ДОКЛАДЫ АН ВШ РФ
2015 январь–март № 1 (26)
ТЕХНИЧЕСКИЕ НАУКИ

УДК 004.432.2

ИСПОЛЬЗОВАНИЕ ПРОЦЕДУР С ОДИНАКОВОЙ СИГНАТУРОЙ ДЛЯ ЭВОЛЮЦИОННОГО РАСШИРЕНИЯ ПРОГРАММ

А.И. Легалов, П.В. Косов, И.А. Легалов
Сибирский федеральный университет

Эволюционная разработка программного обеспечения опирается на различные технические приемы и парадигмы программирования. Применение наследования и виртуализации позволили безболезненно наращивать классы и их функциональность. Добавление новых процедур и функций не вызывает проблем при использовании процедурного и функционального программирования. В более сложных случаях на помощь приходят паттерны проектирования, учитывающие особенности парадигм программирования и ситуации, возникающие при написании фрагментов программного кода. Вместе с тем следует отметить, что эволюционному расширению процедур и функций в настоящее время уделяется недостаточно внимания. В работе рассматривается возможность добавления новой функциональности без изменения уже написанного кода за счет перегрузки процедур с одинаковой сигнатурой. Данный подход базируется на раздельной компиляции таких процедур и связывании их воедино во время окончательной компоновки программы, проходящей на более поздних этапах. Представлены общая идея подхода и возможность ее реализации с использованием традиционных языков программирования. Предлагаются языковые конструкции, которые добавляют в традиционные процедуры опции, задающие перегрузку одинаковых сигнатур. В статье рассмотрены ситуации, в которых применение перегрузки процедур с одинаковой сигнатурой позволяет сделать эволюционную разработку программного обеспечения более гибкой. На конкретных примерах рассмотрены варианты расширения синтаксиса и семантики языков программирования. Предлагаемый подход может использоваться для расширения возможностей процедур и функций в различных парадигмах программирования.

Ключевые слова: эволюционная разработка программ, перегрузка процедур, перегрузка функций, техника программирования, парадигмы программирования.

DOI: 10.17212/1727-2769-2015-1-41-51

Введение

Эволюционная разработка больших программных систем приобретает все большую популярность. Это обусловлено различными факторами, связанными с развитием информационных технологий. В частности:

- существующие методологии разработки программного обеспечения (ПО) ориентированы на инкрементальное наращивание кода;
- современные системы программирования содержат средства, обеспечивающие поддержку эволюционного проектирования;
- эволюционное расширение программных систем экономически более выгодно, чем использование методов, ориентированных на постоянную модификацию уже написанного кода;
- использование эволюционного расширения программ уменьшает количество ошибок, вносимых в написанный и уже отлаженный код, который постоянно

Работа выполнена при поддержке гранта РФФИ № 13-01-00360 «Методы и средства эволюционной разработки программного обеспечения с применением процедурно-параметрической парадигмы программирования».

приходится модифицировать при использовании традиционных методов разработки программного обеспечения

В настоящее время используются разнообразные подходы к эволюционной разработке программ. Ряд их широко применяется на практике, найдя воплощение в различных техниках и парадигмах программирования, среди которых, в частности, следует отметить:

- объектно-ориентированное программирование (ООП) [1];
- аспектно-ориентированное программирование (АОП) [2];
- порождающее программирование [3];
- процедурно-параметрическое программирование (ППП) [4] и другие.

Современные инструментальные средства, поддерживающие эволюционную разработку, обладают определенными ограничениями и не всегда позволяют достичь желаемого эффекта. Поэтому наряду с ними широко применяются методы, обеспечивающие инкрементальное наращивание кода за счет использования разнообразных алгоритмических приемов. В настоящее время эти приемы отражены в образцах (паттернах) проектирования [5], которых стали особенно популярны при использовании методов объектно-ориентированной разработки программ.

