Никлаус Вирт

Проектирование системы с нуля

Niklaus Wirth (1989) Designing a System from Scratch // Structured Programming, № 10.
Р. Богатырев, перевод с англ.

Система Oberon представляет собой однопользовательскую однопроцессную многозадачную систему, ориентированную на рабочую станцию. Она разрабатывалась не на базе уже существующего программного обеспечения, а фактически с нуля. В этой статье освещается последовательность шагов проектирования, которая постепенно привела к построению полной системы. Данный проект в значительной степени использовал такой прием, как раскрутка (bootstrapping), который применялся как по отношению к разработке компилятора, так и по отношению ко всей системе в целом.

Введение

Литературы о приемах, связанных с разработкой систем с нуля, не говоря уже об опыте построения таких систем, крайне мало. Тому есть два объяснения. Во-первых, сейчас, да и в будущем, достаточно небольшое количество людей имеет шанс столкнуться с задачей разработки новой операционной среды. А потому наиболее вероятной наградой за попытку рассказать о полученном опыте будет лишь легкая поддержка. И, во-вторых, изучение этого предмета вряд ли может привести к серьезному научному открытию. Это всего лишь эксперимент.

Автор принял вызов на проектирование и реализацию новой системы с нуля для голого оборудования. При этом потребовались значительные усилия, но взамен были получены и зафиксированы ценные знания, которые крайне трудно получить обычным путем. Основной мотивацией ведения записей эксперимента было твердое убеждение в том, что попытки начать все с нуля — это подчас единственный способ освободиться от установленных ограничений, приобретенных привычек и скверных ошибок. В то время, когда подавляющее большинство людей пытается стандартизировать языки, операционные системы, коммуникационные протоколы, интерфейсы и методы документирования, и занимаются этим довольно долго, прежде чем будут по достоинству оценены, важно отметить, что все еще остается возможность отцепить хвостовой вагон и отправиться в путешествие, которое, правда, может стать источником головной боли и потребовать большой выдержки.

Система Oberon, которая является предметом нашего обсуждения — это универсальная, крайне гибкая и подлинно расширяемая операционная среда, ориентированная на однопользовательскую рабочую станцию [1]. В результате наших работ были созданы текстовый и графический редакторы, файловая система, диспетчер памяти, загрузчик динамического связывания, компоновщик (boot linker), средства архивации на флоппи-диске, компилятор, ассемблер, декодер, набор утилит, а также интерфейсы доступа через сеть к файловому серверу, серверу печати и почтовому серверу. Кроме того, был сделан ряд расширений, связанных с прямым доступом к лазерному принтеру, а также с введением одновременной поддержки текстовым и графическим редакторами монохромного и цветного дисплея. Система также включает в себя драйверы для жесткого диска, флоппи-диска, клавиатуры и мыши, а также для последовательных каналов RS-232 (V24) и RS-485. Модули-утилиты содержат процедуры для преобразования ввода-вывода, для основных математических функций и для тестовых операций с экраном, Ну и, наконец, последнее, но отнюдь не менее существенное — это то, что в ходе проекта родился новый язык Oberon, в значительной степени связанный с языком Modula-2 [2,3].

Сначала казалось вполне резонным программировать систему на Modula-2 или, точнее говоря, на соответствующим образом расширенной версии этого языка. Для того чтобы создать истинно расширяемую систему, обязательным условием является наличие механизма для определения новых типов данных, которые представляют собой расширения существующего типа и при этом остаются с ним совместимыми. Не менее важным было и наше желание увеличить "надежность" языка за счет удаления из него (или, по крайней мере, существенного сокращения) средств обхода правил совместимости типов. И если смотреть на это с учетом ориентации на механизм автоматической утилизации памяти, то такое требование — отнюдь не идеализированное пожелание, а жестокая необходимость. Этого можно достичь только путем удаления из Modula-2

тех средств, которые позволяют выполнять подобные трюки и которые более вольно используются программистами, чем это предполагалось при проектировании Modula-2. Весьма занимательно, что этот высокоуровневый язык был выбран многими программистами просто потому, что он предоставляет достаточно большое количество низкоуровневых средств для нарушения концепции типа.

