Йозеф Темпл

Oberon против C ++

Йозеф Темпл (1995) «Oberon против C++» // Технология программирования, Т.1, No.2, с.

Josef Templ (1994) «Oberon vs. C++» // iX, September (in German).

Сведения об авторе

Йозеф Темпл (Josef Templ) — специалист в области языков программирования. Ранее работал в Швейцарском Федеральном технологическом институте, где защитил докторскую диссертацию по теме «Metaprogramming in Oberon» (1994). Работал над реализацией SPARC Oberon и совместно с профессором Мессенбоком (H.Moessenboeck) над экспериментальным диалектом Object Oberon. В настоящее время работает в университете Йоганна Кеплера in Johannes Kepler University, Linz, Austria.

Аннотация

Настоящая работа посвящена сопоставлению языков Oberon(-2) и C++. Автор весьма убедительно показывает, насколько успешно концепции языка Oberon и их реализация конкурируют с языком C++. Затронуты такие важные аспекты обоих языков, как поддержка объектно-ориентированного программирования, механизмы передачи родовых сообщений, множественное и динамическое наследование, обработчики сообщений, классы и виртуальные функции, обработка исключений, сборка мусора, родовые средства, перегрузка операций. Особое внимание уделяется вопросам обеспечения надежности (safety) в языке программирования. Показано значительное преимущество в этом плане языка Oberon и то, какую опасность здесь представляет собой адресная арифметика C++. Абстрагирование данных посредством модулей, концепция расширения типа и типизированные процедуры — все эти средства Oberon(-2), о которых программисты C++ могут еще только мечтать, позволяют не только добиться эквивалентной выразительной мощи языка при несопоставимо малом его размере, но и наряду с поддержкой строгой статической типизации языка структурного программирования предоставить большую гибкость в выборе того или иного стиля объектно-ориентированного программирования.

Впервые статья была напечатана в 1994 году на немецком языке в журнале iX (Heise Verlag), на английском она вышла в авторском переводе в журнале The ModulaTor (1994, Vol.4, No.9).

Ключевые слова

Oberon, C++, расширение записи, надежность, сборка мусора, обработка исключений, запись-сообщение, обработчик сообщений.

ВВЕДЕНИЕ

Если язык C++ повсеместно принят в области программной индустрии, то язык Oberon пришел на смену языку Pascal в области образования. Сопоставление этих двух языков демонстрирует их концепции и различия.

Когда профессоры Никлаус Вирт (Niklaus Wirth), который хорошо известен многим по созданию языков Pascal и Modula-2, и Юрг Гуткнехт (Jurg Gutknecht) в середине 80-х годов приступили к разработке новой операционной системы для персональных компьютеров, существующие языки программирования не обладали достаточными средствами для построения расширяемых программных систем [Ger94]. Даже такие громоздкие языки, как PL/1 и Ada не могли быть использованы для создания устойчивых и надежных программ, которые впоследствии могли бы расширяться. Небольшие языки вроде Pascal, Modula-2 и C имели (несмотря на немалое число удачных концепций) недостатки в собственной системе типов, чересчур существенные для этой новой тенденции в инженерии программного обеспечения.

Никлаус Вирт сформулировал это требование расширения и построил язык, который был призван вобрать в себя большую выразительную силу и одновременно стать даже проще в изучении и использовании, чем его предшественники. Результатом его работы стал язык Oberon [ReW94]. Настоящая работа призвана прояснить характерные черты Oberon'а и языка C++ [Str92], преемника языка C, который был разработан Бьярном Страуструпом (Bjarne Stroustrup) приблизительно в то же время и примерно по тем же причинам. Сопоставление это видится вполне корректным, поскольку оба языка претендуют на универсальность, строгую типизацию, объектную ориентированность и эффективность их реализации.

C++ в настоящее время практически стал промышленным стандартом, хотя существующие его реализации порой значительно отличаются друг от друга. Oberon снчала появился в академических и учебных кругах, подобно тому, как это было с языком Pascal более 20 лет назад. Сейчас имеются свободно доступные реализации языка для всех ведущих программных и аппаратных платформ, включая DOS, Windows 3.1, Windows NT, OS/2, AIX, Solaris, Ultrix, Irix, HP-UX и Linux.

