрозненных объектов. При фиксированном списке будут выполняться проверки для всех объектов, в то время как вероят-ностный подход предоставляет возможность разбить список на нес-колько частей.

Метод множественного хранения объектов, предложенный в [69], является спорным с точки комплексного решения проблемы ускорения выборки данных. Этот метод позволяет исключить просмотр списков объектов, попадающих на секущие линии УСД. Для доступа к объектам используются не списки объектов, а списки указателей на объекты. Сами объекты помещаются в некотором списке, массиве, буфере или т.п.

Несколько УСД могут содержать ссылки на один и тот же объект, что является как преимуществом, так и серьезным недостатком. Уско-рение выборки достигается за счет усложнения алгоритмов. Чтобы исключить повторную выборку объектов, необходимо использовать ка-кой-либо вид проверки, например механизм флагов. Сброс флагов в исходное состояние может быть выполнен за счет дополнительного обхода дерева, что может удвоить время обработки.

При разработке базы данных были использованы результаты прове-денного анализа. Ее основные особенности следующие:

1. Критерий минимальной длины списка объектов в ячейке УСД заменен критерием минимальных габаритов ячейки. Введение вероят-ностного критерия позволяет либо сэкономить память для УСД за счет отказа от хранения счетчика объектов в ячейке, либо, при отсут-ствии такого, исключить подсчет количества уже существующих в списке объектов.

2. Чтобы исключить перебор и анализ уже внесенных объектов с целью размещения их в новых списках при динамической модификации дерева УСД при формировании БД, объекты сразу вводятся в минимальную покрывающую ячейку. Такое решение позволяет сократить время включения нового ближлежащего объекта при их большом количестве в БД, т.к. включение новых экземпляров происходит в почти готовую структуру. Кроме того, при выполнении мелкомасш-табной эскизной прорисовки содержимого БД есть возможность оценить максимальный габарит всех объектов любого уровня иерархии дерева и исключить проход части ветвей, для которых изображение будет заведомо трансформироваться в одну точку и не изменять картинку на экране. При вводе нового объекта в БД существует три возможных ситуации:

1). Ячейка минимального размера, полностью покрывающая объект, существует в БД;

2). Ячейка минимального размера меньше корневой, но отсутст-вует в БД. В этом случае находится или создается ячейка, покрыва-ющая

предполагаемую, и от нее достраивается иерархия уменьшающихся ячеек, пока не будет создана ячейка требуемого размера;

3). Корневая ячейка дерева УСД имеет меньшие габариты. В этом случае в корне дерева создается ячейка большего размера (верхнего уровня) и предыдущее дерево включается в новую корневую ячейку. Операция повторяется до тех пор, пока в корне дерева не окажется ячейка требуемого размера.

Несмотря на кажущуюся сложность алгоритмов модификации дерева УСД, эти изменения позволяют решить проблемы, возникающие при не-возможности для Q-D деревьев разделить список ячейки на списки подъячеек при сильной близости объектов [75]. С другой стороны, исключается проблема включения объектов, попадающих на границы ячеек - они просто включаются в ячейку приемлемого верхнего уров-ня. В сделанной реализации БД [80] остается лишь одна "резиновая" ячейка на все дерево, в которую включаются объекты, пересеченные осями координат. Этой ячейки может не быть совсем при соответ-ствующем расположении или сдвиге объектов в координатном прост-ранстве. Конечно, образуется некоторая избыточность УСД, но она компенсируется полученными преимуществами.

3. Развивая дальше подход MxM дерева по разбиению ячейки на подъячейки, можно сделать метод разбиения независимым, если зафик-сировать правила обхода дерева УСД. Нет никаких принципиальных ограничений в выборе различных коэффициентов деления и размеров ячейки по осям координат. В реализации БД принято разбиение на равное количество частей по оси X и Y, но размеры минимальной ячейки могут быть различными. Выбор неравных сторон особенно актуален для информации с выраженными канальными свойствами, например, для матричных СБИС.

4. Метод фиксации минимальных размеров позволяет упростить выбор размеров ячейки верхнего уровня, т.к. они получаются путем умножения

размеров ячейки нижележащего уровня, а не делением раз-меров вышележащей.

5. Для упрощения структур данных и, соответственно, алгоритмов обхода дерева в реализации БД [80] принято решение об организации подъячеек в виде списка. Таким образом, каждая ячейка содержит три указателя: один на ячейку данного уровня, один на список подъячеек и один на список объектов, не вошедших в ячейки нижнего уровня (рис.4-6). Такой механизм позволяет экономить память при разбиении ячейки за счет создания не MxM подъячеек, а только необходимого их количества.

6. Организация дерева УСД в виде связанных списков позволяет повысить быстродействие алгоритмов поиска за счет отказа от рекурсии. В приводимых в литературе [72-75] алгоритмах обхода дерева предлагается использовать рекурсивный вызов процедуры при переходе на следующий уровень иерархии. Вообще говоря, это не самое удачное решение с точки зрения быстродействия, т.к. любой вызов процедуры предполагает сохранение контекста вызывающей про-цедуры, начальные установки для вызываемой и формирование факти-ческих параметров. При простоте принятия решения по выбору ячейки УСД при обходе дерева смена вычислительного контекста выглядит достаточно громоздкой. Так как в большинстве ЭВМ для хранения контекста используется стек, могут возникнуть ситуации его переполнения при большой глубине рекурсии для машин класса IBM PC/AT, где стек вместе со статическими данными не может превышать 64-х килобайт практически для всех коммерческих компиляторов. Кроме того, нельзя не отметить тот факт, что для вновь созданного контекста изменяется лишь один параметр - указатель на обраба-тываемую ячейку.

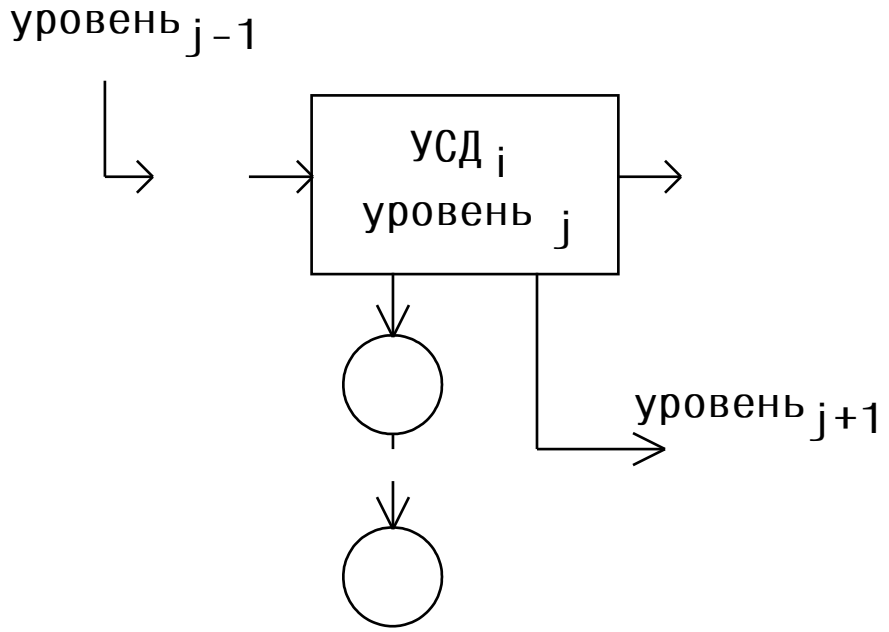


Рис.4-6

Универсальный элемент структур данных

В разработанной СУБД используется алгоритм нерекурсивного обхода дерева УСД. Существование такого алгоритма стало возможным благодаря однородности используемых структур данных и применению указателей на списки. В основе алгоритма лежит использование буфера класса "первый вошел - последний вышел" - стека указателей на ячейки. Максимальная глубина этого стека легко оценивается как $\log_{м}$(Макс. размер ячейки / Миним.размер ячейки), что является не-большим значением. Данный подход позволяет свободно пользоваться рекурсивным вызовом процедур при проходе иерархии высшего порядка - иерархии проектных единиц. Основные шаги алгоритма приводятся ниже:

1) Загрузить в стек адрес первого УСД.

2) Если значение в стеке - нуль, то

2.1) Выгрузить из стека нулевой указатель

2.2) Если в стеке больше нет указателей - выход.

2.3) Заменить текущий указатель в стеке на адрес УСД, следу-ющего в списке за УСД из стека.

2.4) Перейти к п.2.

3) Если критерии поиска не выполняются для УСД из стека, перейти к п. 2.3.

4) Для всех объектов УСД из стека, удовлетворяющих критерию поиска, выполнить запрошенную операцию.

5) Если адрес на субъячейку УСД из стека отличен от нуля, загрузить ее в стек и перейти к п. 2. Иначе перейти к п. 2.3.

Сравнивая представленные в литературе [21,66-81] алгоритмы об-хода дерева УСД, можно заметить, что большинство из них в явном или неявном виде включает проверку критерия поиска для всех ячеек УСД данного уровня, суть которой заключается в вычислении опера-ции пересечения охватывающего прямоугольника УСД и прямоугольника геометрического запроса. В итоге это эквивалентно просмотру пол-ного списка в разработанной БД. Более того, т.к. БД не включает пустые УСД, длина списка может быть короче максимальной, сокращая время поиска. Алгоритмы, использующие быстрый отбор субячеек [67,75], всегда имеют жесткие границы ячеек.

И в заключение приведем описание программного интерфейса на языке C к основным функциям разработанной СУБД.

Инициализация БД.

```
void cminit( int dx, int dx, int m )
```

Параметры dx и dy задают минимальный размер ячейки УСД, m - фактор размножения MxM-дерева.

Создание или поиск ячейки в БД.

```
cmsymbol * cmcreatesymbol( char * name )
```

Параметр name задает строку символов типа ASCII - имя ячейки.

Удаление ячейки из БД.

```
void cmdeletesymbol( cmsymbol * symbol )
```

Параметр symbol задает указатель на дескриптор удаляемой ячейки, для которой удаляются все объекты и пустое дерево УСД. При отсутствии ссылок на данную ячейку удалается сам дескриптор.

Трансформация объектов базы данных.

int cmtransformer(cmsymbol * inputsymbol, int inputlayer, cmsymbol * outputsymbol, int outputlayer, cmbounds * boundingbox, int selection_by_type, int selection_by_state, int operation, int state_to_set, cmt * transformation, int level, int (*useraction)(), int (*usercontext)())

Многооперационная функция для выполнения операций над объектами в БД. Основные типы операций задаются битами в параметре operation, дополнительная обработка выполняется внешней для БД функцией useraction. Область геометрического поиска задается параметром boundingbox, селекция по типу и состоянию задается параметрами selection_by_type и selection_by_state. Объекты выбираются из слоя inputlayer ячейки inputsymbol и, при необходимости, переносятся в слой outputlayer ячейки outputsymbol. Глубина "прозрачной" выборки задается параметром level, при смене уровня иерархии для смены прикладного контекста вызывается функция usercontext. Использо-вание аппарата внешних функций позволяет строить эффективные и мощные программы обработки данных.

