Руслан Богатырев

Оберон. Коротко о главном. Часть 2

Краткие ответы на ключевые вопросы (техническая часть)

1. Правда ли, что в Обероне убраны паскалевские скобки **begin-end**?
2. Почему все зарезервированные слова в Обероне записываются большими буквами?
3. Почему Оберон не имеет оператора **goto**?
4. При отказе от использования **goto** проводились ли исследования о практике применения этого оператора в других языках?
5. Что Вирт исправил в операторе **IF**?
6. Почему из Оберона исключены перечисления и диапазоны?
7. Почему индексы массивов всегда нумеруются от нуля?
8. Почему в Обероне нет цикла **FOR** ?
9. Зачем из Оберона исключены вариантные записи?
10. Почему в Обероне нет обработки исключений?
11. Чем принципиально отличается поддержка ООП в Обероне от ее воплощения в ведущих промышленных языках?
12. Что такое модуль в понимании Оберона и чем модульное программирование помимо отсутствия полиморфизма и наследования отличается от ООП?
13. Почему в Обероне нет ничего подобного P-коду? Его ведь создал Вирт, а идеи использовались потом в Java и .NET.

1. Правда ли, что в Обероне убраны паскалевские скобки begin-end?

Да, это было сделано Виртом еще в языке Modula-2 (1979). Начало синтаксической конструкции (ветвления и цикла) уже определяется первым словом этой конструкции (**IF**, **CASE**, **WHILE**, **REPEAT**, **LOOP**), поэтому слово **BEGIN** здесь излишне. Для тел инициализации модулей и тел процедур скобки **BEGIN-END** используются, но с некоторой модификацией — обязательным указанием после **END** имени соответствующего модуля/процедуры.

2. Почему все зарезервированные слова в Обероне записываются большими буквами?

Этот подход был введен Виртом еще в языке Modula-2 и показал свою высокую ценность в плане четкого выделения структуры исходного текста. Он был заимствован из языка Mesa (1975), разработанного в Xerox PARC. С появлением синтаксически ориентированных редакторов, позволяющих цветом выделять зарезервированные слова, достоинства такого подхода стали не столь явными. Однако, не следует забывать, что цветовое выделение не стандартизировано (ни для одного из популярных языков) и теряется при публикации в печатных и электронных изданиях, а также в ходе электронной переписки. Относительное неудобство переключения регистров при наборе текста легко компенсируется средствами соответствующей системы программирования (IDE), что, в частности, в BlackBox было сделано.

Следует заметить, что непривычность написания зарезервированных идентификаторов большими буквами в языках Оберон-семейства воспринимается многими программистами, выросшими на куда менее строгих правилах Delphi и языков Си-направления, как едва ли не главный барьер на пути постижения Оберона. Очевидно, что это отличительная черта, стиль, в котором никто не вправе отказывать языку. И если именно в ней и заключается основной недостаток языка, значит, он действительно достоин детального изучения.

3. Почему Оберон не имеет оператора goto?

Оператор **goto** был изъят Виртом из Паскаля еще при создании языка Modula-2 (1979). Обоснование изъятия **goto** из языков высокого уровня было предложено в известной работе Эдсгера Дейкстры (Edsger W. Dijkstra "Go To Statement Considered Harmful" // Communications of the ACM, Vol. 11, No. 3, March 1968).

Как показывает практика программирования (особенно последних лет) используют **goto** отнюдь не для оптимизации объектного кода, а, как правило, чтобы сэкономить свое время на кодирование конкретного алгоритма.

Следует отметить:

1. Оператора **goto** нет в таких языках, как Smalltalk (А.Кей, 1972), Modula-2 (Н.Вирт, 1979), Eiffel (Б.Мейер, 1986), Оберон (Н.Вирт, 1987), Modula-3 (DEC SRC, 1988), Оберон-2 (Х.Мессенбок, 1991), Java (Дж.Гослинг, 1995), Компонентный Паскаль (Oberon microsystems, 1997). Это далеко не полный перечень.

2. Оператор **goto** более чем проблемно использовать в блоках контроля и обработки исключений вне зависимости от того, включен ли механизм обработки исключений в язык или он поддерживается на библиотечном уровне.

