Московский государственный университет
имени М. В. Ломоносова

На правах рукописи

Столяров Андрей Викторович

Интеграция разнородных языковых механизмов в рамках одного языка программирования

Специальность 05.13.11 -
математическое и программное обеспечение
вычислительных машин, комплексов и компьютерных сетей

Автореферат диссертации на соискание ученой степени
кандидата физико-математических наук

Москва - 2002

Работа выполнена на кафедре алгоритмических языков факультета вычислительной математики и кибернетики Московского государственного университета им. М. В. Ломоносова.

Научный руководитель:	кандидат физико-математических наук Головин Игорь Геннадьевич.
Официальные оппоненты:	доктор физико-математических наук, профессор Жоголев Евгений Андреевич; кандидат физико-математических наук Ветров Александр Григорьевич.
Ведущая организация:	ГУ "Межведомственный Суперкомпьютерный Центр".

Защита диссертации состоится 29 ноября 2002 г. в 11^{00} часов на заседании диссертационного совета Д 501.001.44 в Московском государственном университете им. М. В. Ломоносова по адресу: 119992, ГСП-2, Москва, Ленинские горы, МГУ, 2-й учебный корпус, факультет ВМиК, аудитория 685.

С диссертацией можно ознакомиться в библиотеке факультета ВМиК МГУ.

Автореферат разослан 29 октября 2002 г.

Ученый секретарь
диссертационного совета
профессор

Н. П. Трифонов

Общаяя характеристика работы

Работа посвящена разработке нового подхода к решению проблемы сочетания разнородных языковых механизмов в рамках одного проекта (одной программы).

Актуальность темы.

Эффективность работы программиста во многом зависит от адекватности используемых изобразительных средств по отношению к решаемой задаче.

Ограничиваясь при реализации проекта одним конкретным языком, мы, как правило, вынуждены довольствоваться наработанными вокруг этого языка изобразительными приемами. При решении частных подзадач это может оказаться неудобным за неимением в избранном языке адекватных средств. Поэтому у многих программистов часто возникает потребность воспользоваться при решении той или иной частной подзадачи средствами языка программирования, отличного от основного языка проекта.

В то же время попытка использования в проекте нескольких языков может обернуться столь высокими накладными расходами, что преимущества применения альтернативных изобразительных средств окажутся сведены на нет.

В современной индустрии программного обеспечения, в основном, используются императивные и объектно-ориентированные языки программирования. Многие программисты имеют навыки и опыт работы с альтернативными языками, но индустриальные языки программирования, как правило, не предоставляют возможностей применения этих навыков. Поэтому разработка методов поддержки практического применения приемов и понятий альтеративных языков программирования в проектах, выполняемых с помощью индустриальных языков программирования, является весьма актуальной задачей.

Цели диссертационной работы.

В работе ставилось целью предложить метод и инструментарий, который позволил бы программистам, владеющим альтернативными языками программирования, на практике применять свои навыки в рамках индустриальных проектов.

Научная новизна.

Предложенный в работе метод позволяет в проекте, основным языком которого является индустриальный язык программирования, применять приемы и понятия, характерные для альтернативных языков программирования.

В отличие от ранее существовавших подходов к решению этой задачи, предложенный метод позволяет работать, оставаясь целиком в рамках основного языка программирования (в качестве которого выступает объектно-ориентированный язык программирования, допускающий переопределение символов стандартных операций для пользовательских типов). Таким образом, метод предполагает использование существующего индустриального языка программирования и существующих систем программирования (включая компилятор), не требуя изменения и/или усложнения привычного инструментария.

Разработанные модели и библиотеки позволяют использовать понятия и приемы, характерные для языков Лисп, Рефал и Дэйталог в проектах на языке Си++, оставаясь целиком в рамках последнего и не применяя таких средств, как дополнительное препроцессирование кода, анализ кода во время исполнения и т.п.

Практическая ценность.

Предложенный метод предполагает использование существующих индустриальных языков и систем программирования без каких-либо модификаций; внедрение метода сводится к освоению библиотеки классов, имеющей сравнительно простую архитектуру и интерфейс.

