Модула-2 и Оберон[1]

Никлаус Вирт

Федеральный технический институт (ETH), Цюрих

wirth@inf.ethz.ch

Аннотация

Это история разработки языков Модула-2 и Оберон. Вместе с предшественниками Алголом-60 и Паскалем, они образуют семейство, называемое алголоподобными языками. Паскаль (1970) отразил идеи структурного программирования, Модула-2 (1979) добавила к этому модульное проектирование систем, а Оберон (1988) обеспечил поддержку объектно-ориентированного стиля программирования. Таким образом, они отразили важнейшие парадигмы программирования последних десятилетий. Далее будут очерчены главные свойства этих языков и рассказано о соответствующих усилиях по их реализации. Разъясняются условия и обстоятельства, в которых были созданы эти языки. Мы подчеркиваем, что самым главным руководящим принципом была простота дизайна. Ясность понятий, экономия свойств, эффективность и надежность реализации стали его следствием.

…

5. Язык Оберон

Язык программирования Оберон стал результатом целенаправленных усилий по повышению мощи Модулы-2 и одновременно по уменьшению её сложности. Оберон – последний представитель семьи алголоподобных языков. Исходным пунктом был Алгол-60, за которым последовали Алгол-W, Паскаль, Модула-2 и, наконец, Оберон [27, 28]. «Алголоподобность» подразумевает процедурную парадигму, строго определенный синтаксис, традиционную математическую символику для выражений (без эзотерических символов ++, ==, /=), блочную структуру, определяющую диапазон действия идентификаторов и реализующую идею локальности, рекурсивность процедур и функций, а также строгую статическую типизацию данных.

Направляющий принцип состоял в том, чтобы сконцентрироваться на базовом и существенном и избавиться от всего эфемерного. Это было безусловно разумно ввиду очень небольшого числа участников проекта. Другим стимулом здесь было осознание злокачественного роста сложности недавно возникших языков программирования, таких как C, C++ и Ада. Они казались еще менее пригодными для обучения, чем даже для работы в промышленности. Даже Модула-2 теперь казалась чересчур перегруженной, содержащей редко используемые свойства, об удалении которых не стоило бы жалеть. Попытка

---

[1] Частичный перевод доклада на Конференции по истории языков программирования, Сан-Диего, 2007; оригинал: http://www.cs.inf.ethz.ch/~wirth/Articles/Modula-Oberon-June.pdf, использована версия, полученная 07 декабря 2007 г.

сконденсировать самые существенные (а не просто удобные и общепринятые) свойства в маленьком языке представлялась полезным (академическим) упражнением.

5.1. Свойства, исключенные из Оберона

Большое число стандартных типов данных не только усложняет компилятор, но также затрудняет обучение и овладение языком. Поэтому типы данных стали главной мишенью нашего энтузиазма в отношении простоты.

Бесспорным кандидатом на удаление были *вариантные записи* Модулы. Введенные с похвальной целью обеспечить гибкость структурирования данных, они в основном стали инструментом злоупотреблений, имеющих целью обход защиты, которую обеспечивала типизация. Это средство позволяет по-разному интерпретировать конкретную запись (т.е. считать ее по-разному составленной из полей-компонент), причем правильная интерпретация указывается значением специального *поля-селектора*. Реально серьезная проблема состояла в том, что это поле-селектор можно было вообще игнорировать. Подробнее об этом можно прочесть в главе 20 книги [13][2].

Перечислительные типы могли показаться достаточно привлекательным и безобидным средством, чтобы сохранить его. Одна проблема выявилась в связи с импортом и экспортом: должен ли экспорт некоторого перечислительного типа также автоматически повлечь за собой и экспорт идентификаторов соответствующих констант, на которые нужно ссылаться, используя в качестве уточняющего префикса имя модуля, в котором они определены? Или нужно, как в Модуле-2, разрешить употреблять эти идентификаторы констант без уточняющего префикса? Первый вариант показался непривлекательным, так как для констант получались бы длинные имена, а второй — из-за того, что идентификаторы могли бы появляться без всякого объявления. По этим причинам перечислительные типы были исключены.

Типы диапазонов так же были исключены. Опыт показал, что их использовали почти исключительно для индексирования массивов. Поэтому проверки диапазонов были нужны для индексирования, а не для присваивания переменным типа диапазонов. Нижняя граница массива была зафиксирована нулем, что сделало проверки индексов более эффективными, а типы диапазонов — еще менее полезными.