В одних языках (Modula-2, Оберон и др.) **goto** часто моделируется связкой **LOOP**-**END** с оператором **EXIT**, в других для этой цели применяют в том числе и программные исключения (exceptions), которые не только позволяют корректно передавать управление на нужную точку, но и нивелировать последствия такого нарушения линейного исполнения кода (утилизация ресурсов, обработчики исключений и т.п.).

При этом следует упомянуть, что обоснованность использования **goto** в языке высокого уровня отстаивается до сих пор. Самый яркий пример последних лет — язык C# (Microsoft, 2000).

4. При отказе от использования goto проводились ли исследования о практике применения этого оператора в других языках?

Есть интересные данные, полученные в ходе исследований Марком Брандисом (Marc Brandis) из ETH Zurich. Результаты работы включены в доклад, представленный на конференции по модульным языкам программирования в Ульме (Joint Modular Language Conference, 1994).

В качестве сравнения брались крупные системы, реализованные на Си.

1. X Window 11 R5 (MIT)
2. GNU C Compiler 2.6.0 (для IBM RS/6000), Free Software Foundation
3. ANSI C Compiler LCC 3.0

Сначала проводилась фильтрация: например, в GNU CC кол-во **goto** было 5701, потом выяснилось, что более 5000 лежали в одном-единственном файле, где реализовывался конечный автомат. Причем текст автомата генерировался автоматически из описания. Естественно из расчета этот фрагмент выбросили.

пакет	исх.файлов	строк	#goto	строк/goto
X Window 11 R5	1747	735038	900	817
GNU CC 2.6.0	111	288510	679	425
LCC 3.0	29	24684	18	1371
Всего	**1887**	**1048232**	**1597**	**656**

Вывод Брандиса: относительно небольшое кол-во **goto** говорит о том, что структурное программирование не забывают и в Си. Более того, выявилась тенденция концентрации **goto** в небольшом числе файлов. Так, напр., ни одного **goto** не было в 92% исходных текстов (смотрелись только *.c) для X Window, в 50% для GNU CC и в 90% для LCC. Брандис приводит

подробный график распределения, из которого видно, что критичные 5, 20, 100 и 200 строк на одно **goto** занимают от 20 до 500 файлов (в основном в X Window).

Выводы исследования:

1. Скрытые **goto** в программах — вполне нормальное явление и не нужно пытаться обязательно от них избавляться. Чаще всего, оптимизирующий компилятор сделает это при необходимости сам.
2. Что касается обычного **goto**, то лучше, когда в языке его нет, но когда он есть, то концентрируется обычно там, где код пишется неаккуратно (плохой стиль конкретного программиста).

В рамках данного исследования изучалась также частота использования различных операторов цикла языка Оберон на основе анализа исходных текстов системы Oberon System (для IBM RS/6000).

По циклам в Обероне выявилось, что из 1479 циклов доля **WHILE** — 1113 (75,2%), **REPEAT** — 232 (15,7%), а **LOOP** — 134 (9,1%). **LOOP** с одной точкой выхода (**EXIT**) Оберон-программисты использовали, чтобы избегать дублирования кода. Но можно легко трансформировать эту структуру в **WHILE**.

Фрагмент

```
LOOP
  stat0;
  IF pred THEN EXIT END;
  stat1
END;
```

эквивалентен

```
stat0;
WHILE ~pred DO
  stat1;
  stat0
END;
```

Интересно отметить, что Гризмер (Griesmer) переписал фрагменты своего компилятора Oberon-V, избавившись от **LOOP** в пользу **WHILE**. При этом он обнаружил, что в большинстве случаев текст стал понятней.

5. Что Вирт исправил в операторе IF?

Оператор **if-then-else**, пришедший из языка Алгол-60 и включенный Виртом в Паскаль, является неоднозначным, когда он вложен в другой **if-then-else**: не очевидно, к какому **if** относится данная **else**-часть.

В своей лекции "Good Ideas — Revised" в МГУ (19.09.2005) Никлаус Вирт привел такой пример:

```
if b0 then if b1 then S0 else S1
```

Из него не ясно, как трактовать данную конструкцию:

1. `if b0 then if b1 then S0 else S1`
2. `if b0 then if b1 then S0 else S1`

В языке Modula-2 эта ошибка была Виртом устранена путем замены **if-then** на **IF-THEN-END** и **if-then-else** на **IF-THEN-ELSE-END**, точнее говоря, включением **END** в состав этой конструкции ветвления

```
IF b0 THEN
  IF b1 THEN S0 ELSE S1 END
END;
```

или

```
IF b0 THEN
  IF b1 THEN S0 END
ELSE S1
END;
```

Оператор стал однозначным, даже если в нем используется много вложенных **IF-THEN-ELSE-END**. Кроме того, Вирт включил в оператор **IF** и **ELSIF**-часть, чтобы удобнее и нагляднее было применять каскадный оператор ветвления без необходимости изменения уровня вложенности.