В то же время разработанные модели позволяют в рамках базового языка записывать выражения, семантически эквивалентные и синтаксически близкие конструкциям альтернативных языков; таким образом, средства программирования с использованием приемов и понятий альтернативных языков оказываются естественными для программиста, владеющего данным альтернативным языком.

Все это позволяет ожидать практического применения предложенного метода и инструментария в индустриальном программировании.

Апробация работы.

Результаты диссертационной работы докладывались:

– на международной конференции 4th Joint Conference for Knowledge-Based Software Engeneering (г. Брно, Чехия, 2000 г.);
– на заседании научно-исследовательского семинара под руководством проф. М. Р. Шура-Бура в МГУ.

Публикации.

По теме диссертации опубликовано 5 печатных работ.

Структура диссертации.

Диссертация состоит из введения, четырех глав, заключения и списка литературы. Общий объем работы составляет 105 страниц (100 страниц без списка литературы). Список литературы включает 60 наименований.

Содержание работы

Во введении рассматривается ситуация с существующими языками программирования и их практическим применением. Анализируется подход к классификации языков программирования по их изобразительным возможностям. Рассматриваются термин *парадигмы программирования* и другие родственные термины, встречающиеся в литературе.

Дается мотивация поиска подходов к проблеме сочетания разнородных языковых механизмов (мультипарадигмального программирования). Приводится обзор существующей литературы на тему мультипарадигмального программирования и смежных работ.

Обобщаются традиционные подходы к решению проблемы сочетания разнородных языковых средств в рамках одного проекта, а именно:

- *Пакеты взаимосвязанных программ* (Unix-style, CORBA/COM, etc).
- *Встраиваемые интерпретаторы* (Embedded SQL, Scheme и т.п.)
- *Расширяемые интерпретаторы* (Tcl/Tk и др.)
- *Трансляция из одного языка в другой* (Scheme и др.)
- *Создание нового языка или расширение существующего* (Leda, Oz, CLOS и др.)
- *Моделирование отдельных возможностей*

Анализируются достоинства и недостатки перечисленных подходов. Формулируется список требований к новому методу:

- *Универсальность* Метод должен позволять использование нескольких различных стилей программирования в рамках одного проекта (одной программы).
- *Языковая целостность.* При импорте очередного стиля должно быть возможно использование всех наиболее заметных достижений программистской культуры, сформировавшейся вокруг языков, породивших данный стиль; иными словами, необходимо импортировать не абстрактный теоретический стиль программирования,

а изобразительные возможности конкретных языков программирования.

– *Удобство.* Средства разработки кода в альтернативном стиле должны быть удобны для человека, владеющего языком-носителем данного альтернативного стиля.
– *Консервативность.* Метод должен предполагать использование существующих систем программирования. В частности, метод не должен быть связан с созданием очередного «принципиально нового» языка программирования, так как этот путь в условиях современного индустриального программирования бесперспективен.
– *Прозрачность.* Метод должен позволять естественное разделение данных между частями программы, написанными в рамках различных стилей, а также обращения к подпрограммам, написанным в одном стиле, из кода, написанного в другом стиле, для любых двух интегрированных стилей.
– *Компилируемость.* Метод не должен быть связан с лексическим и синтаксическим анализом каких-либо частей исходного кода во время исполнения программы.

В первой главе описывается общая идея предлагаемого метода (метода *непосредственной интеграции*). Для иллюстрации рассматривается язык Си++ в качестве базового языка проекта и Лисп в качестве альтернативного.

Благодаря поддержке в языке Си++ концепции абстрактных типов данных и возможности переопределения символов стандартных операций для объектов пользовательских типов язык Си++ может рассматриваться как язык *моделирования алгебр.* С другой стороны, семантика языка Лисп может рассматриваться как *алгебра S-выражений*, одной из операций которой является *вычисление* S-выражения.

Чтобы представить конструкции языка Лисп средствами Си++, достаточно описать класс, способный хранить (инкапсулировать) S-выражение любого поддерживаемого типа и дать набор операций для построения точечных пар и списков из атомарных S-выражений. Используя свойство полиморфности объектов, можно построить иерархию классов с общим предком для хранения различных типов S-выражений, что позволит вводить новые типы S-выражений.