Механизм для определения расширенных типов данных наряду с возможностью устанавливать обработчики в полях записи, которые принадлежат процедурному типу, позволяет классифицировать Oberon как объектно-ориентированный язык. Класс объекта представляется как тип запись, а подкласс — как расширение этого типа. Главным достижением языка Oberon является интеграция концепции класса (которая только кажется новой) с уже признанной концепцией типа данных. Такая интеграция привела к сокращению числа базовых понятий, что в свою очередь, привело к упрощению проектирования и реализации.

Точка отсчета

Целевое аппаратное обеспечение представляло собой рабочую станцию под названием Ceres [4], которая была разработана Гансом Эберле (H. Eberle) и автором этих строк на базе процессора NS-32032, чья регулярная архитектура хорошо подходит для компиляторов. Инструментальными средствами служили рабочая станция Lilith [5] с соответствующей средой программирования языка Modula-2. Эта машина служила для разработки первой версии базового программного обеспечения и ее Modula-компилятор послужил основой для компилятора языка Oberon. Подгрузка программ осуществлялась через последовательный канал RS-232 со скоростью 9600 bps.

Разработка компилятора

Для подобных проектов, как правило, характерно параллельное выполнение нескольких работ, многие из которых, правда, требуют завершения предшествующих им работ. Одной из наиболее четко выделяемых работ является разработка компилятора. Кроме того, эта деятельность закладывает краеугольные камни истории проекта. В этом плане наиболее интересной была работа по переносу компилятора с Lilith на Ceres.

На первой стадии начальная версия компилятора языка Oberon была получена из существующего Modula-компилятора. Эта версия была способна обрабатывать подмножество конструкций Oberon (Oberon0). Это была некая смесь конструкций Oberon и Modula-2 с Oberon-ориентированным синтаксисом. Здесь существенно то, что язык Oberon представляет собой дальнейшее развитие Modula-2 на основе новых понятий расширения типа и включения типа. Oberon0 эти понятия не использовал. Результирующий компилятор генерировал код для процессора NS-32000 и работал на Lilith, являясь по существу кросс-компилятором. Для того чтобы выполнять оттранслированный код, он передавался (подгружался) на Ceres. Разработка необходимых средств для подгрузки является предметом обсуждения следующего раздела. Здесь мы только отметим, что все эти средства могли быть сформулированы на Oberon0 для Ceres (или на Modula—2 для Lilith).

Стадия 2 была преимущественно связана с переводом компилятора Oberon0 на его собственный язык. Трансляция этого компилятора была необходимой предпосылкой для переноса и доступности Oberon на Ceres.

Как только завершился перенос компилятора Oberon0, последовала длинная серия шагов раскрутки. Каждый шаг состоял из двух фаз. Первая включала проектирование и вкрапление частей с добавленными или измененными особенностями. Вторая фаза состояла в том, чтобы использовать появившиеся особенности в самом компиляторе.

Т-диаграммы имеют большое преимущество в смысле наглядности важных принципов, но грешат опасностью их чересчур упрощенной интерпретации. Здесь подавляются многие детали, и человек, не знакомый с этим процессом, может попросту недооценить действительное количество шагов. Основной причиной большого количества таких шагов является, конечно же, способность человека допускать ошибки, которые требуют исправления. Другой причиной служит тот факт, что язык претерпевает изменения. Это связано с новым пониманием путей построения компилятора, обработки и использования новых особенностей языка. Некоторые уже добавленные вещи были впоследствии отменены, когда выяснилась их бесполезность.

Возможность отказываться от введенных средств — это одно из основных достоинств техники раскрутки, и оно не должно остаться без внимания.

Незаменимым инструментом для разработчика компилятора является декодер. Наш декодер, представляющий собой кросс-декодер, был запрограммирован на Modula-2. Он декодировал объектные файлы, поступающие из Ceres по каналу связи. В дальнейшем он был перенесен в среду Oberon.

