

# Проект Оберон

Разработка операционной системы,  
компилятора и компьютера

Обновлённое издание 2013 г.

Никлаус Вирт  
Юрг Гуткнехт

Перевод этой книги в данный момент не завершён. Разрешается его свободное распространение в неизменном виде и использование на свой страх и риск исключительно в ознакомительных целях.

This translation is a work in progress. It may be distributed freely but used at your own risk for evaluation purposes only.

# Предисловие

В этой книге представлены результаты проекта Оберон по созданию интегрированного программного окружения современной рабочей станции. Основными целями проекта, реализованного авторами в 1986-89 гг., были проектирование и разработка с нуля системы, структура которой позволит описать, объяснить и понять её как единое целое. Чтобы ощутить на собственном опыте все аспекты, проблемы и последствия принятых проектных решений, а также иные детали реализации, авторы не только задумали, но и полностью воплотили в коде не только то, что вошло в эту книгу, но и многое другое.

Несмотря на существование многочисленных книг, в которых разъясняются принципы построения и структура операционных систем, ощущается нехватка описаний систем, которые были в реальности разработаны и применялись на практике. Нашим желанием было не только дать рекомендации, как может быть устроена такая система, но и продемонстрировать, как именно их применять. Как следствие, фрагменты программного кода играют в нашей книге важную роль самых подробных пояснений, которые только можно представить. В связи с этим потребовалось особое внимание при выборе подходящего формализма. Язык программирования Оберон был спроектирован не только как эффективное средство реализации системы, но и как формальная нотация для публикации алгоритмов наподобие Алгола 60, созданного три десятилетия назад. Благодаря своей структуре, язык Оберон в равной степени хорош и для демонстрации глобальной модульной архитектуры программных систем.

Несмотря на малые трудозатраты на реализацию системы Оберон и компактность описания, для которого достаточно одной-единственной книги, не следует считать, что она представляет исключительно академический интерес. Напротив, это достаточно развитое программное окружение для рабочих станций нашло себе не только довольных пользователей, но и восторженных поклонников как в академической среде, так и в промышленности. Ядро системы, которое мы рассмотрим далее, содержит в себе менеджеры накопителей данных, файлов, дисплея, текстовых и графических представлений. Его обширные возможности построены на фундаменте тщательно подобранного множества гибких базовых примитивов, обладающих, что ещё более важно, расширяемостью по многим направлениям и пригодных для множества приложений. Эта расширяемость существенно улучшается как использованием языка Оберон с одной стороны, так и эффективностью базового ядра с другой. В основе расширяемости лежит применение объектно-ориентированной парадигмы там, где это представляется выгодным.

В дополнение к ядру системы, будут детально рассмотрены компилятор языка Оберон и графическая система, которые относятся к прикладным программам. На примере компилятора мы увидим, что он может быть спроектирован одновременно как для достижения высокой скорости компиляции, так и генерации эффективного компактного кода. Графическая подсистема продемонстрирует расширяемую архитектуру на основе объектно-ориентированных технологий, которая была успешно интегрирована с существующей текстовой системой. Еще одним дополнением базового ядра послужит сетевой модуль, позволяющий объединить множество рабочих станций в сеть. Также мы покажем, как система Оберон может с удобством применяться в роли сервера, обеспечивающего совместный доступ к файлам, сетевую печать и обмен электронной почтой.

Компактность, регулярность структуры и тщательный контроль эффективности ключевых деталей реализации — залог успеха экономной инженерии программного обеспечения. На примере системы Оберон мы хотели бы опровергнуть правило Рейзера, которое подтверждается опытом разработки практически всех современных операционных систем: *несмотря на существенный прогресс аппаратного обеспечения, рост его производительности не компенсирует замедление программ*. Система Оберон довольствовалась лишь малой долей трудозатрат, которые понадобились для создания популярных коммерческих операционных систем. Её потребность в вычислительной мощности и объёмах накопителей данных столь же мала, но пользователи при этом получают сравнимые с другими системами гибкость и возможности, за исключением, возможно, некоторого внешнего лоска. Мы предлагаем читателю разобраться, как это стало возможным.

Важнее же всего то, что в этой книге мы надеемся представить достойный пример достаточно масштабного программного продукта, который пригодится всем желающим учиться на опыте других.

Мы выражаем нашу искреннюю признательность анонимным авторам многочисленных предложений, советов и благожелательных отзывов. Также мы благодарим наших коллег, Х. Мёссенбёка и Б. Сандерса, и наших associates at the Institut für Computersysteme, которые ознакомились с этой книгой во время работы над ней. Мы благодарны M. Brandis, R. Creliez, A. Disteli, M. Franz, and J. Templ за их успешную работу по переносу системы Оберон на различные модели компьютеров, имеющих в продаже. Доступное аппаратное обеспечение расширило круг читателей, которым принесло пользу изучение нашей книги. Наконец, мы с благодарностью отмечаем вклад Швейцарской высшей технической школы Цюриха, поддержка и атмосфера которой сделали возможной как работу над этим проектом, так и успешное его завершение.

Цюрих, февраль 1992 г.  
Н.В. и Ю.Г.

# Предисловие к изданию 2013 г.

Реакция общества на наши планы по подготовке второго издания была неоднозначной. Некоторые полагали, что книга устарела, что никто более не заинтересуется подобной системой. «Зачем напрягаться?» Другие же решили, что имеется актуальный спрос на книгу такого рода, в которой раскрываются все подробности реализации, в противоположность простому изложению возможных стратегий и подходов. “By all means”!

Немало воды утекло за прошедшие 30 лет. Но даже не принимая во внимание эти перемены, было бы весьма неразумно пытаться предложить и создать систему, которая смогла бы конкурировать с существующими общепринятыми «стандартами». В самом деле, нашлось бы слишком мало желающих ей воспользоваться. Создаётся впечатление, что сообщество в целом погрязло в этих гигантских программных системах и не в силах справиться с их сложностью, большим количеством особенностей и, время от времени, ненадёжностью.

Нетрудно предсказать, что в будущем появятся новые системы, возможно, специализированные для различных узких целей, благодаря чему их размер может оказаться меньшим. Как следствие, возникает вопрос: как их создатели смогут научиться своему ремеслу? Количество подходящей технической литературы невелико и, по моему мнению, обучение в целом происходит «на производстве». Однако, такой способ учёбы утомителен и не оптимален. В то время, как в науке правят бал принципы и законы, которые следует изучить и понять, в работе инженера невозможно обойтись без опыта и практики. Можно ли считать, что преподавание информатики включает законы, которые будут верны практически вечно? Информатика по своей сути обречена строиться на строгом математическом фундаменте, и эта взаимосвязь намного теснее, чем в любых других инженерных отраслях. Тем не менее, в некоторых ключевых вопросах скорее приходится полагаться на практический опыт, изучая удачные разработки.

Основной целью и движущей силой этого проекта было создание книги, способной служить примером операционной системы, которая существует, используется на практике и может быть описана во всех подробностях. В ходе решения этой задачи был сделан вывод, что спроектировать *мощную и в то же время надёжную* систему непросто, но ещё сложнее сделать её при этом настолько простой и ясной, чтобы её можно было изучить и полностью понять. Более всего другого это требует устойчивой концентрации внимания на том, что действительно необходимо, и силы воли отбросить

всё остальное, все те безделушки, которые столь популярны сегодня.

В последнее время всё большее количество людей проявляет интерес к проектированию новых, более компактных, систем. Избыточная сложность популярных операционных систем не только затрудняет их понимание, но и создает благоприятные условия для появления в них «тайных входов». Такие входы позволяют незаметно для пользователей вмешиваться извне в работу системы, устанавливая в неё шпионское и другое вредоносное программное обеспечение, тем самым делая систему уязвимой для атак и повышая вероятность её разрушения. Единственным выходом из сложившейся ситуации представляется построение безопасной системы с самого начала.

Завершая теоретический экскурс, вернемся к практике. Самая объёмная глава издания 1992 г. была посвящена компилятору, транслирующему язык Оберон в машинный код процессора NS32032. Этот процессор более не производится, а его архитектура не считается образцом для подражания. Вместо разработки нового компилятора для какой-либо доступной сегодня архитектуры, было принято решение создать свой собственный процессор, что позволит применить к аппаратному обеспечению приведенные выше принципы простоты и регулярности. Окончательная выгода такого подхода в том, что не только программная, но и аппаратная часть системы Оберон получит полное и строгое описание. Новый процессор был назван RISC. Для реализации аппаратуры применяется язык Verilog.

Решению разработать собственный процессор немало поспособствовала возможность его материального воплощения, которое будет доступно всем желающим. Современный уровень развития программируемых пользователем вентильных матриц (ППВМ) позволяет разместить в единственной микросхеме полноценную реализацию предложенной процессорной архитектуры. Как следствие, наша система может быть построена на основе недорогой отладочной платы. Такая плата производства Digilent включает ППВМ Xilinx Spartan-3 и 1 МБ статического ОЗУ, которого более чем достаточно для системы Оберон вместе с компилятором. Пример аппаратной конфигурации, в которую входят монитор, клавиатура и мышь, показан на фотографии. Сама плата видна в правом нижнем углу.

Еще одним следствием выбора собственной процессорной архитектуры стало то, что главы книги, посвященные компилятору и загрузчику модулей, пришлось полностью переписать. Однако, тем самым нам была предоставлена благоприятная возможность существенно улучшить их ясность. Новый процессор позволил упростить компилятор и сделать его более прямым.

На пути к понятному описанию системы важнейшую роль играет нотация, формализм или язык, который при этом применялся. Алгол 60, опубликованный 50 лет тому назад, был задуман для публикаций, как формализм, при помощи которого возможно определить алгоритм, не ссылаясь при этом на особенности каких-либо компьютеров или даже любых других исполнителей. Полностью достичь этой замечательной цели не удалось, но по крайней мере стала ясна важность *абстракции*, которой следует достигать при помощи нотации, построенной на строгом математическом фундаменте. По крайней мере, Алгол оказался первым языком, чей синтаксис был определен формально. Также Алгол стал результатом зарождающегося понимания того, что программы никогда не следует писать исключительно



ради их исполнения компьютером, а напротив, ради людей, которые будут учиться новому, изучая эти программы.

На протяжении всей моей предыдущей деятельности я пытался создать язык-наследник Алгола, который не только смог бы превзойти его в строгости, но и расширить его применимость от численных методов до программных систем. Путь, начатый с Алгола, прошёл через Паскаль и Модулу к Оберону. Следуя этим путём, мы смогли подойти к цели ближе, чем когда-либо ранее, и ближе, чем какой-либо другой язык программирования. Ключом к успеху послужила постоянная борьба за разумное упрощение.

Язык Оберон, после своего появления в 1988 г., подвергся пересмотру в 2007 г., в ходе которого в основном были удалены конструкции, которые дублировали другие, либо же такие, без которых можно обойтись. Переработка исходных текстов системы в соответствии с новой версией языка была второй, после смены архитектуры процессора, важной причиной многочисленных локальных изменений текста книги. Вкратце, в языке произошли следующие изъятия:

1. Типы данных `LONGINT`, `SHORTINT` и `LONGREAL` были удалены вместе с концепцией включения числовых типов.
2. Удалены операторы `LOOP` и `EXIT` (цикл со многими выходами).
3. Удалён оператор `WITH` (охрана типа на участке).
4. Оператор `RETURN` был удалён как самостоятельная управляющая конструкция и объединён синтаксически с завершением объявления процедуры-функции.

5. Объекты, объявленные в процедуре `P`, более недоступны внутри процедуры `Q`, которая сама является локальной относительно `P`. Другими словами, разрешен доступ к объектам, которые либо строго локальны, либо глобальны.
6. Присваивание значений импортированным переменным не допускается (экспорт в режиме «только для чтения»).
7. Удалены предописания процедур.

В отличие от представленных выше сокращений, появление в языке новых элементов ограничилось единственным случаем: в 2012 г. был добавлен тип данных `BYTE`. Множество его значений составляют целые числа  $x$  такие, что  $0 \leq x < 256$ . Это нововведение предотвращает частое использование типа `CHAR` не по назначению. Тип `BYTE` в основном применяется для объявления массивов и записей в низкоуровневых модулях, чтобы сократить расход памяти.