Вместе с тем следует отметить, что резервы в создании новых методов эволюционного расширения программ еще не исчерпаны. Существуют подходы, которые еще широко не используются, а их реализация требует дополнительных исследований. К одному из таких путей следует отнести возможность эволюционного расширения уже написанных процедур, функциональность которых может расширяться за счет специального их связывания с дополнительным кодом. Ряд исследований в данном направлении представлен в работе [6]. Однако вопросы непосредственного расширения процедур (функций), а также варианты их реализации и использования в языках программирования до конца не исследованы. Далее процедура и функция воспринимаются как синонимы, что вполне соответствует их эквивалентной трактовке в различных языках программирования. В работе предлагается использование процедур с одинаковой сигнатурой, позволяющей рассматривать добавляемые процедуры как расширение ранее написанной процедуры с такой же сигнатурой. Под сигнатурой понимается часть общего объявления процедуры, позволяющая средствам трансляции идентифицировать процедуру среди других [5]. Именно отличие сигнатур позволяет в ряде языков программирования использовать перегрузку имен функций. Вместе с тем применение процедур с одинаковой сигнатурой позволяет получить дополнительные эффекты, связанные с инструментальной поддержкой эволюционного расширения программ в языках программирования.

1. Расширение программ с использованием процедур

Процедуры обеспечивают удобное расширение программы. Это во многом обусловливается их структурой, позволяющей раздельно использовать предварительное описание процедуры и ее реализацию. Помимо этого процедура является одной из основных конструкций, поддерживающих повторное использование. Библиотеки процедур использовались с появлением первых языков программирования.

Процедура состоит из сигнатуры и блока операторов. С точки зрения эволюционного расширения такая форма позволяет многократно повторять одну и ту же сигнатуру с разными реализациями, обеспечивая тем самым наращивание функциональности в избранном контексте. Однако такой прием используется достаточно редко и в большинстве языков программирования считается ошибкой, связанной с неправильным повторным описанием процедуры. Вместе с тем следует

отметить его использование для перегрузки функций с одинаковой сигнатурой в языке программирования Пифагор [7], что обеспечивает эволюционное расширение при функционально-потоковом параллельном программировании. Можно также отметить, что ряд вариантов эволюционного расширения процедур используют некоторую разновидность этого приема. Благодаря локальным переменным внутри процедуры появляется возможность изолировать код от внешнего окружения. Передача данных в процедуру осуществляется посредством формальных параметров, прописываемых в заголовке. Эти аргументы могут обладать дополнительным контекстом, поддерживающим эволюционное наращивание и выбор альтернативной реализации.

В существующих языках программирования можно выделить следующие варианты эволюционного расширения процедур:

- перегрузка процедур (функций) за счет различия в сигнатурах;
- использование процедурных шаблонов;
- применение процедур, связанных с типом;
- использование простых процедур в языке Ада-95;
- расширение обобщенных параметрических процедур;
- применение специальных гнезд;
- перегрузка функций с одинаковой сигнатурой в языке функционально-потокового параллельного программирования Пифагор.

Перегрузка процедур. Это один из простых вариантов, когда в языке программирования существуют процедуры с одинаковыми именами, выполняющие при этом разные действия. Вызов нужной процедуры осуществляется по ее сигнатуре. В любой момент и в любом модуле разработчик может добавить перегрузку. Этот подход реализован в таких языках программирования как C++, Perl, C# и в ряде других. Он реализует одну из форм статического полиморфизма [3]. Достоинством перегрузки является простота подхода. К недостаткам можно отнести поддержку только статического полиморфизма.

Использование процедурных шаблонов. Шаблоны [8] обеспечивают однократное определение процедуры с последующей подстановкой параметров, создаваемых в ходе эволюционной разработки программ. Создание новых процедур на основе шаблона осуществляется во время компиляции, что является основным недостатком этого механизма и ограничивает область применения.

Применение процедур, связанных с типом. Эволюционное расширение процедур, поддерживающих динамический полиморфизм для одного выделенного параметра, реализовано в языке Оберон-2 [9]. Этот механизм обладает особенностями, аналогичными свойствам виртуальных методов ОО языков. Однако им обеспечивается размещение процедур вне типа данных, что позволяет безболезненно наращивать функциональность в отличие от классов ОО языков, которые изменяются при добавлении каждого нового метода. Связанный тип описывается в процедуре в виде отдельного параметра, а при вызове процедуры его значение записывается перед ее именем. Это делает написание вызова процедуры аналогичным написанию вызова метода класса. К недостаткам подхода можно отнести невозможность записи процедуры, связанной с типом вне модуля, описывающего этот тип. Поэтому добавление новых процедур ведет к модификации модуля, в котором располагается тип.