Построение внутреннего ядра (Inner Core)

Внутреннее ядро системы Oberon состоит из файловой системы, загрузчика модулей и средств распределения памяти. Его пять модулей связываются воедино в так называемый файл загрузки (boot file). Здесь мы опишем те шаги, которые позволили сформировать внутренее ядро.

Вряд ли можно поспорить с тем, что началом всех начал для любого программного обеспечения на "голой" машине является начальный загрузчик (boot loader). Поскольку он размещается в ПЗУ и не может изменяться, он должен быть как можно более компактным. Добиться этого не столь сложно, ведь цель загрузчика проста и хорошо определена. Файл загрузки рабочей станции Ceres состоит из последовательности блоков данных, каждому из которых предшествует его размер и адрес, по которому он должен быть размещен.

Начальный загрузчик инициализирует регистры компьютера, очищает память и считывает файл загрузки. Наша первая версия осуществляла считывание через последовательный канал RS-232 и состояла менее, чем из 200 байтов кода. Она была запрограммирована с помощью техники кросс-ассемблирования на компьютере Lilith. Более поздние версии позволяли уже выбирать источник загрузки: жесткий диск, флоппи-диск или же последовательный канал. Сам файл загрузки создавался линкером загрузки (boot linker), который также работал на Lilith.

Построение внутреннего ядра осуществлялось постепенно, по шагам. Каждый шаг добавлял что-то новое в смысле функциональности. На каждом из них в качестве интерпретатора ввода и инициатора вызова тестируемых операций использовался инструментарий команд (command tool). К сожалению, от осторожного подхода к монтажу, который выливается в серию инструментариев команд, никуда не уйти, когда приходится иметь дело с "голой" и незнакомой машиной, у которой отсутствуют даже средства визуализации вывода. Эти инструментарии делались как можно более компактными, ведь "дружественный" интерфейс с пользователем здесь не так уж необходим.

Модуль Tool0 считывал команды с клавиатуры и позволял отображать на экране содержимое различных областей памяти. Он служил для того, чтобы придать уверенности на начальном этапе в правильности жизненно важных программ вывода данных на растровый экран. Модуль Bitmaps содержал процедуры для генерации растра. Модуль IO был полностью ориентирован на преобразование чисел в последовательность символов. Единственный непропорциональный фонт, используемый модулем IO, был выполнен в виде блока, входящего в состав файла загрузки. Модуль Kernel выполнял специальную функцию, которая состояла в том, что он получал управление от начального загрузчика и передавал его первому модулю из файла загрузки (головному модулю). Перед тем, как передать управление, Kernel осуществлял инициализацию ресурсов машины. Он был запрограммирован в ассемблерном коде, поскольку требовалось задействовать также операции, которые не были доступны через компилятор, и поскольку требовалось выполнять обращения к супервизору.

Модуль Tool1 уже работал с диском. Драйвер диска был отнесен к Kernel по той причине, что тот содержал эффективные процедуры по передаче данных сектора во внутренний буфер дискового интерфейса и обратно. Tool1 обеспечивал гарантию функционирования дискового интерфейса и дискового контроллера.

Tool2 служил для тестирования модуля Files. При этом он не работал с именами и каталогами, поскольку единственной его задачей было оттестировать реализацию абстракции File как последовательности байтов. Модуль Kernel теперь был расширен за счет процедур работы с таблицей выделенных секторов диска (таблицей резервирования диска). Поскольку целостность этой информации весьма критична для всей системы в целом, таблица размещалась в области ядра, защищенной аппаратно. Доступ в эту область для записи был возможен лишь при работе в режиме супервизора (ядра).

На следующем этапе появилась работа с каталогами файлов, которая была представлена модулем FileDir. Каталог был выполнен в виде B-дерева степени 12 (т.е. максимум 24 записи на странице). Каждая страница занимала сектор размером 1024 байта. Отсюда следует, что каталог — это отнюдь не файл. В Tool3 появились команды для вставки, поиска и удаления имен, а также для просмотра всего каталога. Модуль FileDir также имел процедуры, осуществляющие инициализацию таблицы резервирования диска при начальном обходе B-дерева и таблицы секторов, которые расположены в заголовке каждого файла. Tool4 использовал для тестирования совокупности средств работы с файлами и с каталогами. Модуль IO был снабжен программами по передаче данных через канал RS-232, что дало возможность осуществлять извлечение и перемещение целых файлов при участии соответствующей программыпартнера на компьютере Lilith.

