Эволюционное расширение программ с использованием процедурно-параметрического подхода

А. И. Легалов[1,*], П. В. Косов[2]
[1]Сибирский федеральный университет, Красноярск, Россия
[2]Компания Glyph Worlds, Красноярск, Россия
[*]Контактный e-mail: legalov@mail.ru

Рассмотрены вопросы эволюционной разработки программ. Особенность предлагаемого подхода заключается в моделировании программных объектов, используемых в процедурно-параметрической парадигме программирования. Моделирование осуществляется за счет использования параметрических массивов, обеспечивающих быстрый выбор альтернативных процедур с соответствии с обрабатываемыми комбинациями специализаций. Это позволяет гибко наращивать как данные, так и функции программы без изменения ранее написанного кода, включая и мультиметоды. Проведено сравнение предлагаемого подхода с процедурным и объектно-ориентированным программированием. Разработанные методы эволюционного расширения программ используются при генерации кода в языке процедурно-параметрического программирования Alien.

Ключевые слова: эволюционная разработка программного обеспечения, парадигма программирования, процедурно-параметрическое программирование, мультипарадигменный стиль.

Введение

Большие программные системы разрабатываются практически во всех предметных областях. Это связано как с использованием высокопроизводительных вычислений, так и с функционированием различных распределенных систем, согласованно взаимодействующих между собой. Опираясь в целом на общие принципы функционирования, большие системы используют разнообразные подходы к разработке программ, что во многом обусловливается спецификой вычислительных архитектур. Вместе с тем следует отметить, что, несмотря на различие используемых вычислительных средств и инструментов, многие из вопросов процесса разработки программного обеспечения имеют общие основы, что, в частности, проявляется при создании продуктов, отвечающих определенным критериям качества. Одним из таких критериев является поддержка эволюционной разработки программ с целью наращивания функциональности без изменения ранее написанного кода. Существующие решения обычно реализуются с использованием средств, непосредственно имеющихся в различных языках программирования, а также с помощью дополнительных библиотечных расширений, например, на основе некоторых паттернов проектирования [1].

Необходимость использования эволюционной разработки программ во многом определяется спецификой их применения в различных областях. Например, в мультиагентных и сетецентрических системах возможно одновременное взаимодействие множества участников, решающих различные задачи, сопровождаемое помимо этого высокопроизводительным компьютерным моделированием. При этом участники взаимодействия могут иметь разную структуру (например, наземные, воздушные, водные средства), что обусловливает при выполнении различных задач использование множества альтернативных алгоритмов. Добавление новых альтернативных взаимодействий может осуществляться уже в процессе эксплуатации систем, что делает актуальной задачу эволюционного расширения ранее написанного кода. Неоднородность систем и вариантов их взаимодействия может при этом различным образом влиять на эволюционное расширение, что ведет к одновременному наличию нескольких версий расширяемых программ. В связи с этим актуальной является задача поддержки безболезненного расширения кода для систем, функциональность которых постоянно изменяется в ходе эксплуатации.

Представленные алгоритмические взаимодействия между множеством альтернативных программных объектов обычно реализуются мультиметодами, обеспечивающими поддержку множественного полиморфизма. Существуют различные подходы к реализации мультиметодов, включая и способы, обеспечивающие их эволюционное расширение. В частности, ряд методов предложен при использовании объектно-ориентированного (ОО) программирования [2–4]. Однако сама природа мультиметода в значительной степени ориентируется на использование множественного полиморфизма, при котором больше подходят процедуры или функции, обрабатывающие несколько обобщенных аргументов. Кроме того, добавление очередного метода в класс ведет к его изменению, что не позволяет поддерживать безболезненное наращивание кода.

Для инструментальной поддержки множественного полиморфизма, обеспечивающего независимое эволюционное расширение данных и процедур, предложена процедурно-параметрическая парадигма программирования [5]. Для ее апробации разработаны экспериментальные языки [6]. Проведено моделирование подхода на языке C++ с применением мультипарадигменного стиля, подтверждающее возможность как эволюционного расширения мультиметодов, так и добавления в программу новых мультиметодов [7]. Это определяет перспективы использования процедурно-параметрической парадигмы программирования в системах, требующих гибкого расширения ранее написанного кода.