6. Почему из Оберона исключены перечисления и диапазоны?

Перечисление (enumeration) — довольно удобный тип, введенный в языке Паскаль (1970) и сохраненный в Modula-2 (1979). Тем не менее, в Обероне он исключен.

Причин несколько: во-первых, при введении в Обероне нового механизма расширения типов выясняется, что этот тип становится особым — его нельзя расширять. Кроме того, поскольку тип перечисление определяется совокупностью идентификаторов констант, то при экспорте требуется передавать все значения (хотя наличие спецификаторов экспорта позволяет экспортировать значения на выбор). Это ввело бы ненужную путаницу. Вирт принял решение отказаться от перечислений, считая, что вполне достаточно в случае острой необходимости имитировать такой тип набором констант.

> Тип перечисление — слишком простое средство, чтобы оно могло выйти из-под контроля. Однако, оно не позволяет распространять расширяемость за пределы модуля. И либо нужно ввести средство для расширения типа перечисление, либо же от типа перечисление надобно отказаться. Причина, по которой мы выбрали второй путь — путь радикального решения — кроется в том, что во все возрастающем числе программ непродуманное использование перечислений (и диапазонов) ведет к демографическому взрыву среди типов, что, в свою очередь, ведет не к ясности программ, а к их многословию. В связи с использованием экспорта и импорта перечисления приводят к исключению из правил, и согласно ему импорт идентификатора типа также приводит к автоматическому импорту всех связанных с типом идентификаторов констант. Это исключение нарушает концептуальную простоту и создает для реализаторов языка неприятные проблемы.
>
> Niklaus Wirth. From Modula to Oberon (1988)

Диапазоны (subrange) также исключены из Оберона, хотя и были представлены в Modula-2. Чаще всего диапазоны использовались в качестве типов индексов для массивов. В Обероне Вирт принял решение вернуться к тому, что было введено еще в Си (1971) — все массивы нумеруются от нуля. Понятно, что "абсолютные адреса" (индексы в рамках диапазонов) легко отобразить на "относительные" (от нуля). Иными словами, эта забота перекладывается на плечи программиста. Но в подавляющем большинстве случаев она не является серьезным ограничением.

> Типы диапазонов были введены в Паскале (и сохранены в Modula-2) по двум причинам: (1) чтобы подчеркнуть тот факт, что переменная принимает значения из ограниченного диапазона базового типа, и чтобы дать возможность компилятору генерировать соответствующий проверочный код для присваиваний; (2) чтобы позволить компилятору выделять минимально необходимое пространство памяти для хранения значений из указанного диапазона. Это желательное свойство в плане использования упакованных записей. Лишь в небольшом числе реализаций используется данное преимущество экономии памяти, поскольку при этом компилятор довольно значительно усложняется. Причины под номером (1) явно недостаточно, чтобы оставить в Обероне работу с диапазонами.
>
> Niklaus Wirth. From Modula to Oberon (1988)

7. Почему индексы массивов в Обероне всегда нумеруются от нуля?

Это следствие исключения из языка перечисления и диапазонов (см. ранее). Достаточно радикальное решение, безусловно упрощающее язык и компилятор, но создающее определенные неудобства программисту. Положительный момент такого решения в том, что он стимулирует всегда рассматривать в качестве нижней границы индекса только 0. Нередко же в качестве нижней границы в Паскале и Modula-2 выбирают, не задумываясь 1, а затем уже, спустя некоторое время, приходят к мысли о том, что 0 более удобен (в частности, при выполнении циклических преобразований с индексом).