Несмотря на существование таких весомых поводов для перемен как на высоком (язык), так и на низком (аппаратура) уровне, прочие части книги остались практически прежними, но не потеряв при этом актуальности. Автор испытывал особое желание продемонстрировать систему в точности в том виде, в котором она существовала 25 лет тому назад, ничего не приукрасив. Главы 3-??, в которых рассмотрено управление задачами, дисплеем и текстом, первоначально написанные Ю. Гуткнехтом, приводятся в данном издании практически без изменений. Существенные изменения претерпели материалы, касающиеся драйверов клавиатуры и мыши. В новой редакции книги они подключаются по интерфейсу PS/2. Дисковый накопитель был заменён одной SD-картой, обмен данными с которой ведётся по шине SPI. Сетевой интерфейс стандарта RS-485 был также заменён на SPI. Главы, посвящённые компилятору и компоновщику, переписаны полностью.

Размер компилятора удалось существенно сократить, в основном, за счет регулярной структуры системы команд процессора RISC. В данный момент он составляет порядка 2900 строк кода и компилирует сам себя в течение 3 секунд, что в очередной раз служит подтверждением его эффективности. Компиляция всей системы в целом занимает менее 10 секунд.

Код проверки ошибок времени выполнения, наличие которого ещё несколько лет назад считалось экстравагантным, а его необходимость подвергалась сомнениям, генерируется автоматически. Помимо прочего, эти проверки касаются диапазонов индексов массивов и разыменования NIL-указателей. Благодаря своей эффективности, они лишь незначительно влияют на скорость исполнения программ, принося взамен очевидную пользу программистам.

Ещё одним выгодным последствием упрощения языка и процессора стало то, что все фрагменты системы, написанные в 1992 г. на языке ассемблера и по этой причине не вошедшие в книгу, теперь удалось выразить на Обероне. Тем самым были оправданы многолетние усилия по проектированию языка высокого уровня, который будет не только мощным и гибким, но и достаточно эффективным для реализации таких частей системы, как драйверы устройств и растровая графика. Этот необходимый шаг оказался последним на пути к тому, чтобы сделать книгу исчерпывающей и завершённой.



## Ссылки

<http://www.inf.ethz.ch/personal/wirth/Oberon/Oberon07.Report.pdf>

<http://www.inf.ethz.ch/personal/wirth/FPGA-relatedWork/RISC.Arch.pdf>

## Благодарности

Автор с признательностью отмечает весомый вклад, сделанный в подготовку данной книги Полом Ридом. Он реализовал интерфейсы стандартов PS/2 и SPI для подключения различных устройств, в том числе SD-карты, которая заменила дисковый накопитель. Благодаря ему в текст были внесены многочисленные улучшения и упрощения. Первоначальное решительное предложение заняться пересмотром книги тридцатилетней давности также исходило от Пола, и он стал основной движущей силой этого процесса. Автор благодарен ему за это.

Никлаус Вирт, сентябрь 2013 г.



# Оглавление

<b>Предисловие</b>	<b>iii</b>
<b>Предисловие к изданию 2013 г.</b>	<b>v</b>
<b>1 История и мотивация</b>	<b>1</b>
<b>2 Основные концепции и структура системы</b>	<b>7</b>
2.1 Введение . . . . .	7
2.2 Концепции . . . . .	8
2.2.1 Отображения . . . . .	8
2.2.2 Команды . . . . .	9
2.2.3 Задачи . . . . .	12
2.2.4 Командные тексты как конфигурируемые меню . . . .	14
2.2.5 Расширяемость . . . . .	15
2.2.6 Динамическая загрузка . . . . .	15
2.3 Структура системы . . . . .	16
2.4 Обзор следующих глав . . . . .	19
<b>3 Управление задачами</b>	<b>25</b>
3.1 Понятие задачи . . . . .	25
3.1.1 Интерактивные задачи . . . . .	26
3.1.2 Фоновые задачи . . . . .	27
3.2 Планировщик задач . . . . .	28
3.3 Понятие команды . . . . .	31
3.3.1 Атомарные действия . . . . .	31
3.3.2 Обобщённое выделение текста . . . . .	33
3.3.3 Обобщённое копирование отображений . . . . .	33
3.4 Инструментарии . . . . .	34



# Глава 1

## История и мотивация

Интересно, как это вообще возможно: прилежно сосредоточиться на работе в такой полдень, полный тепла, солнечного света и голубого неба? Такой риторический вопрос я не раз задавал самому себе в 1985 г. во время годовичного творческого отпуска в Калифорнии. Дома, в Швейцарии, каждый считал бы себя обязанным воспользоваться недолгими солнечными днями, чтобы отдохнуть на лоне природы, прогуляться или заняться любимым спортом. Но здесь, где хорошая погода стоит каждый день, поддаться таким искушениям означало бы покончить с любой работой. Возможно, именно поэтому я не выбрал бы для жизни это место с его манящим приятным климатом?

К счастью, моя работа была достаточно интересной, чтобы погрузиться в неё без особых усилий. Мне выпала редкая привилегия сидеть перед самой совершенной рабочей станцией из когда-либо созданных, изучая секреты, возможно, новейших модных тенденций нашего быстро развивающегося ремесла. Я двигал по экрану цветные прямоугольники. Всё это происходило согласно строгим правилам, обусловленным законами физики и новейшими технологиями. К счастью, совершенный компьютер немедленно сообщил бы о нарушении какого-либо правила. Благодаря встроенной системе их проверки, он действовал наподобие Большого брата, удерживая вас от шагов, ведущих к неприятностям. Ему удавалось то, что ранее считалось невозможным: поддерживать одновременно тысячи ограничений на относительное расположение тысяч прямоугольников. Это называлось «системой автоматизированного проектирования» (computer-aided design, дословно: «проектирования при помощи компьютера», — *прим. перев.*). Слово «помощь» в данном контексте было скорее эвфемизмом, но компьютер не жаловался на такое преуменьшение своей роли.

В то время, как мой взгляд был прикован к разноцветным изображениям на мониторе, и пока я боролся с последствиями допущенной ранее ошибки, в постоянно открытую дверь вошел мой коллега (Ю.Г.). Ему также выпало оставить на время свои привычные обязанности ради работы в той же лаборатории, что и мне, но его лицо выражало скорее досаду, чем счастье. Плитка шоколада в руке служила ему так же, как чашка кофе или трубка табака другим людям: давала возможность расслабиться и отвлечься. Уже не первый раз он появлялся в подобном состоянии духа, так что я мог угадать его причину не говоря ни слова. И эта ситуация, вероятно, могла случиться снова и снова.

В отличие от меня, мой коллега не проводил свое рабочее время, развлекаясь перемещением прямоугольников. Ему была поставлена конкретная задача: разработка компилятора для того же самого компьютера. Как следствие, разбираться с программным обеспечением этой системы приходилось гораздо более глубоко, вплоть до мельчайших подробностей. Как программисту, моему коллеге пришлось выяснять причины довольно частых сбоев, в то время как я всего лишь работал с одной из прикладных программ. По сути, я был всего лишь одним из пользователей! Изучать эти сбои мне было нужно не для их исправления, а всего лишь чтобы их обойти. Каким же образом можно было бы добиться требуемой глубины понимания? В этот момент я осознал, что до сих пор избегал задавать себе этот вопрос. Моё знакомство с этой новейшей системой было поверхностным, не выходящим за рамки требований текущей работы.

Вскоре мне стало ясно, что изучение этой системы практически невозможно. Она поражала не только своими размерами, но и неполнотой документации. Ответы на вопросы, которые требовались немедленно, можно было получить в разговоре с разработчиками системы, которые, к счастью, работали здесь же. Во время этих бесед мы сделали шокирующее открытие, что зачастую не можем их понять. Пояснения были полны жаргонных словечек и отсылки к другим частям системы, которые были ничуть не менее загадочными.

Возникшие при этом разочарование и досада привели к тому, что в нашей работе по реализации компилятора и проектированию микросхем образовались паузы, которые мы решили посвятить выяснению самой сути, фундамента новейших аспектов системы. Что делает её отличающейся от других, обычных систем? Какие из этих концепций необходимы, а какие можно улучшить, упростить или даже отбросить? Где именно они залегают? Возможно ли очистить и выделить суть системы подобно тому, как это происходит с веществами в химических реакциях?

В ходе последующих обсуждений постепенно родилась идея спроектировать все самим. Внезапно, она обрела конкретную форму. Мою первую реакцию можно описать словами «сумасшествие» и «невозможно». Объём требуемых усилий поражал. В конце-концов, мы оба после возвращения домой должны были продолжать преподавательскую работу. Однако, эта мысль укоренилась в нашем сознании, продолжая постепенно овладевать нами.

Некоторое время спустя после возвращения домой, обстоятельства сложились так, что мне было предложено прочесть важный курс, посвященный системному программному обеспечению. Я колебался, ведь по неписаным правилам он должен быть в основном посвящен принципам устройства операционных систем. Мои опасения не были беспочвенными: в конце-концов, я ведь никогда ещё не проектировал такие системы, ни целиком, ни даже частично. Неужели возможно преподавать инженерную дисциплину, не имея в ней практического опыта?

Невозможно? Разве мы уже не создавали компиляторы, операционные системы и редакторы документов силами небольших групп? Разве не было так, причём не один раз, что неадекватная и разочаровывающая программа может быть переписана с нуля, при этом существенно сократившись в размере? Наш мозговой штурм продолжался, с краткими перерывами, несколько недель, и неясные очертания новой системы стали медленно про-

ступать сквозь дымку. Спустя какое-то время было принято решение, кажущееся абсурдным: мы начинаем разработку операционной системы для нашей рабочей станции (которая оказалась намного менее мощной чем та, на которой я занимался укладкой прямоугольников) с нуля.

Основная цель, получение собственного практического опыта и достижение полного понимания каждой детали, естественным образом определила и допустимые трудозатраты: два программиста с неполным рабочим днем. В качестве временных рамок проекта мы благоразумно решили ограничиться тремя годами. Как выяснилось в дальнейшем, эта оценка оказалась весьма точной: мы начали программировать в начале 1986 г., а первая версия системы увидела свет осенью 1988 г.

Несмотря на то, что обычно выбор подходящего названия проекта обычно не считается важным делом и часто отдается на волю случая и внезапных озарений разработчиков, наступил, возможно, удачный момент для рассказа о том, как и почему наш проект получил своё имя. Случилось так, что примерно в то же время, когда мы начинали работу, автоматическая межпланетная станция «Вояджер» попала на первые полосы газет, передавая на Землю восхитительные изображения планеты Уран и её спутников, крупнейший из которых назывался Обероном. С момента её запуска я считал проект «Вояджер» исключительно хорошо спланированным и успешным начинанием. Чтобы отдать ему должное, я выбрал в качестве имени нашего проекта название последнего исследованного «Вояджером» объекта. В самом деле, найдётся не так уж и много инженерных проектов, продукты которых превзошли ожидания как по своей производительности, так и по времени службы: большинство терпит крах намного раньше, особенно в области программирования. Наконец, ещё одним немаловажным доводом стало то, что Оберон известен как король эльфов.

Сознательно запланированная нехватка рабочей силы диктовала нам единственную, но при этом здравую, стратегию: сосредоточиться на необходимых функциях, избегая декора, который представляет собой ни что иное, как дань общепринятым соглашениям и быстро меняющейся моде. Безусловно, в тот момент нам ещё только предстояло выяснить, что же должно войти в состав этого самого необходимого ядра, а затем зафиксировать найденный набор возможностей. Однако, фундамент был заложен. Наше базовое правило приобрело ещё большую важность, когда мы решили, что результат нашей работы должен быть пригодным в качестве учебного пособия. Я вспомнил, с каким чувством Ч. Хоар призывал писать книги так, чтобы в них рассматривались реальные операционные системы, а не их абстрактные полуфабрикаты. В начале 1970-х он жаловался, что в нашей отрасли инженеров постоянно поощряют создавать все новые артефакты не давая им возможности изучить работы их предшественников, хорошо зарекомендовавшие себя на практике. Как же прав он был, даже на сегодняшний день!

После того, как была осознана цель опубликовать результаты нашей работы во всех подробностях, проблема выбора языка программирования предстала в новом свете: он стал критически важным. Наш первоначальный выбор, Модуль-2, оказался не вполне подходящим. Прежде всего, в этом языке отсутствовала возможность адекватно выразить расширяемость, которую мы считали одним из ключевых свойств новой системы. Под «адекватностью» здесь подразумевается машинная независимость. Наши про-

граммы должны быть написаны в стиле, не допускающем отсылок к особенностям конкретных архитектур и средств низкоуровневого программирования, за исключением, возможно, взаимодействия с аппаратурой, где эта зависимость неизбежна.

