УДК 004.052.2

МОДЕЛЬ ДВУХФАЗНОЙ ТРАНСЛЯЦИИ КОДА МУЛЬТИВЕРСИЙ ПРОГРАММНЫХ МОДУЛЕЙ

Грузенкин Д.В., Царев Р.Ю., Кузнецов А.С.
ФГАОУ ВПО «Сибирский федеральный университет», Красноярск, e-mail: gruzenkin.denis@good-look.su

Статья посвящена проблеме автоматизации и ускорения процесса разработки программного обеспечения, в том числе мультиверсионного программного обеспечения. Рассмотрены особенности автоматической и автоматизированной генерации программного кода по диаграммам и блок-схемам различного вида. Приведено описание предметно-ориентированного языка для определения программных алгоритмов. Описаны средства и методы генерации кода на целевом языке программирования по блок-схемам с использованием описания алгоритма на предметно-ориентированном языке в качестве промежуточного представления. В статье предложена новая концептуальная модель двухфазной трансляции кода мультиверсий программных модулей, которая позволяет упростить процесс создания мультиверсионного программного обеспечения, а также программного обеспечения других типов. Приведен практический пример генерации кода на целевом языке программирования Си алгоритма равномерного поиска по его графическому представлению в виде блок-схемы.

Ключевые слова: предметно-ориентированный язык, генерация кода, кодогенерация, кодогенерация на основе блок-схемы, кодогенерация на основе XML

MODEL OF TWO-PHASE TRANSLATION OF MULTIVERSION PROGRAMM MODULES' CODE

Gruzenkin D.V., Tsarev R.Y., Kuznetsov A.S.
Siberian Federal University, Krasnoyarsk, e-mail: gruzenkin.denis@good-look.su

The article concerns the problem of automation and acceleration of the process of development of software, including N-version software. It is considered the features of automated and automatized program code generation based on different types of diagrams and flow-charts. It is presented the domain-specific language for program algorithm description. It is described also the means of code generation on a target programming language based on flow diagram with the use of algorithm description on domain-specific language as intermediate level representation. The paper presents the developed conceptual model of two-phase translation of multiversion program modules' code; it allows simplifying software development process for N-version software and some other kinds of software. The paper presents an example of code generation on the target programming language C based on graphical representation – a flow diagram.

Keywords: domain-specific language, code generation, code generation based on flow diagram, code generation based on XML

Графические или визуальные языки проектирования и программирования уже прочно вошли в «арсенал» средств инженеров-разработчиков и проектировщиков программного обеспечения (ПО), в том числе и мультиверсионного программного обеспечения. Для графических языков разработаны свои среды разработки, отладчики, трансляторы [6, 8].

На сегодняшний день уже решено множество прикладных задач в области программного обеспечения [1, 7]. Причем прикладные задачи зачастую решаются с помощью уже готовых, известных алгоритмов, которые требуется лишь модифицировать для применения их при реализации конкретной задачи [1, 6]. В условиях постоянно растущего темпа жизни, развития новых технологий, увеличения потребностей общества и каждого человека в частности возрастает и потребность в качественном программном обеспечении для удовлетворения нужд пользователей. Скорость создания ПО приходится также постоянно увеличивать за счет использования современных методов и средств проектирования и разработки программного обеспечения [2, 6].

В современной программной инженерии существует практика разработки предметно-ориентированных языков программирования для решения конкретных практических задач и описания объектов предметной области решаемой задачи. Причем для разработки сложных предметно-ориентированных языков применяются специализированные инструменты [6]. Для описания программных алгоритмов были разработаны и широко применяются такие визуальные языки, как блок-схемы и UML-диаграммы деятельности. При этом нет полноценного прикладного унифицированного предметно-ориентированного символьного (текстового) языка для описания программных алгоритмов, кроме учебного алгоритмического языка, предложенного в [4]. Он, как видно из названия, является учебным

и был создан для обучения учащихся средних школ основам информатики, а не для практического применения при разработке программного обеспечения.

Наличие языка, нацеленного на практическое применение, позволит унифицировать представление программных алгоритмов, созданных с помощью различных графических языков, с применением их различных нотаций. Это даст возможность использования одного программного средства для генерации кода описанного алгоритма на целевом языке программирования, вместо необходимости в подборе под конкретный тип диаграммы своего инструментария кодогенерации. В свою очередь, эта возможность позволит значительно ускорить разработку ПО, и особенно мультиверсионного ПО так как на его создание затрачиваются значительные средства при разработке различных версий программных модулей, которые должны отличаться друг от друга способом реализации, например алгоритмом, используемым для решения конкретной задачи.