Использование простых процедур в Ada-95. Простые процедуры языка Ada-95 [10] являются аналогами процедур, связанных с типом. Основное отличие заключается в том, что параметр, обеспечивающий поддержку динамического полиморфизма, находится в общем списке параметров на первой позиции. При вызове процедуры подставляемое значение также ставится на первое место.

С одной стороны, это унифицирует запись процедуры, делая ее похожей на обычную процедуру. С другой стороны, возможно затруднение восприятия таких процедур и то, что первый параметр в них обладает свойствами, отличными от свойств других параметров. Так же, как и в Обероне-2, местоположение простой процедуры ограничено пакетом, в котором располагается тип полиморфного параметра. Помимо этого все простые процедуры некоторого типа должны располагаться сразу же за описанием этого типа, что затрудняет эволюционное расширение.

Расширение обобщенных параметрических процедур. Данный способ используется в ППП [11] и обеспечивает гибкое добавление новых обработчиков при добавлении специализаций. Задаваемая изначально обобщающая процедура определяет сигнатуру, в которую обобщенные параметры входят наряду с обычными. Тело обобщенной параметрической процедуры задает обработку по умолчанию или отсутствует. Обработчики специализаций задаются вне обобщенной процедуры (в новых модулях, обеспечивающих эволюционное расширение программы), а связь с ней осуществляется посредством идентичности сигнатур. Сигнатуры обработчиков отличаются признаками специализаций, дополнительно указываемыми в обобщенных параметрах. Представленный подход поддерживает гибкое расширение процедур при любом числе полиморфных параметров, что обеспечивает прямое эволюционное расширение мультиметодов.

Применение специальных гнезд. Специальные гнезда могут присутствовать в процедурах так же, как и в блоках операторов. Вставка новых операторов обеспечивает реализации методов, аналогично применяемым в аспектно-ориентированном программировании [2]. Это поддерживает простое повторное использование и наращивание функциональности процедур. Подобный прием является перспективным для гибкой разработки программного обеспечения.

Перегрузка функций с одинаковой сигнатурой. Подход предложен в функционально-потоковом языке параллельного программирования Пифагор. Он учитывал специфику используемых структур данных, в частности параллельного списка [7]. Это обеспечило интерпретацию множества функций с одинаковой сигнатурой как единого целого, каждая из которых обрабатывала один и тот же аргумент, а полученное множество результатов образовывало параллельный список. Вариативность вычисления, вплоть до выдачи только одного значения, обеспечивается проверкой поступающего аргумента и выдачи пустого значения, если этот аргумент не удовлетворяет требуемым условиям. Следует подчеркнуть специфику языка и предлагаемых в нем подходов, включая и данный метод расширения. Вряд ли он напрямую применим в традиционном императивном программировании, ориентированном на строгую типизацию данных. Вместе с тем данный метод вполне можно использовать для расширения функциональности процедур.

2. Особенности перегрузки процедур с одинаковой сигнатурой

Проведенный анализ способов построения процедур позволил предложить метод их расширения, который можно использовать в различных парадигмах. Он основан на перегрузке процедур с одинаковой сигнатурой, что позволяет гибко наращивать функциональность.

Основная идея подхода заключается в написании процедур, отличающихся от уже написанной процедуры телом и указанием порядка ее выполнения относительно других таких же процедур. То есть необходимо использовать некоторую систему приоритетов, обеспечивающих расстановку тел процедур в требуемом порядке так, что если приоритет перегружающей процедуры выше, чем у основ-

ной, она выполнится раньше. Этим обеспечатся соответствующие префиксные действия. При меньшем приоритете процедура выполнится после выполнения основной процедуры.

Для обеспечения возможностей использования процедур с одинаковой сигнатурой, размещаемых, например, в едином пространстве имен, необходимо предусмотреть их соответствующее синтаксическое описание. Рассмотрим возможные варианты на примере языка процедурно-параметрического программирования [12], синтаксис которого близок к синтаксису, используемому в языках семейства Oberon [9]. Пусть P будет некоторой расширяемой процедурой. Для ее идентификации в качестве расширяемой используется дополнительная опция, задающая приоритет:

```
PROCEDEURE P(<аргументы>):<тип_результата>;
PRIORITY := <значение>;
<тело_процедуры>
```

Аргументы процедуры определяют список формальных параметров, являющийся частью сигнатуры. Тело задает описания и блок операторов. Статус позволяет определить специфику перегружаемой процедуры. В простейшем случае он может являться числом с фиксированной точкой, задающим приоритет. Это позволяет достаточно просто добавлять новую процедуру в любое желаемое место, варьируя значением целой и дробной части числа. При равенстве приоритетов двух процедур с одинаковой сигнатурой предполагается, что их взаимное размещение осуществляется транслятором или компоновщиком случайным образом. Подобный метод эквивалентен использованию внутренних гнезд и специальных методов подключения к ним (рис. 1).