Как следствие, в Модуль-2 была добавлена возможность, известная ныне как расширение типов. Также мы пришли к выводу, что этот язык содержит различные конструкции, которые нам не нужны, которые на самом деле не вносят ничего важного в его выразительную силу, но в то же время увеличивают сложность компилятора. Компилятор же нужно не только реализовать, но и описать его, сделать доступным для изучения и понимания. Исходя из этого, было принято еще одно решение начать с чистого листа, на этот раз в области проектирования языков программирования, следуя тем же принципам: концентрироваться на существенном, отбрасывать всё остальное. Новый язык, который до сих пор сохранил заметное сходство с Модуль-2, получил то же самое имя, что и система в целом: Оберон [1, 2]. В отличие от своего предшественника, он более лаконичен и, прежде всего, представляет собой значительный шаг на пути к возможности программирования на высоком уровне абстракции без применения машинно-зависимых средств.

Проектирование системы началось поздней осенью 1985 г., а её реализация — в начале 1986 г. В качестве инструментальной машины использовалась рабочая станция Lilith, языком программирования которой была Модуль-2. В первую очередь был разработан кросс-компилятор, за которым последовали модули ядра системы совместно с необходимыми средствами тестирования и загрузки данных. Одновременно велась работа над подсистемой управления дисплеем и текстовой подсистемой. Конечно же, возможности их протестировать у нас не было, и мы на собственном опыте поняли, насколько отсутствие отладчика и, более того, компилятора, способствует аккуратному программированию.

Нашим следующим шагом был перевод компилятора на Оберон. Это удалось сделать достаточно быстро благодаря тому, что такая возможность была заложена изначально. После появления компилятора на целевой машине, рабочей станции Seges, и ввода в строй средств редактирования текстов, пуповина, связывающая нас с Lilith, могла быть перерезана. Система Оберон стала реальностью, по крайней мере в черновом её варианте. Это произошло примерно в середине 1987 г., и вскоре было опубликовано описание системы [3], за которым в 1991 г. последовали документация для программистов и руководство пользователя [5].

Завершение работы над системой потребовало ещё одного года, в ходе которого приоритет отдавался объединению рабочих станций в сеть с возможностью обмена файлами [4], централизованной печати, а также инструментам сопровождения. Поставленная цель — построить работающую систему за три года — была достигнута. В середине 1988 г. система была представлена более широкому сообществу, и стала возможной работа над прикладными программами. Были реализованы служба электронной почты и графическая подсистема, началась разработка универсальных средств подготовки документации. Подсистема управления дисплеем была расширена для поддержки двух экранов, включая цветные. Одновременно с этим отзывы пользователей позволили также улучшить и другие существующие части системы. Начиная с 1989 г., Оберон заменил Модуль-2 в наших ввод-



ных курсах программирования.

## Ссылки

1. N. Wirth. The programming language Oberon. *Software — Practice and Experience* 18, 7, (July 1988) 671-690.
2. M. Reiser and N. Wirth. *Programming in Oberon — Steps beyond Pascal and Modula*. Addison-Wesley, 1992.
3. N. Wirth and J. Gutknecht. The Oberon System. *Software — Practice and Experience*, 19, 9 (Sept. 1989), 857-893.
4. N. Wirth. Ceres-Net: A low-cost computer network. *Software — Practice and Experience*, 20, 1 (Jan. 1990), 13-24.
5. M. Reiser. *The Oberon System — User Guide and Programmer's Manual*. Addison-Wesley, 1991.



## Глава 2

# Основные концепции и структура системы

### 2.1 Введение

Успешное проектирование и реализация с нуля полноценной операционной системы при разумных затратах сил и времени невозможны без некоторого количества оригинальных базовых принципов. В начале этой главы мы рассмотрим важнейшие концепции, лежащие в основе системы Оберон, и проектные решения, определившие её облик. Затем с их помощью мы представим структуру системы. На данном этапе мы не будем вдаваться в детали, ограничившись описанием самых крупных её элементов, модулей, их композиции и зависимостей. В конце главы читателя ждёт обзор остальной части книги. Его цель — помочь понять роль, место и значимость различных частей системы, рассмотренных в отдельных главах.

Основное назначение операционной системы — представление компьютера пользователю и программисту на некотором уровне абстракции. Например, память может рассматриваться как совокупность переменных заданных типов данных, которые могут быть затребованы по необходимости, дисковый накопитель — как множество последовательностей литер (или байтов), известных как файлы, дисплей представлен прямоугольными областями — *отображениями*, клавиатура — входным потоком литер, а мышь — парой координат и множеством состояний кнопок. Каждая абстракция характеризуется определённым набором свойств и множеством допустимых операций. Задача системы — реализовать их и организовать выполнение с учётом доступных ресурсов данного компьютера. Эта задача получила название *управление ресурсами*.

Каждая абстракция естественным образом скрывает детали, в чем и состоит её суть. Это скрывание может происходить на различных уровнях. Например, некоторые области памяти компьютера или некоторые устройства могут быть недоступными в зависимости от режима его работы (пользовательский/системный), или же язык программирования управляет доступом к тем или иным сущностям при помощи встроенных в него возможностей. Программное решение, безусловно, гораздо гибче и мощнее; на самом деле, аппаратные ограничения доступа играют в нашей системе пренебрежимо

малую роль. Важность такого скрывания в том, что оно позволяет гарантировать выполнение определённых свойств нашей абстракции, называемых её *инвариантами*. Абстракция — ключ к любой форме модульности, без которой, в свою очередь, никакие попытки обеспечить надёжность и корректность не увенчаются успехом. Очевидно, целью проектирования системы Оберон было построение модульной структуры на основе целевых абстракций. Доступность подходящего языка программирования — необходимое условие, и важность его выбора не может быть переоценена.

## 2.2 Концепции

### 2.2.1 Отображения

Абстрактные представления памяти как множества переменных и дискового накопителя как множества файлов достаточно чётко определены и имеют смысл в любой компьютерной системе. В то же время, абстракции, касающиеся устройств ввода и вывода, приобрели важность лишь после того, как интенсивность взаимодействия человек-компьютер превысила определённый порог. Высокая *интерактивность* требует хорошей пропускной способности, которую в человеческом организме обеспечивает только зрение. Как следствие, компьютерные устройства отображения графической информации должны соответствовать зрительным возможностям человека. Этого удалось достичь в середине 1970-х при помощи дисплеев высокого разрешения, которые, в свою очередь, обязаны своим появлением наличию всё более быстрой и дешёвой памяти. Это событие может считаться одним из немногих прорывов в истории развития компьютерных технологий. Типичная пропускная способность современного дисплея имеет порядок 100 МГц. Одним из первых последствий их появления стало применение к графическому выводу принципов абстракции и управления ресурсами. В системе Оберон дисплей разделяется на *отображения*, также называемые *окнами*, или, точнее, на *фреймы*, прямоугольные области на одном или нескольких экранах. Отображение обычно состоит из двух фреймов: строки заголовка, в которой содержится имя и меню команд, и основного фрейма, содержащего некоторый текст, векторную либо растровую графику или другие объекты. Отображение само по себе также является фреймом, которые могут быть вложены на любую глубину.

Система предоставляет процедуры создания фрейма (отображения), а также его перемещения и закрытия. Она создает новое отображение в заданном месте и при необходимости выдает подсказку о наилучшем его возможном расположении. Также система ведет учёт открытых отображений. Этот процесс называется *управлением отображениями* и происходит отдельно от отрисовки их содержимого.

Помимо высокой пропускной способности при выводе визуальной информации, достижение хорошей интерактивности требует также и гибкости ввода. Безусловно, при этом не нужна столь же широкая полоса пропускания, но ограничение скорости ввода с клавиатуры на уровне около 100 Гц явно недостаточно. Прорыв в этой области произошёл с появлением так называемой *мыши*, устройства координатного ввода, которое появилось примерно в одно время с дисплеями высокого разрешения.

Это произошло отнюдь не по воле счастливого случая. Именно наличие дисплея высокого разрешения и соответствующей программной поддержки позволяет мышке раскрыть весь свой потенциал. Сама по себе мышь ни что иное, как очень простое устройство, регистрирующее своё перемещение по столу. Эта информация позволяет компьютеру обновлять положение отметки на экране, известной как курсор. Поскольку обратная связь происходит посредством человеческого зрения, большая точность от мыши не требуется. Например, когда человеку требуется указать некоторый объект на экране, такой как литера, он передвигает мышь столько, сколько нужно, пока курсор не совпадет с объектом. Это составляет разительный контраст с диджитайзером, от которого ожидаются точные координаты. Система Оберон существенно образом полагается на наличие мыши.

Снабдить мышь кнопками было, вероятно, самой лучшей идеей. Благодаря возможности подать сигнал той же рукой, которая перемещает курсор, пользователь получает полное ощущение контекстно-зависимого управления. Эта контекстная зависимость реализуется программно: путём делегирования обработки сигнала процедуре, называемой *обработчиком* или интерпретатором, связанной с отображением, в границах которого в данный момент находится курсор. При помощи подходящего программного обеспечения достигается удивительная гибкость вызова команд. Такие интерфейсные решения, как всплывающие и выпадающие меню и т.п., обладают достаточной мощностью даже при наличии всего одной кнопки мыши, однако для множества приложений многокнопочная мышь гораздо предпочтительнее, поэтому система Оберон предполагает доступность трех кнопок. Возможность назначить каждой из кнопок различные функции могла бы привести ко множеству недоразумений, если бы каждое приложение занималось этим по-своему. Однако, эта проблема легко устраняется при помощи некоторых «глобальных» соглашений. Так, в системе Оберон левая кнопка мыши в основном используется для *отметки* местоположения (установки курсора), средняя кнопка — для вызова различных *команд* (речь о которых пойдет далее), а правая — для *выбора* объектов на экране.

Не так давно в моду вошла возможность использовать перекрывающиеся окна, тем самым имитируя документы, наваленные грудой на столе. Мы считаем эту метафору не слишком убедительной. Частично скрытые окна обычно требуется поднимать наверх и делать полностью видимыми до того, как произвести над ними любые действия. Незначительные преимущества, которыми обладает такая схема, составляют разительный контраст с существенными усилиями по её реализации. Она может служить хорошим примером того, как выгода усложнения несоизмерима с его ценой. Как следствие, мы остановились на решении, которое гораздо проще реализовать, но которое при этом не имеет и существенных недостатков по сравнению с перекрывающимися окнами: размещение отображений в виде «мозаики», как показано на рис. 2.1.

### 2.2.2 Команды

Контекстно-зависимые команды с фиксированным (в каждом отдельно взятом отображении) смыслом следует дополнить командами более общего вида. Традиционно, такие команды вводятся с клавиатуры и представляют собой имя программы, которую следует исполнить. В системе Оберон ре-

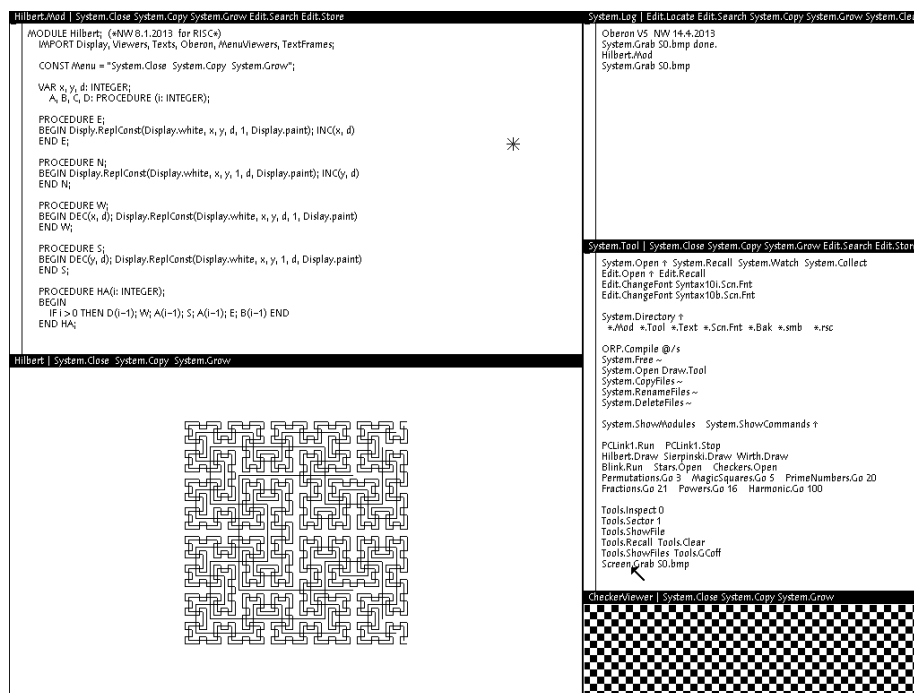


Рис. 2.1. Дисплей системы Оберон с мозаикой отображений.