Предметно-ориентированный язык описания алгоритмов

В качестве графического языка, на основании представления которого будет производиться генерация кода на предметно-ориентированном языке, для записи программного алгоритма на первом этапе разработки были выбраны блок-схемы алгоритмов. Данный выбор был обусловлен применением ГОСТ 19.701-90 для стандартизации условных обозначений [3], что является важным аспектом для устранения неоднозначности понимания и начертания элементов блок-схемы.

К промежуточному представлению программного алгоритма на предметно-ориентированном языке предъявляются следующие требования:

1) простота для понимания человека;
2) хорошая формализация;
3) четкая структурированность;
4) легкость создания программного алгоритма на основе графического представления.

В качестве промежуточного представления программного алгоритма было выбрано представление алгоритма на XML, так как представление данных в XML-формате отвечает всем вышеуказанным требованиям.

Каждой фигуре, представленной в [3], ставится в соответствие элемент (тег) с ее названием, переведенным на английский язык. В зависимости от свойств графического элемента каждый тег может иметь определенный набор атрибутов и/или определенное количество наследников (вложенных в него тегов). На основе данных положений был создан предметно-ориентированный язык промежуточного представления программных алгоритмов на базе XML.

Промежуточное представление программного алгоритма, созданное с помощью специальных программных средств на основе блок-схемы, сложно использовать непосредственно для генерации кода на целевом языке. Поэтому XML-представление разбивается на последовательность строк, или лексем. Множество лексем разбивается на непересекающиеся подмножества (лексические классы) [5, 8].

Транслятор предметно-ориентированного языка использует традиционную схему взаимодействия синтаксического и лексического анализаторов, когда синтаксический анализатор обращается к лексическому за очередной лексемой. Задача синтаксического анализа заключается в построении синтаксической структуры XML-документа для последующей генерации на ее основе программного кода на целевом языке программирования. Построение синтаксической структуры документа осуществляется путем нахождения порождения (если таковое существует) конкретной языковой конструкции, для чего применяется заданная грамматика языка. Более наглядно процесс создания кода на целевом языке программирования на основе блок-схемы алгоритма приведен на рис. 1. Для упрощения на рис. 1 не показано взаимодействие с окружением транслятора.

Практический пример кодогенерации

В качестве примера представлена генерация кода программы на языке Си для нахождения экстремума с помощью одного из методов оптимизации – метода равномерного поиска. Блок-схема, на основании которой производится генерация кода, представлена на рис. 2. Можно отметить, что на рис. 2 приведена укрупненная блок-схема алгоритма.

На основе блок-схемы (рис. 2) создается промежуточное XML-представление на предметно-ориентированном языке, которое приведено в листинге 1. В листинге 2 представлен код программы на целевом языке программирования Си.

Из приведенного примера видно, что промежуточный код на предметно-ориентированном языке прост и понятен как для человека, так и для автоматизированных средств разбора, благодаря его формализованности и структурированности.