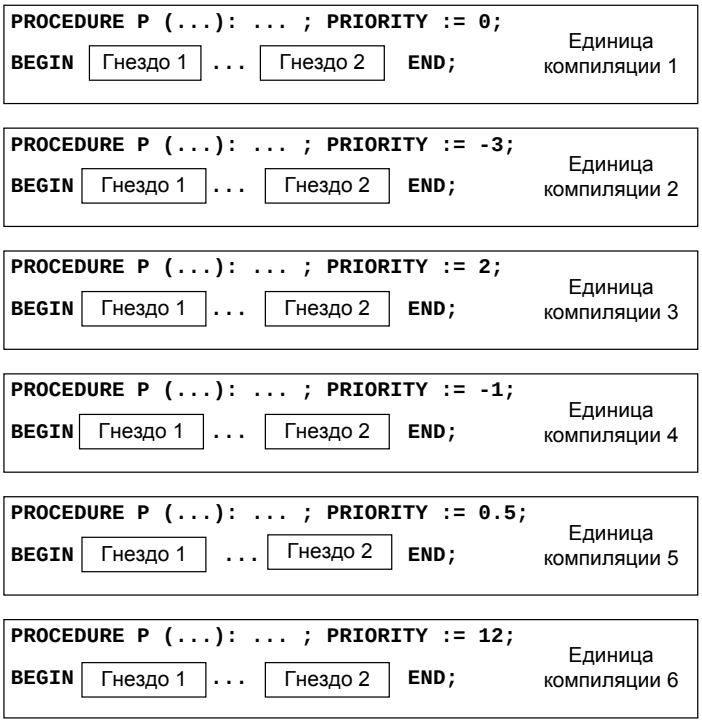

Рис. 1 – Распределение процедур с одинаковой сигнатурой
Fig. 1 – Allocation of procedures with the identical signature

Гнездо для подключения может выбираться в соответствии с приоритетом, приписываемым каждой из процедур, который, например, может задаваться действительным числом. В результате сборки всех процедур с одинаковой сигнатурой образуется двоичное дерево, определяющее порядок их подключения и обхода (рис. 2).

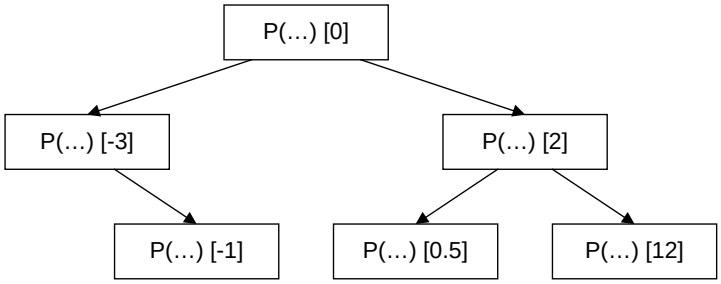

Рис. 2 – Дерево процедур с одинаковой сигнатурой
Fig. 2 – A tree of procedures with the identical signature

Вызов процедур начинается с корневой вершины, после чего управление передается в левое поддерево процедур, подключенных к первому гнезду. После выполнения кода в любой из процедур вызывается процедура, подключенная ко второму гнезду, в которой управление вновь передается левому поддереву и т. д. Обход дерева в заданном порядке обеспечивает выполнение всех тел процедур.

Однако для его реализации можно воспользоваться и другими решениями. В простейшем случае достаточно использовать обычный одномерный массив указателей на процедуры. В результате компоновки процедур, находящихся в различных единицах компиляции, формируется список указателей, определяющий порядок их выполнения в соответствии с приоритетами. Порядок обхода указателей может осуществляться специальным диспетчером, последовательно вызывающим собранные процедуры в соответствии с их приоритетами и, по сути, являющимся базовой процедурой.