Ввод объектов в базу данных.

int cmcreate...(cmsymbol * symbol, int layer, void * descriptor, cmarray * array, char * property)

Набор функций создает объекты, описанные параметром descriptor, в слое layer ячейки symbol, назначая им параметры мультипликации array и атрибуты property.

4.4. Методы ускорения "прозрачной" выборки данных из иерархической БД.

Преимущества в экономии памяти, предоставляемые иерархической организацией БД, особенно важны для больших современных проектов в САПР. Однако, как было замечено в предыдущей главе, выборка данных из таких БД требует дополнительных затрат времени на прохождение

иерархии. В настоящей работе рассматривается ряд методов ускорения алгоритмов выборки данных.

Сначала уточним ряд терминов. В отечественной литературе и документации отечественных САПР часто названием "ячейка" обозначается как логически организованный набор графических данных, так и ссылка на такой набор. Поэтому для различения этих понятий под ячейкой будем понимать именно набор данных, а ссылку на него будем обозначать часто используемым термином "привязка".

При выборке объектов согласно параметрам привязки тривиальный метод заключается в копировании их в текущую ячейку с последующей трансформацией координат. Для того, чтобы удалить временные объек-ты после выполнения операции, их следует каким-либо образом поме-тить. Если среди вновь созданных объектов встречаются привязки, копирование и трансформация координат повторяются. Преимущество метода - в использовании простого нерекурсивного цикла, а главный недостаток - потребность в большом объеме памяти для временных объектов.

В разработанной БД [80] используется метод "прозрачной" выборки данных, не требующий затрат памяти, за исключением памяти для временного хранения одного объекта. Его суть заключается в организации пообъектной выборки данных и выполнением операций над каждым объектом, трансформированным в систему координат исходной ячейки. Предварительно запишем функцию трансформации координат, которая служит и главным параметром привязки, в следующем виде:

$$\overline{P} = f_j(\overline{S}_j, \overline{R}_j, \overline{T}_j, \overline{P}) \qquad (4\text{-}6)$$

где

$\overline{P}$ - вектор-координата точки

$\overline{S}_j$ - вектор масштабных и/или зеркальных преобразований

$\overline{R}_j$ - вектор поворота

$\overline{T}_j$ - вектор смещения центров координат

Предложенный метод предполагает выборку объектов данной ячейки обычным способом, вслед за чем для всех привязок, удовлетворяющих критерию поиска, выполняются следующие действия:

1) Вычисление новой функции трансформации координат для уровня j+1 по известной трансформации текущего уровня J и трансформации $f_{пр}$, задаваемой привязкой:

$$\bar{P} = f_j(\bar{S}_j, \bar{R}_j, \bar{T}_j, f_{пр}(\bar{S}_{пр}, \bar{R}_{пр}, \bar{T}_{пр}, \bar{P})) = \\ = f_{j+1}(\bar{S}_{j+1}, \bar{R}_{j+1}, \bar{T}_{j+1}, \bar{P}) \quad (4\text{-}7)$$

2) Вычисление новой области поиска. В связи с тем, что выполняется проецирование координат текущего уровня на более глубокий, используется трансформация, обратная трансформации привязки. Для обеих точек на диагонали прямоугольной области применяем:

$$\bar{P} = f_{пр}^{-1}(\bar{S}, \bar{R}, \bar{T}, \bar{P}) \quad (4\text{-}8)$$

3) Запускаем обычный поиск в привязанной ячейке, используя пересчитанную область поиска и трансформацию координат.

Для работы алгоритма требуется задание начальной трансформа-ции, в качестве которой может выступать либо тождественная транс-формация для простой выборки объектов, либо соответствующая опера-ции для начальной ячейки, как в случае рисования, копирования и т.д. Время переключения с одной ячейки на другую определяется временем вычисления новых параметров для функции выборки и време-нем смены вычислительного контекста при ее вызове. Как будет показано в следующей главе, для трансформаций с поворотами, кратными 90 град., временные затраты достаточно малы. В случае произвольных поворотов возрастает время вычисления обратного преобразования координат. При достаточных ресурсах ЭВМ, что имеет место в ОС UNIX или VMS, целесообразно хранить обратную транс-формацию в дескрипторе привязки для ускорения расчетов.

Рассмотрим другой аспект ускорения алгоритмов обхода дерева УСД, основанный на избирательном вычислении критериев геометри-ческого поиска. Метод использует детерминированность изменения размеров ячеек УСД в дереве и поэтому применим ко всем деревьям класса MxM. В основе метода лежит применение троичной логики для вычисления результата пересечения прямоугольников:

0 - отсутствие пересечения;

1 - охватывающий прямоугольник УСД полностью покрывается (входит) в прямоугольник поиска;

2 - все остальные случаи пересечения.

Небольшое усложнение вычислений в троичной логике с лихвой компенсируется исключением проверки для списка объектов и субъ-ячеек, когда УСД покрываются полностью областью поиска. Алгоритм обхода дерева разработанной БД в подобной ситуации осуществляет простой перебор списка объектов. Этот метод позволяет разрешить дилемму, свойственную многим схемам БД, когда параметры базы данных для быстрого локального поиска не обеспечивают высокой скорости для глобальной обработки объектов, и наоборот.

Оценим эффект ускорения при применении избирательной проверки. Разобьем все дерево поиска типа MxM и глубиной L (для удобства примем, что уровень L соответствует корню дерева, уровень 0 - листьям) на два участка: [0,1], когда область поиска полностью покрывает поддерево УСД, и от]1,L], когда требуется проверка критерия. Будем считать, что объекты находятся только в листьях дерева УСД.

На каждом уровне n в диапазоне]1,L] любому алгоритму требу-ется проверка MxM элементов УСД. Выразим это так:

$$T_{п} = T_{УСД}(L - 1)M^2 \qquad (4\text{-}9)$$

где

$T_{УСД}$ - время вычисления критерия для одного УСД

Проверка критерия в диапазоне уровней **[0,l]**, если она выполняется, производится над всеми УСД поддерева, начинающегося с уровня **l**, и может быть оценена суммой:

$$T_{л} = \sum_{n=0}^{l} M^{2n} \quad (4\text{-}10)$$

Определим получаемый эффект K (4-11) как отношение времени обхода дерева УСД с полной проверкой ко времени обхода с частичной проверкой. При выполнении локальных запросов (l=0) эффект не наблюдается, но при выполнении глобальных операций (l=L-1) ускорение алгоритма может быть существенным, особенно когда время выполнения операции над объектами мало. Следует заметить, что значение уровня L при иерархической обработке включает в себя сумму всех проходимых уровней в иерархии и поэтому несмотря на небольшие локальные значения глобальный уровень может быть существенным.

$$K = (T_{п} + T_{л})/T_{п} = 1 + T_{л}/T_{п} = 1 + \sum_{n=0}^{l} M^{2 \bullet n} \Big/ (L - l) \bullet M^{2} = \quad (4\text{-}11)$$

$$= 1 + \frac{1}{L - l} \bullet \sum_{n=0}^{l} M^{2 \bullet (n-1)}$$

Заметим, что отказ от ненужной проверки критериев поиска сводит к минимуму непроизводительные затраты времени при выборке объектов из БД. В результате алгоритм выборки приближается к максимально возможной скорости, эквивалентной простой безусловной поочередной обработке объектов, находящихся в одном последова-тельном списке.

Эффект, получаемый описанным выше методом, обусловлен тем, что имея некоторую обобщенную информацию, можно принять решение об упрощенной проверке критерия. В качестве такой информации выс-тупает охватывающий прямоугольник, хранимый в дескрипторе ячейки, в ячейке УСД и в описании сложных контурных фигур. Его вычисление несложно, а

эффект достаточно существенен. Поэтому охватывающий прямоугольник является неотъемлемой частью структур данных большинства современных графических БД. Обобщая и развивая этот подход дальше, можно выделить другие наиболее часто используемые характеристики объектов, которые следует накапливать в управляющих структурах данных. Эти признаки в сильной степени определяются спецификой приложения базы данных. В разработанной БД [80], больше ориентированной на проектирование СБИС, дополнительно к охватыва-ющему прямоугольнику используется информация о типе объекта (прямоугольник, многоугольник, шина постоянной ширины, линия или привязка), а также признаки состояния объекта (существует, удален, выбран для обработки, помечен и др.). Эта информация пред-ставляется в битовом виде, занимает немного памяти (2 байта при реализации БД) и легко накапливается и изменяется. Эффективность применения выявляется на показанных в следующей главе алгоритмах контроля электрических цепей в топологии, а также во многих других практических случаях.

Имея подобную информацию для каждой отдельно взятой ячейки в иерархически организованном проекте, можно легко накапливать необ-ходимые признаки с учетом привязанных ячеек, и причем на всю глубину иерархии. В ряде случаев (контроль проектных норм, "проз-рачное" рисование и т.д.) это позволит отказаться от прохода части ветвей дерева иерархии ячеек проекта.

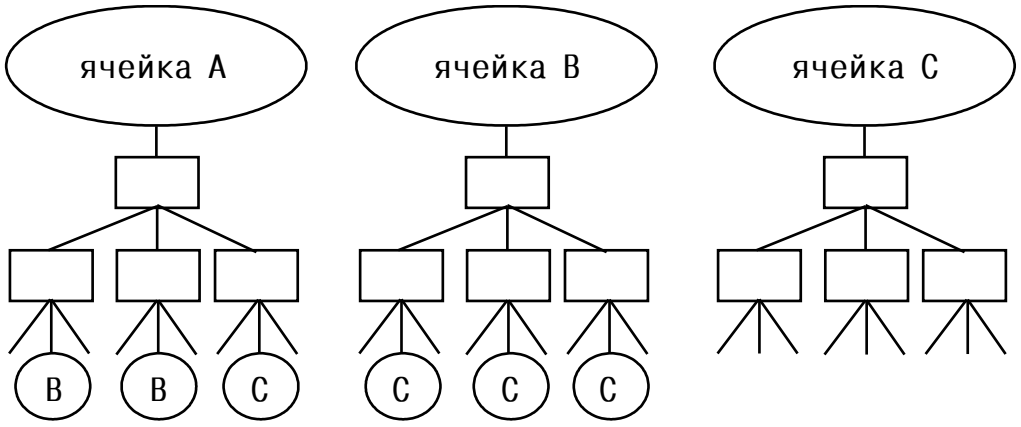


Рис.4-7

Пример иерархии ячеек без объектов в листьях деревьев

Покажем эффективность подхода на некотором гипотетическом примере, изображенном на рис.4-7. Допустим, что требуется "прозрачная" выборка всех прямоугольников, начиная с ячейки A, хотя на самом деле их нет ни в ячейке A, ни в ячейках B и C. Обычный алгоритм выборки просмотрит дерево УСД ячейки A, после чего для каждой привязки ячейки B повторит выборку. Далее, при анализе ячейки B каждый случай привязки ячейки C вызывает аналогичную выборку. Даже при отсутствии требуемых объектов обычная смена вычислительного контекста при рекурсивном повторении выборки для большого количества привязок вносит значительную и непонятную для пользователя задержку реакции программы. Этот эффект особенно заметен при обработке информации, импортируемой из других САПР, когда проектная иерархия неоптимальна для базы данных. Эксперимен-тальная проверка показала уменьшение времени до 60% при выполнении операции рисования.