Следует однако отметить, что современные реалии требуют использования языков программирования, давно зарекомендовавших себя на практике. Это обусловливается наличием множества разработанных для них библиотек, поддерживающих широкое применение как в разнообразных предметных областях, так и на различных последовательных и параллельных архитектурах. Поэтому одним из перспективных направлений видится создание библиотеки, обеспечивающей поддержку процедурно-параметрических конструкций для существующих языков программирования, скрывающей за соответствующим программным интерфейсом особенности реализации. Использование такой библиотеки в рамках имеющейся языковой инфраструктуры позволило бы обеспечить поддержку как последовательного, так и параллельного программирования.

Вместе с тем разработке такой библиотеки обычно предшествует анализ реализуемости, выражающийся в построении экспериментальных прототипов и примеров, мо-

делировании предполагаемых свойств. В работе рассматривается возможность гибкой поддержки процедурно-параметрической парадигмы для языка программирования C++, широко используемого как в высокопроизводительных, так и в распределенных вычислениях.

1. Особенности семантической модели процедурно-параметрических конструкций

Спецификой процедурно-параметрической парадигмы программирования является расширение процедурного подхода, т. е. в отличие от объектно-ориентированного стиля присутствует разделение на данные и функции. При этом независимое эволюционное расширение как данных, так и процедур при совместном их использовании обеспечивает поддержку множественного полиморфизма. Для эволюционного расширения абстрактных типов данных предложены обобщенные записи, обработка которых обеспечивается обобщающими параметрическими процедурами [8].

Традиционные процедурные и объектно-ориентированные языки не поддерживают процедурно-параметрические конструкции, что ведет к необходимости их моделирования с применением существующих средств. Вместе с тем возможность использования мультипарадигменного стиля в языке программирования C++ позволяет осуществить это моделирование достаточно эффективно за счет раздельного безболезненного расширения как данных, так и процедур.

1.1. Моделирование расширяемых данных

Расширение данных, в ряде случаев соответствующее построению обобщенных записей процедурно-параметрической парадигмы, можно получить за счет применения наследования к обычным структурам. При этом отсутствие внутри структур виртуальных методов и конструкторов не позволяет использовать объектно-ориентированный полиморфизм, но избавляет данные структуры от избыточности, обеспечивая реализацию полиморфизма за счет имитации процедурно-параметрического полиморфизма внешними функциями.

Особенности реализации данного подхода можно рассмотреть на простом примере программы, обрабатывающей геометрические фигуры. Обобщенным понятием является такое понятие, как “фигура”. Предполагается, что в ходе расширения в программу могут постепенно добавляться различные фигуры-специализации (треугольник, прямоугольник, круг и т. д.). Основы этих специализаций можно сформировать заранее независимо друг от друга. Например, для треугольника она будет выглядеть следующим образом:

```
struct Triangle {
    int a, b, c;
};
```

Для обработки этого треугольника можно создать соответствующие функции, обеспечивающие его инициализацию, создание, удаление и др.:

```
    // Инициализация существующего треугольника
    void Init(Triangle& t, int a, int b, int c);
    // Создание треугольника с инициализаций сторон
    Triangle* CreateTriangleAndInit(int a, int b, int c);
    // Ввод данных в существующий треугольник из потока
    void In(ifstream &ifst, Triangle& t);
    // Вывод данных о треугольнике в поток
    void Out(ofstream &ofst, Triangle& t);
```

Данная информация представляется в соответствующем заголовочном файле, определяя интерфейс для работы с треугольником. Описание реализации функций осуществляется в отдельной единице компиляции. Для идентификации обобщенных фигур в процедурно-параметрической парадигме используется признак, который обеспечивает индексацию всех добавляемых фигур. Его реализацию можно осуществить за счет структуры, описывающей фигуру, выступающую в роли базы для всех порождаемых специализаций:

```
struct Figure {
    int mark;
};
```

Специализацию фигуры-треугольника (как и любой другой фигуры) можно сформировать, объединив воедино уже существующую основу (треугольник) и фигуру, используя, например, наследование:

```
struct FigTriangle: Figure {
    Triangle t;
};
```