> Понятие определяемого программистом типа индексов для массивов также не вошло в Оберон: теперь все индексы могут быть только целыми числами. Более того, нижняя граница индексов теперь строго равна 0 и в

> описании массива фигурирует только количество элементов, а не пара значений, определяющих границы индексов. Этот отход от уже устоявшейся традиции, восходящей своими корнями еще к языку Алгол-60, наглядно демонстрирует принцип удаления из языка всего несущественного. Спецификация произвольной нижней границы вряд ли придает языку дополнительную выразительную силу. В то же время это является скорее ограниченным видом отображения индексов, которое ведет к скрытым вычислениям, не сравнимым с удобством использования. Скрытые накладные расходы проступают особенно сильно при проверке границ и при работе с динамическими массивами.
>
> Niklaus Wirth. From Modula to Oberon (1988)

8. Почему в Обероне нет цикла FOR ?

Это не совсем так. Цикл **FOR** использовался в языке Modula-2, но в Обероне сначала был изъят. Это обусловлено тем, что большое количество ошибок программирования возникает именно при использовании данного цикла.

Из-за простоты формы его использованием нередко злоупотребляют в тех случаях, когда другие виды операторов цикла (с предусловием — **WHILE** и с постусловием — **REPEAT**) более предпочтительны. Особый случай — переменная цикла.

Для надежности область ее видимости (и область существования) должна ограничиваться рамками цикла (быть неопределенной, а еще лучше — невидимой, до и после цикла). Именно такой подход был взят на вооружение в языке Modula-3 (1988), разработанном в DEC Systems Research Center и Olivetti на базе языка Modula-2. Почему — понятно. Многие ошибки возникают из-за того, что за рамками цикла используется эта переменная, притом, что ее значение должно быть неопределено.

> Отказ от оператора FOR — другой пример разрыва древней традиции. Замысловатый механизм оператора FOR, принятый в языке Algol-60, был значительно упрощен в языке Паскаль (и в Modula-2). Незначительная практическая ценность этого оператора привела к тому, что в Обероне он отсутствует.
>
> Niklaus Wirth. From Modula to Oberon (1988)

В Обероне Вирт вначале пошел на радикальный шаг — принял решение вообще отказаться от использования цикла **FOR**, поскольку его легко могут заменить циклы **WHILE**, **REPEAT** и универсальный **LOOP** (точнее, остался от Modula-2, где он всегда был и прекрасно себя зарекомендовал). С его помощью реализовывать можно абсолютно любой цикл. Но в том-то и состоит принцип разумного минимализма (вспомните изречение Эйнштейна, вынесенное в эпиграф описания Оберона — оно там не просто так) — избыточность даже минималистскому языку не противопоказана. **WHILE** и **REPEAT** прекрасно дополняют друг друга, а **LOOP** служит для особых случаев. **FOR** проще всего реализовывать через **WHILE** с заведением явной переменной цикла (и всеми явными, а не скрытыми, проблемами, которые вытекают из ее применения за границами цикла).

В языке Оберон-2 (и затем в Компонентном Паскале) цикл **FOR** был восстановлен в своих правах, оставаясь при этом сомнительным компромиссом удобства работы.

Важно отметить, что в новой книге "Programming in Oberon" (2004) Никлаус Вирт упоминает **FOR** как полноценный оператор языка Оберон. По всей видимости, консерватизм и сопротивление программистов были столь сильны, что Вирт вынужденно пошел на уступку:

> Рекомендуется, чтобы оператор FOR использовался только в простейших случаях, в частности, операнды выражения, определяющего диапазон значений переменной цикла, не должны изменяться внутри цикла. Кроме того, и сама переменная цикла не должна подвергаться модификации внутри цикла. Значение переменной цикла после завершения оператора FOR должно считаться неопределенным.
>
> Niklaus Wirth. Programming in Oberon (2004)

9. Зачем из Оберона исключены вариантные записи?

Функциональность вариантных записей, используемых в Паскале и Modula-2, сохраняется в Обероне за счет введения концепции расширяемых типов данных (type extensions), это уникальное отличие Оберона.

Приведем пример:

```
T  = RECORD x, y: INTEGER END;
T0 = RECORD (T) z: REAL END;
T1 = RECORD (T) b: BOOLEAN; s: CHAR END;
```

Это означает, что базовый тип T, содержащий поля x и y, имеет два производных типа (расширения, обогащения). T0 в итоге содержит поля x, y и z. Тип T1 — поля x,y, b, s.

Расширение типа действует как в отношении статических типов (на основе полей комбинированного типа — **RECORD**), так и динамических (на основе полей ссылочного типа — **POINTER**).

Если в языке оставлять вариантные записи (как избыточность по отношению к расширению типа), то это серьезное препятствие к реализации полноценного сборщика мусора.