В связи с тем, что информация в иерархическом проекте может динамически меняться, обобщающая информация должна обновляться перед каждой обработкой данных в БД. Обновление локальных признаков в ячейке и в дереве УСД может выполняться алгоритмами СУБД во время выполнения операций, приводящих к изменению содержи-мого БД, что сильно облегчает задачу. Поэтому рассмотрим алгоритм обновления относительно иерархии проектных единиц:

1) Определяем максимальную глубину залегания (длину пути) для каждой ячейки в иерархии. Алгоритмы вычисления широко известны в теории графов и здесь не приводятся.

2) Для каждой ячейки на максимальной глубине просматриваем все привязки и складываем их обобщенные признаки с локальными призна-ками, формируя обобщенные признаки данной ячейки.

3) Уменьшаем глубину на 1. Если отрицательное число - выход, иначе повторение п.2.

4.5. Способы трансформации координат.

Трансформация координат служит для выполнения разнообразных операций над графическими объектами баз данных, а именно: перемещение в пространстве (трансляция), вращение, зеркальное отражение осей координат, масштабирование, а также любые комбинации перечисленных элементарных преобразований. В векторной форме элементарные операции имеют вид:

$$\overline{P}_{j+1} = \overline{F} \cdot \overline{P}_j \qquad (4\text{-}12)$$

где:

$\overline{P}_{j+1}$ и $\overline{P}_j$ - точки в n-мерном пространстве,

$\overline{F}_j$ - квадратная матрица размерности n преобразования координат.

В разработанном программном обеспечении БД использовалась двумерная матрица, хотя это не являлось принципиальным ограничением для метода. Приведем вид коэффициентов матрицы для элементарных операций:

1. Вращение против часовой стрелки на угол φ:

$$\begin{aligned} f_{11} &= f_{22} = cos\varphi \\ f_{12} &= +sin\varphi \\ f_{21} &= -sin\varphi \end{aligned} \qquad (4\text{-}13)$$

2. Масштабирование или зеркальное отражение оси X $(|S_y| = 1)$ и/или масштабирование или зеркальное отражение оси Y $(|S_x| = 1)$:

$$\begin{aligned} f_{11} &= f_{12} = S_x \\ f_{21} &= f_{22} = S_y \end{aligned} \qquad (4\text{-}14)$$

Главная особенность применения трансформаций в САПР - это их относительность. Операции задаются относительно базовой точки, часто называемой точкой привязки, и приводятся к целевой точке. Вторая особенность - требуемое преобразование часто является комп-лексным,

т.е. состоит из композиции элементарных операций. Свой-ства матричных операций позволяют представить элементарные опера-ции в виде единого преобразования вида:

$$\bar{P}_{j+1} = \bar{F} \cdot (\bar{P}_j - \bar{0}) + \bar{T} \qquad (4\text{-}15)$$

где вновь введенные обозначения:

$\bar{0}$ и $\bar{T}$ - базовая и целевая точки в n-мерном пространстве.

Заметим, что (4-15) может определять различный порядок выпол-нения операций масштабирования и поворота. Для определенности по-ложим сначала зеркальное отражение и масштабирование относительно базовой точки, а затем поворот относительно базовой точки. Для большинства операций с БД такой порядок удовлетворителен, при не-обходимости его можно легко поменять, т.к. изменятся только коэффициенты матрицы.

Недостаток уравнения (4-15) в том, что требуется выполнение трех матричных операций. В литературе [50-54] приводится метод ис-пользования так называемых однородных координат, который базиру-ется на увеличении размерности уравнения на единицу путем ввода фиктивного единичного компонента в вектор координаты и вычислении дополнительных коэффициентов матрицы. Для размерности 2 уравнение (4-15) примет вид:

$$\begin{vmatrix} x \\ y \\ 1 \end{vmatrix}_{j+1} = \begin{vmatrix} +s_x \cdot cos\varphi & -s_y \cdot sin\varphi & t_x \\ +s_x \cdot sin\varphi & +s_y \cdot cos\varphi & t_y \\ 0 & 0 & 1 \end{vmatrix} \cdot \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}_j \qquad (4\text{-}16)$$

где трансляция координат выражается как:

$$\begin{aligned} t_x &= -x_o \cdot s_x \cdot cos\varphi + y_o \cdot s_y \cdot sin\varphi + x_t \\ t_y &= -x_o \cdot s_x \cdot sin\varphi - y_o \cdot s_y \cdot cos\varphi + y_t \end{aligned} \qquad (4\text{-}17)$$

Приведенное выше уравнение (4-16) выполняется за одну матрич-ную операцию, но справедливости ради надо заметить, что в таком виде вычисления эффективны только при аппаратурной, а не прог-раммной реализации. В практике программироания всегда используются только две первых строки уравнения. Кроме того, для многих прило-жений, в том числе и для разработанной БД, коэффициенты должны быть представлены в плавающем формате. Это необходимо для вычис-ления матриц обратных преобразований и для обеспечения произвольных углов поворота, когда sin и cos суть вещественные числа. Кроме того, коэффициенты в формулах (4-16) могут выйти за разрядную сетку целого числа.

Во многих САПР, особенно в мироэлектронике, координаты точек хранятся в целом виде, что обусловлено необходимостью обеспечить точность, когда даже ошибка в младшем разряде числа может означать серъезное нарушение в проекте. Поэтому вычисления по уравнению (4-16) неявно включают две операции по изменению типа числа. В качестве альтернативы для САПР электроники предложено использовать целочисленные дробные коэффициенты взамен вещественных. Такой под-ход не увеличивает объем памяти для хранения матрицы коэффициентов и основывается на следующих фактах, справедливых для большинства операций с графикой типа топологии, чертежей электрических схем и т.п.:

1) повороты ограничены углами, кратными 90 град.;

2) в случае поворотов с произвольными углами точность поворота ограничивается некоторым значением;

3) коэффициенты масштабирования невелики, а в случае рисования изображения могут корректироваться без потери качества.

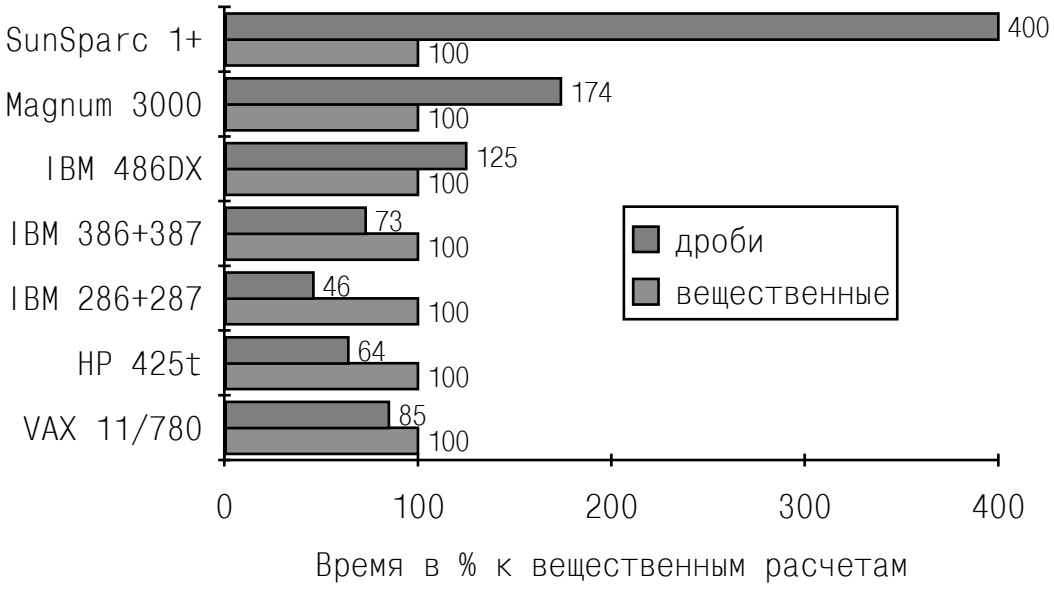


Рис.4-8

Относительная скорость вычисления трансформаций на разных ЭВМ

Применение дробей усложняет алгоритмы вычислений, т.к. особое внимание должно уделяться размещению результата в разрядной сетке целого числа. Однако для ряда ЭВМ эксперименты показали эффективность вычисления трансформаций с дробными коэффициентами, что показано на диаграмме рис.4-8. Время вычисления с плавающими коэффициентами принято за 100% для каждой ЭВМ. Текст программы приведен в приложении.

Для самых распространеных ЭВМ российского парка компьютеров - машин ряда VAX и младших моделей IBM PC/AT - вычисления дробей дают выигрыш, однако для перспективных современных ЭВМ на базе RISC-процессоров (Magnum, Iris Indigo, Sparc) вычисления следует выполнять с плавающей точкой. Полученный эффект во многом объясняется тем, что новые компьютеры в значительной степени ориентированы на обработку изображений, где трансформация координат - основная операция. Более того, микропроцессоры этих машин выполняют преобразование типов целое-вещественное и вещественное-целое аппаратно одной командой, что дополнительно говорит в пользу применения вещественных коэффициентов. В разработанной СУБД ис-

пользуется механизм макроопределений для включения типов данных и операций, дающих максимальный эффект.

4.6. Методы контроля проектных норм топологии в иерархической БД.

Проектирование топологии СБИС, печатных плат и микросборок включает не только строго детерминированные автоматические этапы, но и работы, выполняемые разработчиком вручную. В связи с этим возможно появление ошибок, которые могут быть не замечены при большом объеме информации. Необходимость строгого контроля обус-ловлена дороговизной производства фотошаблонов, пластин и опытных образцов. Для СБИС длительность производства кристаллов может превосходить время проектирования. Методы контроля рисунка фото-шаблонов для одноуровневого представления достаточно хорошо разра-ботаны [87-93]. Существующие алгоритмы решения этой задачи выполняются достаточно длительное время для больших проектов, что исчисляется часами работы ЭВМ. Для ускорения проверки используются методы простейшего фрагментирования и проверки по частям, инкре-ментальной проверки и другие. В данной главе будет показано, как аналогичные средства могут быть реализованы на базе разработанной СУБД [80].