Окончательным шагом первой стадии разработки явилось создание загрузчика под названием Modules. В его обязанности входили загрузка, инициализация и выгрузка модуля, а также вызов всех процедур (которые являются частью загруженного модуля), представленных в Tool5. Сам загрузчик представляет собой рекурсивную процедуру. Он практически идентичен линкеру загрузки, за исключением того, что последний работает на Lilith и осуществляет загрузку в виртуальные сегменты, которые затем заносятся в файл.

В соответствии с таким подходом необходимо: выполнять редактирование и компиляцию на компьютере Lilith, компоновать модули в файл загрузки, осуществлять загрузку с предварительной передачей файла по каналу связи и, наконец, производить тестирование с помощью команд инструментария (tool). Теперь появилась возможность передавать отдельные объектные файлы и выполнять их компоновку/загрузку под управлением Tool5 на рабочей станции Ceres. Tool5 не только обладает необходимыми средствами для достижения автономии на первом этапе, но и предоставляет достаточный набор средств для работы (с использованием кросс-компилятора на компьютере Lilith).

В этот момент модуль Tool, выступавший в качестве головного модуля файла загрузки был заменен загрузчиком. Это предусматривало, что при начале работы Kernel передает управление Modules, в инициализации которого содержится обращение к самому загрузчику с фиксированным параметром "Tool5" (в дальнейшем — "Oberon"). Загрузчик и все импортируемые им модули формируют файл загрузки. Этот набор модулей получил название Inner Core (внутреннее ядро). В него намерено не включены модули Bitmaps и IO, которые загружаются и связываются на компьютере Ceres, поскольку они импортируют Tool, а не Modules. В новый модуль Input были встроены драйвер клавиатуры (ранее размещавшийся внутри Tool) и драйвер мыши.

На протяжении всей разработки модуль Kernel также претерпевал эволюционные изменения. Введение загрузчика было обусловлено появлением двух ранее не использовавшихся понятий: прерывания и виртуальной адресации. Прерывание (вместо опроса устройства) теперь использовалось для ввода с клавиатуры, давая возможность перехватывать специальный символ в драйвере клавиатуры (Input) и генерировать программное прерывание (дабы вывести процессор из ошибочного цикла). Kernel теперь экспортирует процедуру для установки обработчиков прерываний. Он также инициализирует контроллер аппаратных прерываний.

Использование виртуальной адресации, т. е. отображение адресов из виртуального пространства в физическое, выполняется аппаратурой MMU, которая нужна в связи с отложенной загрузкой (delayed loading). Oberon не использует принудительное разбиение на страницы — метод, который потерял свою актуальность с появлением первичной памяти большого объема. При отложенной загрузке предусматривается, что модуль B, импортируемый модулем A, физически не загружается в момент загрузки модуля A. Вместо этого используется просто дескриптор, содержащий все данные, относящиеся к клиентам модуля B. Одновременно для B выделяется необходимое виртуальное пространство, а не физическая память. Последняя резервируется до того момента, когда появится сигнал о попытке доступа ко все еще отсутствующему коду модуля B.

Таким образом, включение загрузчика также было вызвано активацией отображения виртуальных адресов и обеспечением процедур, устанавливающих требуемые таблицы настройки. Файл загрузки загружается, конечно, в тот момент, когда действие MMU блокировано. А потому начальная часть системы полностью отображается на физическую память.

Другим важным расширением модуля Kernel был диспетчер динамической памяти, точнее говоря, сборщик мусора. На ранних шагах был необходим простой диспетчер с последовательным

выделением блоков памяти. Дело в том, что относительно короткие сеансы тестирования никогда не требовали всего объема доступной памяти (2 Мбайт). Наш сборщик мусора базируется на традиционном принципе помечивания (mark-scan). Нюанс его состоял в том, что нужно было быть осторожным с координацией и интеграцией его работы с компилятором и загрузчиком, которые генерировали и выделяли дескрипторы, представляющие типы данных. Эти дескрипторы сами динамически выделяли записи, доступные сборщику мусора.