Для обеспечения параметризации необходимо, чтобы с каждой из вновь создаваемых специализаций было связано уникальное значение признака, которое бы заносилось в поле признака (mark), наследуемое от фигуры. Для автоматического формирования такого признака используется переменная, одновременно играющая роль признака специализации:

```
namespace {
    int specNumber = 0;
}
```

Использование безымянного пространства имен позволяет закрыть прямой доступ к этой переменной из внешних единиц компиляции. Ее изменение осуществляется специальным регистратором специализации за счет вызова функции GetSpecNumAndIncrement, возвращающей значение признака зарегистрированной специализации и увеличивающей счетчик зарегистрированных специализаций на единицу.

```
int GetSpecNumAndIncrement() {
    return specNumber++;
}
```

Сформированный признак обеспечивает впоследствии индексацию специализации и ее регистрацию в таблицах обобщенных параметрических процедур.

```
namespace {
    // Класс, регистрирующий специализацию треугольника
    class RegFigTriangle {
    public:
        RegFigTriangle(const char* regInfo) {
            cout << regInfo << endl;
            regMark = GetSpecNumAndIncrement();
        }
    };
    RegFigTriangle regFigTriangle("Registration of: FigTriangle");
}
```

Доступ к признаку каждой специализации осуществляется за счет дополнительной функции, которая, как и счетчик числа обобщений, имеет доступ к соответствующей переменной признака в безымянном пространстве имен.

```
namespace {
    // Признак, зарегистрированный для специализации треугольника
    int regMark = -1;
}
// Функция, возвращающая значение признака фигуры
int GetRegMarkFigTriangle() {
    return regMark;
}
```

Добавление каждой специализации и ее регистрация могут осуществляться в процессе расширения программы поэтапно и независимо за счет подключения к проекту дополнительных заголовочных файлов и единиц компиляции со специализациями без изменения ранее написанного кода, что обеспечивает требуемое эволюционное расширение данных.

1.2. Добавление параметрических процедур

Семантическая модель параметрической процедуры формируется из моделей обобщенной параметрической процедуры и обработчиков специализаций. Обобщенная параметрическая процедура задает общий интерфейс для всех добавляемых обработчиков каждой из специализаций. Обычно ее реализация опирается на задание многомерного массива, размерность которого определяется числом обобщенных аргументов в вызове процедуры и зависит от размерности реализуемого мультиметода. При наличии одного аргумента, характеризующего монометод [9], многомерный массив вырождается в вектор, а реализуемый при этом процедурно-параметрический полиморфизм соответствует по возможностям объектно-ориентированному полиморфизму. Регистрация каждого из обработчиков специализаций может осуществляться в своей единице компиляции. Регистрируемые функции, являющиеся обработчиками специализаций, используют для

фиксации своего местоположения в массиве в качестве индексов значения признаков, автоматически сформированных во время регистрации специализаций обобщения.

В качестве примера рассмотрим формирование обобщенной функции удаления геометрической фигуры. В новом заголовочном файле, определяющем интерфейс данной функции, прописываются параметрический массив и тип обработчика специализации.

```
// Указатель на функцию удаления обобщенной (любой) фигуры
typedef void (*DeleteFigureFunc)(Figure* f);
// Массив функций-специализаций удаления фигур
extern DeleteFigureFunc deleteFigure[];
```

Сам массив описывается в новой единице компиляции, задающей реализацию функции. Помимо этого в нем также имеется описание функции-обертки, скрывающей механизм параметрического вызова.

```
// Массив для регистрации функций удаления фигуры
DeleteFigureFunc deleteFigure[10];
// Функция-обертка удаления обобщенной фигуры
void DeleteFigure(Figure* pf) {
    DeleteFigureFunc func = deleteFigure[pf->mark];
    func(pf);
}
```

Удаление конкретной фигуры реализовано в единице компиляции, добавляемой в ходе формирования соответствующего обработчика специализации.

```
// Функция удаления специализации для фигуры-треугольника
void DeleteFigTriangle(FigTriangle* pft) {
    delete pft;
}
```