Типы множеств оказались мало полезными в Паскале и Модуле-2. Множества, реализованные как строки битов длиной в «слово» использовались редко, хотя объединение и пересечение могли вычисляться одной логической операцией. В Обероне мы заменили общие типы множеств единственным предопределенным типом SET с элементами от 0 до 31.

После длинных дискуссий было решено (в 1988) объединить текст, описывающий интерфейс модуля (definition), с текстом, описывающим его реализацию. С педагогической точки зрения это решение, возможно, было ошибочным. Ясно, что интерфейсы должны быть сначала спроектированы как контракты между проектировщиком и клиентами модуля. Вместо этого, теперь все экспортируемые идентификаторы должны просто помечены в их объявлениях (звездочкой). Преимуществом этого решения было то, что отдельный текст описания интерфейса стал излишним, и что компилятору было больше не нужно проверять согласованность двух текстов. Важный аргумент в пользу слияния состоял в том, отдельный текст описания интерфейса мог быть сгенерирован автоматически по тексту модуля.

---

[2] Н.Вирт, Программирование на Модуле-2.

Возможность квалифицированного импорта Модулы-2 так же была отвергнута. Теперь при каждом использовании импортированного идентификатора ему должно предшествовать имя модуля, в котором он определен. Это оказалось весьма полезным при чтении программ. Список импорта теперь содержит только имена модулей. Нам кажется, что это хороший пример искусства упрощения: уже упрощенная версия механизма модулей из языка Mesa была еще более упрощена без потерь для его смысла: упрятывания информации и типобезопасной раздельной компиляции.

Количество *низкоуровневых средств* было резко уменьшено, в частности, были удалены функции преобразования типов. Небольшое число оставшихся низкоуровневых функций было изолировано в особом псевдомодуле, имя которого должно стоять в хорошо видимом списке импорта каждого модуля, в котором такие низкоуровневые средства используются.

Устранение всех потенциально опасных возможностей явилось в высшей степени важным шагом к настоящему языку высокого уровня. Непроницаемая проверка типов (так же через границы модулей), строгая проверка индексов во время выполнения, проверка NIL-указателей, а также концепция безопасного расширения типов позволяют программисту полностью полагаться только на правила самого языка. Больше не нужно помнить о том, как устроен используемый компьютер, как транслируется язык программирования, или какое используется представление данных. Наконец была достигнута давняя цель, состоящая в том, чтобы язык был определен без упоминания о механизме исполнения. Полное абстрагирование от машины и настоящая переносимость стали реальностью. Кроме того, полная безопасность по типам данных является (эта истина часто игнорируется) безусловной необходимостью для автоматического управления памятью (сборщика мусора).

Упомянем одно свойство, которое, как теперь ясно, следовало добавить: это финализация, подразумевающая автоматическое выполнение указанной процедуры, когда происходит выгрузка модуля или утилизация записи (объекта) при сборке мусора. Включение финализации обсуждалось, но было сочтено, что ее стоимость и усилия по реализации были чрезмерны в сравнении с ее полезностью. Очевидно, её важность была недооценена, особенно в том, что касается выгрузки модулей.

5.2. Новые средства, введенные в Оберон.

В сущности, в Оберон введено всего два средства: расширение типов и включение типов. Это удивительно, принимая во внимание большое количество отсеянного.

Понятие *включения типов* связывает вместе все арифметические типы. Каждый из них определяет диапазон значений, которые может принимать переменная данного типа. Оберон содержит пять арифметических типов:

longreal ⊇ real ⊇ longint ⊇ integer ⊇ shortint

Идея здесь в том, что значения типов, стоящих в этой цепочке правее, могут быть присвоены переменным типов, стоящих левее. Поэтому, если даны объявления

var i: integer; k: longint; x: real

то разрешены присваивания k:=i и x:=i, тогда как i:=k и k:=x — нет. Теперь, оглядываясь назад, относительно большое число арифметических типов кажется ошибкой. Вероятно, могло бы хватить двух типов integer и real. Принятое решение оправдывалось важностью экономии памяти в то время, а также тем, что в целевом процессоре имелись наборы инструкций для всех пяти типов. Конечно, описание языка не запрещает одинаково реализовать integer, longint и shortint или real и longreal.