Для ускорения контроля в иерархически организованных проектах современные маршруты проектирования вводят жесткие ограничения, например, рисунок топологии данного уровня не должен переходить за границы привязанных ячеек, что позволяет выполнять проверку без развертывания нижних уровней. Такой подход позволяет упростить и ускорить проверку за счет неполного использования площади крис-талла. В настоящей работе предложены методы контроля проектных норм на базе разработанной БД, в основе которых лежит метод "проз-рачной" выборки объектов. Преимущества такого подхода, во многом обеспеченные возможностями СУБД, заключаются в следующем:

1) экономия памяти за счет использования БД "как есть", без развертывания информации в одноуровневое представление;

2) быстрый поиск соседних фигур, основанный на пространствен-но-отсортированном хранении фигур в БД, обеспечивающем быстрый поиск;

3) унификация операций контроля за счет приведения всех фигур к типу многоугольника позволяет обойтись одним алгоритмом провер-ки;

4) диалоговый или автоматический режим проверки с графической визуализацией результатов.

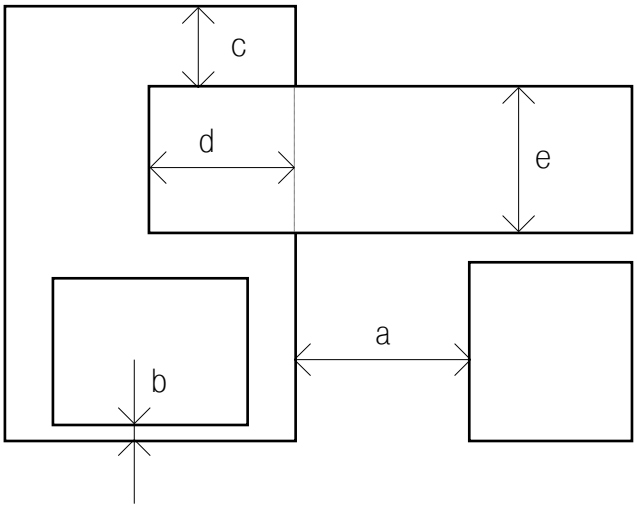


Рис. 4-9

Типичные геометрические соотношения фигур

Рассмотренные в литературе методы контроля топологии включают метод сканирования (растровый метод), метод полос, матриц провер-ки, метод объединяющих многоугольников и ряд других [87,89]. Спи-сок проверяемых правил в основном состоит из анализа следующих геометрических соотношений фигур (рис.4-9): внешний (a) и внутрен-ний (b,c) зазор, наложение (d), минимальный размер (e) и определе-ния факта пересечения ребер.

Наиболее часто упоминаемый метод для вычисления зазоров и пересечений заключается в коррекции размеров на величину конт-рольного значения и проверке на пересечение контура с соседними фигурами. Следует заметить, что для многоугольников с произвольным

наклоном сторон коррекция размеров не является простой задачей, т.к. изменение размеров должно выполняться строго по нормали к ребрам фигуры. Учитывая дискретность координат, можно ожидать дополнительной погрешности расчетов, обусловленной округлением положения концов ребер до единичного значения. Для СБИС это может быть источником ложных ошибок.

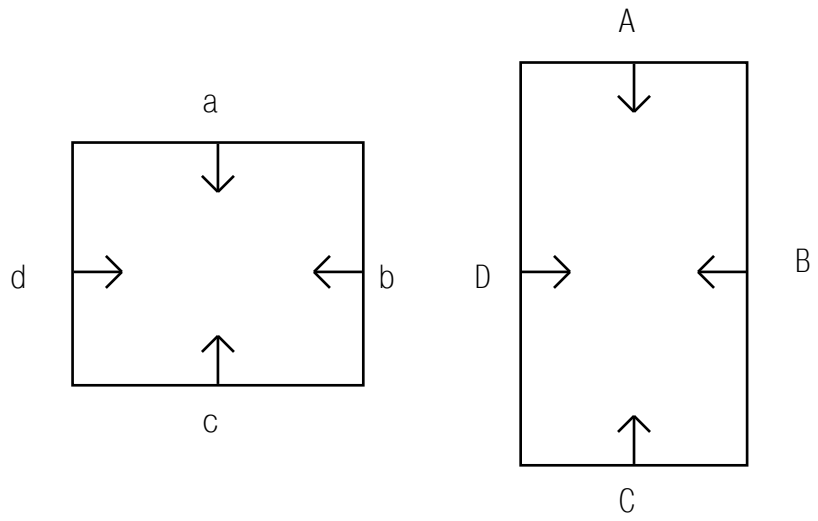


Рис.4-10

Геометрические соотношения ребер фигур с учетом нормалей

Если обратить внимание на результирующее отношение двух произвольных непересекающихся контуров, можно заметить, что этот результат определяется соотношением пары наименее удаленных ребер. Рассмотрим пример на рис. 4-10, на котором для простоты изображены два прямоугольника. Существенные комбинации всех ребер и их тип следующие:

1) [d,B] - наложение;

2) [d,D] и [b,B] - внутренний зазор;

3) [b,D] - внутренний зазор.

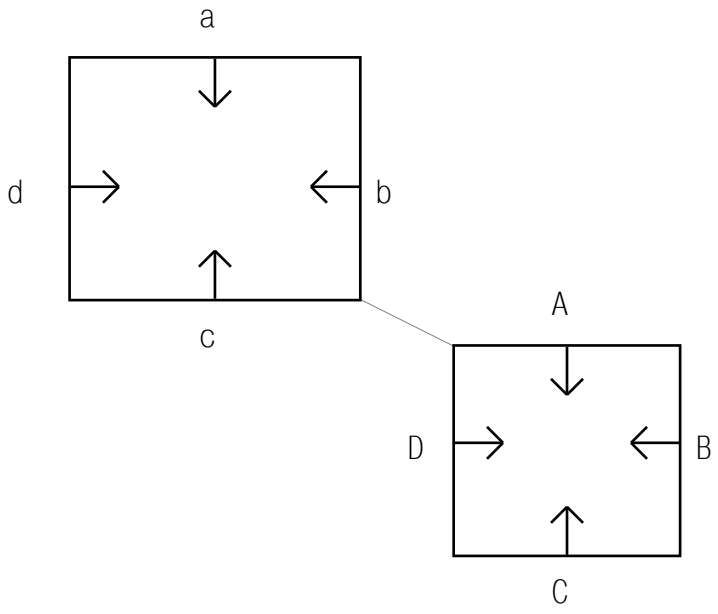


Рис.4-11

Определение расстояния при отсутствии соотношений ребер

Из рассмотрения исключены сочетания ребер, которые не дают хотя бы одной нормальной проекции крайних точек на другое ребро. Такими являются (рис.4-10) ребра a, c, A, и C. Из существенных размеров минимальное значение дает пара [b,D], что и следовало ожидать.

В тех случаях, когда ни одна пара ребер не дает существенное значение расстояния, результат есть внешний зазор, определяемый как расстояние между двумя ближайшими точками излома контуров фигур (рис.4-11).

В случае пересечения контуров проверяемых фигур (рис.4-12) результирующее отношение неоднозначно, т.к. не определяется минимальным существенным отношением всех ребер. Однако это не является недостатком метода, т.к. в практике проверки проектных норм (например, в технологии КМОП) требуются все определенные значения.

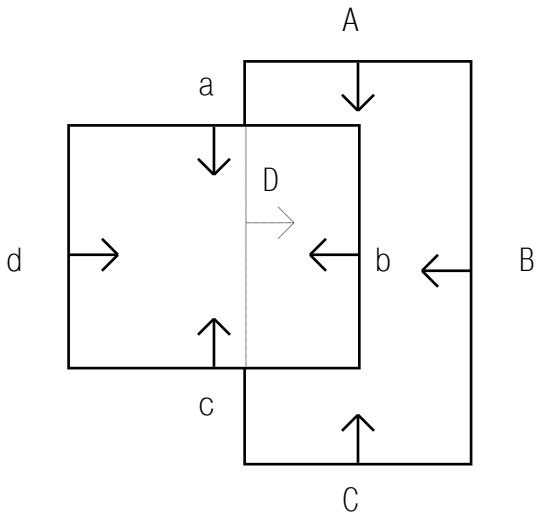
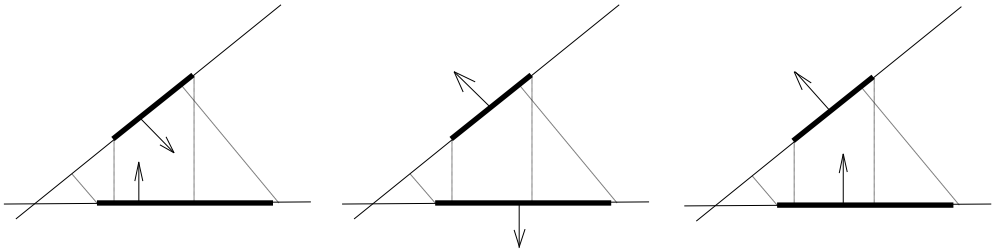


Рис.4-12

Геометрические соотношения ребер фигур при пересечении контуров

На основании изложенного подхода для проверки используется вычисление отношений между ребрами контуров, базирующееся на вычислении следующих основных геометрических соотношений (рис. 4-13):



а) наложение б) внешний зазор в) внутр. зазоры

Рис.4-13

Типы соотношений ребер

1. Расстояние между ребрами, нормали которых направлены навстречу друг другу, вычисляется как минимальное расстояние от концов ребер по перпендикуляру к другому ребру (рис.4-13а). Размер считается недействительным, если основание перпендикуляра лежит за пределами крайних точек ребра.

2. Зазор между ребрами, нормали которых направлены в противоположную друг другу сторону, вычисляется как минимальное расс-

тояние от концов ребер по перпендикуляру к другому ребру (рис.4-13б). Размер считается недействительным, если основание перпенди-куляра лежит за пределами крайних точек ребра.

3. Зазор покрытия или вложения между ребрами, нормали которых направлены в одну сторону, вычисляется как минимальное расстояние от концов ребер по перпендикуляру к другому ребру (рис.4-13в). Размер считается недействительным, если основание перпендикуляра лежит за пределами крайних точек ребра.

Для вычисления расстояния между точкой и ее нормальной проек-цией на ребро воспользуемся расчетами и формулами из [50,96], где используется уравнение общего вида первого порядка:

$$A \cdot x + B \cdot y + C = 0 \quad (4\text{-}18)$$

Удобство этой формулы в легкости определения коэффициентов и в том, что она позволяет задавать прямые под любым углом наклона. Для определения расстояния приведем (4-18) в нормальную форму, для чего разделим все коэффициенты на $\sqrt{A^2 + B^2}$. Коэффициент C становится расстоянием по нормали к началу координат (0,0), а два других коэффициента становятся суть cos и sin угла наклона:

$$a \cdot x + b \cdot y + c = 0 \quad (4\text{-}19)$$

Проведем через искомую точку (x_0, y_0) прямую, параллельную ребру. Коэффициенты A и B этой прямой совпадают к коэффициентами прямой, проходящей по ребру. Коэффициент C определяется по урав-нению (4-19):

$$c_o = -(a \cdot x_o + b_o \cdot y) \quad (4\text{-}20)$$

Определим искомое расстояние как модуль разности расстояний от обеих прямых до начала координат, выражаемый через коэффициенты нормального уравнения прямой:

$$d = |c - c_0| = |a \cdot x_0 + b \cdot y_0 + c| = \frac{|A \cdot x_0 + B \cdot y_0 + C|}{\sqrt{A^2 + B^2}} \qquad (4\text{-}21)$$

где коэффициенты вычисляются по координатам ребра [96]:

$$A = y_1 - y_2$$
$$B = x_2 - x_1 \qquad (4\text{-}22)$$
$$C = -(Ax_1 + By_1) = -(Ax_2 + By_2)$$

Эти коэффициенты могут быть уменьшены путем деления на наибольший общий делитель A и B. Для ребер (прямых), параллельных осям координат или для ребер с углом наклона, кратным 45 град., эти коэффициенты примут значения из множества { -1, 0, +1 }. Для целочисленных вычислений с произвольно наклоненными ребрами нормирование коэффициентов должно предотвратить переполнение разрядной сетки при вычислениях.