Для регистрации данной функции формируется дополнительная обертка, что обеспечивает согласование интерфейса с обработчиком параметрических обобщений и его регистрацию в массиве по индексу, определяемому из специализации. При этом регистрация специализации и реализуемая обертка локализованы только этой единицей компиляции за счет безымянного пространства имен.

```
namespace {
    // Обертка функции удаления специализации треугольника
    void DeleteFigTriangleSkin(Figure* pf) {
        // Проверка на всякий случай
        DeleteFigTriangle(reinterpret_cast<FigTriangle*>(pf));
    }

    // Регистратор функции удаления фигуры-треугольника
    class RegDeleteFigTriangleSkin {
    public:
        RegDeleteFigTriangleSkin(const char* regInfo) {
```

```
            cout << regInfo << endl;
            // Регистрация функции удаления треугольника
            // в параметрической таблице
            deleteFigure[GetRegMarkFigTriangle()] =
                DeleteFigTriangleSkin;
        }
    };
    // Объект, обеспечивающий регистрацию функции удаления
    RegDeleteFigTriangleSkin
        regDeleteFigTriangleSkin("DeleteFigTriangleSkin");
}
```

Добавление новых функций не вызывает дополнительных проблем. Для этого могут использоваться новые заголовочные файлы и новые единицы компиляции. Эволюционное расширение мультиметодов отличается только тем, что вместо одномерного массива используется многомерный. Регистрация новых обработчиков специализаций осуществляется по группе аргументов-специализаций в используемом многомерном массиве.

2. Варианты использования и предоставляемые возможности

Для анализа возможностей предлагаемого подхода разработан ряд тестовых примеров добавления новых программных объектов в уже написанный код. Выделен ряд типичных ситуаций, часто встречающихся на практике [10]:

1) расширение обобщений специализациями и, как следствие, расширение обрабатывающих их обобщающих процедур;
2) добавление новых процедур, обеспечивающих дополнительную функциональность;
3) добавление новых полей данных в существующие типы и изменение в соответствии с этим процедур, осуществляющих обработку измененных программных объектов;
4) добавление новых процедур, предназначенных для обработки только одной из специализаций некоторого обобщения;
5) создание нового обобщения на основе существующих специализаций;
6) добавление в программу мультиметодов, осуществляющих обработку двух или более обобщенных параметров;
7) изменение мультиметодов при добавлении новых специализаций в обобщения, используемые в качестве аргументов мультиметодов.

Ниже приводятся варианты сравнения с чистыми процедурным и объектно-ориентированным подходами.

2.1. Добавление в обобщение новых специализаций

Процедурный подход. В традиционных процедурных языках для прямого построения обобщений обычно используются объединения (C, C++) или вариантные записи (Pascal, Modula-2), что не обеспечивает эволюционного расширения и требует модификации уже написанного кода. Применение при процедурном подходе расширений типов данных обеспечивает безболезненное добавление новых специализаций в таких языках программирования, как Оберон, Оберон-2, Component Pascal, Ada. Аналогичным образом проблема расширения решается и в языке C++ применением наследования к структурам.

Добавление новых специализаций в существующие обобщения ведет к необходимости добавления обработчиков этих специализаций, которые обычно реализуются как ветви условного оператора или эквивалентного ему оператора выбора (переключения), вставляемые в имеющиеся процедуры, обрабатывающие это обобщение, что не способствует эволюционному расширению написанного кода.

Объектно-ориентированный подход. Для добавления специализаций в языках программирования Оберон-2, Component Pascal и Ada применяется расширение типа. Для добавления новых обработчиков специализаций используются процедуры, связанные с типом. В C++, C#, Java и других объектно-ориентированных языках в качестве специализации создается производный класс, а обработчики специализации реализуются посредством виртуальных методов, расширяющих обобщающий виртуальный метод базового класса. Таким образом, объектно-ориентированный подход поддерживает безболезненное расширение в данной ситуации.

Моделирование процедурно-параметрического подхода (ППП) обеспечивает добавление новых специализаций через наследование структур. Необходимые обработчики специализаций также легко расширяют уже существующие обобщающие процедуры за счет дописывания новых реализаций.

2.2. Добавление процедур с дополнительной функциональностью

Процедурный подход и моделирование ППП. Использование внешних процедур обеспечивает простое решение данной задачи. Не возникает никаких проблем с их безболезненным добавлением в новых единицах компиляции. Подобный подход успешно применим практически во всех языках процедурного программирования. При этом моделирование процедурно-параметрического подхода дает возможность создавать дополнительные функции с учетом последующего добавления в программу новых специализаций, чего не позволяет сделать чистый процедурный подход.