Возможность порождения кода, обеспечивающего реализацию данного подхода, можно рассмотреть на простом примере, написанном на языке программирования C++. Для подключения процедур с одинаковой сигнатурой будем использовать контейнер, содержащий указатели на соответствующие функции. Для упрощения примера используем контейнер multimap из стандартной библиотеки. В файле Foo.h определим тип функции и контейнер:

```
typedef int (*foo)();
typedef multimap<double, foo> FooListType;
```

Для инкапсуляции доступа воспользуемся вспомогательными функциями:

```
// Получение указателя на сформированный список функций
FooListType* GetFooList();
// Регистрация функций в
// списке функций с одинаковой сигнатурой
void AddFoo(double weight, foo f);
```

Помимо этого объявим прототип функции, играющей роль базовой функции с одинаковой сигнатурой, и класс, обеспечивающий регистрацию добавляемых функций:

```
// Функция, осуществляющая обход зарегистрированных функций
// с одинаковой сигнатурой.
// Обеспечивает реализацию основной
// идеи расширения функциональности
int Foo();
// Класс, обеспечивающий регистрацию добавляемой функции
class RegFoo {
public:
    RegFoo(double weight, foo f);
};
```

Реализация представленных описаний выделена в отдельную единицу компиляции и выглядит следующим образом:

```
#include "Foo.h"
FooListType* GetFooList() {
    static FooListType fooList;
    return &fooList;
}
void AddFoo(double weight, foo f) {
    FooListType* pFooList = GetFooList();
    pFooList->insert (pair<int,foo>(weight, f));
}
int Foo() {
    int tmp;
    FooListType* pFooList = GetFooList();
    for(FooListType::iterator ii = pFooList->begin();
           ii != pFooList->end(); ++ii)
    {
        foo f = (*ii).second;
        tmp = f();
        cout << tmp << endl;
    }
    return tmp;
}
// Конструктор, обеспечивающий регистрацию добавляемой функции
RegFoo::RegFoo(double weight, foo f) {
    AddFoo(weight, f);
};
```

Любая добавляемая функция Foo_i размещается в своей единице компиляции Foo_i.cpp и скрыта от внешнего влияния в неименованном пространстве имен. При этом все функции реализуются по единому шаблону и регистрируются в контейнере в соответствии с их приоритетами:

```
#include "Foo.h"
// Неименнованное пространстов скрывает функцию
// и все манипуляции с ней от внешнего мира
namespace {
    // Функция, расширяющая функционал каркаса
    int Foo_i() {
        cout << "Execution of Foo_i" << endl;
        return 1;
    }

    // Регистратор функции с заданным весом
    RegFoo toRegFoo_i(1.0, Foo_i);
}
```

Количество функций, добавляемых таким образом, может быть произвольным. Вызов функции из программы осуществляется прозрачно, обеспечивая при этом последовательный вызов всех зарегистрированных функций с одинаковой сигнатурой. Например:

```
int i = Foo();
```

Приведенный пример показывает, что для основного случая существуют достаточно простые варианты инструментальной поддержки процедур с одинаковой сигнатурой. Вместе с тем представленный подход может быть использован и в более интересных случаях, инструментальная поддержка которых также не представляет особых сложностей.

3. Использование процедур с одинаковой сигнатурой

Организация сквозной функциональности. Добавление сквозной функциональности является основной особенностью аспектно-ориентированного программирования. Предположим существование нескольких процедур, каждая из которых предназначена для выполнения своих задач и, следовательно, имеет собственную сигнатуру. Определим для этих процедур приоритет BASE, как условие возможного расширения с использованием одинаковой сигнатуры:

```
PROCRDURE P1(i: INTEGER); PRIORITY (BASE);…
PROCRDURE P2(x: REAL); PRIORITY(BASE);…
PROCRDURE P3(c: CHAR); PRIORITY(BASE);…
```

Пусть некоторая процедура Aspect используется для контроля за рядом параметров в этих процедурах. Тогда ее добавление может быть осуществлено следующим образом:

```
PROCRDURE P1(i: INTEGER); PRIORITY(FIRST); BEGIN Aspect() END;
PROCRDURE P2(x: REAL); PRIORITY(< BASE); BEGIN Aspect() END;
PROCRDURE P3(c: CHAR); PRIORITY(> BASE); BEGIN Aspect() END;
```

В приведенных примерах демонстрируется возможность задания приоритета в относительных и абсолютных значениях путем использования специальных символов и ключевых слов. В результате расширения каждой из трех процедур процедура Aspect будет запущена: перед всеми процедурами, расширяющими процедуру P1; раньше выполнения тела базовой процедуры P2; после выполнения тела базовой процедуры P3.