Построение внешнего ядра (Outer Core)

Как уже было сказано, Inner Core (внутреннее ядро) представляет собой набор модулей, связанных с помощью линкера загрузки. Начальный загрузчик производит загрузку этого набора и передает управление (через Kernel) обычному загрузчику, который подгружает определенный модуль, а именно Oberon. При этом автоматически загружаются и все модули, импортируемые модулем Oberon. Полученная конфигурация носит название Outer Core. При использовании Tool5 внешнее ядро состояло из внутреннего ядра и модулей Bitmaps, IO, Input и Tool5.

Внешнее ядро законченной системы включало в себя преимущественно модули, отвечающие за обработку текста, и визуализаторы (viewer). Они участвуют в формировании гибкого редактора и системы многочисленных визуализаторов, которые выполнены в объектно-ориентированном стиле. Каждый визуализатор имеет свой собственный интерпретатор команд. Командный цикл в центральном модуле (раньше Tool, сейчас Oberon) все еще использует опрос устройств ввода (клавиатуры, мыши, в дальнейшем и сети), однако фактическая интерпретация поступившей информации больше не концентрируется в командном цикле, а распределяется по интерпретаторам, которые устанавливаются как соответствующие обработчики в визуализаторах.

Одно из важных преимуществ нашей схемы кроется в том, что различные головные модули (определяющие внешнее ядро) могут быть использованы без изменения внутреннего ядра (файла загрузки). Правда, здесь есть одна серьезная неприятность, поскольку компоненты внутреннего ядра не имеют ссылок на дисплей. (Модуль IO импортировался только инструментарием Tool.) Из этого следует, что никакого визуального вывода не будет, пока устанавливается внешнее ядро. Единственным признаком проведения фазы загрузки был пустой экран!

Перенос компилятора на Ceres производился параллельно с разработкой внешнего ядра. Эта фаза оказалась гораздо более трудоемкой, чем мы предполагали. Некоторые средства, недоступные в версии компилятора Lilith, все еще ждали своей реализации. По мере их появления требовалось перепрограммировать и адаптировать многие уже используемые части.

Начало третьей стадии было отмечено тем, что внешнее ядро стало полностью работоспособным, при этом сам Oberon мог быть использован (и оттестирован) для подготовки и редактирования текстов, а не только для компиляции и выполнения программ. Как только этого удастся достичь, система станет (почти) автономной.

Первой утилитой, полностью разработанной под Oberon, была файловая система для дискет, которая преимущественно использовалась для архивации файлов. Без нее связь с Lilith и проведение архивации были бы невозможны. Следующие утилиты уже обеспечивали доступный через сеть интерфейс с сервером печати, файловым и почтовым сервером.

Сервисные средства

Внутренее ядро размещается в слинкованном виде на загрузочном треке диска и загружается начальным загрузчиком, который по выбору производит загрузку с диска, с дискеты или через последовательный канал RS-232. Основная часть процесса загрузки, следовательно, выполняется обычным загрузчиком системы Oberon. Сначала он собирает модули, образующие внешнее ядро, и затем, передавая управление Oberon'у, подгружает все модули, импортируемые модулем Display. Часть инициализации Display открывает визуализатор и высвечивает текст, соответствующий командам для открытия остальных визуализаторов. Таким образом, загрузка состоит из трех фаз.

1. Внутреннее ядро (Inner Core). Оно загружается с помощью начального загрузчика, слинкованного в файл загрузки.

2. Внешнее ядро (Outer Core). Оно загружается с помощью загрузчика Oberon, который инициализируется в процессе инициализации модуля Modules.
3. Система (The System). Она загружается с помощью загрузчика Oberon, который инициализируется в процессе инициализации модуля Oberon.