Несравненно более важным новым свойством было *расширение типов* [26, 39]. Вместе с полями процедурных типов в записях оно составляет техническую основу для объектно-ориентированного стиля программирования. Эта идея более известна под антропоморфным обозначением *наследование*. Рассмотрим тип записи (класс) *Window* (T0) с координатами x, y, шириной w и высотой h. Его объявление таково:

T0 = **record** x, y, w, h: integer **end**

T0 может быть базой для *расширения* (подкласса) TextWindows (T1), объявляемого так:

T1 = **record** (T0) t: Text **end**

Здесь подразумевается, что T1 содержит (наследует) все свойства (x, y, w, h) его *базового типа* T0, и в дополнение к ним содержит текстовое поле t. При этом также подразумевается, что все объекты типа T1 являются и объектами типа T0, что дает возможность строить неоднородные структуры данных. Например, элементы дерева могут быть определены как имеющие тип T0. Однако конкретные присваиваемые элементы могут иметь любой тип, являющийся расширением T0, например, T1.

Требуется только одна новая операция – *проверка типа*. Если дана переменная v типа T0, то булевское выражение v **is** T0 используется для определения эффективного текущего типа значения, хранящегося в v. Эта проверка выполняется во время выполнения программы.

Одного только расширения типов (вместе с процедурными типам) достаточно для программирования в объектно-ориентированном стиле. *Объектный тип* (класс) объявляется как запись, содержащая поля процедурных типов, так называемые *методы*. Например:

type viewer = **pointer to record** x, y, w, h: integer;

move: **procedure** (v:viewer; dx, dy: integer);

draw: **procedure** (v:viewer; mode: integer);

end

Операция рисования определенного объекта v типа viewer вызывается посредством v.draw(v,0). Первое вхождение v здесь служит для квалификации метода рисования как принадлежащего типу viewer, второе v определяет конкретный рисуемый объект.

В этом выявляется тот факт, что объектно-ориентированное программирование — это, в сущности, стиль, основанный на процедурном программировании и наследующий ему. Удивительно, что многие не сочли Оберон объектно-ориентированным языком просто потому, что в нем не использована соответствующая новая терминология. В 1990 году Мессенбек возглавил усилия по исправлению этого очевидного недостатка и по реализации незначительного расширения языка, названного Оберон-2 [34]. В Обероне-2 методы, то есть процедуры, связанные с типом записей, явно описывались как принадлежащие записи и подразумевали особый параметр, обозначающий объект, к которому должен быть применен метод. Как следствие, такие методы считались константами и поэтому потребовался дополнительный механизм их переопределения для подклассов.

…

7. Заключение и размышления

Моей давней целью было продемонстрировать, что систематическое проектирование с использованием соответствующего языка приводит к экономным и эффективным

программам, требуя лишь небольшую долю обычно затрачиваемых ресурсов. Эта цель была успешно достигнута. Я твердо убежден на основании многолетнего опыта, что структурированный язык способствует построению структурированных систем. Кроме того, было показано, что чистый и компактный проект целой программной системы может быть описан и объяснен в одной книге [37]. Вся Система Oberon, включая компилятор языка, текстовый редактор и оконный интерфейс занимает менее 200 килобайт оперативной памяти и компилирует сама себя менее чем за 40 секунд на 25-мегагерцовом компьютере.

Сейчас, в 2007 году, кажется, что такие числа вряд ли имеют значение. Когда ёмкость оперативной памяти измеряется в сотнях мегабайт, а вместительность жестких дисков — в десятках гигабайт, 200 килобайт — величина пренебрежимая. Когда частота процессоров измеряется в гигагерцах, скорость компиляции не имеет значения. Другими словами, чтобы пользователю пришлось ждать, программное обеспечение должно быть по-настоящему паршивым. Невероятный прогресс в аппаратном обеспечении оказал глубокое влияние на разработку программного обеспечения. Хотя благодаря этому прогрессу компьютеры достигли феноменальной производительности, его влияние на дисциплину программирования оказалось в целом довольно негативным. Благодаря ему качество и эффективность программного обеспечения страшно упало, так как низкая производительность маскируется быстрым аппаратным обеспечением. В преподавании понятие экономии памяти и процессорных циклов стало излишним. Фактически программирование больше не рассматривается как серьезный предмет; ему можно научиться мимоходом или, еще хуже, полагаясь на существующие программные «библиотеки».

Все это находится в резком контрасте с временами Алгола-60 и Фортрана. Языки должны были быть точно определены, а их недвусмысленность доказана; они должны были давать логичную, непротиворечивую основу для доказательства корректности программ, а не только подчиняться набору синтаксических правил. Такая честолюбивая цель могла быть достигнута, только если эта основа достаточно мала и проста. Современные языки, наоборот, все время растут. Их размер и сложность просто за пределами логического обоснования. Они практически недоступны человеческому пониманию. Технические руководства своей толщиной отталкивают любого, кто попытается просветиться с их помощью. Как следствие программированию обучаются не на основе правил и логических рассуждений, а скорее методом проб и ошибок. Слава интерактивным инструментам.