Использование возвращаемого результата. Представленный подход является достаточно простым для реализации и обеспечивает гибкое расширение уже написанных процедур без изменения их внутренней реализации. Достаточно лишь обеспечить последовательность их выполнения в соответствии с указанными приоритетами. Однако при использовании перегрузки процедур с одинаковой сигнатурой необходимо реализовать механизм, обеспечивающий использование фактических параметров и возвращаемых результатов.

Использование фактических параметров проблем не вызывает. Они последовательно передаются от одной перегруженной процедуре к другой. При этом параметры-переменные могут изменяться, что обеспечивает при необходимости учет модификации данных в процедурах, выполняемых по цепочке. Однако использование результата требует введения дополнительных языковых конструкций, обеспечивающих идентификацию возвращаемого значения. В простом случае с возвращаемым результатом можно сопоставить ключевое слово RESULT, тип которого можно автоматически сопоставить с типом возвращаемого параметра.

В этом случае каждая вызываемая процедура через этот параметр может имеет доступ к значению, полученному в предшествующей процедуре.

Применение данного подхода позволяет использовать передаваемый параметр для анализа и принятия решений, изменяющих последовательное выполнение цепочки процедур с одинаковой сигнатурой. В частности, при определенных значениях можно прекращать вычисления до завершения выполнения всех перегруженных процедур.

Заключение

Предложенный метод эволюционного расширения, опирающийся на использование процедур с одинаковой сигнатурой, позволяет гибко наращивать функциональность по различным направлениям. Вместе с тем следует отметить, что данный подход требует дальнейшей проработки. В частности, необходимо разобраться, каким образом осуществлять обработку возвращаемого результата в первой из цепочки процедуре, когда самого результата еще не было. Помимо этого до конца не ясно, какую роль может играть базовая процедура, которая помимо выполнения своих локальных действий может исполнять и функции централизованного диспетчера. Решение этих проблем и уточнение семантики рассмотренного механизма позволят добавить в арсенал программиста новые методы и инструментальные средства, повышающие эффективность процесса разработки ПО.

ЛИТЕРАТУРА

1. **Буч Г.** Объектно-ориентированный анализ и проектирование с примерами приложений на C++ / пер. с англ. под ред. И. Романовского, Ф. Андреева. – 2-е изд. – М.: Бином; СПб.: Невский диалект, 1998. – 560 с.
2. Eclipse AspectJ: Aspect-oriented programming with AspectJ and the eclipse AspectJ development tools / A. Colyer, A. Clement, G. Harley, M. Webster. – Boston, Massachusetts: Addison Wesley Publ., 2004. – 504 p.
3. **Чарнецки К., Айзенекер У.** Порождающее программирование: методы, инструменты, применение: пер. с англ. – СПб.: Питер, 2005. – 736 с. – (Для профессионалов).
4. **Легалов А.И.** Процедурно-параметрическая парадигма программирования: возможна ли альтернатива объектно-ориентированному стилю? / Красноярский государственный технический университет. – Красноярск, 2000. – 43 с. – Депонировано в ВИНИТИ 13.03.2000, № 622-В2000.
5. Приемы объектно-ориентированного проектирования: паттерны проектирования: пер. с англ. / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. – СПб.: Питер, 2001. – 368 с.
6. **Горбунов-Посадов М.М.** Расширяемые программы. – М.: Полиптих, 1999. – 336 с.
7. **Легалов А.И., Привалихин Д.В.** Эволюционное расширение программ в функциональном языке параллельного программирования // Вестник Красноярского государственного университета. Физико-математические науки. – 2004. – № 5/2. – С. 40–48.
8. **Остерн М.Г.** Обобщенное программирование и STL: использование и наращивание стандартной библиотеки шаблонов C++: пер. с англ. – СПб.: Невский диалект, 2004. – 544 с.
9. **Moessenboeck H., Wirth N.** The programming language Oberon-2 / Institut für Computersysteme. – Zurich: ETH, 1996. – 27 p.
10. **Barnes J.** Programming in Ada 95. – 2nd ed. – Harlow, Addison-Wesley Publ., 1998. – 720 p.
11. **Легалов А.И., Швец Д.А.** Процедурный язык с поддержкой эволюционного проектирования // Научный вестник НГТУ. – 2003. – № 2 (15). – С. 25–38.
12. **Легалов А.И., Бовкун А.Я., Легалов И.А.** Расширение модульной структуры программы за счет подключаемых модулей // Доклады Академии наук высшей школы Российской Федерации. – 2010. – № 1 (14). – С. 114–125.