Эффективность подхода во многом обусловлена высокой скоростью локальной выборки данных из БД. В качестве прямоугольной области поиска соседних фигур используется охватывающий прямоугольник исходной фигуры, расширенный на величину контрольного значения. Вычисление координат самого прямоугольника для любых типов фигур не требуется, т.к. он хранится в БД и при иерархической выборке подвергается только однократной трансформации.

Как показали практические испытания, метод хорошо работает для локальных фигур. На основании проведенных тестов (табл.4-2) можно сделать вывод о скорости выполнения проверки порядка O(N), где N - количество проверяемых контуров. В таблице представлены результаты выполнения контроля зазоров для равномерно расположенного массива прямоугольников. В среднем для каждого прямоугольника выполнялась

проверка с восемью "соседями". Проверка проводилась на рабочей станции Magnum-3000.

Таблица 4-2

N	100	1000	10000	100000
время,с	~1	9	94	940

В качестве другого теста выполнялся контроль 50 правил для аналого-цифровой схемы площадью 3 кв. мм (3 мкм проектные нормы, КМОП), на который было затрачено 5 мин. на ЭВМ IBM PC/AT-486. Для протяженных контуров, когда охватывающий прямоугольник становится большим, предпочтительно изменить порядок анализа соотношений ребер. Вместо проверки пары фигур можно выполнять проверку соседей для каждого ребра, что для ненаклоненных ребер уменьшит область поиска по одной из координат до значения контрольной величины, по другой координате - до длины ребра.

Использование электрического контекста [63] позволяет выпол-нять проверку на короткие замыкания простым и эффективным спосо-бом. В отличие от специальных алгоритмов [95], для контроля используется обычное правило минимального внешнего зазора, а до-полнительным условием для отбора соседней фигуры служит различие идентификаторов электрических узлов. Метод успешно работает для иерархической топологии, используя возможности СУБД по вычислению истинного электрического узла конкретной привязки.

Кроме диалогового задания области и правил для локальной проверки, полный контроль может выполняться на основе файла с заранее записанными правилами. Формат правил и их типы для редактора Layout Windows [97] приведены в приложении.

4.7. Выводы.

Особенности иерархического способа хранения и обработки инфор-мации в САПР электронных изделий принижают преимущества известных

схем пространственной сортировки. Использование быстродействующей оперативной базы данных на принципах модифицированного MxM-дерева позволило создать эффективные и компактные структуры, требующие меньшего количества памяти при сохранении скоростных качеств СУБД. Ускорение выборки достигнуто путем аккумулирования и обработки обобщенных признаков объектов и методом трансформации координат с применением дробей. Высокая скорость СУБД (получен порядок O(N) при контроле проектных норм) позволяет использовать алгоритмы, традиционно требующие значительного времени для выполнения.

Адаптивность СУБД к пространственному расположению фигур обеспечивается как традиционными свойствами квадро-деревьев, так и списковой организацией управляющих структур, что позволяет менять ключевые параметры БД без изменения алгоритмов управления. Использование ряда новых методов построения дерева позволяет решить ряд проблем, свойственных квадро-деревьям с множественным хранением объектов.

Вычисление трансформации координат средствами СУБД существенно упрощает прикладные алгоритмы, а аппарат применения внешней функ-ции обеспечивает гибкость обработки информации. Метод псевдорекурсивного обхода дерева с использованием стека указателей позволяет экономить память системного стека, необходимую для рекурсивной обработки иерархии проектных единиц (ячеек).

ГЛАВА 5

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АДАПТИВНОЙ ЗАКАЗНОЙ САПР: ГРАФИЧЕСКИЙ РЕДАКТОР LAYOUT WINDOWS

5.1. Структура и взаимодействие компонент графического редактора.

Графический редактор Layout Windows включает в себя реализацию всех представленных выше концепций, методов и алгоритмов. По выполняемым функциям он относится к редакторам широкого применения и может использоваться в микроэлектронике, дискретной электронике, машиностроении, архитектуре и т.д. Однако ряд специфических воз-можностей типа: контроль проектных норм, контроль электрических соединений, установка элементов и прокладка трасс соединений специализируют редактор в области электроники. Использование Н-интерфейса обеспечивает 100% переносимость на любую вычислительную и сетевую среду. Оконный интерфейс служит для гибкой настройки и управления окнами редактора, что позволяет легко адаптировать его на решаемую задачу. В итоге реализуется концепция открытого для пользователя инструментального средства.

Редактор Layout Windows написан на языке С и содержит около 25000 строк исходного текста. Структура редактора приведена на рис. 5-1. Ядром служат функции обслуживания событий в графических окнах. В основе функционирования лежит предположение об асинх-ронном наступлении событий, определяемом пользователем. Но в связи с тем, что большинство действий по вводу и редактированию требуют строго определенной последовательности шагов, сами операции выпол-няются в

монопольном режиме: у оператора есть возможность продол-жить, закончить или прервать текущую операцию, прежде чем начать следующую. При этом, однако, остается полная свобода действий по управлению оконным интерфейсом, т.к. они не нарушают порядок собы-тий, видимый функцией обработки событий. Оконный интерфейс, а именно его функции главного цикла и запроса события, содержит фильтр "своих" событий, что разделяет редактирование топологии окон и прикладные операции, делая их полностью независимыми.

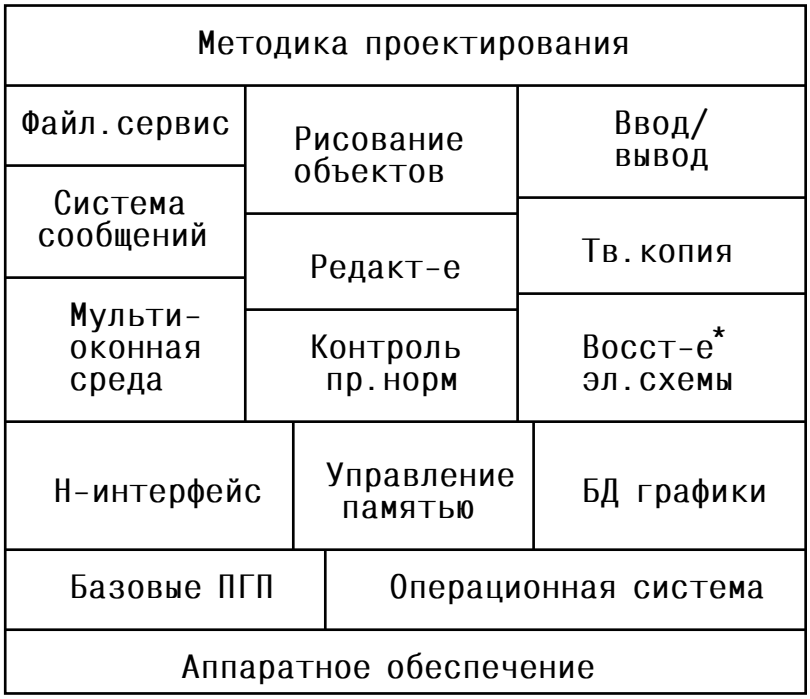


Рис.5-1

Структура программного обеспечения редактора Layout Windows

Редактор использует описанную выше СУБД многослойного типа, что обеспечивает хранение и обработку элементов одного класса в одном "срезе" базы данных. Это, во-первых, упростило и ускорило ряд алгоритмов, в частности рисование. Во-вторых, многослойная структура обеспечивает наиболее естественную селекцию слоев по признакам "активный" - "видимый" - "все слои", которые организу-ются вне БД. В качестве параметров базы данных используются размеры минимальной ячейки УСД и коэффициент размножения ячеек, одинаковый по обеим осям

координат. Изменение параметров доступно пользователю, что позволяет получить эффективное сочетание затрат памяти на БД и скорости локальной выборки в каждом конкретном случае.

Подсистема контроля проектных норм функционирует в ручном или автоматическом режиме. В первом случае правило контроля, численные параметры и слои для проверки задаются пользователем интерактивно, во втором случае используется внешний файл правил с заранее заданными нормами. В любом случае контроль выполняется либо в локальной выбранной области, либо глобально по всей топологии. Кроме того, имеется возможность ограничиться только текущим уровнем иерархии или задать "прозрачную" выборку примитивов на всю глубину. Перечисленные режимы могут комбинироваться в любом сочетании для достижения поставленных целей. Проверка выполняется путем полного перебора примитивов, входящих в область проверки, однако соседний примитив ищется исключительно в локальной области габаритов первого, что существенно сокращает время выполнения операции контроля.

Подсистема восстановления / идентификации электрических узлов выполняет три задачи. Первая обусловлена необходимостью контроля импортированной топологии, для которой следует выполнить восстановление электрической схемы. Часть этапов выполняется автоматически, некоторые требуют вмешательства оператора. Следующая операция реализует механизм подключения, или наследования, идентификаторов цепей, что необходимо при ручной прокладке соединений. Для этого в заданной точке выполняется поиск примитива или элемента с ранее назначенными цепями (узлами). Эта цепь наследуется вновь создаваемой трассой. И последняя решаемая задача - это модификация электрического контекста при выборке примитивов из нижних уровней иерархии. В ее основе лежит переназначение локальных электрических цепей образца ячейки на действительные цепи согласно списка подключения в привязке ячейки (элемента). Эта задача является ключевой при большинстве операций "прозрачной" выборки.

Для автоматического поиска файлов в большинстве ОС использу-ют-ся поименованные переменные типа строки символов, в которых содержится список каталогов для поиска. Сложности обслуживания этих переменных и ограничения систем типа MS-DOS (длина строки не более 128 символов и т.д.), поставили задачу разработки анало-гичного сервиса, управляемого программой.

Автоматическая защита файлов основана на запрете записи (моди-фикации) для файлов, находящихся в любом каталоге, отличном от текущего (рабочего). Для получения доступа по чтению (неразруша-ющий доступ) используется полная спецификация файла, включающая идентификатор устройства, путь по иерархии каталогов и имя файла. При отсутствии первых двух поиск производится по установленному списку каталогов. Если запрошен доступ по записи, в спецификации файла оставляется только его имя, что автоматически приведет к записи файла в текущий каталог. Такой метод позволяет сохранять в неизменном виде файлы общего пользования. Изменяя и записывая такой файл, образуется локальная копия, специфичная для данного проекта. Естественно, для разделения проектов необходимо создавать и использовать для них отдельные каталоги, однако эти возможности есть во всех современных ОС.