ализован свой собственный, гораздо более гибкий, способ, который будет рассмотрен в этом разделе.

Прежде всего заметим, что общепринятая трактовка программы как текста, компилируемого в виде отдельной сущности, делает её слишком большой для элементарного действия, пригодного в качестве одной команды, наподобие вставки фрагмента текста при помощи мыши. В системе Оберон понятие элементарного действия отделено от понятия единицы трансляции. В роли действия выступает *команда*, представленная экспортированной процедурой, а единицей трансляции служит *модуль*. Таким образом, модуль может (причём обычно так оно и есть) реализовывать несколько либо даже много команд. Такую команду можно вызвать в любой момент, если навести курсор мыши на её имя *в любом тексте*, доступном в каком-либо отображении на дисплее, после чего нажать среднюю кнопку мыши. Имя команды имеет вид *M.P*, где имя *P* — идентификатор процедуры, а имя *M* — модуля, в котором эта процедура была определена. Как следствие, вызов любой такой команды может потребовать загрузки одного или нескольких модулей, если *M* еще не находится в оперативной памяти. Дальнейшие обращения к *M.P* будут происходить без какой-либо задержки, поскольку модуль *M* в этом случае будет уже загружен. Ещё одним последствием будет то, что модули никогда не выгружаются автоматически, поскольку очередная команда может к ним обратиться.

Назначением каждой команды служит изменение состояния каких-то ее аргументов. В традиционных системах, как правило, они задаются текстом, следующим за именем команды. Система Оберон поступает так же. Строго

говоря, команды Оберона представляют собой процедуры без параметров, но система обеспечивает им возможность получения текстовой позиции, из которой они были вызваны. Благодаря этому команда может прочесть и интерпретировать текст, следующий за её именем, то есть, свои фактические параметры. Однако, как чтение, так и интерпретация должны быть запрограммированы явно.

Текстовые параметры должны ссылаться на объекты, существующие до выполнения команды. Довольно часто в такой роли выступают результаты выполнения предыдущих команд. В большинстве операционных систем такими объектами, обеспечивающими взаимодействие команд, служат *файлы*. Система Оберон расширяет это соглашение: в роли связки между последовательными командами могут выступать не только файлы, но и любые глобальные переменные, поскольку модули не выгружаются после завершения работы команды, как это сказано выше.

Эта впечатляющая гибкость при неосторожном её применении открывает ящик Пандоры. Причина в том, что состояние глобальных переменных может полностью определить или повлиять на результаты работы очередной команды. Эти переменные образуют *скрытое состояние* в том смысле, что пользователь в общем случае не знает об их существовании и не имеет простого способа узнать их значение. Положительный же аспект использования глобальных переменных как интерфейса между командами в том, что некоторые из них могут быть видимыми на дисплее. Все отображения вместе со своим содержимым сведены в иерархическую структуру данных, корень которой хранится в глобальной переменной модуля *Viewers*. Части этой структуры, таким образом, составляют *видимое состояние* и отлично подходят на роль аргументов команд.

Одним из правил, которые могли бы быть включены в воображаемый Кодекс разработчика Оберона, было бы избегать скрытых состояний и вводить поменьше глобальных переменных. Однако, мы считаем это правило не догмой, а всего лишь руководством к действию. Существуют очень полезные исключения из данного правила, даже если переменные не имеют видимых частей.

Остается открытым вопрос, как назначать видимые объекты на роль аргументов команды. Самый очевидный случай — использовать последний выбранный объект. Процедура для получения результата выбора реализована в модуле *Oberon*. (Может работать только с выделенным текстом). Ещё одна возможность — ссылаться на положение в тексте курсора. Она применяется при вставке нового текста. Так, нажатие клавиши на клавиатуре считается отдельной командой и результатом ее работы будет вставка литеры в позицию, указанную курсором.

Имеется специальная возможность обращаться с отображениями как с аргументами команд: так называемый маркер-звёздочка. Он появляется в месте, указанном курсором мыши, после нажатия на клавиатуре клавиши, которой назначена эта операция (SETUP). Процедура *Oberon.MarkedViewer* находит отображение, в области которого располагается маркер. Когда команды, принимающие это отображение в качестве параметра, встречаются где-либо в тексте, их упоминание обычно тоже помечено звёздочкой. Использование содержимого выбранного отображения (будь то текст, графика или что-либо иное) в качестве параметра задаётся процедурой, реализующей команду.

Наконец, нельзя не отметить ещё одну замечательную возможность. Она основана на свойстве глобальных переменных сохранять свои значения на протяжении всего времени работы системы, а её польза становится очевидной при возникновении сбоев при выполнении команд. Ошибки в их работе вызывают прерывание. Появление такого прерывания свидетельствует об аномальном завершении команды. В худшем случае, глобальное состояние при этом может потерять свою целостность, но оно не потеряно, и очередная команда может начать свое выполнение поверх этого состояния. Обработчик прерывания открывает особое отображение, в котором выводится последовательность вызова процедур вместе с текущими значениями их локальных переменных. Эта информация позволяет программисту обнаружить причину ошибки.

### 2.2.3 Задачи

Как было показано выше, система Оберон отличается исключительной гибкостью механизма вызова команд. В число команд входят как вставка отдельной литеры или пометка какого-либо объекта на экране, так и вычисления, которые требуют часов или даже дней. Ещё большим отличием служит возможность выбора в качестве аргументов команд произвольных объектов, не ограничиваясь одними файлами. Самое же главное — фактическое отсутствие скрытого состояния. Несмотря на имеющуюся возможность, на практике состояние системы определяется тем, что видит пользователь.

Благодаря этому пропадает необходимость помнить долгую историю введенных ранее команд, запущенных программ, выбранных режимов работы и т.п. По нашему мнению, наличие в системе каких-либо режимов служит отличительным признаком её недружелюбия. В данный момент уже должно стать очевидным, что наша система обеспечивает своему пользователю возможность работать над несколькими задачами одновременно. Эти задачи представлены отображениями, содержащими текст, графику или другие видимые объекты. Пользователь переключается между задачами неявным образом, выбирая очередное отображение как аргумент очередной команды. Отличительные черты этой концепции — явный контроль пользователя над переключением задач и команды как неделимые элементарные действия.

В то же самое время, мы отнесли бы систему Оберон к классу однопроцессных (однопоточных) систем. Как следует понимать такой очевидный парадокс? Вероятно, лучше всего его можно пояснить на примере основного режима функционирования. Всё своё время, не занятое исполнением команд, процессор проводит в цикле непрерывного опроса источников событий. Этот цикл называется *центральным циклом*; он содержится в модуле *Oberon*, который по сути может считаться сердцем всей системы. В качестве двух предопределённых источников событий выступают клавиатура и мышь. При получении события от клавиатуры, управление передаётся обработчику так называемого *отображения в фокусе*, то есть такого, в котором в данный момент расположен текстовый курсор. События от мыши приходят обработчику отображения, в котором находится её курсор. Всё это возможно организовать как единственный непрерывный процесс.

Наличие одного процесса подразумевает невозможность его прерывания. Как следствие, команды не могут взаимодействовать с пользовате-



лем. Вся интерактивность заключается в выборе очередной команды для исполнения. Таким образом, в типичной программе для системы Оберон отсутствуют операции пользовательского ввода. Входные данные берутся командой из полученных при вызове параметров, и к этому моменту они уже окончательно сформированы.

Такой подход может показаться излишне ограничительным. На практике это не так, если оставаться в рамках однопользовательской работы. В таком режиме диалог с компьютером задаётся исключительно пользователем. Человек в состоянии поддерживать одновременное взаимодействие с несколькими процессами только в одном случае: когда выполнение команд требует существенного времени. Мы считаем, что времяёмкие операции лучше выполнять автономно: в распределенной системе на предназначенных для этого вычислительных серверах.

Основной выгодой работы в однопроцессной системе можно считать то, что переключение задач происходит исключительно в точках, заданных пользователем. Это устраняет необходимость сохранять состояние процесса до возобновления его выполнения. По той же причине задачи не могут влиять друг на друга неожиданным и неуправляемым образом из-за изменения совместно используемых данных. Благодаря этому, при проектировании системы становится возможным обойтись без всех тех защитных механизмов, которые призваны исключать возможные помехи. Достигнутое тем самым упрощение очень значительно.

Существенным отличием системы Оберон от многопроцессных систем является переключение задач только между командами, в то время как в последних оно может произойти практически после каждой машинной инструкции. Очевидно, это порождает разницу в сложности элементарных действий. В системе Оберон они достаточно крупные, что вполне приемлемо для однопользовательской системы.

В системе реализована возможность вставить в центральный цикл дополнительные команды опроса. Это необходимо при появлении новых источников событий. Замечательным примером служит поддержка сети, когда команды могут быть посланы с других рабочих станций. При выполнении центрального цикла просматривается список так называемых *дескрипторов задач*. Каждый дескриптор связан с командной процедурой. Два предопределённых события активируются только при наступлении заданных условий, т.е. наличия ввода с клавиатуры или изменения состояния мыши. Добавленные задачи обязаны предоставить *their own guard in the beginning of the installed procedure*.

Пример команд, порождаемых сетевой активностью (также называемых *запросами*), приводит нас к следующему вопросу: что произойдет, если в момент прихода запроса процессор будет занят исполнением другой команды? Очевидно, запрос будет потерян, если не принять каких-либо мер. Эта проблема легко решается буферизацией ввода. Буферизация реализована в драйверах всех устройств ввода, будь то клавиатура или сетевой интерфейс. Готовность устройства ввода вызывает прерывание, его обработчик считывает данные и буферизует их. Подчеркнём, что механизм обработки прерываний полностью скрыт в драйверах — компонентах системы самого низкого уровня. Механизм обработки аппаратных прерываний не использует средства выбора и переключения задач. После обработки прерывания управление возвращается в ту же самую точку, делая этот механизм про-

зрачным с точки зрения обычных программ. Подобно каждому другому правилу, своё исключение найдется и для этого: если обработчик прерывания от клавиатуры обнаруживает, что нажата клавиша останова, управление возвращается в центральный цикл.

#### 2.2.4 Командные тексты как конфигурируемые меню

Очевидно, что концепции отображений, задающих свою собственную интерпретацию щелчков мышью, команд, которые можно вызвать из любого видимого на дисплее текста, любого видимого объекта, который можно выбрать в качестве интерфейса между командами, самих команд, которые подразумевают отсутствие диалога с пользователем и выполняются «в один присест» — всё это оказало существенное влияние на стиль программирования для системы Оберон и коренным образом изменило способ пользования компьютером. Лёгкость и гибкость, с которыми фрагменты текста могут выбираться, перемещаться, копироваться, назначаться на роль команд и их аргументов, радикально снижают потребность что-либо набирать на клавиатуре. Доминирующим устройством ввода становится мышь, клавиатура служит лишь для ввода текстовых данных. Это подчёркивается использованием так называемых *командных текстов*, композиций часто используемых команд, которые обычно выводятся в группе системных отображений (на рис. 2.1 — более узкая группа в правой части экрана). Потребность вводить с клавиатуры команды попросту исчезает! Как правило, они уже где-то видны на дисплее и готовы к вызову. Зачастую пользователь составляет свой собственный командный текст для каждого проекта, над которым работает. Эти тексты можно считать примером персональных меню, каждое из которых может быть гибко настроено.

Самая очевидная выгода редкого набора команд вручную в том, что их можно называть осмысленными словами. Например, операцию копирования можно назвать *Copy* вместо *cp*, переименования — *Rename* вместо *rn*, вывод содержимого каталога файлов — *Directory* вместо *ls*. Уходит нужда запоминать бесконечный список таинственных сокращений, который служит еще одним признаком недружественных систем.

Однако, влияние концепций Оберона не ограничивается стилем пользования компьютером. Они также задают способ проектирования программ с учётом их взаимодействия со средой. Так, определение в ядре системы абстрактного типа *Text* предполагает его использование для передачи входных и выходных данных вместо файлов во многих случаях. Полученное преимущество кроется в возможности непосредственного редактирования этого текста. Например, вывод команды *System.Directory* создает требуемую сводку информации о файлах в виде (отображаемого) текста. Его часть или сам текст в целом могут быть выбраны и скопированы в другие тексты обычными командами редактирования (щелчками кнопок мыши). Кроме того, текст может быть подан на вход компилятора. Тем самым открывается возможность скомпилировать программу, запустить её, а затем изменить текст и скомпилировать его ещё раз, ничего при этом не сохраняя на диске. Вездесущая возможность редактирования текстов вместе с длительным временем существования глобальных данных (в некоторых отображениях) позволяет опустить множество шагов, которые не приносят ничего существенного в процесс решения реальной задачи.