АБСТРАГИРОВАНИЕ ДАННЫХ ПОСРЕДСТВОМ МОДУЛЕЙ

Синтаксические различия языков — это не предмет для нашего сравнительного анализа. Вместо этого мы попытаемся сфокусировать свое внимание на на механизмах абстракции и на надежности, обеспечиваеиое тем или иным языком. Абстрагирования означает освобождение от излишних деталей. Это наиболее важное средство для управления сложностью. В плане программирования абстрагирование означает игнорирование реализации и учет лишь спецификации или интерфейса. Программы в языке Oberon состоят из набора модулей, которые взаимодействуют через механизм экспорта-импорта и которые позволяют скрывать от клиентных модулей детали реализации.

Модули в Oberon'е служат для выражения таких концепций, как абстрактные структуры данных, абстрактные типы данных или просто для реализации библиотек различных функций. Модули также выступают в роли единиц компиляции, а в системе Oberon и в роли единиц расширения системы. Иными словами, модуль может быть подгружен по запросу на этапе выполнения и, тем самым, обеспечить расширение функционирующей программы за счет новой функциональности. Более того, пользователь может вызывать экспортируемые процедуры без параметров в качестве команд операционной системы. Модуль может также содержать тело, которое обычно используется для инициализации глобальных переменных.

На рис.1 приведен небольшой пример. В спецификации импорта (IMPORT) указываются все импортируемые и, таким образом, доступные модули. Клиент модуля M может использовать только те объекты, которые помечены знаком экспорта (*), идущим следом за именем объекта. (например, тип T*). Чтобы можно было различать объекты, имеющие в разных модулях одинаковые имена, Oberon требует использовать перед именем объекта имя экспортирующего его модуля (например, M.P). Это позволяет избежать конфликтов имен и помогает в чтении программы, поскольку всегда понятно, откуда тот или иной объект.

```
MODULE M;
```

```
IMPORT M1, M2 := MyModule;

TYPE
T* = RECORD
f1* : INTEGER;
f2 : ARRAY OF CHAR
END;

PROCEDURE P* (VAR p: T);
BEGIN
M1.P(p.f1,p.f2);
END P;

END M.
```

рис.1 Пример модуля в языке Oberon. В спецификации импорта указаны все импортируемые модули. Экспортируемые объекты помечены звездочкой.

Импортируемые модули могут быть переименованы в спецификации импорта. Это позволяет сокращать длинные имена, а также экспериментировать с различными вариантами модуля без внесения многочисленных изменений в его клиентов (M2 := MyModule). Поля записи можно экспортировать выборочно (f1*), при этом можно одни поля экспортировать, а другие оставлять приватными.

C++ ИМИТИРУЕТ МОДУЛИ С ПОМОЩЬЮ ПРЕПРОЦЕССОРА

Язык C++ не имеет концепции модуля, но имитирует ее хорошо известным способом через C-препроцессор (cpp) и через соответствующие программные соглашения (header-файлы). Глобальное пространство имен, которое C++ унаследовал от C, не исключает конфликтов имен на этапе компоновки программы. Чтобы избежать этой проблемы, для имитирования модульного пространства имен иногда используют классы. В случае взаимосвязанных классов и процедур, которые ссылаются более чем на один класс, должны использоваться так называемые друзья (friends). Они представляют своего рода goto областей видимости — конструкцию, которая позволяет обходить обычные правила видимости языка C++. Друзья делают имена видимыми там, где в обычном случае их нельзя было бы увидеть. Поскольку этот механизм остается по-прежнему неудовлетворительным для построения больших программных систем, в комитете по стандартизации языка C++ [Str94] обсуждается вопрос введения пространств имен (namespaces) — расширений механизмов областей видимости. Однако, пространства имен все еще опираются на C-препроцессор, а потому их нельзя рассматривать как настоящую концепцию модуля на уровне языка.