Разработанный набор программ в качестве дополнительного сервиса переопределяет стандартные функции языка Си open() и fopen(), а также предоставляет ряд сервисных функций для системно-независимых операций со спецификацией файла. Используются возмож-ности большинства реализаций языка С, позволяющие задавать специ-фикацию файла в UNIX-подобном формате, независимо от типа ОС. В результате обеспечивается системно-независимый уровень взаимодей-ствия, расширенный дополнительным поиском файлов. Одним из прило-жений этой подсистемы является реализация простого механизма по-иска и открытия файлов инициализации при старте редактора.

Во время сеанса работы пользователь может оперативно изменить список каталогов для поиска файлов с данными, что удобно при стар-те программы, когда недостающие в текущем каталоге файлы извлека-ются из каталога, где находится сам редактор. Имеется возможность задать уникальный список каталогов при вводе файлов. Этот режим удобен для загрузки файлов топологии из библиотек.

Подсистема памяти, как она описана выше, в основном исполь-зуется для операционной системы типа MS-DOS и оболочки MS Windows. Подсистемой переопределяются стандартные функции malloc() и free() согласно моделям памяти программы. Кроме того, обеспечиваются счетчики для учета выделенной и освобожденной памяти. Для ряда платформ предоставляется отсутствующая в библиотеках функция memcpy().

Оконный интерфейс, описанный ранее, использует запись и чтение конфигурации окон в файле, что позволяет сохранять расположение и прочие параметры окон для следующего сеанса.

Окно топологии служит для послойного отображения рисунка слоев, порядок которых и образец рисования заданы в таблице слоев. Использование полупрозрачных и контурных образцов позволяет видеть нижележащие слои без изменения окраски, что выгодно отличает редактор от программ с хаотичным порядком слоев при рисовании. Картинка в окне дополняется координатной сеткой заданного по осям шага, причем сетка накладывается либо сверху, либо снизу изобра-жения. Независимо от сетки обеспечивается округление координат съема, что позволяет выводить более крупную сетку, не затеняя изображения.

Глубина развертывания иерархии при рисовании варьируется от 0 (только текущий уровень) до 9 или выводится вся иерархия на полную глубину. Для удобства редактирования печатных плат введены режимы "вида сзади" с инверсией изображения по оси X или Y, что позволяет работать естественным образом с обратной стороной. В заголовке окна

выводится имя файла, флаг изменений, шаг сетки и округления, а также прочие индикаторы.

Окно таблицы слоев служит для отображения состояния и нагляд-ного и быстрого управления. В таблице отображаются образцы рисова-ния видимых слоев и индикаторы активности. Прочая информация может быть представлена в четырех вариантах, в зависимости от пожеланий пользователя: 1) номер слоя и его обозначение, 2) только номер слоя, 3) только обозначение слоя и 4) без номера и обозначения. Наличие образца рисования позволяет легко и быстро изменить цвет, заливку и штриховку слоя, а также поменять порядок слоев. Для оперативного управления таблицей обычное указание слоя в таблице меняет его активность или видимость. Таблица при необходимости может быть сохранена в файле для последующей загрузки в данном состоянии.

Окно координат автоматически отображает прикладные координаты точки съема, абсолютное приращение от предыдущей точки и знаковые приращения по осям координат.

Функции создания и редактирования графических примитивов явля-ются мощным интерактивным средством, расширяющим возможности СУБД. Использованные алгоритмы используют принцип инвариантности окон, т.е. допустима операция между двумя любыми окнами топологии. А именно: операция может быть начата в одном окне, продолжена или завершена в других. Это позволяет равноценно использовать как общий план, так и увеличенное изображение в нескольких окнах. Для ввода примитивов есть одно ограничение: окна должны отображать один и тот же объект (файл) базы данных.

БД поддерживает только 5 типов примитивов: прямоугольник, многоугольник, шина постоянной ширины, линия ширины 0 и привязка ячейки. Дополнительно можно ввести правильный многогранник, коль-ца, дуги и текст. Для текста используется набор файлов, созданных в самом редакторе, с изображением символов фиксированного размера. Имена файлов фиксированы и соответствуют шрифту H-интерфейса.

Все вводимые примитивы могут дополняться атрибутом - текстовой строкой произвольного содержания. Данная версия Layout Windows ориентирована на обработку атрибутов как признаков выводов элементов и идентификаторов электрических цепей, что специализирует редактор на проектировании электроники. Замена модулей обработки атрибутов может поменять специализацию редактора на другую область. В редакторе синтаксис атрибутов выбран таким образом, чтобы они подчинялись стандартным правилам расширения формата CIF версии 2.0:

1) Для примитивов класса "вывод (порт) элемента":

6 <номер вывода> <обозначение вывода>;

5 <обозначение локального узла>;

2) Для привязки класса "элемент":

5 <обозначение> <обозначение узла вывода 1> ...

<обозначение узла вывода N>;

3) Для примитивов класса "соединение":

5 <обозначение локального узла>;

Кроме признаков, все вводимые примитивы могут обладать свойствами мультипликации для описания регулярных областей. Определяющие параметры состоят из пары векторов направления и коэффици-ентов повторения (рис.5-2). Это позволяет задавать косоугольные массивы элементов и адекватно трансформировать массивы при поворотах, не кратных 90 градусам.

Создание примитивов выполняется либо сразу во всех слоях, либо только в видимых, либо только в активных. Используемые слои задаются пользователем посредством меню.

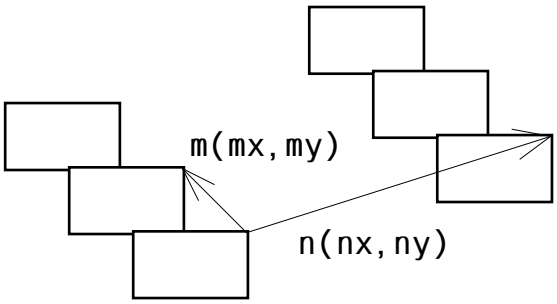


Рис.5-2

Параметры произвольной мультипликации

Редактирование подразделяется на три типа операций: трансформация координат, изменение свойств и изменение атрибутов. Отбор примитивов выполняется по двум независимым критериям: 1) активные - видимые - все слои и 2) в точке - в прямоугольной области - все координатное пространство - только помеченные фигуры. Установка требуемого режима позволяет выполнять как локальные, так и глобальные операции единым образом. Отобранные фигуры обводятся по контуру, чтобы можно было убедиться в правильности отбора и при необходимости отменить операцию до ее выполнения.

Трансформация координат включает операции сдвига, копии, деформации и удаления примитивов. За исключением последней, все они могут быть функционально усилены для более сложных действий: зеркальное отражение осей координат, поворот, масштабирование и коррекция размеров на постоянную величину. Результат трансформации остается в исходных слоях, либо перемещается в заданный пользователем слой. Таким образом можно выполнять как обычные простые операции, так и сложные, которые в других программах требуют нескольких шагов.

Изменение свойств позволяет изменять (задавать вновь) пара-метры мультипликации, развертывать мультипликацию до отдельных примитивов, а также заменять привязку содержимым ячейки. Возможна модификация шины и многоугольника. Изменение атрибутов заключается в ручном редактировании строки атрибутов.

Редактор достаточно легко реализует отмену только что выполненной операции по созданию или корректировке примитивов. В ее основе лежит свойство БД помечать специальным флагом все вновь вводимые или удаляемые примитивы и откладывать действительное удаление до специального вызова СУБД. Таким образом, обратная замена флагов удаленных и созданных примитивов фактически означает возвращение базы данных в исходное состояние. Конечно, при этом расходуется дополнительная память ЭВМ, но эти затраты соответ-ствуют риску потери примитивов при ошибочном выполнении операций для ответственных данных.

Файловая подсистема редактора Layout Windows построена по модульному типу, предоставляя возможности легкого добавления или замены программного кода по вводу / выводу прикладной информации. В настоящее время поддерживаются несколько диалектов формата SOURCE, формат CIF в версии 2.0 и в модифицированном варианте для записи каждой ячейки в отдельный файл. Выбор штатных форматов был обусловлен (но не ограничен) необходимостью сопряжения как с отечественными, так и с зарубежными САПР. Возможности подсистемы позволяют провести выборочную селекцию примитивов аналогично операциям по редактированию, а также задать масштаб при записи, развернуть (раскрыть) иерархию и преобразовать шины и прямоуголь-ники в многоугольники. В режиме подтверждения для записываемых файлов можно задать другую спецификацию файла. При вводе обычно автоматически вводится вся иерархия ячеек, если не установлена глубина чтения иерархии.

В связи с тем, что БД выступает в роли промежуточного "формата", редактор позволяет естественным образом выполнить преобразование форматов данных. Например, набор исходных двоичных файлов в формате SOURCE может быть выведен в виде одного текстового файла в формате CIF 2.0 и обратно.

Для комплексной оценки редактора было проведено измерение времени локальной и глобальной прорисовки одноуровневой и иерархически организованной топологии. Сравнение с программой Chipgraph версии 7 (фирма Mentor Graphics) на ЭВМ HP425t дало одинаковые результаты (с учетом погрешности измерений). Результаты аналогичных испытаний для программ Layout Windows, KIC и "Пульт" версии 2.04 на ЭВМ IBM PC/AT-386 приведены в табл.5-1.

Таблица 5-1

Программа	лок.из иерархии	лок.из одноур.	глоб.из одноур.
Layout Windows	6	4	21
KIC	178	10	24
Пульт 2.04	23	14	22

Примерное равенство скоростей глобальной одноуровневой прорисовки доказывает, что организация дерева УСД в виде списков и использование буферных графических программ обеспечивает такую же высокую скорость, как и альтернативные способы. Лучшие результаты для локальной прорисовки демонстрируют преимущества СУБД.

5.2. Использование графического редактора в системах автоматизированного проектирования.

Многофункциональность редактора предполагает его широкое применение в типовом маршруте проектирования СБИС. Возможности идентификации и экстракции электрической схемы позволяют применять его как простой редактор электрических схем. При разработке топологии матричных СБИС автоматическими или полуавтоматическими методами с помощью редактора можно выполнять ручную корректировку размещения элементов и прокладку трасс. При разработке заказных СБИС ручным способом редактор позволяет выполнить бездефектное рисование / редакторование топологических слоев в режиме контроля проектных норм. Этот контроль может быть выполнен над всем проектом без приведения

иерархии в одноуровневое представление. Использование различных форматов данных при вводе и выводе топологии позволяет выполнить преобразование форматов с промежуточным визуальным конт-ролем.

Использование редактора в в/ч 11135 в составе системы проек-тирования полузаказных СБИС "Аист" позволило реализовать систему полуавтоматической трассировки кристалла СБИС, включающую, в частности, программу автоматической трассировки и редактор "Layout Windows". С помощью редактора формируется задание области локаль-ной или глобальной трассировки. Результаты последующей автома-тической трассировки вставляются в "вырезанное" окно области трас-сировки. После корректировки проводников цикл проектирования может быть повторен.