### 2.2.5 Расширяемость

Важной целью разработки системы Оберон было достижение расширяемости. Система должна с лёгкостью дополняться новыми возможностями путём добавления модулей, использующих уже доступные ресурсы. Не менее важно, что это позволит сократить систему до набора реально используемой в данный момент функциональности. Например, редактор документов, работающий с документами без графики, не должен требовать загрузки развитого графического редактора, рабочая станция без подключения к сети — сетевого программного обеспечения, а офисная система — компилятора или ассемблера. Аналогично, система, реализующая новую разновидность экранного фрейма, не должна включать процедуры управления отображениями, содержащими такие фреймы. Вместо этого, следует пользоваться существующими средствами управления отображениями. Непрерывный рост требований к объёму памяти многих популярных систем вызван нарушением этих фундаментальных инженерных принципов. Потребность системы в памяти объёмом несколько мегабайт, хоть и считается многими приемлемой, абсурдна и может быть ещё одним признаком недружелюбия к пользователям или, возможно, излишнего дружелюбия к производителям. Причина такого аппетита — все та же: неудовлетворительная расширяемость.

Мы не ограничиваемся исключительно процедурной расширяемостью, которую легко реализовать. Следует подчеркнуть, что в ходе расширения возможно не только добавление новых процедур и функций, но и определение новых типов данных на основе тех, которые уже доступны в системе: расширяемость по данным. Например, в графической подсистеме следует определять фреймы с поддержкой графики на основе других фреймов, доступных в модуле базовой поддержки дисплея, расширив их нужными для графики атрибутами.

Такой подход требует поддержки со стороны языка программирования. В языке Оберон реализовано в точности то, что требуется; этот механизм получил название *расширение типа*. Решение о разработке нового языка было принято именно ради него. Модуль-2 была бы вполне подходящим выбором, если бы не отсутствие такой возможности. Влияние языка на структуру системы было очень велико, а полученный результат — самым благоприятным. По мере продвижения проекта, различные дополнения были реализованы с поразительной лёгкостью. Одно из них описано в конце этой книги. Базовая система, тем не менее, обладает очень скромными потребностями в ресурсах (см. таблицу в конце разд. 2.3).

### 2.2.6 Динамическая загрузка

Вызов команд, реализованных в модулях, которые в данный момент не находятся в оперативной памяти, подразумевает возможность их загрузки, причём со всеми другими модулями, от которых они зависят. Обращение к загрузчику может произойти и в других ситуациях: it may also occur through programmed procedure calls. Такая возможность незаменима для достижения настоящей расширяемости. Модули должны загружаться по требованию. Например, редактор документов может загрузить пакет поддержки графики, если в текущем документе обнаружится графический элемент, но

В системе Оберон понятие программы-компоновщика отсутствует. Модуль связывается с импортируемыми модулями только при загрузке в оперативную память, а не до этого. Как следствие, каждый модуль присутствует в системе лишь в двух экземплярах: в оперативной памяти (связанным) и на внешнем накопителе (несвязанным, в виде файла). Устранение множества копий одного и того же модуля в нескольких объектных файлах, сгенерированных в процессе компоновки — ключевой аспект экономии внешней памяти. Громадные скомпонованные исполнимые файлы в системе Оберон не встречаются, и каждый модуль может свободно использоваться многократно.

Если попытаться разбить систему на хорошо различимые составные части, самыми крупными окажутся модули. Как следствие, описывать структуру системы лучше всего на их основе. Благодаря явному описанию интерфейсов модулей, нетрудно выразить их взаимную зависимость в виде ориентированного графа, дуги которого представляют импорт одного модуля другим.

Иерархия модулей базовой системы показана в таблице прямого импорта и в виде графа на рис. 2.2. Для упрощения рисунка на нем не отображены операции прямого импорта в случае, когда существует другой путь, связывающий эти вершины. Например, модуль *Files* импортирует *Kernel*, но этот прямой импорт не показан, поскольку найдётся другой путь в графе от *Kernel* к *Files*, проходящий через *FileDir*.

[illegible]

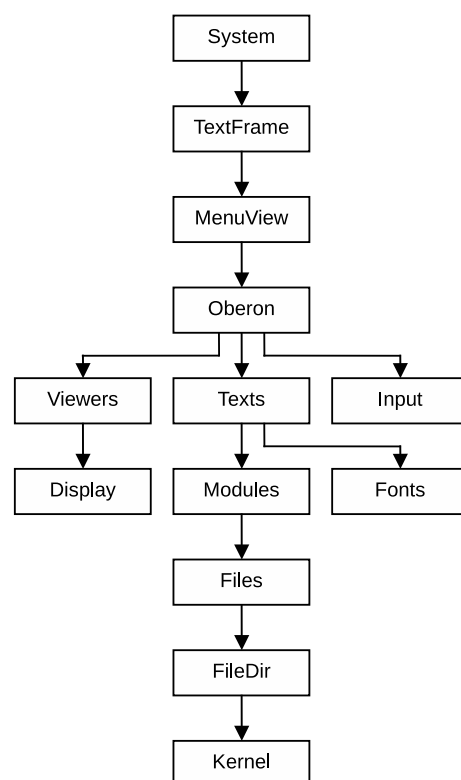


Рис. 2.2. Структура ядра системы Оберон.

Модули, имена которых имеют форму множественного числа, как правило содержат определение абстрактного типа, который экспортируется совместно со своими операциями. Примером такого модуля могут служить *Files*, *Modules*, *Fonts*, *Texts*, *Viewers*, *MenuViewers* и *TextFrames*. Модули с именами в форме единственного числа обычно обозначают ресурс, которым управляет этот модуль, будь то глобальная переменная или внешнее устройство. Эта переменная или устройство скрыты (не экспортируются), а доступ к ним осуществляется через экспортированные данным модулем процедуры. В качестве примеров можно привести все драйвера устройств, такие как *Input* для клавиатуры и мыши, *Kernel* для памяти и дисковых накопителей, а также *Display*. Исключение из правила составляют командные модули, имена которых в основном выбираются в соответствии с основными действиями, которые они представляют, подобно *System* и *Edit*.

Как уже было сказано ранее, модуль *Oberon* представляет собой сердце всей системы. Он реализует центральный цикл, который получает управление каждый раз после завершения выполнения очередной команды, как нормального, так и аварийного. Модуль *Oberon* экспортирует некоторые вспомогательные процедуры, но главные из его экспортов — процедура вызова команд (*Call*) и доступа к тексту, задающему параметры команды, посредством переменной *Oberon.Par*. Помимо этого, в нём реализованы глобальные экспортируемые переменные: *текст журнала*. Текст журнала обычно служит для выдачи подсказок и коротких сообщений об ошибках команд. Он выводится в *отображение журнала*, которое открывается автоматически в ходе инициализации модуля *System*. Ещё модуль *Oberon* содержит пару глобальных маркеров, видимых на дисплее: *курсор мыши* и *указатель-звёздочку*. Он экспортирует процедуры для рисования и стирания этих маркеров, а также позволяет installation of different patterns for them.

Система, изображенная на рис. 2.2, предоставляет базовые возможности создания и редактирования текстов, которые могут затем сохраняться в файлах. Все прочие функции реализуются модулями, которые должны быть добавлены обычным путём, загрузкой по требованию. Они включают, среди прочего, компилятор, обмен данными по сети, редакторы документов и все прочие программы, какие только могут быть созданы пользователями. Особое внимание, которое при проектировании системы было уделено модульности, борьбе с ненужными «рюшами» и концентрации на самом необходимом в составе ядра, было вознаграждено достижением особой компактности. Несмотря на то, что такое свойство может казаться неважным в наше время непрерывного удешевления памяти, мы считаем его крайне желательным. Особенно бы хотелось обратить внимание читателя на то, что между размером системы и её надёжностью существует взаимосвязь. Также, мы не считаем хорошей инженерной практикой чрезмерное потребление ресурсов лишь потому, что они дешёвы. Таблица, приведённая ниже, содержит список модулей ядра и основных приложений вместе с размером их кода (в машинных словах), статических переменных (в байтах) и количеством строк исходного текста.

module	code	data	lines
Kernel	1123	8244	263
FileDir	1963	60	352
Files	2360	148	505
Modules	1214	112	226
Input	186	32	79
Fonts	628	56	115
Display	1033	84	190
Viewers	1324	104	206
Texts	2906	204	537
Oberon	1679	288	410
MenuViewers	1513	56	208
TextFrames	5786	292	874
System	2134	72	418
Edit	1096	1104	232
	24945	10856	4615
ORS	1762	992	319
ORB	2348	408	437
ORG	6699	34976	1125
ORP	5994	144	974
	16803	36520	2855
Graphics	3484	564	685
GraphicFrames	2832	288	498
Draw	690	40	164
Rectangles	649	40	118
Curves	1765	72	241
	9420	1004	1706

## 2.4 Обзор следующих глав

Реализация системы велась в направлении снизу вверх. Понятно, что модули более высокого уровня импортируют модули более низкого и не могут функционировать при их отсутствии. С другой стороны, описание системы лучше вести в обратном порядке, сверху вниз. Причина этого в том, что система проектировалась с учётом области её применения и ожидаемых функциональных возможностей. Декомпозиция в иерархию модулей обусловлена использованием вспомогательных функций и абстракций, более детальное объяснение которых отложено на будущее до момента, когда их назначение станет полностью ясным. Исходя из этих соображений, продолжим знакомство с системой, продвигаясь именно сверху вниз.

Главы 3—?? описывают *внешнее ядро* системы. Глава 3 сосредоточена на динамических аспектах. В частности, в этой главе представлены такие фундаментальные исполнимые сущности, как *задача* и *команда*. В модели управления задачами системы Оберон делается различие между категориями *интерактивных* и *фоновых задач*. Интерактивные задачи представлены на дисплее прямоугольными областями, которые называются *отображениями*. Фоновые задачи такой связи с дисплеем не требуют. Они выполняются с низким приоритетом в моменты, когда взаимодействие с пользо-

вателем не ведётся. Хорошим примером фоновой задачи может служить сборщик мусора. Как интерактивные, так и фоновые задачи назначаются на исполнение единственным процессом при помощи планировщика задач. Команды в системе Оберон — явные атомарные единицы интерактивных действий. Они реализуются в виде экспортированных процедур без параметров и заменяют собой более тяжеловесное понятие программ, известное нам по более традиционным операционным системам. Далее в этой главе будет дано определение программного инструментария как набора логически связанных команд. В завершение рассматриваются основы инструментария управления системой.

Глава ?? объясняет подсистему управления дисплеем. Она начинается обсуждением нашего выбора иерархической мозаичной стратегии для размещения отображений. Затем следует детальное изучение роли отображений в системе Оберон. Тип *Viewer* рассмотрен как класс объектов с открытым интерфейсом на основе передачи сообщений, реализующий концептуальную основу для глубокой расширяемости. Далее будет показано, что отображения суть специальный случай так называемых *фреймов*, которые могут быть вложенными. Рассматривается категория отображений стандартного вида, содержащих фрейм меню и фрейм содержимого. Следующая тема этой главы — работа с курсором. А cursor in Oberon is a marked path. Как менеджер отображений, так и обработчик курсора работают в абстрактном пространстве логического дисплея без привязки к конкретным физическим мониторам. Это позволяет действовать унифицированно, независимо от количества и характеристик доступных мониторов. Например, автоматически достигается беспрепятственный переход курсора через границы экранов. В дальнейшем рассматривается лаконичный, но полный набор растровых операций, используемых для отрисовки текста и графики. Глава завершается обзором дисплейного инструментария.