В результате мы получим запись лисповских программ и данных на Си++. После компиляции текста транслятором Си++ в сегменте данных основной программы появятся конструкции Лиспа, уже переведенные во внутреннее представление. Чтобы заставить эти конструкции «ожить», необходимо описать функцию (возможно, метод того же класса), производящую вычисление значений S-выражений по правилам языка Лисп. Таким образом, с помощью библиотеки классов, не изменяя собственно язык Си++, можно дать программисту возможность использования понятий и приемов языка Лисп в программе на Си++.

Одним из условий постановки задачи является удобство работы с кодом, написанным в альтернативных парадигмах. В связи с этим при реализации алгебры языка Лисп необходимо предоставить пользователю адекватные средства конструирования лисповских списков.

Один из известных способов описания произвольных S-выражений состоит в описании точечных пар с помощью явного вызова функции `cons`, Такая форма записи, позволяя строить произвольные лисповские списки, при этом требует их записи в форме, совершенно не похожей на запись в Лиспе. Например, список `(1 2 3)` мог бы быть записан в форме

```
cons(new l_int(1), cons(new l_int(2),
     cons(new l_int(3), &nil)));
```

где `l_int` – имя класса для представления лисповских целых констант, `nil` – объект, используемый для представления пустого списка, `cons` – имя функции, создающей новый объект типа «точечная пара».

Низкая наглядность такого представления делает его неприменимым для практического программирования. Чтобы удовлетворить условиям постановки задачи, необходимы иные средства.

Допустим, мы описали некоторый класс (назовем его `SExpr`), инкапсулирующий любое S-выражение и снабдили его конструкторами для порождения S-выражений типов целая константа, строка и точечная пара.

```
class SExpr {
      . . .
  public:
      . . .
    SExpr(int i);           // для целых
    SExpr(const char *);    // для строк
    SExpr(const SExpr &car, const SExpr &cdr);
                            // для точечных пар
      . . .
};
```

Предположим также, что один из экземпляров класса `SExpr` описывает пустой список.

Наибольшей наглядности при представлении списков можно достичь путем переопределения для класса `SExpr` операции «запятая». Операцию можно перегрузить таким образом, чтобы, например, лисповский список

```
("The Beatles" "Let It Be" 1969)
```

мог быть представлен в виде

```
(SExpr("The Beatles"), SExpr("Let It Be"), SExpr(1969))
```

или, благодаря автоматическим преобразованиям в Си++, более компактно:

```
(SExpr("The Beatles"), "Let It Be", 1969)
```

Преобразования первого элемента списка к типу `SExpr` достаточно для того, чтобы компилятор, обрабатывая всю конструкцию слева направо, применял операцию «запятая», определенную для класса `SExpr`, автоматически преобразуя каждый следующий операнд к типу `SExpr` через соответствующий конструктор. Но и такая сравнительно компактная конструкция выглядит неудобочитаемой из-за наличия лишних скобок в вызове конструктора `SExpr`.

Кроме того, такие средства приводят к сложностям в случае, если первым элементом списка будет, в свою очередь, список, например, `((25 36) 49)`. Конструкция `((SExpr(25),36),49)` в Си++ будет в точности эквивалентна конструкции `(SExpr(25),36,49)`. Можно описать искомую структуру данных выражением `(SExpr((SExpr(25),36)),49)`, однако такая конструкция оказывается неприемлемой для восприятия человеком.

Наконец, введенные средства не позволяют описать список из одного элемента.

Устранить перечисленные недостатки позволяет введение отдельной унарной операции, превращающей любое S-выражение в список из одного элемента. Необходимо отметить, что, поскольку из соображений наглядности желательно иметь возможность строить такие одноэлементные списки из числовых и строковых констант языка Си++, переопределением одной из стандартных унарных операций проблема не решается. Поэтому для обозначения такой операции вводится вспомогательный класс, в котором переопределяется одна из бинарных операций (например, `|`). Описав переменную-объект этого класса с коротким именем (например, `L`), мы сможем представлять лисповские списки достаточно наглядно:

```
(L| "Here I go", 25)          //   ("Here I go" 25)
(L| (L| 25, 36), 49)          //   ((25 36) 49)
(L| (L| 5, 35), (L| 6, 36)) //   ((5 25) (6 36))
```

Аналогичным образом могут быть введены операции для конструирования точечных пар и точечных списков, для представления лисповского символа «апостроф» и т.п.