Эта схема позволяет добиться большой гибкости. С ее помощью можно получить полностью различные конфигурации, либо же только различные компоненты. Однако, неудобство состоит в том, что разрушение любого из многочисленных объектных файлов, которые нужны на фазах 2 и 3, будут приводить к сбою процесса загрузки. Поэтому необходим специальный сервисный инструментарий для восстановления потерянных или разрушенных файлов. Этот же инструментарий (который назван Oberon0) используется для установки необходимых файлов на новую машину (диск). Инструментарий был легко построен на базе Tool4 путем расширения последнего процедурами чтения с дискет. Эти процедуры практически идентичны тем, которые находятся в системном модуле Diskette.

Мы надеемся, что такой многофазный процесс загрузки является не столько результатом эволюции разработки системы, сколько действительно необходимым средством обнаружения сбоев. Так, к примеру, фаза 1 опирается исключительно на ПЗУ-резидентный начальный начальный загрузчик; фаза 2 — на обычный загрузчик; фаза 3 — на загрузчик, систему работы с текстом и систему визуализаторов. Этот принцип последовательного насыщения компонентами соблюдался и при проектировании аппаратной части. Процесс аппаратной загрузки начинается с распространения сигнала сброса (reset) и обработки его путем передачи управления начальному загрузчику, который не использует периферийных устройств (за исключением, конечно, того, откуда производится загрузка). Выбор устройства — источника загрузки осуществляется внутренним переключателем, который остается в одной и той же позиции кроме тех случаев, когда нужен сервис. При этом даже не нужна клавиатура.

Модуль, который собирается вместе со всеми импортируемыми им модулями в файл загрузки и который не содержит загрузчика, мы называем сервисным средством (service tool). Другое сервисное средство, которое носит название DiskCheck, используется в тех ситуациях, когда диск содержит сбойные файлы и сектора. DiskCheck осуществляет интерпретацию команд, предназначенных для проверки отдельных секторов, для сохранения, форматирования и восстановления треков, для анализа каталогов и заголовков файлов, для форматирования всего диска и, главное, для сборки мусора, т.е. для восстановления каталогов и неиспользуемых секторов, которые остались вследствие сбоя диска.

Заключительные замечания

Оглядываясь назад, можно смело сказать, что эксперимент удался. Несмотря на множество трудностей построение системы с нуля принесло громадное удовлетворение в отличие от работ по "латанию" существующих программных комплексов и "замазыванию" ошибок и неточностей. Да и эффект в смысле обогащения знаниями также был весьма значителен.

Нет другого, более сильного стимула для аккуратного программирования, чем отсутствие отладчика. Столь широко распространенное смещение акцента от языков программирования в сторону сред программирования (в особенности отладчиков), которое произошло за последнее десятилетие, является весьма прискорбным фактом и крайне вредно в смысле обучения программированию. С другой стороны, упорное отстаивание ясных концепций и обдуманное отвержение излишних "звоночков и гудочков" в языке Oberon принесло явную пользу. Использование неструктурированного языка или языка без "водонепроницаемых" правил совместимости типов в конечном итоге приведет к ошибкам в проекте. Единственное, что было написано на ассемблере, — так это ядро и операции работы с растровой графикой. Программирование этих частей требует утроенного внимания. В частности, находящийся в ядре сборщик мусора представляет собой программу, которую просто нельзя отладить, независимо от того, есть отладчик или нет. Добиться его корректности можно только путем четкого и аккуратного проектирования.

Очевидно, что отладчики лечат только следствие, а не саму болезнь. К сожалению, бывают такие обстоятельства, когда приходится смириться с болезнью, хотя, начиная с нуля, мы предполагали избежать этого в нашем проекте. Даже когда есть свое собственное аппаратное обеспечение, приходится принимать компоненты такими, какими они есть. Так, при построении интерфейса к периферийным устройствам, в частности, это касается драйверов для интерфейсных чипов,

бывали эпизоды, когда казалось, что наши планы рушатся. Такие чипы стали за последнее время довольно сложными, приспосабливаясь к тем болячкам сложного программного обеспечения, которые стали уже притчей. Ярким примером этого в нашем проекте стал контроллер последовательного канала (Zilog 8530). Это семейство чипов характеризуется повышенной сложностью из-за непродуманного и избыточного универсализма, приводящего к невнятному поведению и к длиннющим, тяжелым для восприятия, неполным, а, подчас, даже и некорретным спецификациям. Здесь уже программирование — это не творческая деятельность, а чистый эксперимент, осуществляемый методом проб и ошибок. А потому наш совет — избегать использования таких компонентов везде, где это только возможно.