Другой аспект языка C++, который часто подвергается критическим замечаниям, — отсутствие порядка инициализации. Эта проблема также решается модулями языка Oberon. В отличие от include-механизма cpp, отношение импорта запрещает наличие циклических связей. Таким образом, импортируемые модули всегда будут проинициализированы до своих клиентов.

Для системного программирования Oberon предоставляет псевдомодуль SYSTEM, который содержит в себе операции, зависящие от реализации и от аппаратной платформы. Те модули, которые импортируют модуль SYSTEM, заведомо являются непереносимыми и ненадежными, но их легко можно распознать по наличию в списке импорта ключевого слова SYSTEM. C++ позволяет использовать низкоуровневые операции без специального выделения таких программ. И когда программы переносятся с одной аппаратной платформы на другую, это может привести к неприятным сюрпризам и долгим сеансам работы с отладчиком.

НАДЕЖНОСТЬ В ЯЗЫКАХ ПРОГРАММИРОВАНИЯ

В наши дни уже давно никто и забивает себе голову думой о том, что его электробритва вдруг может быть включена в розетку со слишком высоким напряжением. Более того, в случае короткого замыкания или аналогичного нарушения корректного подсоединения контактов существуют дополнительные пробки. Удивительно, но подобные идеи безопасности до сих пор не нашли своего отражения в большинстве языков программирования. Конечно, далеко не каждая ошибка программирования может быть предотвращена еще при создании языка. Но, тем не менее, предотвращение определенных ошибок в классах и выявление ошибок этапа выполнения остаются важными показателями качества [Kir94, Str94]. И Oberon, и C++ опираются на понятие жесткой типизации. Однако, их подход к этому вопросу сильно отличается. В Oberon'е, как и в Pascal'е, переменная связывается с произвольным сложным типом; в C++, так же как и в C, тип связывается с произвольным сложным адресатом (l-значение, lvalue). Это l-значение работает, как прототип для использования переменной и неявно задает тип переменной. Инвертируя объявление и изолируя переменную, можно восстановить ее тип. Ниже приводится конкретный пример объявления указателя «v» на структуру «x»:

```
struct x *v;
```

Это означает, что l-значение «*v» имеет тип «struct x». Звездочка обозначает операцию разыменования (dereferenciation), а потому тип переменной «v» может быть воспринят как указатель на «struct x». В Oberon'е это выглядит так:

```
VAR v: POINTER TO x;
```

Переменная в этом описании уже отделена. В случае более сложных описаний подход Oberon'а окажется еще проще и будет иметь более регулярную структуру. Как бы там ни было, и в том, и в другом языке с каждой переменной связан тип, который определяет набор значений и применимые к ним операции. Так что многие случаи ошибочного использования переменных и процедур могут быть обнаружены до выполнения программы, что поможет избежать загадочных аварийных завершений программы.

АДРЕСНАЯ АРИФМЕТИКА В C++ — ИСТОЧНИК ОПАСНОСТИ

Для тех ошибок, которые не могут быть выявлены до выполнения программы, Oberon делает еще один шаг вперед в плане гарантии надежности типов и целостности памяти даже во время выполнения. Необходимые «пробки», например, проверка границ массивов, может быть реализована практически без накладных расходов в смысле времени выполнения и размера программы. C++ определяет массив аналогично указателю на первый элемент и разрешает использовать адресную арифметику (pointer arithmetic). На практике это исключает возможность проверки индексов. Еще один пробел в плане обеспечения надежности для C++ существует в управлении динамической памятью, где Oberon

гарантирует целостность памяти средствами автоматической сборки мусора.

В отличие от BASIC'а и большинства script-языков и языков четвертого поколения Oberon и C++ дают возможность конструировать динамические структуры данных, которые связываются друг с другом с помощью указателей. Такие структуры могут не только расширяться, но и сжиматься. В последнем случае программист на C++ должен явно высвобождать неиспользуемую память. Для решения этой задачи C++ предлагает понятие деструктора, который автоматически активируется всякий раз, когда память объекта утилизируется.