USING IDENTICAL SIGNATURE PROCEDURES FOR EVOLUTIONARY EXTENSION OF PROGRAMS

Legalov A.I., Kosov P.V., Legalov I.A.
Siberian Federal University, Krasnoyarsk, Russian Federation

Evolutionary development of software is based on different techniques and programming paradigms. The application of inheritance and virtualization made it possible to easily build up classes and their functionality. Adding new procedures and functions is not a problem when using procedural and functional programming. In more complex cases design patterns taking into account the peculiarities of programming paradigms and situations that arise when writing a piece of the code can help . However, it should be noted that the evolutionary expansion of procedures and functions is currently neglected. In this paper we consider the possibility of adding new functionality by overloading procedures with an identical signature without changing the already written code. This approach is based on a separate compilation of such procedures and linking them together during the final assembly of the program at later stages. The paper presents a general idea of the approach and the ability to implement it using traditional programming languages. We propose language constructs that add specific options initializing the overloading of identical signatures in the traditional procedures. The paper describes situations in which the overloading of procedures with an identical signature makes evolutionary software development more flexible. Possible options for extending the syntax and semantics of programming languages are given using specific examples. The proposed approach can be used to extend the capabilities of procedures and functions in different programming paradigms.

Keywords: evolutionary development of software, overloading of procedures, overloading of functions, programming paradigms.

DOI: 10.17212/1727-2769-2015-1-41-51

REFERENCES

1. Booch G. *Object-oriented analysis and design with applications.* 2nd ed. Redwood City, California, USA, Benjamin-Cummings Publ., 1994. 608 p. (Russ. ed.: Buch G. *Ob"ektno-orientirovannyi analiz i proektirovanie s primerami prilozhenii na C++*. 2-e izd. Moscow, Binom Publ., Saint Petersburg, Nevskii dialekt Publ., 1998. 560 p.).
2. Colyer A., Clement A., Harley G., Webster M. Eclipse *AspectJ: Aspect-oriented programming with AspectJ and the eclipse AspectJ development tools*. Boston, Massachusetts, USA, Addison Wesley, 2004. 504 p.
3. Czarnecki K., Eisenecker U. *Generative programming: methods, tools and applications*. Reading, Boston, Massachusetts, USA, Addison-Wesley, 2000. 864 p. (Russ. ed.: Charnetski K., Aizeneker U. *Porozhdayushchee programmirovanie: metody, instrumenty, primenenie*. Saint Petersburg, Piter Publ., 2005. 736 p.).
4. Legalov A.I. *Procedural parametric programming paradigm. Can it be possible as alternative to object-oriented style?* Krasnoyarsk, 2000. 43 p. Available from VINITI, no. 622-V2000. (In Russian)
5. Gamma E., Helm R., Johnson R., Vlissides J. *Design patterns: elements of reusable object-oriented software*. Boston, Massachusetts, USA, Addison-Wesley, 1998. 395 p. (Russ. ed.: Gamma E., Khelm R., Dzhonson R., Vlissides Dzh. *Priemy ob"ektno-orientirovannogo proektirovaniya: patterny proektirovaniya*. Saint Petersburg, Piter Publ., 2001. 368 p.).
6. Gorbunov-Posadov M.M. *Rasshiryaemye programmy* [Expandable programs]. Moscow, Poliptikh Publ., 1999. 321 p.
7. Legalov A.I., Privalikhin D.V. Evolyutsionnoe rasshirenie programm v funktsional'nom yazyke parallel'nogo programmirovaniya [Evolutionary expansion of programs in a functional language for parallel programming]. *Vestnik Krasnoyarskogo gosudarstvennogo universiteta. Fiziko-matematicheskie nauki – Bulletin of the Krasnoyarsk state university. A series of physical and mathematical sciences*, 2004, no. 5/2, pp. 40–48.
8. Austern M.H. *Generic programming and the STL: using and extending the C++ standard template library*. Boston, Massachusetts, USA, Addison-Wesley, 1999. 548 p. (Russ. ed.: Ostern M.G. *Obobshchennoe programmirovanie i STL: ispol'zovanie i narashchivanie standartnoi biblioteki shablonov C++*. Saint Petersburg, Nevskii dialekt Publ., 2004. 544 p.).
9. Moessenboeck H., Wirth N. *The programming language Oberon-2*. Institut für computersysteme, Zurich, ETH, 1996. 27 p.