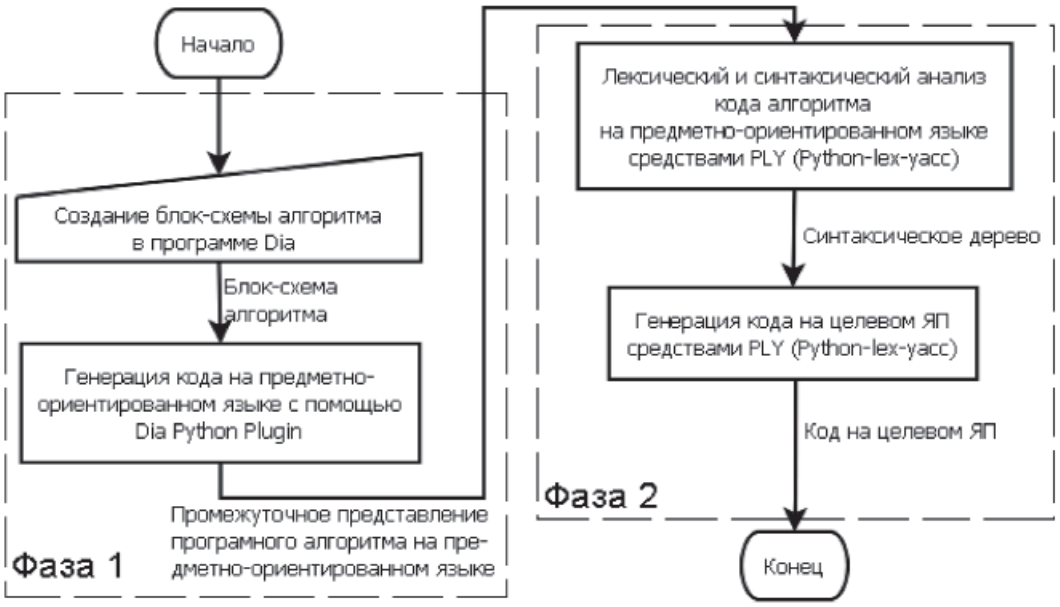

Рис. 1. Блок-схема процесса автоматизированной кодогенерации

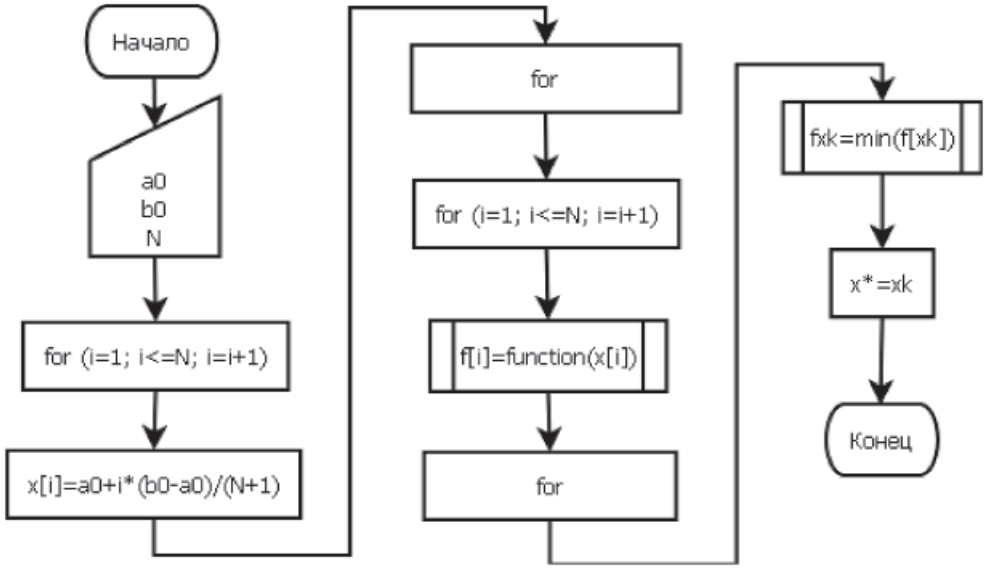

Рис. 2. Блок-схема алгоритма равномерного поиска

Полученный в результате исходный код программы на целевом языке программирования, в данном случае на языке Си, также приводится в удобочитаемом виде, что позволяет при необходимости с легкостью его модифицировать.

Заключение

На текущий момент существует ряд генераторов кода на различных целевых языках программирования, как на основе блок-схем, так и UML-диаграмм различного вида. Отличительной особенностью предлагаемого в статье подхода является концепция двухфазной генерации программного кода на целевом языке программирования с использованием промежуточного представления на предметно-ориентированном языке описания алгоритмов. Такой подход дает возможность полного контроля процесса кодогенерации, а также позволяет генерировать по одному алгоритму код одновременно на нескольких целевых языках программирования. Данная концепция может быть применена при создании мультиверсионного программного обеспечения с целью диверсификации версий модулей, как на уровне алгоритмов, так и языков реализации.

```
<?xml version="1.0" encoding="UTF-8"?>
<version id="0">
  <docAuthor id="1">Gruzenkin Denis</docAuthor>
  <algorithm id="2">
    <algorithm_name id="3">Метод равномерного поиска</algorithm_name>
    <preparation_process id="4">
      <modified_commands id="5">
        <modified_command id="6">a0</modified_command>
        <modified_command id="7">b0</modified_command>
        <modified_command id="7">N</modified_command>
      </modified_commands>
    </preparation_process>
    <start_loop_process id="8">
      <condition id="9">
        <preparation_process id="10">
          <modified_commands id="11">
            <modified_command id="12">i=1</modified_command>
          </modifinded_commands>
        </preparation_process>
      </condition>
      <condition id="13">
        <preparation_process id="14">
          <modified_commands id="15">
            <modified_command id="16">i=i+1</modified_command>
          </modified_commands>
        </preparation_process>
      </condition>
      <condition id="17">i&lt;=N</condition>
    </start_loop_process>
    <process id="18">x(i)=a0+i*(b0-a0)/(N+1)</process>
    <stop_loop_process id="8"></stop_loop_process>
    <start_loop_process id="19">
      <condition id="20">
        <preparation_process id="21">
          <modified_commands id="22">
            <modified_command id="23">i=1</modified_command>
          </modified_commands>
        </preparation_process>
      </condition>
      <condition id="24">
        <preparation_process id="25">
          <modified_commands id="26">
            <modified_command id="27">i=i+1</modified_command>
          </modified_commands>
        </preparation_process>
      </condition>
      <condition id="28">i&lt;=N</condition>
    </start_loop_process>
    <predefined_process href="function.c" id="29">
      <operations id="30">
        <operation id="31">f(i) = function(x(i))</operation>
      </operations>
    </predefined_process>
    <stop_loop_process id="19"></stop_loop_process>
    <preparation_process id="4">
      <modified_commands id="5">
        <modified_command id="7">fxk</modified_command>
      </modified_commands>
    </preparation_process>
    <predefined_process href="function.c" id="29">
      <operations id="30">
        <operation id="31">fxk = min(f(xk))</operation>
      </operations>
    </predefined_process>
    <preparation_process id="4">
       <modified_commands id="5">
         <modified_command id="7">x_=fxk</modified_command>
       </modified_commands>
     </preparation_process>
  </algorithm>
</version
```