Деструкторы, однако, не решают проблемы, в частности, тогда, когда память из-под объектов возвращается чересчур рано или слишком поздно. Для выявления и устранения таких ошибок приходится проводить долгие часы с отладчиком. Для расширяемых программных систем не спасет и это. Нетрудно показать, что в этом случае программист не может знать корректного времени высвобождения объекта. Следовательно, отнюдь не только ради удобства Oberon опирается на концептуально бесконечную кучу, которая позволяет явно запрашивать память, но не возвращать ее.

OBERON СО ВСТРОЕННОЙ СБОРКОЙ МУСОРА

В отличие от программиста исполняющая система (runtime system) может легко определить, когда объект больше не используется и утилизировать связанную с ним память. Этот прием, называемый также автоматической сборкой мусора (automatic garbage collection), реализует иллюзию бесконечной кучи и ведет к значительному росту продуктивности работы. Быть может, именно сборка мусора, а отнюдь не синтаксис, явилась той притягательной силой, которая влекла к себе пользователей таких языков, как Smalltalk и Lisp. И не секрет, что введение сборки мусора — это горячая тема обсуждений в сообществе программистов C++. Однако, из-за адресной арифметики ввести ее в язык C++ гораздо сложнее, если вообще возможно.

РАСШИРЕНИЕ ЗАПИСИ — КОНЦЕПЦИЯ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

Грубо говоря, то Oberon и C++ являются объектно-ориентированными расширениями существующих языков. Подходы к этому у них, правда, различаются. C++ в сущности поддерживает объектно-ориентированное программирование (ООП) в стиле а-ля Simula-67. Oberon не поддерживает конкретного ООП-стиля, а оставляет программисту право выбирать соответствующий стиль, подходящий его конкретной задаче. Все эти стили базируются на понятии расширения записи (record extension), которое заменяет вариантные записи (union в C) своих предшественников. Расширение записи означает, что новый тип запись может быть определен как расширение уже существующего.

Базовый тип и расширенный типснизу вверх совместимы, так что все операции, применимые к базовому типу, могут быть также применены и к расширенному типу. Обратное неверно. В Oberon'е можно выделить два фундаментальных ООП-стиля. Они различаются тем, что сообщение (message) представляется явно, как структура данных в языке Oberon, или неявно в виде процедурного вызова.

В первом случае сообщения представляются в виде записей (message records), которые передаются в процедуру (обработчик сообщений) явно через параметры-переменные. Обработчик обычно связан с объектом-получателем с помощью процедурной переменной (см. рис.2). Объекты обычно размещаются в куче и становятся доступны через указатели.

```
TYPE
   Object = POINTER TO ObjectDesc;
   ObjectMsg = RECORD END;
   Handler = PROCEDURE (O: Object; VAR M: ObjectMsg);
   ObjectDesc = RECORD
   handle: Handler
   END;
```

рис.2 Записи-сообщения явно передаются процедуре-обработчику, которая связана с объектом-получателем через процедурную переменную.

Применяя расширение записи к сообщениям, можно создавать иерархические типы сообщений. Например, тип сообщения DisplayMsg строится на основе базового типа ObjectMsg (см. рис.3). Возможна и дальнейшая специализация DisplayMsg. Обработчик разбирается в сообщениях раного вида с помощью оператора IS пробы типа и реагирует на сообщение объектно-ориентированным путем.

На первый взгляд использование записей-сообщений и обработчиком кажется неудобным и недостаточным. Однако, они имеют определенные достоинства, которые объясняют, почему они представляют в языке Oberon доминирующий ООП-стиль.

К достоинствам относится следующее.

- Сообщения могут вводиться по мере надобности. Нет необходимости объявлять их вместе с базовым типом.
- Сообщения могут обрабатываться родовым образом без
  знания их типа и интерпретации их содержания. Объект-контейнер, например, может переадресовывать сообщения своим элементам без знания всех этих сообщений.
- Возможны родовая широковещательная рассылка (generic broadcast), переадресация (forwarding) и делегирование (delegation).
- Эффект расширяемого списка параметров можно получить с помощью расширения записей-сообщений.