Необходимым условием подобного проекта является аккуратное планирование шагов таким образом, чтобы не было мостиков к предыдущим шагам, которые бы разрушали сделанное ранее. И несмотря на то, что было бы ошибкой оставлять все предыдущие стадии на попечение так называемой системы управления процессом разработки программного обеспечения (software engineering management), все же обходиться без них весьма опасно, особенно тогда, когда нет полной уверенности в последующих шагах.

Постоянной проблемой было поддержание соответствия в различных конфигурациях модулей. Расхождения обнаруживаются путем сравнения на этапе загрузки ключей модулей. Возникающие несоответствия особенно тяжелы в момент начальной загрузки.

Наконец, мы хотим обратить внимание на тот факт, что процесс концептуального проектирования системы шел сверху вниз, тогда как фактическое построение осуществлялось снизу вверх. Создание сложных систем полностью опирается на симбиоз этих двух подходов.

Благодарности. Автор крайне признателен Юргу Гуткнехту (J. Gutknecht), соавтору языка Oberon, за его бесценный вклад. Он оказал значительное воздействие на формирование структуры всей системы, был непреклонен в отстаивании ясных и последовательных принципов и концепций, а также программировал (наряду с другими частями) весь комплекс средств, поддерживающих работу с текстом, графикой и визуализаторами.

Литература

1. Wirth N. (July 1988) The Oberon System // Bericht 88, Informatik, ETH Zuerich.
2. Wirth N. (July 1988) From Modula to Oberon // Software: Practice and Experience, 18(7), 661-670.
3. Wirth N. (July 1988) The Programming Language Oberon // Software: Practice and Experience, 18(7), 671-690.
4. Eberle H. (Jan. 1987) Hardware Description of the Workstation Ceres // Tech. Bericht 77, Informatik, ETH Zuerich.
5. Ohran R. (Aug. 1984) Lilith and Modula-2 // BYTE, 9(8), 181-194.

Об авторе. Никлаус Вирт (Niklaus K. Wirth) — профессор Швейцарского федерального технологического института (ETH) в Цюрихе, который Вирт закончил в 1958 г. и где получил специальность инженера в области электроники. Затем он продолжил свое обучение в Лавальском университете (Laval University) в Квебеке (Канада). В университете Калифорнии в Беркли (University of California at Berkeley, США) в 1963 г. Вирт защитил диссертацию. До 1967 г. работал доцентом на вновь образованном факультете компьютерных наук в Стенфордском университете (Stanford University, США), где он разработал язык PL360 и, в сотрудничестве с рабочей группой IFIP Working Group 2.1, язык Algol-W. В том же 1967 г. становится доцентом в университете Цюриха (University of Zurich), а в 1968 г. переходит в ETH, где в период с 1968 по 1970 годы разрабатывает язык Паскаль. Среди последующих проектов — разработка и реализация персонального компьютера Lilith, высокопроизводительной 16-разрядной рабочей станции с растровым дисплеем, создание языка Modula-2 (1978-1982 г.), и 32-разрядной рабочей станции Ceres (1984-1986 г.). Затем им были созданы языки Oberon и Oberon-2 (совместно с профессором Х.Мессенбоком), а также операционная система Oberon (1986-1989 г.). В 1984 г. профессор Вирт был удостоен почетной премии Алана Тьюринга (Turing Award), в 1989 г. — премии Max Petitpierre Prize, а также премии Science and Technology Prize от IBM Europe. Один из последних его проектов — система Lola System для разработки электронных схем (1995 г.).