Объектно-ориентированный подход. Метод, реализующий дополнительную функциональность, приходится вставлять внутрь класса, что ведет к изменению интерфейса последнего. Сама реализация метода в ряде языков (например, C++) может быть описана в новой единице компиляции, что уменьшает объем изменений, вносимых в программу. Однако большинство современных ОО языков (C#, Java), ориентированных на автоматическую генерацию интерфейсов модулей, используют запись реализации методов внутри класса, что увеличивает объем изменений уже написанного кода. Таким образом, объектно-ориентированная парадигма напрямую не поддерживает эволюционного добавления новых методов, расширяющих функциональность класса.

2.3. Добавление новых полей в существующие типы

Прямое добавление новых полей в запись или класс ведет к его изменению независимо от избранного стиля программирования. Поэтому при необходимости модификации типа с сохранением возможности безболезненно расширять программу используются обходные пути, опирающиеся на косвенное связывание. В этом случае осуществляется связывание через указатель на базовый тип, к которому подключается объект производного типа, содержащий дополнительные поля. При обработке такого объекта необходимо использовать явное приведение типа или дополнительные функции. Если подобные действия приходится применять в ранее написанных модулях, то эволюционное

расширение программы невозможно. Однако часто программа, использующая модифицированный тип, может быть расширена таким образом, что его обработка осуществляется только в добавленных единицах компиляции. Поэтому данный подход может использоваться для всех трех парадигм программирования с применением присущих им технических приемов.

2.4. Добавление новых процедур для обработки конкретных специализаций внутри обобщений

Данная ситуация возникает, когда необходимо использовать только одну из специализаций, например, при обработке элементов контейнера, ссылающегося на обобщенный тип, через который нужно обеспечить доступ только к одному из специализированных типов.

Процедурный подход. В случае расширений типа используются процедуры, в которых осуществляется явная проверка производного типа по указателю на базовый тип. Метод легко добавляется в новом модуле. Не вызывает проблем использование этого же приема в процедурных языках, не имеющих поддержки полиморфизма типа (например, в С). В этом случае обобщение обычно строится таким образом, что имеет признак, который и используется при идентификации специализации.

Объектно-ориентированный подход. Применение объектно-ориентированных методов опирается на использование виртуальных функций, осуществляющих обработку специализированного объекта требуемого типа. Однако такая функция для использования виртуализации должна добавляться в базовый класс, что не способствует эволюционному расширению программы. Помимо этого, базовый класс перегружается дополнительной информацией о производном классе, что не способствует инкапсуляции. Поэтому для обработки специализаций при объектно-ориентированном подходе чаще всего применяется процедурный прием, основанный на явной проверке типа.

Моделирование ППП. Простейшим вариантом является использование процедурного подхода, примененного к обобщенному типу. Отличие заключается в проверке признака специализации вместо проверки производного типа подключенного объекта. Поэтому при процедурно-параметрическом подходе не возникает проблем в решении данной задачи.

2.5. Создание обобщения из существующих специализаций

Процедурный подход. Зачастую в программе требуется сформировать новое обобщение при уже существующих специализациях. Процедурный подход в этом случае обеспечивает прямое решение на основе объединения (язык С) или вариантной записи (языки Pascal, Modula-2). Данный тип может формироваться в модулях, добавляемых в программу, что не вызывает проблем с эволюционным расширением. Вместе с тем следует отметить, что языки, использующие для построения обобщений только расширение типа (Oberon, Oberon-2), не обеспечивают прямого решения этой проблемы. Необходимы дополнительные построения, чтобы достичь искомого решения.

Объектно-ориентированный подход. Использование наследования обладает теми же недостатками, что и расширение типа. Поэтому построение обобщения обычно ведется на основе создания нового базового класса и порождения от него производных классов, включающих в себя необходимые существующие специализации. Решение достаточно простое, но не является прямым.

Моделирование ППП. Рассмотренный в работе вариант моделирования обобщений базируется на наследовании, в которое включаются основы специализации. В принципе ничто не мешает включить эти основы в другие создаваемые обобщения, что позволяет безболезненно расширять программу.

2.6. Добавление мультиметодов

Процедурный подход. Так как мультиметод является обычной процедурой, его добавление не вызывает никаких проблем. Следовательно, процедурный подход поддерживает эволюционное добавление мультиметодов.