Глава ?? вводит понятие текста. Наличие абстрактного типа данных *Text*, интегрированного в систему Оберон, является её визитной карточкой. В этой главе будут обсуждаться многочисленные фундаментальные концепции. Например, текст может быть возвращён в качестве результата работы одной команды, отредактирован пользователем и подан на вход следующей команде. Сами команды имеют текстовое представление в виде своего имени *M.P*, за которым тоже в виде текста задан список параметров. Как следствие, любая команда может быть вызвана напрямую из текста (называемого *командным*) простым выбором с помощью мыши. Однако, суть этой главы в том, чтобы представить текстовую подсистему Оберона как пример опытной разработки модульного программного обеспечения. Вопросы управления текстом и его отображения удалось в ходе этого исследования отделить друг от друга. Как менеджер текстов, так и возможности дисплея по его выводу задаются абстрактными публичными интерфейсами над скрытыми структурами данных. Финал главы затрагивает вопросы управления шрифтами и инструментарий редактирования текста системы Оберон.

Главы ??—?? посвящены *внутреннему ядру*, по-прежнему следуя в направлении сверху вниз. В главе ?? рассматривается загрузчик программных модулей и поясняются мотивы появления типа данных *Module*. Также глава включает вопросы управления памятью, содержащей код программы, и определяет формат хранения скомпилированных модулей в виде объект-



ных файлов. В дальнейшем в ней обсуждаются вопросы связывания раздельно компилируемых модулей и ссылок на объекты, определённые в других модулях.

Глава ?? посвящена подсистеме работы с файлами, критически важной части системы, поскольку доступ к файлам требуется практически каждой программе. Глава разбита на две не связанные между собой части. В первой определяется тип *File* и описывается структура файлов, то есть способ их хранения на дисковых накопителях with its sequential characteristics, во второй — понятие каталога файлов и его организация в виде В-дерева, позволяющего быстрый поиск.

Вопросы управления оперативной памятью служат предметом главы ??.

Централизованное управление памятью было одним из ключевых проектных решений, гарантирующих эффективное и экономное её использование. В главе объясняется разбиение памяти на конкретные области. Однако, её основная тема — распределение динамической памяти в области под названием *куча*. Приводятся подробности реализации алгоритмов выделения (предопределённая процедура *NEW*) и освобождения (известного как «сборка мусора») памяти.

Драйвера устройств находятся на самом низком уровне иерархии модулей. Они описаны в главе ??, которая содержит драйвера для некоторых популярных стандартов сопряжения аппаратуры. Прежде всего, это последовательный синхронный интерфейс PS/2. Он служит для подключения клавиатуры и мыши. Второй интерфейс — SPI, стандарт синхронной двунаправленной последовательной шины. С его помощью происходит обмен данными с SD-картой (флеш-памятью) и работа с сетью. Наконец, стандарт RS-232 обычно используется для простых и медленных устройств. Обмен данными при этом двунаправленный и асинхронный.

Вторая часть книги включает в себя главы ??—?? и посвящена самым первым приложениям базовой системы Оберон. По этой причине каждая из глав не зависит от других и ссылается только на главы 3—??.

Несмотря на то, что система Оберон хорошо подходит для работы на автономных рабочих станциях, возможность объединения компьютеров в сеть считается фундаментальной. Модуль *Net* обеспечивает передачу файлов между рабочими станциями, объединёнными в сеть топологии «шина». Его реализация описана в главе ??.

В ней рассматриваются не только проблемы доступа по сети, обработки сбоев передачи данных и сетевых коллизий, но и поиска корреспондентов по именам. Найденные решения воплощены в удивительно компактном модуле, который использует сетевой драйвер из главы ??.

После объединения в сеть рабочих станций, возникает необходимость в центральном сервере. Такой сервер, предназначенный для обмена файлами, сетевой печати и хранения электронной почты, представлен в главе ??.

Его реализация основана на расширении модуля *Net* из главы ?? и может служить впечатляющим примером применения задач, рассмотренных в разделе 2.2. В завершение отметим, что сервер работает на компьютере без присмотра пользователя. Это обстоятельство выдвигает повышенные требования устойчивости не только к сбоям во время передачи, но и к получению данных, не соответствующих предписанным форматам.

Реализация этих сетевых служб наглядно показывает, что принятая в системе Оберон однопоточная архитектура не обязана ограничиваться одно-

пользовательским режимом. Необходимость непрерывной обработки каждого полученного запроса вплоть до его завершения может быть приемлемой при условии, что никакой запрос не займёт процессор слишком долго, что в основном и происходит в представленных сетевых приложениях. Запросы, приходящие во время обработки, помещаются в очередь. Как следствие, процессор обрабатывает их последовательно, без переключения с одного на другой. Это благотворно сказывается на общей производительности в силу отсутствия накладных расходов на частое переключение задач.

В главе ?? описан компилятор языка Оберон. Он транслирует исходный текст программы на Обероне в объектный код, т.е. в последовательности машинных команд целевого компьютера. Лежащие в его основе принципы и технологии рассматриваются в работе [6]. Перед изучением компилятора следует ознакомиться как с его входным языком, так и с целевой RISC-архитектурой. Они представлены в приложении к этой книге.

Несмотря на то, что компилятор выглядит как прикладная программа, он играет особую роль, поскольку система (и сам компилятор) реализованы на его входном языке. Вместе с текстовым редактором он послужил основным средством разработки системы. Использование сравнительно прямолинейных алгоритмов синтаксического анализа и организации символьных таблиц сделало его весьма компактным. Однако, основной вклад в достигнутый результат принадлежит определению языка в котором отсутствуют сложные конструкции и редко используемая «синтаксическая глазурь».

Как сам компилятор, так и посвящённая ему глава, могут быть разбиты на две основные части. Первая включает средства работы со входным языком без привязки к какому-либо целевому компьютеру: *лексический* и *синтаксический анализаторы*. Эта часть, как следствие, наиболее интересна читателям. Вторая же часть по сути не зависит от входного языка, но существенно связана с системой команд целевого компьютера. Она называется *кодогенератором*.

Тексты играют в системе Оберон главную роль, и поэтому встроенный редактор занимает в ней не менее важное место. В главе ?? мы опишем ещё один редактор, предназначенный для работы с графическими объектами. Для начала, в виде объектов представляются только горизонтальные и вертикальные линии, а также короткие заголовки. Основное отличие от редактирования текста заключается в том, что координаты графических объектов не могут быть автоматически вычислены из координат их предшественников, поскольку графические объекты образуют скорее множество, чем последовательность. Каждый графический объект имеет свои собственные, независимые от других, координаты. Эта казалось бы небольшая разница имеет далекоидущие последствия и проникает в самую глубину архитектуры. Как следствие, текстовый и графический редакторы практически не имеют между собой ничего общего, за исключением, возможно, декомпозиции на три части. В модуле нижнего уровня определяется структура данных для представления текста или графики, соответственно, вместе с нужными для работы с ней процедурами, такими как поиск, вставка или удаление. Средний модуль реализует соответствующий фрейм и все относящиеся к нему процедуры отображения и обработчики событий мыши и клавиатуры. Модули верхнего уровня — инструментальные: *Edit* и *Draw*, соответственно. Представленный графический редактор особенно интересен тем, что он служит убедительным примером расширяемости системы Оберон.

Графический редактор полностью интегрируется в систему, он встраивает свои графические фреймы в стандартные отображения с полосой меню и использует возможности текстовой подсистемы для заголовков элементов. Наконец, новые разновидности элементов могут быть реализованы простым добавлением модулей, без доработки и даже без перекомпиляции существующих. В самой главе ?? найдется пара таких примеров: прямоугольники и окружности.

Система Draw активно использовалась для подготовки принципиальных электрических схем. Также это приложение демонстрирует концепцию, которая может быть полезной и в других программах: рекурсивное определение понятия объекта. Множество объектов само по себе может считаться объектом, в том числе получить собственное имя. Такой объект называется *макросом*. Реализация поддержки макросов представляет собой нетривиальную задачу, если ставится цель сделать её также расширяемой, то есть, чтобы макрос никоим образом не ссылался на типы своих составляющих даже при чтении файлов, в которых макросы были ранее сохранены.

В главе ?? представлены ещё два инструмента: первый из них применяется для установки системы Оберон на новый компьютер, а второй — для восстановления файловой системы после сбоев. Несмотря на сравнительную редкость использования, инструмент установки был незаменим при разработке системы, а инструменты сопровождения и восстановления данных бесценны при возникновении сбоев, которых, увы, не избежать! Материал главы ?? встречается в литературе достаточно редко.

Глава ?? посвящена инструментам, которые не используются системой Оберон, но могут оказаться нужными в некоторых приложениях. Один из них реализует передачу данных по протоколу на основе стандарта RS-232, описанного в главе ???. Другой — стандартный набор основных математических функций. Наконец, третий инструмент предназначен для создания новых макросов системы Draw.

Третья часть этой книги посвящена подробному описанию аппаратных средств. В главе ?? определяется процессор, для которого генерирует код наш компилятор. Целевой компьютер имеет действительно простой процессор с регулярной архитектурой, который называется RISC, исполняет всего лишь 14 машинных инструкций и не выпускается промышленностью. Вместо этого он реализован на основе ППВМ (программируемой пользователем вентиляционной матрицы), благодаря чему структура процессора может быть описана в книге до малейших деталей. Он представляет собой достаточно прямолинейную реализацию архитектуры фон Неймана с регистровым файлом и арифметико-логическим устройством, включающим блок операций над числами с плавающей запятой. Популярные усовершенствования, такие как конвейеризация и кэш-память, не применялись ради сохранения прозрачности и простоты конструкции. Схема процессора определяется на языке *Verilog*.

Глава ?? описывает аппаратное окружение, в котором работает процессор. Оно состоит из интерфейсов оперативной памяти и всех внешних устройств.

## Литература

1. N. Wirth. The programming language Oberon. *Software — Practice and Experience* 18, 7, (July 1988) 671-690.
2. M. Reiser and N. Wirth. *Programming in Oberon — Steps beyond Pascal and Modula*. Addison-Wesley, 1992. ISBN 0-201-56543-9
3. N. Wirth and J. Gutknecht. The Oberon System. *Software — Practice and Experience*, 19, 9 (Sept. 1989), 857-893.
4. N. Wirth. Ceres-Net: A low-cost computer network. *Software — Practice and Experience*, 20, 1 (Jan. 1990), 13-24.
5. M. Reiser. *The Oberon System — User Guide and Programmer's Manual*. Addison-Wesley, 1991. ISBN 0-201-54422-9
6. N. Wirth. *Compiler Construction*. Addison-Wesley, Reading, 1996. ISBN 0-201-40353-6

## Глава 3

# Управление задачами

Возможность выполнить любую мыслимую *задачу* — именно это в конечном счёте и превращает вычислительное устройство в гибкий универсальный инструмент. Как следствие, вопросы представления и управления работой задач — среди важнейших решений, которые предстоит принять при проектировании любой операционной системы. Безусловно, не следует ожидать, что единственный подход окажется идеальным для всех систем и во всех сценариях их использования. Например, вероятнее всего различные решения понадобятся в случае закрытой системы на базе мейнфрейма, обслуживающей большое количество пользователей в режиме разделения времени, и персональной рабочей станции, с которой работает один пользователь в режиме активного диалога.

В случае системы Оберон мы сознательно сосредоточились на втором варианте. Более точно, мы ориентировали возможности Оберона по управлению задачами на случай однопользовательской интерактивной персональной рабочей станции с возможностью подключения к локальной сети.

Глава начинается разделом 3.1, в котором будет дано точное определение технического термина *задача*. Раздел 3.2 продолжит её детальным объяснением стратегии планирования выполнения задач. Далее, в разделе 3.3 будет введено понятие *команды*. Наконец, раздел 3.4 содержит обзор предопределённых системных инструментариев, то есть коллекций однородных команд, посвящённых одному и тому же вопросу, например, управлению и диагностике системы, управлению дисплеем или работе с файлами.

### 3.1 Понятие задачи

В целом, все задачи в системе Оберон могут быть отнесены к двум категориям: *интерактивных задач* и *фоновых*. Попросту говоря, интерактивные задачи привязаны к определённым областям на дисплее и зависят от взаимодействия пользователя с их содержимым. С другой стороны, фоновые задачи оперируют на уровне системы в целом и не связаны ни с какими видимыми сущностями.

### 3.1.1 Интерактивные задачи

Каждая интерактивная задача представлена так называемым *отображением*. Отображения лежат в основе интерфейса дисплейной подсистемы. Их довольно разнообразные роли сведены в абстрактный тип данных *Viewer*. Подробнее работа дисплейной подсистемы разбирается в главе ?? . На данный момент достаточно знать, что отображения представляются в виде прямоугольников на экране и они неявно служат *carriers* интерактивных задач. Рис. 3.1 демонстрирует типичный для системы Оберон вид экрана, который может быть разбит максимум на семь отображений, соответствующих семи независимым активным интерактивным задачам.