> В системах, опирающихся на использование автоматической утилизации памяти (сборку мусора), вариантная запись является большим камнем преткновения. Работа с ней является сложной и нескладной. Попутно отметим, что объявление вариантной записи может содержать несколько взаимоисключающих вариантов на одном и том же уровне и что они могут быть вложенными. Практически невозможно использовать сборщик мусора в отношении вариантных записей без введения дополнительных ограничений.
>
> Niklaus Wirth. Type Extensions (1988)

> Концепция задуманной операционной системы потребовала высокодинамичного централизованного распределения памяти, операющегося на технологию сборки мусора. И хотя Modula-2 в принципе никак не препятствует встраиванию соответствующего сборщика мусора, наличие в языке вариантных записей создает серьезные препятствия. Поскольку новый механизм расширения типов сделал вариантные записи излишней возможностью, то логичным решением было попросту изъять этот ненужный элемент. Такой шаг привел к ограничению (подмножеству) языка Modula-2.
>
> Niklaus Wirth. From Modula to Oberon (1988)

10. Почему в Обероне нет обработки исключений?

Для простой по своей архитектуре системы Oberon, которая была реализована на языке Оберон, опираясь на механизм команд (экспортируемые модулями процедуры без параметров), в этом не было необходимости. Это явилось первым побудительным мотивом отказаться от использования обработки исключений в языке Оберон.

Вторая причина состоит в том, что обработка исключений (exception handling), близкая по своей природе к оператору **goto** и программным прерываниям, является достаточно опасным средством, при этом потребность использования исключений острее всего проявляется в наиболее критичных задачах (системы реального времени).

Для них обработка исключений, как показал опыт использования предшественника Оберона — языка Modula-2, может быть эффективно вынесена на уровень библиотек. Результаты подобного исследования для обработки исключений в среде параллельных процессов были опубликованы в журнале Structured Programming (1993), в редколлегии которого были Дональд Кнут и Никлаус Вирт, группой отечественных специалистов из Московского авиационного института — это была первая публикация советских ученых в столь авторитетном издании. См. I.Egorov, R.Bogatyrev, D.Petrovichev. Yet Another Approach to Modula-2 Implementation of Exception Handling Mechanism // Structured Programming, 1993, #14(1), pp. 23-36.

Другой подход был предложен проф. Ханспетером Мёссенбёком, автором языка Оберон-2, и его коллегами из Университета в Линце им. Кеплера (Австрия). См. Markus Hof, Hanspeter Mossenbock, Peter Pirkelbauer. Zero Overhead Exception Handling Using Metaprogramming // 1996.

11. Чем принципиально отличается поддержка ООП в Обероне от ее воплощения в ведущих промышленных языках?

Если сформулировать коротко, то

- Модули (module) и расширения типа (type extension) против концепции классов (class) и пространств имен (namespace).
- Процедурные переменные против процедурных констант.
- Подход с акцентом на экземпляры (instance-centred approach) против подхода с акцентом на классы (class-centred approach)

> Механизм классов предусматривает, что все процедуры, применимые к объектам класса, будут определены вместе с описанием данных. Эта догма проистекает из понятия абстрактного типа данных, но она служит серьезным препятствием при разработке больших систем, где крайне желательна возможность добавить новые процедуры, определенные в дополнительных модулях. Очень неудобно обязательно переопределять класс только лишь потому, что какой-то метод (процедура) должен быть добавлен или изменен, в особенности когда это изменение требует перекомпиляции описания класса и всех его клиентных модулей.
>
> Niklaus Wirth. From Modula to Oberon (1988)