Объектно-ориентированный подход. В этом случае реализация мультиметода опирается на множественную диспетчеризацию [11], при которой между классами, играющими роль обобщенных аргументов мультиметода, возникают тесные взаимодействия, задаваемые через методы. Поэтому добавление мультиметода ведет к изменению описания класса, что не способствует эволюционному расширению. Поскольку использование диспетчеризации вызывает большие изменения во взаимодействующих классах, обычно при реализации мультиметодов применяются подходы, базирующиеся на явной проверке типов или смешанном варианте, когда тип первого аргумента выявляется полиморфно, а тип второго определяется явной проверкой [2–4]. Однако подобные решения все равно не обеспечивают поддержку эволюционного расширения, так как приходится изменять содержимое класса.

Моделирование ППП. Для гибкой реализации мультиметодов используются обобщающие параметрические процедуры [6–9]. Являясь по сути внешними процедурами, они обеспечивают безболезненное добавление мультиметодов в программу.

2.7. Изменение мультиметодов при добавлении специализаций

Процедурный подход. Выбор обработчика комбинации специализаций мультиметода осуществляется на основе явной проверки типов обобщенных аргументов внутри условных операторов или операторов выбора. Поэтому добавление новой специализации к любому обобщенному аргументу мультиметода изменяет один или несколько условных операторов, не обеспечивая эволюционного расширения программы в данной ситуации [7, 9].

Объектно-ориентированный подход. Добавление новой специализации связано с созданием нового производного класса [4]. Включение этого класса в общую группу, содержащую мультиметод, ведет для обеспечения диспетчеризации к добавлению дополнительных методов во все классы группы. Поэтому прямое добавление новых специализаций в мультиметод не поддерживается. Стоит отметить существование алгоритмических решений, обеспечивающих безболезненное расширение мультиметодов в частных случаях на основе объектно-ориентированного полиморфизма [2–4] в основном за счет сочетания процедурного и объектно-ориентированного подходов.

Моделирование ППП. Данная парадигма изначально разрабатывалась для поддержки безболезненного расширения мультиметодов при добавлении новых специализаций. Это обеспечивается за счет описания всех комбинаций аргументов в обработчиках специализаций, размещаемых независимо от обобщающей процедуры. Поэтому каждая новая специализация ведет лишь к написанию соответствующих обработчиков в новых единицах компиляции. Метод основан на достаточно простом механизме построения параметрических отношений, одна из возможных реализаций которого приведена в [7].

2.8. Общие замечания

Разработка программ показала, что моделирование процедурно-параметрической парадигмы позволяет гибко расширять уже написанный код, применяя в большинстве случаев методы процедурного программирования. Объектно-ориентированные механизмы в основном были использованы для запуска процедур регистрации, что обусловлено спецификой языка программирования C++, в котором запуск других функций (кроме конструкторов классов) невозможен до запуска главной функции. Вместе с тем следует отметить, что в языках программирования с модульной структурой (например, в Modula-2, Oberon, Python), поддерживающих запуск кода в теле модуля до момента запуска кода головного модуля, можно было бы обойтись только процедурным подходом.

Объектно-ориентированный подход обеспечивает прямое эволюционное расширение программы в наименьшем числе из рассмотренных ситуаций. Однако его текущая популярность вполне объяснима. Во-первых, объектно-ориентированная парадигма дает возможность добавлять новые специализации (ситуация 1), т. е. в той ситуации, которая считается одной из наиболее типичных при расширении кода. Во-вторых, в большинстве других случаев (ситуации 3–6) на практике, даже при использовании ОО языков, применяются методы процедурного и мультипарадигменного программирования. Ситуация 2 зачастую сглаживается тщательной проработкой интерфейса на этапе проектирования. Помимо этого во многих случаях легко можно пойти на небольшие изменения классов (ситуации 2, 4, 6), так как подобная модификация больше касается процесса компиляции, а не изменения модульной структуры программы. Можно пойти и на алгоритмические ухищрения (ситуации 3, 4, 7), которые не приводят к большому увеличению кода, но сохраняют требуемую гибкость, описанную в ситуации 1.