Похоже, мир терпит такое развитие событий. При такой ошеломительной мощи «железа», можно позволить себе расточительность в отношении пространства и времени. Но проблемы возникнут с другим: с удобством использования и с надежностью. Громадность современных коммерческих систем ставит пределы пониманию и порождает ошибки, приводя к ненадежным продуктам. Начали появляться признаки, что конец терпению приходит. За последние годы я узнал о растущем числе компаний, принявших Оберон в качестве основного инструмента программирования. Их общие черты — небольшая команда разработчиков и небольшое количество преданных клиентов, требующих программное обеспечение высокого качества, надежное и легкое в использовании. Создание такого программного обеспечения требует от создателей полного понимания своих продуктов. Естественно, такое понимание должно охватывать операционную систему и библиотеки, на основе которых строится программа. Но постоянное усложнение коммерческого программного обеспечения сделало такое понимание невозможным. Эти компании нашли в Обероне реальную альтернативу.

Не удивительно, что эти компании состоят из небольших команд опытных программистов, достаточно компетентных, чтобы принимать смелые решения, и пользующихся доверием небольшой группы довольных клиентов. Не удивительно и то, что такие маленькие системы как Оберон находят основное применение в создании встроенных систем для сбора данных и

управления в реальном времени. Здесь не только экономичность является главной заботой, но еще более — надежность и устойчивость.

Тем не менее, такие клиенты и такие приложения остаются исключением. Рынок отдает предпочтение языкам коммерческого происхождения, вне зависимости от их технических достоинств или дефектов. Инерция рынка огромна, поскольку ее определяет множество обратных связей, устроенных по принципу порочного круга. Поэтому законным является вопрос, имеют ли смысл и значение исследования по созданию новых языков программирования. И этот вопрос следует задать.

Новые идеи по усовершенствованию дисциплины программирования приходят из практики. Их нужно выразить в какой-то нотации, которая в конце концов образует конкретный язык, который должен быть реализован и проверен «в полевых условиях». Понимание, достигнутое таким путем, проникает в новые версии широко используемых коммерческих языков, медленно, очень медленно в течение десятилетий. Можно утверждать, что Паскаль, Модула-2 и Оберон успешно внесли свой вклад в этот процесс.

Однако их главная идея выражена в заглавии Сообщения о языке Оберон: «Делай как можно проще...» Это пожелание еще не нашло полного понимания [41]. Похоже, что коммерческие интересы направлены в другую сторону. Время рассудит.

Благодарности

Проектирование, реализация и развития Модулы-2 и Оберона были результатом командных усилий на протяжении длительного времени. Я чувствую себя обязанным всем, кто принимал во всем этом участие как член команды или как пользователь, обеспечивавший ценную обратную связь. Я сделал правилом реализовывать и тестировать язык *до* публикации. поэтому работа моих коллег имела огромную важность, и я выражаю им свою благодарность. На самом деле разработчики (в случае Оберона и я сам) были нашими первыми «подопытными кроликами» и источниками ценной информации.

Прежде всего я должен поблагодарить Федеральный технический институт (ETH) в Цюрихе за постоянную молчаливую поддержку и доверие нашим исследованиям. На протяжении трех десятилетий четыре или пять ассистентов (аспирантов) помогали мне, работая на должностях в институте. Наши проекты давали им темы для диссертациям и практику, сделавшую их отличными инженерами-программистами. Мне редко приходилось искать финансирование, и мне удалось избежать написания бесчисленных заявок и отчетов по грантам. Поэтому я мог сосредоточиться на преподавании и исследованиях. Я был свободен от давления индустрии и предрассудков, мог осуществлять такие проекты, какие считал правильными, и необязательно такие, какие требовалось или которые были модными.

Считаю нужным подчеркнуть, что старая традиция ETH поддерживать тесную связь между преподаванием и научными исследованиями сыграла весьма важную роль. Мне сразу принесла пользу необходимость придерживаться в своих построениях простоты и ясности, без которых невозможно эффективное преподавание. Тем самым мне удалось реализовать долгосрочные ценности простого, логичного проектирования. Обстановка для наших исследований была идеальной.