В НИИ "Научный Центр" и РНИИТМЭ редактор используется для ручного проектирования топологии коммутационных плат. СП BSD/ Silicon применяет редактор для контроля проектных норм и преобра-зования форматов топологической информации между отечественными и зарубежными САПР.

Использование редактора на указанных предприятиях подтверждено актами о внедрении, приведенными в приложении.

5.3.Выводы.

Разработка графического редактора топологии "Layout Windows" преследовала 2 цели: практически опробовать методы создания заказ-ных САПР с помощью открытой адаптивной оболочки и создать мощное инструментальное средство редактирования топологии. Подводя итог теоретической и практической работы, можно уверенно утверждать о том, что обе поставленные цели достигнуты.

Использование стабильных интерфейсов адаптивной оболочки поз-волило вести параллельную разработку подпрограмм на разнородной вычислительной технике. Перенос редактора в новые системы выпол-нялся в кратчайшие сроки, что было обусловлено отсутствием принци-

пиальных изменений в исходных текстах прикладных программ. Редак-тор Layout Windows является мощным интерактивным средством инже-нера-тополога, которое успешно функционирует в различных вычисли-тельных средах. Успешное внедрение в промышленности является этому твердым доказательством.

ЗАКЛЮЧЕНИЕ

К основным результатам, полученным в данной работе, необходимо отнести следующее:

1. Обоснована эффективность структуры заказной САПР на базе открытых адаптивных системно-независимых программных интерфейсов, обеспечивающая высокую мобильность прикладных программ. Показано, что использование буферного программного обеспечения позволяет сократить трудоемкость и время разработки заказных САПР. В работе структурно выделены компоненты оболочки ЗакСАПР и выполнена их реализация для современных вычислительных систем и сред: ОС UNIX, VMS, MS-DOS и Windows (NT).

2. На основе метода последовательного конструирования выпол-нено определение оптимального функционального состава и разрабо-тана спецификация интерактивно-графического интерфейса. Предложены инвариантные к вычислительным средам структуры данных ресурсов, что позволило уменьшить количество функций управления интерфей-сом. Выполнена реализация для большинства типов современных ЭВМ.

3. Предложены алгоритмы и схема передачи управления в прило-жении Windows, которые реализуют интерактивно-графический интер-фейс и позволяют использовать стандартное построение прикладных программ. В результате появилась возможность переноса программ на базе Н-интерфейса в среду Windows(NT) без изменения исходного текста, что принципиально невозможно для других программ.

4. Предложена обобщенная модель динамической памяти, описы-вающая отношения управления между прикладной программой и различ-ными вычислительными средами. На основе модели разработаны специ-фикация и функциональный состав системно-независимого управления

динамической памятью и выполнена его реализация для UNIX, MS-DOS (EMS/XMS) и Microsoft Windows. Это позволяет единым образом программировать использование динамической памяти при полной гарантии переносимости на уровне исходных текстов.

5. Предложена оригинальная структура компактной графической базы данных, ориентированная на обработку иерархически организо-ванной информации. В отличие от известных схем, данная структура обеспечивает экономию памяти для пространственной сортировки объектов при сохранении высокой скорости локальной и глобальной выборки, свойственной квадро-деревьям. Это позволяет реализовать эффективную СУБД и прикладные программы на ее основе для ЭВМ класса IBM PC/AT.

6. Предложены методы аккумулирования и использования обобщен-ной информации о графических объектах, ускоряющие выборку данных из иерархической БД. Их использование особенно эффективно для несбалансированных иерархий данных, когда с помощью разработанных методов удается избежать излишнего обхода деревьев и получить почти двукратное ускорение выполнения операций.

7. Разработаны оригинальные алгоритмы и программы контроля проектных норм топологии с использованием разработанной СУБД, не требующие развертывания иерархии в одноуровневое представление. Применение СУБД позволяет получить экспериментально подтвержденную линейную зависимость времени контроля от количества контуров.

8. Разработанные методы, алгоритмы и интерфейсы применены в графическом редакторе топологии Layout Windows, используемом в составе САПР СБИС, микросборок и печатных плат.

Практическая ценность выполненной работы и разработанного программного обеспечения подтверждена внедрением в промышленности.

ЛИТЕРАТУРА

1. Соколов А.Г. Принципы построения математического обеспече-ния гибких САПР электронных схем: Диссертация ... ученой степени доктора технич. наук: 05.13.12 / МИЭТ. - М.,1989.

2. Казеннов Г.Г., Соколов А.Г. Эволюция САПР БИС. Тр. Всес. конф. АН СССР и Минвуза СССР "Современные вопросы информатики, вычислительной техники и автоматики", М., апр. 1985.

3. Казеннов Г.Г. Структура, основные требования и принципы построения САПР микроэлектронных приборов. М., Высшая школа, 1978.

4. Норенков И.П., Маничев В.Б. Системы автоматизированного проектирования электронной и вычислительной аппаратуры. М., Высшая школа, 1983.

5. Петренко А.И., Семенков О.И. Основы построения систем авто-матизированного проектирования. Киев, Висша школа, 1985.

6. Казеннов Г.Г., Баталов Б.В., Беляков Ю.Н. Соколов А.Г. и др. Машинное проектирование интегральных полупроводниковых схем. М., ЦНИИ "Электроника", Вып.5(71), 1972.

7. Ильин В.П. Основы автоматизации схемотехнического проекти-рования. М., Энергия, 1979.

8. Соколов А.Г., Бравов В.В., Иващенко О.И., Перминов В.Н. и др. Гибкие системы автоматизированного схемотехнического проекти-рования. Электронная промышленность, №4, 1987.

9. Казеннов Г.Г., Соколов А.Г. Принципы построения САПР и АСТПП. М., Высшая школа, 1989.

10. Бравов В.В. Исследование и разработка алгоритмов управле-ния проблемными модулями и обработки информации в интерактивных системах

проектирования фрагментов СБИС. Диссертация ... ученой степени канд. технич. наук. - М., МИЭТ, 1986.

11. Дал У., Дейкстра Э. Структурное программирование. Пер. с англ. М., Мир, 1985.

12. Дж.Хьюз, Дж.Мичтом, Структурный подход к программированию. Пер. с англ. М., Мир, 1980.

13. Корячко В.П., Курейчик В.М., Норенков И.П. Теоретические основы САПР. М.,Энергоатомиздат, 1987.

14. Tom Williams, "Software Tools Integrate the Management of Complex Design Projects", Comuter Design, vol.30, no.11, 1991.

15. K.R. Dittrich, A.M. Kotz and J.A. Mulle, "Database Support for VLSI Design: The Damascus Approach", Proc. CompEuro87, 1987.

16. A.H.V. de Lima, R.C.B. Martins, R. Stern and L.M.F. Cameiro, "GARDEN - An Integrated and evolving environment for ULSI / VLSI CAD Applications", IBM Systems Journal, no.4, 1989.

17. P. van der Wolf, P. Bingley and P. Dewilde, "On the Architecture of a CAD Framework: The NELSIS Approach", Proc. 1st Europ. DAC, 1990.

18. Steven E. Schulz, "Frameworks: Debunking the Myths". Electronic Design, no.16, 1991.

19. Debbie Lienhart, "Frameworks and the CAD Framework Initiative". В сборнике "Engineering Data Management: The Technology for Integration" - proc. of the 1990 ASME Int. Computers in Engi-neering Conf.

20. N. van der Mejs, T.G.R. van Leuken, P. van der Wolf, I. Widya and P. Dewilde, "A Data Management Interface to Facilitate CAD/IC Software Exchanges", Proc. Int. Conf. on Comp. Design, 1987.

21. James Martin, "Computer Data-Base Organization", Second ed., Prentice-Hall, 1977. Им. перев. Дж. Мартин "Организация баз данных в вычислительных системах", М.,Мир,1980.

22. P. van der Wolf and T.G.R. van Leuken, "Object Type Oriented Data Modelling for VLSI Data Management", Proc. 25th DAC, 1988.

23. Software Portability: An Advanced Course, Cambridge University Press, 1977. Им. перевод: Мобильность программного обеспечения. М., Мир, 1980.

24. Robert L. Glass, Software Reliability Guidebook, Prentice-Hall, 1979. Им. перевод: Р. Гласс, Руководство по надежному программированию. М., Финансы и статистика, 1982.

25. Е.А. Бутаков, Методы создания качественного программного обеспечения ЭВМ. М., Энергоатомиздат, 1984.

26. David King, Creating Effective Software. Computer Program Design Using the Jackson Methodology. Prentice-Hall, 1988. Им. перевод: Д. Кинг, Создание эффективного программного обеспечения. М., Мир, 1991.

27. В.В. Липаев, А.И. Потапов, Оценка затрат на разработку программных средств. М., Финансы и статистика, 1988.

28. M. Granger, F. Minssieux, "OS/2. Concepts et mise enoeuvre", Systemes, 1989. Имеется перевод: Гранже М., Менсье Ф. OS/2: Принципы построения и установка: Пер. с франц. - М.: "Мир", 1991. - 232 с.

29. Coats R.B., Vlaeminke I., "Man-computer interfaces", Black Well Scientific Publications, 1987. Имеется перевод: Коутс Р., Влейминк И. Интерфейс человек-компьютер: Пер. с англ. - М.: "Мир", 1990, 501 с.

30. Jim Gettys, Ron Newman and Robert Scheifler, Xlib - C Language X Interface, X Window System, X Version 11, Release 3, X Distribution Set, MIT, 1988.

31. Adrian Nye, Xlib Programming Manual for Version 11, OReilly & Associates, Inc., 1990.

32. Domain/C Language Reference, Hewlett-Packard, 1990.

33. RISCompiler and C Programmers Guide, MIPS Computer Systems, 1989.

34. SPARCompiler C++ 3.0.1. Programmers Guide, Sun Microsystems Inc., 1992.

35. Zortech C++ Compiler V3.0. Compiler Guide, Symantec corp., 1991.

36. RISC/os Programmers Guide, Volume I, MIPS Computer Systems, 1990.

37. Н.А. Вешняков, С.А. Кожемякин, В.П. Терешков, "Структура резидентного программного обеспечения графической станции". Техника средств связи, серия Техника телевидения, вып.4, 1990.

38. HP7475A Graphics Plotter. Operational and Interconnection Manual, Hewlett-Packard, 1990.

39. EPSON FX-850/1050. Users Guide, Seico Epson Corp., 1989.

40. Giles Billingsley and Ken Keller, "KIC: A Graphics Editor for Integrated Circuits", Programmers Manual, Univ. of California at Berkley, 1987.

41. Managing BSD System Software, Apollo Computer Inc., 1988.

42. G.Enderle, K.Kansy, G.Praff, "Computer Graphics Programming. GKS - The Graphics Standard". Springer-Verlag, 1984. Им. перев.: Т.Эндерле, К.Кэнси, Т.Прафф. Программные средства машинной графики. Международный стандарт GKS. Пер. с англ. - М., "Радио и связь", 1988.