Чтобы строить дальнейшие рассуждения на более твёрдом грунте, определим тип *Viewer* на языке Оберон, слегка добавив абстракции:

```
Viewer = POINTER TO ViewerDesc;
ViewerDesc = RECORD
    X, Y, W, H: INTEGER;
    handle: Handler;
    state: INTEGER
END;
```

где  $X$ ,  $Y$ ,  $W$  и  $H$  задают прямоугольную область экрана, отведённую отображению: координаты верхнего левого угла  $X$ ,  $Y$ , ширину  $W$  и высоту  $H$ . Переменная *state* содержит информацию о текущей видимости отображения (visible, closed, covered), в то время как *handle* представляет его функциональный интерфейс. Тип обработчика

```
Handler = PROCEDURE (V: Viewer; VAR M: ViewerMsg);
```

где *ViewerMsg* — некоторый базовый тип сообщений, точное определение которого в данный момент не представляет интереса:

```
ViewerMsg = RECORD ... (*basic parameter fields*) END;
```

Отметим, что далее будет использоваться объектно-ориентированная терминология. Такой подход вполне оправдан, поскольку *handle* имеет процедурный тип (является *обработчиком*) и значение, определяемое конкретным отображением. Вызов процедуры вида  $V.handle(V, M)$  может, следовательно, интерпретироваться как посылка сообщения, которое будет обработано методом отображения-получателя  $V$ .

Мы признаём, что наш подход на основе обработчиков существенно отличается от традиционной объектно-ориентированной модели. Традиционная модель *замкнута* в том смысле, что данный класс объектов может обрабатывать только фиксированный набор сообщений. В отличие от неё, парадигма обработчиков *открыта*, потому что она определяет только корень (*ViewerMsg*) потенциально бесконечного дерева типов сообщений, построенного при помощи расширения. Например, конкретная реализация обработчика может принимать сообщения типа *MyViewerMsg*, где

```
MyViewerMsg = RECORD (ViewerMsg)
    mypar: MyParameters
END;
```



Рис. 3.1. Типичный вид экрана системы Оберон с группой инструментов справа.

является расширением типа *ViewerMsg*.

Следует ещё раз отметить исключительную гибкость нашей открытой объектно-ориентированной модели. В частности, расширение множества типов сообщений, которые может обработать объект, является всего лишь деталью его реализации, то есть, не оказывает никакого влияния ни на интерфейс объекта с точки зрения компилятора, ни на целостность системы. В то же время, нельзя не признать, что такая хорошая расширяемость не даётся даром. Цена, которую придётся за неё заплатить — необходимость явной *диспетчеризации сообщений* во время работы системы. В следующих главах мы воспользуемся этим свойством.

Возвращаясь к вопросу реализации механизма задач, добавим, что каждая посылка сообщения отображению вызывает запуск или возобновление интерактивной задачи, которую оно представляет.

### 3.1.2 Фоновые задачи

Фоновые задачи Оберона априори не связаны с каким-либо конкретным агрегатом в системе. С технической точки зрения они представляют собой экземпляры абстрактного типа данных, заданного декларацией типов *Task* и *TaskDesc* вместе с базовыми операциями *NewTask*, *Install* и *Remove*:

```
Task = POINTER TO TaskDesc;
TaskDesc = RECORD state: INTEGER; handle: PROCEDURE END;
```

```
PROCEDURE NewTask(h: PROCEDURE; period: INTEGER): Task;
PROCEDURE Install (T: Task);
PROCEDURE Remove (T: Task);
```

Процедуры *Install* и *Remove* вызываются явно, чтобы изменить состояние конкретной задачи с *offline* на *idle* и с *idle* на *offline*, соответственно. Installed tasks поочередно переходят в состояние *active*, то есть, исполняются. Назначенные задачам обработчики представляют собой простые процедуры без параметров, которые реализуют как требуемые действия, так и проверку условий для их исполнения с одним исключением: возобновление задачи может быть отложено на некоторое время. Длительность задержки указывается в миллисекундах при создании задачи.

Следующие два примера конкретных фоновых задач, возможно, смогут улучшить понимание наших объяснений. Первый из них — общесистемная процедура сборки мусора, освобождающая неиспользуемую память. Второй пример — сетевой монитор, принимающий входящие данные из локальной сети. В обоих случаях состояние задач полностью определяется глобальными системными переменными. Мы вернёмся к их рассмотрению в главах ?? и ??, соответственно.

Невозможно завершить этот раздел, не сделав важного вывода. Передача управления между задачами реализована в системе Оберон как обыкновенные вызовы и возвраты из обычных же процедур (точнее, переменных процедурного типа). Вытеснение одной задачи другой невозможно. Из этого можно заключить, что периоды активности задач образуют последовательность и могут выполняться единственным потоком управления. Подобное упрощение воздаёт сторицей: исчезает необходимость блокировки при совместном доступе к ресурсам, а тупики попросту невозможны.

## 3.2 Планировщик задач

Начнём этот раздел с формулировки общего предположения: будем считать, что в любой заданный момент времени в системе найдётся некоторое количество well-determined задач, готовых к исполнению. Напомним, что все задачи делятся на две категории: интерактивные и фоновые. Они существенно отличаются между собой условиями активации и возобновления, а также приоритетом выполнения. Интерактивные задачи активируются и возобновляются исключительно действиями пользователя и выполняются с высоким приоритетом. В отличие от них, фоновые задачи опрашиваются с низким приоритетом.

Как нам уже известно, интерактивные задачи активируются посылкой сообщений. Типы сообщений, используемых для этой цели: *InputMsg* и *ControlMsg*, уведомляющие о событиях от клавиатуры и мыши, соответственно. В несколько упрощённом виде их объявления выглядят так:

```
InputMsg = RECORD (ViewerMsg)
    id: INTEGER;
    X, Y: INTEGER;
    keys: SET;
    ch: CHAR
END;
```



```
ControlMsg = RECORD (ViewerMsg)
  id: INTEGER;
  X, Y: INTEGER
END;
```

Поле *id* задаёт точный вид запроса, получение которого возобновило выполнение задачи. В случае сообщения типа *InputMsg* возможны запросы *consume* (литеру, сохранённую в поле *ch*) и *track* (мышь, начиная с состояния, заданного полями *keys*, *X* и *Y*). Для сообщения типа *ControlMsg* альтернативами будут *mark* (отображение, включающее точку с координатами *X*, *Y*) или *neutralize*. Операция пометки состоит в перемещении глобальной системной отметки (обычно представленной на экране звездочкой) в текущую позицию курсора мыши. Нейтрализация отображения заключается в удалении с него всех отметок и графических атрибутов.

Все аспекты управления задачами реализованы в одном модуле под названием *Oberon*. В частности, в нём объявляется абстрактный тип данных *Task* и типы сообщений *InputMsg* и *ControlMsg*. Однако, самый большой вклад этого модуля — планировщик задач (часто называемый «циклом Оберона»), который может считаться центром динамики всей системы.

Переход к изучению планировщика во всех подробностях потребует некоторой предварительной подготовки. Мы начнём с понятия *отображения в фокусе*. По определению, это такое отображение, в которое направляется ввод с клавиатуры. Отметим, что отображению в фокусе соответствует задача в фокусе, благодаря прямой взаимосвязи отображений с задачами.

В модуле *Oberon* реализованы такие сущности, относящиеся к отображению в фокусе: глобальная переменная *FocusViewer*, процедура *PassFocus* для передачи фокуса другому отображению и вариант *defocus* сообщения *ControlMsg* для уведомления прежнего отображения в фокусе о его переходе.

Подробности реализации абстрактного типа данных *Task* скрыты от его клиентов. Достаточно знать, что все дескрипторы задач сохраняются в циклическом списке, где каждый узел ссылается на задачу, которая была активирована предыдущей. Гарантируется, что список не может быть пустым, потому что упомянутая ранее задача сборки мусора ставится на выполнение в ходе загрузки системы и играет в списке роль барьера.

Ниже приводится немного абстрагированная версия реального кода планировщика, работающего со списком задач. Он расположен в процедуре *Loop* модуля *Oberon*.

```
получить координаты и состояние кнопок мыши;
REPEAT
  IF клавиатура готова ко вводу THEN прочесть литеру
  IF литера = escape THEN
    отправить всем отображениям сообщение neutralize
  ELSIF литера = mark THEN
    отправить сообщение mark отображению, указанному мышью
  ELSE отправить сообщение consume отображению в фокусе
  END;
  получить координаты и состояние кнопок мыши
  ELSIF хотя бы одна кнопка мыши нажата THEN
```

```

REPEAT
    отправить сообщение track отображению, указанному мышью;
    получить координаты и состояние кнопок мыши
UNTIL все кнопки отпущены
ELSE (*ни одна кнопка не нажата*)
    отправить сообщение track отображению, указанному мышью;
    извлечь очередную задачу из списка и сделать её текущей;
    вызвать её обработчик (если прошёл заданный период времени)
    получить координаты и состояние кнопок мыши
END
UNTIL FALSE

```

Система выполняет без вытеснения последовательность процедур (задач). Интерактивные задачи возобновляют работу при получении входных данных с клавиатуры, мыши или любого другого источника. Фоновые задачи обслуживаются в циклическом порядке, интерактивным задачам даётся приоритет.

До этого момента мы сознательно избегали в своих объяснениях ситуаций незапланированного поведения программ. Пришло время рассказать, каким образом продолжается работа системы в случае аварийного завершения задачи или, другими словами, возникновения *прерывания*. Рассуждая на уровне задач (снова чуть более абстрактно, чем это реализовано на деле), мы можем выделить три последовательных действия для восстановления после программного сбоя:

```

восстановление после программного сбоя =
BEGIN сохранить текущее состояние системы;
    вызвать зарегистрированный обработчик прерываний;
    возобновить работу планировщика задач
END

```

В целом, состояние системы определяется значениями всех глобальных и локальных переменных в заданный момент времени. Обработчик прерывания обычно открывает дополнительное отображение с информацией о причине прерывания и сохранённым состоянием системы. Отметим, что в коде восстановления, рассмотренном выше, фоновые задачи после сбоя удаляются из списка. Это эффективно предупреждает появление многократно повторяющихся ошибок. Очевидно, что подобные меры не требуются интерактивным задачам, поскольку их возобновление происходит под контролем пользователя.

Завершим раздел кратким изложением существенных свойств подсистемы управления задачами: система Оберон является многозадачной и основана на модели двух категорий задач. Интерактивные задачи взаимодействуют с подсистемой дисплея и выполняются с более высоким приоритетом. Фоновые задачи работают сами по себе и имеют более низкий приоритет. Возобновление работы задач реализовано путём послышки сообщения и, в конечном итоге, вызова обработчика, заданного процедурной переменной. Они выполняются последовательно в рамках единственного потока управления.

### 3.3 Понятие команды

Операционная система образует универсальную платформу, поверх которой созданы пакеты прикладного программного обеспечения. С точки зрения разработчиков приложений эта платформа выглядит как «интерфейс» к системе и, в частности, к аппаратному обеспечению. К сожалению, подобные интерфейсы в составе обычных операционных систем нередко предлагают слишком примитивные механизмы на основе понятий «программного прерывания» и «команды вызова супервизора», а также применяют файлы для обмена данными. Эта ситуация выглядит особенно нелепо на фоне развития языков программирования высокого уровня в направлении всё большей абстракции.

Во время работы над системой Оберон наибольшее внимание было уделено устранению *семантического разрыва* между прикладным и системным программным обеспечением. Итогом этих усилий стало появление весьма выразительных и целостных *программных интерфейсов приложения* в виде явной иерархии определений модулей. Возможно, самым важным и достойным внимания результатом этого подхода стали коллекции очень мощных общедоступных *абстрактных типов данных*, например, *Task*, *Frame*, *Viewer*, *File*, *Font*, *Text*, *Module*, *Reader*, *Scanner*, *Writer* и т.д.

#### 3.3.1 Атомарные действия

Самой важной функцией, общей для всех операционных систем, можно считать исполнение *программ*. Чтобы уточнить понятие *программы* в том смысле, который вкладывается в это понятие системой Оберон, нам придётся рассмотреть два аспекта: *статический* и *динамический*. С точки зрения статики, программа в системе Оберон представляет собой пару  $(M^*, P)$ , где  $M$  — произвольный модуль,  $P$  — экспортированная процедура без параметров, доступная в модуле  $M$ , а  $M^*$  обозначает иерархию, включающую как сам модуль  $M$ , так и все его прямо или косвенно импортированные модули. Отметим, что в общем случае две иерархии,  $M^*$  и  $N^*$ , могут иметь общую часть даже если  $M$  и  $N$  — различные модули. Точнее, их пересечение является надмножеством операционной системы.