Чтобы сделать возможным применение такого метода, базовый язык проекта должен поддерживать концепцию абстрактных типов данных и позволять переопределение стандартных операций. Кроме уже рассмотренного языка Си++ таким условиям удовлетворяет, например, язык Ада-95. Однако последний не позволяет добиться столь же высокой наглядности, поскольку количество символов инфиксных операций в нем существенно меньше, нежели в языке Си++. Кроме того, язык Си++ гораздо более распространен в современной индустрии. В этой связи для дальнейшего изложения язык Си++ выбирается в качестве единственного базового языка.

Во второй главе описывается реализованная в ходе работы библиотека классов Си++, предназначенная для импорта в Си++ изобразительных механизмов, характерных для языка Лисп.

В начале главы дается краткий обзор известных диалектов языка Лисп и их сравнительный анализ. Рассматривается ситуация, в которой язык Лисп используется как второстепенный язык в проекте, основным языком которого является императивно-объектный язык программирования. Предлагается диалект языка Лисп (Intelib Lisp), специально предназначенный для моделирования в рамках предложенного метода интеграции языковых механизмов.

Далее в главе описывается реальная библиотека классов Си++, моделирующая изобразительные механизмы предложенного диалекта Лиспа. Приводится иерархия основных классов библиотеки.

Введенные средства позволяют, оставаясь в рамках языка Си++, строить выражения, семантически эквивалентные и синтаксически близкие соответствующим конструкциям языка Лисп. Сказанное можно проиллюстрировать на примере функции, исходный код которой в синтаксисе языка Лисп приведен на рис. 1.

Используя классы и операции, введенные библиотекой, можно в программе на языке Си++ построить фрагмент кода, показанный на рис. 2. Необходимо пояснить, что `L` – это имя переменной типа `LListConstructor`; `Evaluate()` – метод класса `LReference`, ответственного за работу с S-выражениями. `DEFUN`, `COND`, `ATOM`, `AND`, `NIL`, `CAR` и `CDR`

```
(defun isomorphic (tree1 tree2)
 (cond ((atom tree1) (atom tree2))
       ((atom tree2) NIL)
       (t (and (isomorphic (car tree1)
                           (car tree2))
               (isomorphic (cdr tree1)
                           (cdr tree2))))))
```

Рис. 1: Код функции ISOMORPHIC на языке Лисп

```
#include "intelib.h"
LSymbol ISOMORPHIC("ISOMORPHIC");
void LispInit_isomorphic() {
  static LSymbol TREE1("TREE1");
  static LSymbol TREE2("TREE2");
  (L|DEFUN, ISOMORPHIC, (L|TREE1, TREE2),
    (L|COND,
      (L|(L|ATOM, TREE1), (L|ATOM, TREE2)),
         (L|(L|ATOM, TREE2), NIL),
         (L|T, (L|AND,
           (L|ISOMORPHIC, (L|CAR, TREE1),
                          (L|CAR, TREE2)),
           (L|ISOMORPHIC, (L|CDR, TREE1),
                          (L|CDR, TREE2)))))).Evaluate();
}
```

Рис. 2: Код функции ISOMORPHIC на языке Си++

– заранее описанные переменные класса `LSymbol`, моделирующие семантику соответствующих стандартных функций Лиспа.

Следует особо подчеркнуть, что код, приведенный на рис. 2, является корректным кодом на языке Си++. Трансляция этого кода производится обычным компилятором Си++; никакого предварительного преобразования кода не требуется.

Далее во второй главе отмечается, что использование рассматриваемых конструкций Си++ может оказаться неудобным в случае, если требуется реализовать некоторый программный модуль целиком в рамках лисповской техники программирования. Описывается вспомогательный транслятор, введенный для решения этой проблемы. Транслятор воспринимает на входе файл на языке Intelib Lisp в традиционном синтаксисе языка Лисп и генерирует программный модуль (заголовочный файл и файл реализации) на языке Си++, использующий возможности описанной библиотеки.