Имеет место четкое различие между созданием подтипов (расширение записи) и созданием подклассов (наследование кода). Наследование кода, включающее в себя множественное и даже динамическое наследование, может быть достигнуто путем программирования в разработчике соответствующего механизма диспетчеризации сообщений (см. ELSE-часть на рис.3).

```
TYPE
   CopyMsg = RECORD (ObjectMsg)
   deep: BOOLEAN;
   cpy: Object
   END;


   DisplayMsg = RECORD (ObjectMsg)
   F: Frame;
   x, y: INTEGER;
   END;
```

```
PROCEDURE HandleMyObject (O: Object; VAR M: ObjectMsg); BEGIN
  IF M IS CopyMsg THEN ...
  ELSIF M IS DisplayMsg THEN ...
  ...
  ELSE Objects.Handle(O,M)
  END END HandleMyObject;
```

рис.3 Расширение записи может быть также применяться в отношение записей-сообщений, что ведет к построению иерархии типов сообщений.

Доминирующая роль записей-сообщений очевидным образом проглядывает из таких систем, как MacOS, X11 и Windows, где они выступают в качестве записей-событий. В этих системах, однако, записи-сообщения выражаются как нерасширяемые вариантные записи (unions).

Если эффективность более важна, чем гибкость, то имеются и другие механизмы. В языке Oberon-2, который поддерживается многими коммерческими производителями, к этим механизмам относятся также типизированные процедуры (type-bound procedures), схожие с виртуальными функциями языка C++. Так, например, процедура Display может быть связана с типом Line следующим образом:

```
PROCEDURE (L: Line) Display (F: Frame; x,y: INTEGER);
```

C++ вводит объектно-ориентированное программирование через специальную синтаксическую конструкцию — класс — которая выступает в роли своеобразных текстовых скобок, замыкающих внутри себя расширяемое описание структуры, а также функции, связанные с этой структурой. И хотя записи-сообщения и обработчики в этом случае в принципе также возможны, но этот прием уже будет непрактичным, ведь здесь отсутствует операция пробы типа.

В обычном ООП-стиле языка C++ с классами и виртуальными функциями наследование кода не может быть выражено явно, в виде обработчиков сообщений Oberon'a. Следовательно, язык уже содержит несколько важных отношений наследования, включая множественное наследование и виртуальные базовые классы. Однако, эти предопределенные механизмы не могут соперничать с гибкостью явно запрограммированного обработчика сообщений. К примеру, здесь возможна и родовая переадресация сообщений.

ОБРАБОТКА ИСКЛЮЧЕНИЙ

Большинство разработчиков языков согласно с тем, что операции ввода-вывода, потоки управления (threads), семафоры и аналогичные средства не должны определяться внутри языка, поскольку существует слишком много различных концепций и ни одна из них не подходит для всех случаев жизни. Это отнюдь не означает, что программисты не должны использовать эти концепции. Просто концепции должны поддерживаться средствами соответствующих модулей, а не языковыми конструкциями. Эта же идея применима и к обработке исключений в языке Oberon, в то время как в C++ конкретный механизм обработки исключений уже встроен в язык.

РОДОВЫЕ СРЕДСТВА

Программа называется родовой (generic), если она не является специфичной для конкретной задачи программирования. В языках программирования с жесткой типизацией сами типы обычно являются постоянными, т.е. они специфичны. В то же время можно мыслить в терминах программных компоненты, таких как процедуры и классы, которые параметризуются типами. Наиболее известными примерами таких родовых программ являются классы-контейнеры (списки, множества, деревья), которые состоят из элементов конкретного типа.

C++ позволяет использовать шаблоны (templates), т.е. строительные блоки, которые для увеличения возможности многократного использования программного кода могут параметризовываться даже типами. К сожалению, среди известных приемов не существует более хорошего, чем расширение шаблонов для всех различных комбинаций аргументов. Шаблоны фактически представляют другой вид препроцессора, который уже знает о правилах областей видимости C++. Поддержка и сопровождение расширенных шаблонов в дальнейшем усложняет их реализацию и использование. В существующих реализациях эта проблема остается в основном неразрешимой, а интенсивное использование шаблонов из-за неумышленного дублирования кода часто ведет к весьма непредсказуемому размеру этого кода.