> Объявление класса в объектно-ориентированных языках выглядит подобно объявлению записи с дополнительными объявлениями процедур (или их заголовков). Такой подход обладает некоторыми достоинствами и ведет к определенным последствиям. В языках Modula-2 и Оберон соответствующее поле процедурного типа предполагает особую роль переменной и, следовательно, фактическая процедура должна присваиваться этой переменной явно всякий раз, когда порождается новый экземпляр записи. Это можно рассматривать либо как нагрузку (и как источник ошибок), либо как дополнительную степень свободы (и мощи). К тому же большинство типичных приложений привязывает одну и ту же процедуру (обработчик) ко всем экземплярам класса: взгляд с позиции методов ориентирован на классы; взгляд с позиции Оберона — на экземпляры.
>
> <...>
>
> Различия между подходами с акцентом на классы (class-centred) и с акцентом на экземпляры (instance-centred) по отношению к описанию методов можно рассмотреть под другим углом. В первом случае методы объявляются как (процедурные) константы, во втором — как (процедурные) переменные. Ограниченные возможности первого подхода уже проявились в отношении подклассов, т.е. расширений типов. Обычно подкласс делает методы отличными от методов своего суперкласса. И так как методы задаются в виде констант, вызывается новая реализация, которая переопределяет описание суперкласса. В соответствующих реализациях переопределение (перегрузка) осуществляется за счет выделения своей таблицы методов для каждого подкласса. В ориентированном на экземпляры подходе языка Оберон не требуется ни подобный дополнительный механизм, ни дополнительная символика переопределения, так как все это достигается через обычное явное присваивание.
>
> <...>
>
> Как результат, Оберон концептуально проще, и его реализации не нагружаются дополнительными механизмами работы с классами. С другой стороны, объектно-ориентированные языки могут предоставлять несколько более удобную символику и обеспечивать безопасность за счет гарантии неизменности объявленных методов для всех экземпляров класса, что выражается в улучшенной эффективности отложенных вызовов. Правда, это стоит рассматривать как незначительное преимущество, поскольку есть уверенность в том, что использование объектно-ориентированной парадигмы должно быть осознанным и там, где это в самом деле нужно. При проектировании операционной системы [Oberon] мы обнаружили, что практически вся система была успешно запрограммирована в традиционном стиле, а объектно-ориентированный стиль затронул лишь систему визуализаторов, которая предоставляла распределенное управление.
>
> Niklaus Wirth, Modula-2 and Object-Oriented Programming (1990)

Детали подхода Вирта к поддержке ООП в Обероне изложены в его новой книге "Programming in Oberon" (October 2004), в 4-й главе, посвященной ООП.

В родоначальнике ООП, языке Simula, использовались поля данных (прообраз комбинированного типа **record** языка Паскаль), но не было понятия методов (они появились в Smalltalk) — поведенческим аспектом класса занималась сопрограмма (coroutine).

Сопрограмма была положена в основу квазипараллелизма языка Modula-2, но в Обероне этот механизм не используется (вынесен за пределы языка).

Оберон предлагает использовать схему, близкую Simula: вместо набора методов, реализуемых через поля процедурного типа (т.е. коммутаторов процедур) заводится одна выделенная процедура-обработчик (handler). Она имеет два параметра: объект, к которому применяется, и сообщение (message). Сообщение выражается в виде расширяемого комбинируемого типа (**RECORD**), это определяет протокол класса. Подобный подход показал свою эффективность и наглядность при реализации системы Oberon (1985—1995).

При таком подходе объект (object) представляется указателем на запись. Запись содержит всего одно поле процедурного типа — handle. При инициализации (порождении объекта) этому полю явно присваивается процедура-обработчик, которая в точности удовлетворяет спецификации процедурного типа по набору формальных параметров процедуры. Обработчик имеет два параметра: объект-адресат и сообщение. Сообщение определяется то действие, которое требуется применить по отношению к объекту-адресату сообщения. Обработчик создается из коммутатора сообщений, реализуемого через выделенный условный оператор **IF-ELSIF-END**. Сообщения могут отправляться в режиме широковещательной рассылки всем объектам с

разнородной структурой данных без знания характера этой структуры. Неприемлемые сообщения будут просто игнорироваться.

12. Что такое модуль в понимании Оберона и чем модульное программирование помимо отсутствия полиморфизма и наследования отличается от ООП?

Расширенный ответ на этот вопрос можно найти в статье Р. Богатырев "Модульное программирование: Terra Incognita" (2005).

13. Почему в Обероне нет ничего подобного P-коду? Его ведь создал Вирт, а идеи использовались потом в Java и .NET.

Для обеспечения мобильности кода нет никакой необходимости в реализации виртуальных машин, подобных Java VM. Проект Оберон наглядно доказал, что более эффективен и перспективен подход кодогенерации на лету, когда генерация машинного кода из компактного промежуточного представления, полученного после компиляции, осуществляется самим загрузчиком с конкретизацией под определенную аппаратную платформу. Этот подход был разработан Микаэлем Францем (Michael Franz), аспирантом Никлауса Вирта, защитившим на эту тему диссертацию в 1994 г. в ETH Zurich.