В третьей главе описана объектно-ориентированная модель рефал-машины, выполненная в виде надстройки над уже описанной во второй главе библиотекой классов.

В начале главы приведена мотивация применения метода непосредственной интеграции к языку Рефал в качестве альтернативного и Си++ в качестве базового.

Отмечается, что древовидные данные, состоящие из строк терминальных символов, символов-меток, символов-чисел и структурных скобок (то есть выражения, допустимые на входе в рефал-функцию), очевидным образом изоморфны лисповским деревьям, состоящим из строковых констант, символов и чисел. Например, рефальское выражение

```
'abc' symb ('def' 25)
```

соответствует в смысле рассматриваемого изоморфизма лисповскому

```
("abc" symb ("def" 25))
```

Если говорить строже, рефальские объектные выражения изоморфны подмножеству лисповских S-выражений, а именно – множеству списков, в которых нигде не встречается двух символьных строк подряд.

Рассмотренный изоморфизм позволяет использовать для представления рефальских выражений уже существующие средства, разработанные для Лиспа. Приведенное выше выражение можно представить на Си++ как

```
(L|"abc", SYMB, (L|"def", 25))
```

Атомарные S-выражения и точечные списки предлагается считать не соответствующими никаким выражениям языка Рефал и при попытке использования в качестве таковых фиксировать ошибку. Что касается списков с двумя и более строковыми константами подряд, то такие списки можно считають эквивалентными спискам, в которых следующие подряд константы заменены на их конкатенацию.

Введенный изоморфизм позволяет смоделировать языковые механизмы Рефала, пользуясь уже наработанными для Лиспа средствами формирования списочных выражений.

Код на языке Рефал:

```
Pal {                = True;
      s.1            = True;
      s.1 e.2 s.1 = <Pal e.2>;
      e.1            = False;  }
```

Код на языке Си++:

```
LSymbol PAL("PAL");
LSymbol True("True");
LSymbol False("False");
RfVariable_E e_1("e.1"), e_2("e.2");
RfVariable_S s_1("s.1");
RFUNC(PAL) [L               ] [L| True]
           [L| s_1          ] [L| True]
           [L| s_1, e_2, s_1] [L| (R| PAL, e_2)]
           [L| e_1          ] [L| False] ;
```

Рис. 3: Пример кода на языке Рефал и соответствующего кода на языке Си++

Далее в главе описывается расширение ранее созданной библиотеки классов, позволяющее моделировать языковые изобразительные механизмы языка Рефал-5. Приводятся средства, вводимые для представления *активных выражений* Рефала-5 (выражений, заключенных в угловые скобки), а также для представления *WHERE-предложений* и *WITH-предложений*.

Вводимая перегрузка некоторых стандартных операций позволяет в рамках программ на Си++ описывать выражения, семантически эквивалентные и относительно близкие по виду к конструкциям языка Рефал. Пример оригинального фрагмента кода на языке Рефал и соответствующего ему кода на языке Си++ приведен на рис. 3.

Присутствующие в Рефале-5 `E-`, `S-` и `T-`переменные реализованы в библиотеке как частные случаи обобщенного понятия Рефал-переменной, каковому уделено особое внимание. Введенные средства позволяют описывать в виде классов языка Си++ новые типы Рефал-переменных, исходя из потребностей конкретной задачи. Так, в описываемом далее трансляторе, создающем модуль Си++ по исходному коду в традиционном синтаксисе Рефала-5, используются Рефал-переменные, описыва-

Код на языке Дэйталог:

```
father(john, george).
father(george, alex).
father(alex, alan).
father(alan, poul).
grandfather(X, Y) :- father(X, Z), father(Z, Y).
```

Код на языке Си++

```
#include "il_dlog.h"
LSymbol john("john"), george("george"),
        alex("alex"), alan("alan"), poul("poul");
DlVariable X("X"), Y("Y"), Z("Z");
DlPredicate father, grandfather;
void DatalogInit_father_grandfather() {
    father(john, george) <<= true;
    father(george, alex) <<= true;
    father(alex, alan) <<= true;
    father(alan, poul) <<= true;
    grandfather(X, Y) <<= father(X, Z), father(Z, Y);
}
```

Рис. 4: Пример кода на языке Дэйталог и соответствующего кода на языке Си++

емые классами `RfVariable_Refid` (сопоставляется со строкой, допустимой в качестве идентификатора в Рефале), `RfVariable_Strconst` (сопоставляется с заключенной в апострофы текстовой константой) и т.п.