По этой причине Oberon не включает в язык механизм шаблонов, а перепоручает задачу расширения фрагментов кода программисту. В принципе препроцессор шаблонов может быть также сделан и для Oberon'a, но как отдельное инструментальное средство.

ПЕРЕГРУЗКА ФУНКЦИЙ И ОПЕРАЦИЙ

Одним из центральных проектных решений языка C++ была возможность определять новые типы данных, которые выглядят в точности как встроенные типы. Следовательно, при этом необходимо допускать существование определенных пользователем операций, таких как + и -. Вполне естественно распространить эту идею также и на перегрузку функций.

В отличие от C++ одним из центральных проектных решений языка Oberon было то, что импортируемые объекты должны всегда помечаться спереди именем модуля, чтобы упростить процесс чтения исходного текста. Это идет вразрез с перегрузкой функций и операций. Поэтому Oberon использует перегруженные функции и операции только для встроенных языковых типов. Такое ограничение также помогает гарантировать, что перегруженные операции достаточно схожи, чтобы оправдать перегрузку, и оно помогает сократить возможное неумышленное ухудшение эффективности. Обратите внимание, что перегрузка, даже имея под собой корни устоявшейся математической концепции, обладает и снекоторыми подводными камнями. Искушенный читатель, быть может, захочет определить, какая из следующих двух функций вызывается (если таковая, конечно, есть) и что произойдет, если одну из этих

---

[1] Это не совсем верно в отношении второй и третьей возможностей. Информация о типе на этапе выполнения после недавних нововведений в языке C++ стала реальностью, где она носит название RTTI (RunTime Type Information). RTTI реализуется не для всех типов C++, а только для классов, имеющих одну или несколько виртуальных функций, т.е. для полиморфных типов. Что касается сборки мусора, то для C++ это действительно непростая задача и она не является штатной возможностью языка. В то же время имеются различные реализации алгоритмов как в виде отдельных библиотек, так и путем встраивания сборки мусора в библиотеку поддержки (run-time library). Достаточно упомянуть работы Joel F. Bartlett (1989, TN-12-89) и G.M.Yip (1991, TR-8-91) из Western Research Laboratory фирмы Digital. В остальном аргументация достаточно убедительна. /Прим.переводчика/

функций изъять.

```
void f(char*); void f(int); ... f(0);
```

Подводя итог, можно сказать, что исключительная лаконичность описания языка Oberon (менее 20 страниц) не ведет к ущербности в выразительной силе языка. Как раз наоборот, Oberon уже обладает некоторыми возможностями, о которых программисты C++ могут только мечтать. Скажу лишь о некоторых: модули, информация о типе на этапе выполнения и сборка мусора[1]. Сборка мусора (garbage collection) является просто необходимой для устойчивых и надежных расширяемых программных систем, и в будущем она будет играть все более и более важную роль. Остается надеется, что великолепные университетские реализации Oberon будут поддержаны столь же хорошо реализованными промышленными системами программирования, которые предоставят программистам-практикам реальную альтернативу языку C++.

СРАВНЕНИЕ OBERON И C++

Критерий	Oberon	C++
проба типа	+	-
тип BOOLEAN	+	- [2]
модули	+	- [2]
выделение системной зависимости	+	-
определенный порядок инициализации	+	-
сборка мусора	+	-
динамические массивы	+	-
run-time проверки	+	-
полностью надежные типы	+	-
потребность в препроцессоре	-	+
исключения	-	+
шаблоны	-	+
перегрузка	-	+
типизация	явная	неявная
уровни приоритетов операций	4	17
описание языка (страниц)	20	150

ЛИТЕРАТУРА

[Ger94] Gericke R. (1994) «Wider den Schnickschnack» // Oberon-System.

[Kir94] Kirsch Ch. (1994) «Entwicklertrauma: Marktuebersicht. Tools fuer die C-Entwicklung» // iX, No.6.

[ReW94] Reiser M., Wirth N. (1994) «Programming in Oberon» // Addison-Wesley.

[Str92] Stroustrup B. (1992) «C++ Programming Language» // Addison-Wesley.

[Str94] Stroustrup B. (1994) «The Design and Evolution of C++» // Addison-Wesley.

[2] Требуются дополнительные пояснения