Вместе с тем следует отметить, что моделирование процедурно-параметрического подхода позволяет гибко наращивать программу практически во всех рассмотренных ситуациях и может эффективно применяться в тех случаях, когда использование классов с множеством реализуемых в них методах может оказаться нецелесообразным.

Заключение

Предложенный в работе подход к эволюционному расширению программ был использован в генераторе кода языка процедурно-параметрического программирования Alien. Вместе с тем следует отметить, что широкому использованию данного языка, как и многих других новых языков программирования, препятствует отсутствие библиотечной поддержки, развитие которой может занять достаточно длительное время. В связи с этим дальнейшие варианты развития предложенного подхода могут быть связаны с созданием библиотек, обеспечивающих процедурно-параметрическую поддержку в широко распространенных универсальных языках программирования, в частности в C++. Разработка этой библиотеки может опираться на использование макросов и шаблонов, что позволит создать достаточно эффективную предметно-ориентированную оболочку для написания эволюционно расширяемых программ. В настоящее время ведутся работы в этом направлении.

Помимо этого, повышение эффективности написания кода и надежности его применения могут быть достигнуты за счет включения в существующий язык специальных ключевых слов и конструкций, распознаваемых метакомпилятором, запускаемым

до основного компилятора языка. Подобный подход используется, например, разработчиками библиотеки Qt. Это в перспективе позволит еще больше повысить возможности по использованию предлагаемого подхода для разработки эволюционно расширяемого программного обеспечения.

Благодарности. Работа выполнена при финансовой поддержке РФФИ (договор № 13-01-00360 "Методы и средства эволюционной разработки программного обеспечения с применением процедурно-параметрической парадигмы программирования").

Список литературы / References

[1] Приемы объектно-ориентированного проектирования. Паттерны проектирования: Пер. с англ. / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. СПб.: Питер, 2001. 368 с.
Design patterns. Elements of reusable object-oriented software / E. Gamma, R. Helm, R. Johnson, J. Vlissides. Addison-Wesley Professional, 1994. 353 p.

[2] **Александреску А.** Современное проектирование на C++: Пер. с англ. М.: Изд. дом "Вильямс", 2002. 336 с.
Alexandrescu, A. Modern C++ Design: Generic Programming and Design Patterns Applied. Addison-Wesley Professional, 2001. 352 p.

[3] **Мейерс С.** Наиболее эффективное использование C++: 35 новых рекомендаций по улучшению ваших программ и проектов: Пер. с англ. М.: ДМК Пресс, 2000. 304 с.
Meyers, S. More effective C++: 35 New ways to improve your programs and designs. Addison-Wesley Professional, 1995. 336 p.

[4] **Легалов А.И.** ООП, мультиметоды и пирамидальная эволюция // Открытые системы. 2002. № 3. С. 41–45.
Legalov, A.I. OOP, Multimethods and Pyramidal Evolution // Open Systems J. 2002. No. 3. P. 41–45. (In Russ.)

[5] **Легалов А.И.** Процедурно-параметрическая парадигма программирования. Возможна ли альтернатива объектно-ориентированному стилю? Красноярск, 2000. Деп. рук. № 622-В00. Деп. в ВИНИТИ 13.03.2000. 43 с.
Legalov, A.I. Procedurally-parametric programming paradigm. Is it possible to be as an alternative to the object-oriented style? Krasnoyarsk, 2000. No. 622-B00. Dep. VINITI 13.03.2000. 43 p. (In Russ.)

[6] **Легалов А.И., Швец Д.А.** Процедурный язык с поддержкой эволюционного проектирования // Науч. вест. НГТУ. 2003. № 2(15). С. 25–38.
Legalov, A.I., Schvetc, D.A. Procedural language with support for evolutionary design // Science Bulletin the NSTU. 2003. No. 2(15). P. 25–38. (In Russ.)

[7] **Легалов А.И.** Мультиметоды и парадигмы // Открытые системы. 2002. № 5. С. 33–37.
Legalov, A.I. Multimethods and paradigms // Open Systems J. 2002. No. 5. P. 33–37. (In Russ.)

[8] **Легалов И.А.** Применение обобщенных записей в процедурно-параметрическом языке программирования // Науч. вест. НГТУ. 2007. № 3(28). С. 25–37.
Legalov, I.A. Using of generalized records in procedural-parametric programming language // Science Bulletin the NSTU. 2007. No. 3(28). P. 25–37. (In Russ.)