В главе также предлагается методика вызова Рефал-конструкций из программы, написанной на Лиспе, т.е. вычисление произвольного S-выражения в терминах Рефала (вызов Рефал-машины представляется как вызов специальной формы Лиспа) и наоборот, вызова лисповских функций из активных выражений Рефала.

В четвертой главе рассматривается применение метода непосредственной интеграции к языку Дэйталог в качестве дополнительного и Си++ в качестве основного.

В начале главы предлагается избрать язык Дэйталог в качестве представителя логического стиля программирования. Кроме относительной простоты реализации, причиной такого выбора является изначальная нацеленность языка Дэйталог на работу с релационными базами данных.

Описывается Дэйталог-машина, реализованная в виде библиотеки классов Си++. Для представления данных, над которыми осуществляется работа, используются введенные во второй главе модели лисповских S-выражений. Вводятся классы для представления дэйталоговских понятий *предикат*, *атом* и т.п. Описываются средства наглядного представления конструкций языка Дэйталог выражениями языка Си++. Пример соответствующего отображения приведен на рис. 4.

Язык Дэйталог предлагается снабдить встроенными предикатами `DLCONS/3` и `DLLISPCALL/2`. Первый из них позволяет конструировать и анализировать списочные S-выражения. Второй представляет прямое обращение к Лисп-машине, т.е. позволяет вычислить произвольное S-выражение в терминах языка Лисп.

В заключении приведен краткий обзор проделанной работы, сформулированы основные результаты и намечены дальнейшие пути развития исследования.

Основные результаты работы

1. Предложен метод, позволяющий, не выходя за рамки базового объектно-ориентированного языка программирования, допускающего переопределение стандартных операций, программировать, используя понятия и приемы, характерные для некоторых альтернативных языков.

   Метод заключается в построении средствами базового языка модели семантики альтернативного языка, позволяющей формировать выражения, семантически эквивалентные и синтаксически близкие конструкциям альтернативного языка.

2. В рамках предложенного метода разработаны объектно-ориентированные модели алгебры S-выражений языка Лисп, Рефал-машины и Дэйталог-машины. Каждая модель вводит специфический набор операций для конструирования выражений, синтаксически близких конструкциям соответствующего языка.

3. Создана библиотека классов Си++ InteLib, реализующая вышеперечисленные модели и позволяющая практически осуществлять программирование в стиле языков Лисп, Рефал и Дэйталог, оста-

ваясь при этом целиком в рамках языка Си++ и используя существующие для этого языка системы программирования.

Публикации

По теме диссертации опубликованы следующие работы:

1. E. Bolshakova and A. Stolyarov. Building functional techniques into an object-oriented system. // *Knowledge-Based Software Engineering. Proceedings of the 4th JCKBSE*, volume 62 of *Frontiers in Artificial Intelligence and Applications*, pages 101–106, Brno, Czech Republic, September 2000. IOS Press, Amsterdam.

2. E. I. Bolshakova and A. V. Stolyarov. Extending an object-oriented language with Lisp programming techniques. // *Memoria del 3er Encuentro Internacional de Ciencias de la Computacion ENC'01*, volume 2, pages 903–911, Aquascalientes, Mexico, September 2001.

3. И. Г. Головин, А. В. Столяров. Объектно-ориентированный подход к мультипарадигмальному программированию. *Вестник МГУ*, сер. 15 (ВМиК), №1, 2002 г., стр. 46–50.

4. А. В. Столяров. Расширенный функциональный аналог языка Рефал для мультипарадигмального программирования. // Л. Н. Королев, ред., *Программные системы и инструменты. Тематический сборник*, том 2, стр. 184–195. Издательский отдел факультета ВМиК МГУ, Москва, 2001.

5. А. В. Столяров. Интеграция изобразительных средств альтернативных языков программирования в проекты на C++. Рукопись депонирована в ВИНИТИ 06.11.2001, №2319-В2001, Москва, 2001.