Динамическое же поведение программы в системе Оберон определяется как атомарное действие (часто называемое *командой*) над глобальным состоянием системы, где под *атомарностью* подразумевается «отсутствие взаимодействия с пользователем». Такое определение естественно следует из нашей модели невытесняющего планирования задач на основе единственного потока управления. Мы можем обосновать это следующим образом. Когда традиционная интерактивная программа требует ввода данных от пользователя, текущая задача обычно вытесняется в пользу другой задачи, производящей требуемые данные. Как следствие, традиционную программу можно рассматривать как последовательность атомарных действий, которая может прерываться действиями, возможно принадлежащими другим программам. В то время, как в традиционных системах прерывание работы программы может случиться в любой момент, в системе Оберон это происходит только после завершения задачи (команды).

Подытоживая сказанное выше, в системе Оберон программы представляются в виде *команд*, то есть экспортированных процедур без параметров,

которые не взаимодействуют с пользователем.

Возвращаясь к вопросу вызова и исполнения программ, мы приходим к такому псевдокоду:

```
вызвать программу (M*, P) = BEGIN
    загрузить иерархию модулей M*; вызвать команду P
END
```

Системный интерфейс к механизму команд реализован во всё том же модуле *Oberon*. Его основную операцию словами можно изложить как «вызвать команду по имени и передать ей список фактических параметров»:

```
PROCEDURE Call (name: ARRAY OF CHAR; par: ParList; VAR res: INTEGER);
```

где *name* — имя желаемой команды в формате *M.P*, *par* — список фактических параметров, а *res* — код завершения. Но на самом деле мы разделили установку параметров и собственно вызов. Установка параметров производится вызовом

```
PROCEDURE SetPar (F: Display.Frame; T: Texts.Text; pos: INTEGER);
```

а собственно исполнение команды — вызовом

```
PROCEDURE Call (name: ARRAY OF CHAR; VAR res: INTEGER);
```

Пара (*T*, *pos*) задаёт текст, в котором содержится список параметров и расположение начала этого списка в тексте, *F* — отображение, из которого вызывается команда. Отметим появление ещё одного абстрактного типа данных *Text*, который импортируется из модуля *Texts*. Подробному обсуждению текстовой подсистемы Оберона посвящена глава ??, а в данный момент будет достаточно трактовать текст как последовательность литер.

Список фактических параметров передается модулем *Oberon* команде, которая будет вызвана, посредством экспортированной глобальной переменной *Par*:

```
Par: RECORD vwr: Viewers.Viewer;
    frame: Display.Frame;
    text: Texts.Text;
    pos: INTEGER
END
```

В принципе, помимо списка параметров, команда оперирует всем состоянием системы в целом, к которому может получить доступ при помощи абстрактного модульного интерфейса. Доступ к списку параметров — один из компонентов этого интерфейса. Еще один компонент — так называемый *системный журнал*, в котором ведётся общесистемный протокол выполнения команд и возникающих исключительных ситуаций в хронологическом порядке. Журнал представлен глобальной переменной типа *Text*:

```
Log: Texts.Text;
```

В данный момент читателю уже должно стать очевидным, что команды в своей реализации могут опираться на богатый арсенал абстрактных глобальных средств, отражающих структуру состояния системы и обеспечивающих доступ к её текущему состоянию. Другими словами, они могут

рассчитывать на высокую степень интеграции с системой. Как следствие, система Оберон предлагает исключительно широкий спектр взаимно интегрированных возможностей. Например, характерной особенностью системы служит полная интеграция абстрактных типов данных *Viewer* и *Text*, с которыми мы уже сталкивались ранее. Подробнее мы их рассмотрим в главах ?? и ??.

Модуль *Oberon* поддерживает интеграцию этих типов со следующим концептуальными возможностями системы, первые две из которых нам уже знакомы: стандартный список параметров для команд, системный журнал, обобщённое выделение текста, обобщённое копирование отображений. В данный момент следует уточнить, что мы подразумеваем под словом «обобщённый». Его следует считать равнозначным фразе «интерпретация зависит от конкретного отображения (интерактивной задачи)», и оно обычно используется там, где речь идет об отправке сообщений или других действиях с объектом, который нам точно не известен.

Далее мы кратко обсудим упомянутые обобщенные возможности, оставаясь на текущем уровне абстракции и понимания.

### 3.3.2 Обобщённое выделение текста

Выделенный фрагмент задаётся указанием текста-источника, диапазона литер внутри этого текста и временной меткой. Если не сказано иное, «выделенный фрагмент текста» всегда означает «фрагмент, выделенный самым последним». Получить его программным путём возможно при помощи вызова процедуры *GetSelection*:

```
PROCEDURE GetSelection (VAR text: Texts.Text; VAR beg, end, time: LONGINT);
```

Возвращаемые значения содержат интервал литер в тексте, который начинается в позиции *beg* и завершается в позиции *end* – 1, а также временную метку. Реализация этой процедуры включает в себя отправку широковеЩательного сообщения *запрос выделенного текста* всем отображениям. Объявление этого сообщения имеет вид:

```
SelectionMsg = RECORD (ViewerMsg)
    time: INTEGER;
    text: Texts.Text;
    beg, end: INTEGER
END;
```

### 3.3.3 Обобщённое копирование отображений

Обобщённое копирование называется также воспроизведением и клонированием. Это самая простая обобщенная операция, которую можно себе представить. Аналогично сказанному выше, ещё один вариант типа *ViewerMsg* используется для передачи запросов нужного вида:

```
CopyMsg = RECORD (ViewerMsg) vwr: Viewers.Viewer END
```

Получатели сообщения *запрос копии* обычно создают копию самих себя и передают её отправителю при помощи поля *vwr*.

Подытожим изложенное в этом разделе: операционная система Оберон с точки зрения пользовательских приложений выглядит как очень выразительный абстрактный модульный интерфейс. Этот интерфейс предоставляет множество мощных абстрактных типов данных наподобие *Viewer* или *Text*. Богатый арсенал глобальных типов данных и обобщённых средств предназначен для достижения высокой степени интеграции в систему. Программы в системе Оберон реализуются в виде так называемых команд, то есть экспортированных процедур без параметров, которые не взаимодействуют с пользователем. Набор команд, реализуемый модулем, является его пользовательским интерфейсом. Параметры команд передаются через глобальный список, предоставленный вызывающей задачей посредством модуля *Oberon*. Команды оперируют глобальным состоянием системы.

### 3.4 Инструментарии

Модули, как правило, выступают в одной из трех различных ролей. Во-первых, это модули, инкапсулирующие некоторые данные и обеспечивающие доступ к ним исключительно через экспортированные процедуры и функции. Хороший пример такого модуля — *FileDir*, скрывающий содержимое каталога файлов и защищающий его от возможного разрушения. Во-вторых, это модули, определяющие *абстрактные типы данных*, которые экспортируют тип и операции над ним. Типичный пример — модули *Files*, *Modules*, *Viewers* и *Texts*. Наконец, третья разновидность модулей — простые наборы процедур, объединённых общим назначением, например, модуль *RS-232*, реализующий связь через последовательный порт.

Модули в системе Оберон приобретают ещё одну роль — *инструментария*. По определению, инструментарием называется модуль, реализующий исключительно коллекцию команд в смысле, заданном выше. Принципиальное отличие инструментариев от других модулей в том, что они находятся на самом верху иерархии. Можно говорить, что инструментарий «импортируется» пользователем системы во время работы с ней. Другими словами, определения, находящиеся в этом модуле, задают *пользовательский интерфейс*. Типичные примеры таких модулей — *System* и *Edit*. Существует эмпирическое правило, что для каждой темы или приложения существует свой инструментарий.

В качестве примера инструментария приведём модуль *System* с дополнительными комментариями:

DEFINITION System;

(\*Управление системой, главы 3 и 8\*)

```
PROCEDURE SetUser; (*идентификация*)
PROCEDURE SetFont; (*для вводимого текста*)
PROCEDURE SetColor; (*для вводимого текста и графики*)
PROCEDURE SetOffset; (*для вводимого текста*)
PROCEDURE Date; (*установка и показ даты и времени*)
PROCEDURE Collect; (*сборка мусора*)
```

(\*Управление дисплеем, глава 4\*)

```
PROCEDURE Open; (*отображение*)
```

```

PROCEDURE Close; (*отображение*)
PROCEDURE CloseTrack;
PROCEDURE Recall; (*последнее закрытое отображение*)
PROCEDURE Copy; (*отображение*)
PROCEDURE Grow; (*отображение*)
PROCEDURE Clear; (*очистить журнал*)

(*Управление модулями, глава 6*)
PROCEDURE Free; (*заданные модули*)
PROCEDURE ShowCommands; (*заданного модуля*)
PROCEDURE ShowModules; (*список загруженных модулей*)

(*Работа с файлами, глава 7*)
PROCEDURE Directory;
PROCEDURE CopyFiles;
PROCEDURE RenameFiles;
PROCEDURE DeleteFiles;

(*Получение информации о системе, глава 8*)
PROCEDURE Watch; (*задачи, память, использование дисков*)
END System;

```

Важным следствием нашего подхода к построению интегрированной системы является возможность создания универсального интерактивного *командного интерпретатора*, привязанного к отображениям с *текстовым содержанием*. Если текст подчиняется приведенному ниже синтаксическому правилу (выраженному в расширенной форме Бэкуса-Наура), будем называть его *командным текстом*:

$$\text{CommandTool} = \{ [\text{Comment}] \text{CommandName} [\text{ParameterList}] \}.$$

При наличии списка параметров, он передаётся в вызываемую команду через поля *text* и *pos* глобальной переменной *Par*, экспортированной из модуля *Oberon*. Поскольку интерпретация списка параметров отдаётся на усмотрение команды, его формат никак не зафиксирован. Однако, следующие правила и соглашения были выработаны для стандартизации пользовательского интерфейса:

1. Элементами текстового списка параметров служат универсальные лексемы, наподобие имён, строк, целых и вещественных чисел, а также специальные символы.
2. Символ «^» в списке текстовых параметров указывает на то, что выделенный фрагмент текста должен быть использован как продолжение этого списка. В частном случае, когда этот символ следует непосредственно за именем команды, весь список параметров представлен выделенным фрагментом.
3. Символ «\*» в списке текстовых параметров ссылается на помеченное в данный момент отображение. Как правило, этот символ заменяет собой имя файла. В этом случае содержимое отображения, помеченного системным маркером (звёздочкой) обрабатывается командным интерпретатором вместо содержимого файла.

4. An *at*-character “@” in the textual parameter list indicates that the selection marks the (beginning of the) text which is taken as operand.
5. Наконец, символ «~» служит признаком конца текстового списка параметров переменной длины.

Поскольку командные тексты — самые обыкновенные тексты с возможностью их редактирования (чем они разительно отличаются от меню в традиционных системах), они могут дорабатываться «на лету», обеспечивая большую гибкость системы. Обратимся снова к рис. 3.1, на котором показан типичный вид экрана системы Оберон. На нём можно выделить две области, в которых отображения размещены друг под другом. Слева находится *группа пользовательских отображений* большей ширины, а справа — более узкая *группа системных отображений*. В пользовательской группе показаны три документа: текст, векторное и растровое изображения. В системной группе мы видим отображение системного журнала и два отображения командных текстов, в одном из которых — стандартные инструменты управления системой, а в другом — персональный набор инструментов пользователя.

В завершение конкретизируем понятия команды и командного текста на примере раздела управления системой инструментария *System*. В его состав входят такие команды, как *SetUser*, *Date*, *SetFont*, *SetColor*, и *Collect*, предназначенные для управления общесистемными средствами. В частности, эти команды выбирают текущего пользователя, отображают и устанавливают системные дату и время, задают системный шрифт для текстового ввода и системные цвета, а также иницируют сборку мусора.

Подытожим материал главы. Инструментарием называется модуль системы Оберон, обладающий определёнными свойствами. Он реализует набор команд. Инструментарии находятся в самом верху иерархии модулей и своим содержимым задают пользовательский интерфейс системы. Командные тексты представляют собой последовательности вызовов команд, записанные в текстовом виде. Они могут произвольно редактироваться и приспособляться к потребностям пользователя. При стандартном для системы Оберон расположении элементов графического интерфейса на экране, командные тексты выводятся в группе системных отображений.