[9] **Легалов А.И.** Эволюция мультиметодов при процедурном подходе. Адрес доступа: http://www.softcraft.ru/coding/evp/evp.shtml (дата обращения 23.09.2014).

Legalov, A.I. Evolution of multi methods in procedural approach. Available at: http://www.softcraft.ru/coding/evp/evp.shtml (accessed 23.09.2014). (In Russ.)

[10] **Легалов А.И., Легалов И.А., Солоха А.Ф.** Эволюционное расширение программ при различных парадигмах программирования // Тр. 16-й Байкальской Всерос. конф. "Информационные и математические технологии в науке и управлении". Ч. 3. Иркутск: ИСЭМ СО РАН, 2011. С. 42–49. (ISBN 978-5-93908-094-1).
Legalov, A.I., Legalov, I.A., Soloha, A.F. Evolutionary expansion of programs for different programming paradigms // Proc. of 16th Baikal's All-Russian Conf. "Informational and mathematical technologies in science and management". Vol. 3. Irkutsk: ISEM SB RAS, 2011. P. 42–49. (In Russ.)

[11] **Элджер Дж.** C++: библиотека программиста: Пер. с англ. СПб.: Питер, 1999. 320 с.
Alger, J. C++ for Real Programmers. AP Professional, 1998. 388 p.

[12] **Легалов А.И., Косов П.В.** Особенности организации и использования обобщенных записей языка процедурно-параметрического программирования Alien // Тр. 13-й Всерос. науч.-практ. конф. "Проблемы информатизации региона 2013". Красноярск: ИВМ СО РАН, 2013. С. 187–194.
Legalov, A.I., Kosov, P.V. Features of the organization and the use of generalized records in procedurally-parametric programming language Alien // Proc. of 13th All-Russian Scientific-practical Conf. "Problems of Regional Informatization 2013". Krasnoyarsk: ICM SB RAS, 2013. P. 187–194. (In Russ.)

Поступила в редакцию 30 декабря 2015 г.

Evolutionary extension of programs using the procedural-parametric approach

Legalov, Alexander I.[1,*], Kosov, Pavel V.[2]

[1]Siberian Federal University, Krasnoyarsk, 660041, Russia
[2]Glyph Worlds, Krasnoyarsk, 660001, Russia

*Corresponding author: Legalov, Alexander I., e-mail: legalov@mail.ru

Methods of evolutionary expansion of the previously written code are widely used during development of large software systems. Evolutionary expansion increases the reliability of programs and allows to avoid mistakes that can be made by modification of already functioning modules. Methods of evolutionary development are widely used in various programming paradigms. Modern software developers employ their own opportunities for paradigms and their special libraries. The latter includes design patterns.

Procedural-parametric programming paradigm was proposed to increase the flexibility of the process of evolutionary expansion of programs. Specific feature of this paradigm lies in separation for presentations of data and procedures, which in this case can be expanded through the creation of new specializations and their handlers without alteration of the previously written code. This provides evolutionary support for multiple polymorphism used in multimethods without direct analysis of types of arguments. The special programming languages were developed for ensuring this support. However extensive application of popular languages that are supported by a huge number of different libraries prevents expansion of this approach.

To increase the popularity of the procedure-parametric approach we propose methods of its application in the modern multiparadigm programming languages. The article presents examples of the implementation of evolutionary extensible and automatically installed data. It is shown how procedures and functions which support not only single, but also multiple polymorphism could be created and painlessly expanded.

Examples of the implementation of scalable data and procedures that use parametric arrays are presented. The demonstration of a flexible development of a program is presented for some typical extension related situations. It is shown that the proposed approach provides more flexible development of a code compared to traditional programming methods. The proposed approach is used in the implementation of the code generator for procedural-parametric programming language. It also becomes the basis of the library, which supports the evolutionary developed programs. The proposed approach is realized with the help of C++ programming language.

Keywords: evolutionary software development, programming paradigms, procedural-parametric programming, multiparadigm style.

Acknowledgements. The work was supported by the RFBR (agreement No. 13-01-00360 “Methods and tools for evolutionary software development using procedural-parametric programming paradigms”).

Received 30 December 2015