43. Н.А.Маркова, Графический стандарт и его внедрение на ПЭВМ. "Микропроцессорные средства и системы", № 1, 1989, с.65.

44. В.Б. Бетелин, Г.В. Лебедев, "О проблеме мобильности графического программного обеспечения". В кн. Вопросы кибернетики. Автоматизированные системы ввода-вывода графической информации и их приложения. - М., 1987.

45. Peter Norton, Paul L. Yao, "Windows 3.0: Power Programming Teqniques", Bantam Books.

46. С.А. Гладков, Г.В. Фролов, Программирование в Microsoft Windows: В 2-х ч. М., ДИАЛОГ-МИФИ, 1992.

47. Charles Petzold, Programming Windows, Microsoft Press.

48. Microsoft C Professional Development System. Version 6.0. C Reference.

49. Н.В.Варанкин, А.Г.Соколов, О.Г.Тютюкин, Адаптивный графический интерфейс обеспечивает переносимость программных средств. Материалы конф. "Автоматизированное проектирование радиоэлектронной аппаратуры". Каунас, июнь 1992.

50. S. Harrington, "Computer Graphics. A Programming Approach", McGraw-Hill, 1987.

51. Роджерс Д. Алгоритмические основы машинной графики: Пер. с англ. - М.: "Мир", 1989. - 504 с.

52. Bruce A. Artwick, "Applied Concepts in Microcomputer Graphics", Prentice-Hall, 1984.

53. Дж. Фоли, Вен Дем, Основы интерактивной машинной графики: В 2-х кн. Пер с англ. М., Мир, 1985.

54. Б.С. Уокер и др., Интерактивная машинная графика. Пер. с англ. М., Машиностроение, 1980.

55. OSF/Motif. Style Guide. Revision 1.1. Prentice Hall, Englewood Cliffs, New Jersey 07632, 1991.

56. OPEN LOOK Graphical User Interface Application Style Guidelines, Sun Microsystems, Inc., Addison-Wesley, 1990.

57. Проектирование пользовательского интерфейса на персональ-ных компьютерах. Стандарт фирмы IBM. Вильнюс, DBS LTD, 1992.

58. OPEN LOOK Graphical User Interface. In: Solaris Open Windows. Open Windows V3 Collection. Release Reports and White Papers, SunSoft, 1991.

59. В.В. Бравов, Н.В. Варанкин, Л.Л. Савченко, " Реалии и перспективы развития графических редакторов для IBM PC". СБ. научных трудов МИЭТ. САПР СБИС: проблемы автоматизации. М., 1991.

60. ChipGraph Users Manual. Software Version 7.0. Mentor Graphics Corp., 1989.

61. "ПУЛЬТ". Иерархическая система редактирования топологии БИС. Версия 1.3 . Руководство пользователя. М., 1990.

62. Michel Lucas, Yvon Gardan, Techniques Graphiques et C.A.O., Hermes Publishing, 1983. Им. перевод: И. Гардан, М. Люка, Машинная графика и автоматизация конструирования. Пер. с франц. М., Мир, 1987, 272 с.

63. Н.В. Варанкин, А.Г. Соколов, "Система проектирования топологии заказных БИС". Электронная промышленность, № 3, 1993.

64. Bjarne Stroustrup, The C++ Programming Language, Addison-Wesley, 1986. Им. перевод: Б.Страуструп, Язык программирования C++. М., Радио и связь, 1991.

65. Microsoft MS-DOS: Operating System version 5.0. Users Guide and Reference. Microsoft Corp., 1991.

66. J.K.Ousterhout and D.Unger, "Measurements of a VLSI Design", Proc. 19th DAC, June 1982.

67. CD - A Package of C Procedures for Managing CIF Databases. См. [40]

68. J.L.Bentley and J.H.Friedman, "A survey of algorithms and data structures for range searching", ACM Compt. Surveys, vol.11,no.4,1979.

69. R.L.Brown, "Multiple storage quadtrees: a simpler faster alternative to bisector list quadtrees", IEEE Trans. CAD, vol.5, July 1986.

70. Jay Banerjee and Won Kim, "Supporting VLSI Geometry Operations in a Database System", Proc. Int. Conf. Data Eng., pp.409-415, June 1986

71. R.A.Finkel and J.L.Bentley, "Quad-trees - A data structure for retrieval on composite keys", Acta Inform., vol.4, no.1-9, 1974.

72. Gershon Kedem, "The quad-CIF tree: A data structure for hierarchical on-line algorithms", Proc. 19th DAC, June 1982.

73. Josette Berger, "Quad-tree hierarchy for circuit data retrieval in structured design", Proc. Int. Conf. Computer Design, 1985.

74. Anucha Pitaksanonkul, Suchai Thananawastein and Chidchanok Lursinsap, "Comparisons of Quad Trees and 4-D Trees: New Results", IEEE Trans. CAD, vol.8, no.11, Nov. 1989.

75. Wim De Pauw and Ludo Weyten, "Multiple storage Adaptive Multi-Trees", IEEE Trans. CAD, vol.CAD-9, no.3, March 1990.

76. Jon L.Bentley, "Multidimentional binary search trees used for associative searching", Commun. ACM, vol.18, no.9, Sept. 1975.

77. J.H.Friedman, Jon L.Bentley and R.A.Finkel, "An algorithm for finding best matches in logarithmic expected time", ACM TOMS, vol.3, no.3, Sept. 1977.

78. Jon L.Bentley, "Multidimentional binary search trees in database applications", IEEE Trans. Softw. Eng., vol.SE-5, no.4, July 1979.

79. Jonathan B.Rosenberg, "Geographical data structures compared: A study of data structures supporting region queries", IEEE Trans. CAD, vol.CAD-4, Jan. 1975.

80. Н.В.Варанкин, А.Г.Соколов, Высокоскоростная графическая база данных для САПР электронных изделий. Материалы конф. "Автоматизированное проектирование радиоэлектронной аппаратуры". Каунас, июнь 1992.

81. Jeffrey D. Ullman, Computational Aspects of VLSI. Rockville, MD, Computer Sci. Press, 1984. Им. перев. Дж.Д.Ульман "Вычислительные аспекты СБИС". М., "Радио и связь", 1990.

82. J.K.Ousterhout, "Corner-stiching: A Data Structuring Technique for VLSI Layout Tools", IEEE Trans. CAD, vol.CAD-3, no.1, Jan 1984.

83. Marple D., Smulder M., Hegen H., "Tailor: A Layout System Based on Trapezoidal Corner Stiching", IEEE Trans. CAD, vol.CAD-9, no.1, Jan 1990.

84. J.K. Ousterhout, G.T. Hamachi, R. N. Mayo, W.S. Scott and G.S. Taylor, "The Magic VLSI Layout System", IEEE Design & Test, Feb. 1985, pp. 19-30.

85. З.А.Лившиц, Д.Г.Титов, Алгоритмы работы с тайловыми представлениями топологии СБИС. Автометрия, № 3, 1991.

86. Pei-Yung Hsiao and Wu-Shiung Feng, "Using a Multiple Storage Quad Tree on a Hierarchical VLSI Compaction Scheme", IEEE Trans. CAD, vol.CAD-9, no.5, May 1990.

87. Jeffrey D. Ullman, Computational Aspects of VLSI, Computer Science Press, 1984. Им. перев. Дж.Д.Ульман, Вычислительные аспекты СБИС. М., "Радио и связь", 1990.

88. М.Ватанабэ, К.Асада, К.Кани, Т.Оцуки, Проектирование СБИС. Пер. с яп. М.,Мир,1988.

89. В.З.Фейнберг, Геометрические задачи машинной графики больших интегральных схем. М., Радио и связь, 1987.

90. G.S. Taylor and J.K. Ousterhout, "Magics Incremental Design Rule Checker", Proc. 21st DAC, 1984, pp.160-165.

91. C.M. Baker and C. Terman, "Tools for Verifying Integrated Circuit Design", Lambda 1:3, 1980.

92. Dracula Users Reference Manual, CADENCE, Inc., 1986.

93. R.A. Cottrel, "IC Layout Verification", in Comp.-Aided Tools for VLSI System Design, G.Russel, ed., Peregrinus/IEE, 1987.

94. K. Yoshida, "Layout Verification", in Layout Design and Verification, 1986.

95. Joel W. Gannet, "SHORTFINDER: A Graphical CAD Tool for Locating Net-to-Net Shorts in VLSI Chip Layouts", IEEE Trans. on CAD, vol.9, no.6, June 1990.

96. Stephen C. Johnson, "Hierarchical Design Validation Based on Rectangles", Proc. Conf. on Adv. Reseach in VLSI, MIT, 1982.

97. Г.Корн, Т.Корн, Справочник по математике для научных работников и инженеров. Пер. с англ. М., Наука, 1984.

98. Мультиоконный графический редактор Layout Windows. Версия 1.26. Руководство пользователя. Зеленоград, 1993.

99. C.Mead and L.Conway, Introduction to VLSI systems. Reading, MA: Addison-Wesley, 1980.

100. Варанкин Н.В., Кораблев И.С., Орсич И.А., Соколов А.Г. Применение интеллектуальных систем для проектирования элементной базы БИС. В сб. науч. трудов МИЭТ "САПР СБИС: Проблемы автоматизации", М, МИЭТ, 1991.

101. Д.Кнут, Искусство программирования для ЭВМ. В 3-х кн. М., Мир, 1976-1978.

102. П.Нортон, Персональный компьютер фирмы IBM и операционная система MS-DOS. Пер. с англ. М., Радио и связь, 1991.

103. Лянгасов С.И., Варанкин Н.В. Программы преобразования и графического анализа данных. В сб. науч. трудов МИЭТ "Алгоритмическое обеспечение и проектирование микропроцессорных управляющих систем", М, МИЭТ, 1982.

104. В.А.Селютин, Автоматизация проектирования топологии БИС. М., Радио и связь, 1983.

105. Варанкин Н.В., Орсич И.А., Соколов А.Г., Шумилов В.В. Комплекс программ многоуровневого моделирования фрагментов КМОП БИС на ПЭВМ IBM PC. В сб. науч. трудов МИЭТ "САПР СБИС: Проблемы автоматизации", М, МИЭТ, 1991.

106. J. Miller, K. Groning, G. Schulz and C. White, "The Object-Oriented Integration Methodology of the Cadlab Work Station Design Environment", Proc 26th DAC, 1989.

107. D.S. Harrison, P. Moore, R.L. Spickelmier and A.R. Newton, "Data Management and Graphics Editing in the Berkley Design Environment", Proc IEEE ICCAD-86, 1986.

ПРИЛОЖЕНИЯ

1. Акты внедрения результатов диссертационной работы в промышленности и учебном процессе.

2. Программа демонстрации возможностей H-интерфейса.

3. Программа определения относительной скорости вычисления трансформаций.

4. Описание формата файла правил контроля проектных норм топологии.