10. Barnes J. *Programming in Ada 95*. 2nd ed. Harlow, Addison-Wesley, 1998. 720 p.
11. Legalov A.I., Shvets D.A. *Protsedurnyi yazyk s podderzhkoi evolyutsionnogo proektirovaniya* [Procedural language with support for evolutionary design]. *Nauchnyi vestnik NGTU – Science bulletin of the Novosibirsk state technical university*, 2003, no. 2 (15), pp. 25–38.
12. Legalov A.I., Bovkun A.Ya., Legalov I.A. *Rasshirenie modul'noi struktury programmy za schet podklyuchaemykh modulei* [Extension of program's modular structure at the expense of attachable modules]. *Doklady Akademii nauk vysshei shkoly Rossiiskoi Federatsii – Proceedings of the Russian higher school Academy of sciences*, 2010, no. 1 (14), pp. 114–125.

СВЕДЕНИЯ ОБ АВТОРАХ

Легалов Александр Иванович – родился в 1956 году, д-р техн. наук, профессор, заведующий кафедрой вычислительной техники Сибирского федерального университета. Область научных интересов: технологии разработки программного обеспечения, языки программирования, параллелизм. Опубликовано свыше 160 научных работ. (Адрес: 660074, Россия, Красноярск, ул. акад. Киренского, 26. Email: legalov@mail.ru).

Legalov Alexander Ivanovich (b. 1956) – Doctor of Sciences (Eng.), Professor, Head of the Computer Technique Department in the Siberian Federal University. His research interests are currently focused on software engineering, programming languages, and parallelism. He is author of over 160 scientific papers. (Address: 26, Kirensky St., Krasnoyarsk, 660074, Russian Federation. Email: legalov@mail.ru).

Косов Павел Владимирович – родился в 1982 году, аспирант кафедры вычислительной техники, Сибирский федеральный университет. Область научных интересов: техника и методы программирования, языки программирования, трансляторы. Опубликовано 3 научные работы. (Адрес: 660074, Россия, Красноярск, ул. акад. Киренского, 26. Email: kosov_@mail.ru).

Kosov Pavel Vladimirovich (b. 1982) – a postgraduate student of the Computer Technique Department in the Siberian Federal University. His research interests are currently focused on techniques and methods of programming. He is author of 3 scientific papers. (Address: 26, Kirensky St., Krasnoyarsk, 660074, Russian Federation. Email: kosov_@mail.ru).

Легалов Игорь Александрович – родился в 1982 году, канд. техн. наук, доцент кафедры информационных систем Сибирского федерального университета. Область научных интересов: технологии разработки программного обеспечения, языки программирования, трансляторы. Опубликовано 11 научных работ. (Адрес: 660074, Россия, Красноярск, ул. акад. Киренского, 26. Email: igor@legalov.ru).

Legalov Igor Alexandrovich (b. 1982) – Candidate of Sciences (Eng.), Associate Professor of the Information Systems Department in the Siberian Federal University. His research interests are currently focused on software, programming languages, translators. He is author of 11 scientific papers. (Address: 26, Kirensky St., Krasnoyarsk, 660074, Russian Federation. Email: igor@legalov.ru).

Статья поступила 04 декабря 2014 г.
Received December 04, 2014

To Reference:

Legalov A.I., Kosov P.V., Legalov I.A. Ispol'zovanie protsedur s odinakovoi signaturoi dlya evolyutsionnogo rasshireniya programm [Using identical signature procedures for evolutionary extension of programs]. *Doklady Akademii nauk vysshei shkoly Rossiiskoi Federatsii – Proceedings of the Russian higher school Academy of sciences*, 2015, no. 1 (26), pp. 41–51. doi: 10.17212/1727-2769-2015-1-41-51