Листинг 1. XML-представление алгоритма равномерного поиска

```
void main()
{
        float a0;
        scanf("%f", &a0);
        float b0;
        scanf("%f", &b0);
        int N;
        scanf("%d", &N);
        int* f= (int*) malloc(N * sizeof(int));
        int* x= (int*) malloc(N * sizeof(int));
        for(int i=1;i<=N;i=i+1)
        {x[i]=a0+i*(a0-b0)/(N+1);}
        for(int i=1;i<=N;i=i+1)
        {f[i]=function(x[i]);}
        fxk=min(f);
        float x_;
        x_=fxk;
    free(x); free(f);
}
```

Листинг 2. Программный код исходного алгоритма на целевом языке С

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 16-57-46016 СТ_а.

Список литературы

1. Артамошин С.А. Подходы к созданию автоматизированных систем, предназначенных для обработки информации ограниченного распространения / С.А. Артамошин, С.И. Нагорный, В.В. Донцов // Спецтехника и связь. – 2009. – № 2. – С. 41–48.

2. Брукс Ф. Мифический человеко-месяц, или Как создаются программные системы. – СПб.: Символ-Плюс, 2010. – 304 с.

3. ГОСТ 19.701-90 Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. Дата введ. 01.01.1992. – М.: Государственный стандарт Союза ССР, 1990.

4. Ершов, А.П. Алгоритмический язык в школьном курсе информатики и вычислительной техники // Микропроцессорные средства и системы – 1985. – № 2. – С. 48–51.

5. Кузнецов, А.С. Генерация компиляторов мультисинтаксических языков программирования мультиверсионных систем / А.С. Кузнецов, И.В. Ковалев // Программные продукты и системы. – 2008. – Вып. 4 (84). – С. 101–103.

6. Сухов, А.О. Сравнение систем разработки визуальных предметно-ориентированных языков // Математика программных систем: межвузовский сборник научных статей. – Пермь: Пермский государственный национальный исследовательский университет, 2012. – Вып. 9. – С. 84–111.

7. Царев Р.Ю. Методология многоатрибутивного формирования мультиверсионного программного обеспечения сложных систем управления и обработки информации: монография; Краснояр. гос. аграр. ун-т. – Красноярск, 2011. – 210 с.

8. Aho A.V., Lam M.S., Sethi R., Ullman J.D. Compilers: Principles, Techniques and Tools Addison Wesley; 2nd edition 2006, 1000 p.

9. Beazley, D.M. PLY (Python Lex-Yacc) Home page // URL: http://www.dabeaz.com/ply/ (Дата обращения: 03.12.2015).

10. Dia Home page // URL: http://wiki.gnome.org/Apps/Dia (Дата обращения: 03.12.2015).

References

1. Artamoshkin S.A., Nagornyi S.I., Dontsov V.V. *Podhody k sozdaniju avtomatizirovannyh sistem, prednaznachennyh dlja obrabotki informacii ogranichennogo rasprostranenija* [Approaches to automated system creation, which aimed to limit access information processing]. Specialized machinery and communication, 2009, no. 2, pp. 41.

2. Brooks F. *Mificheskij cheloveko-mesjac ili kak sozdajutsja programmnye sistemy* [The mythical Man-Month: Essays on Software Engineering]. St. Petersburg: Symbol-Plus, 2010, 304 p.

3. *GOST 19.701-90 Edinaja sistema programmnoj dokumentacii. Shemy algoritmov, programm, dannyh i sistem. Uslovnye oboznachenija i pravila vypolnenija. Data vvedenija 01.01.1992* [Unified system for program documentation. Schemes of algorithms, programs, data and systems. Nomenclature and regulations]. Moscow: State standard of the USSR, 1990, 23 p.

4. Ershov A.P. *Algoritmicheskij jazyk v shkolnom kurse informatiki vychislitelnoj tehniki* [Algorithmic language in the school course of computer science]. Microprocessor tools and systems, 1985, no. 2, pp. 48–51.

5. Kuznetsov A.S. *Generatsiya kompilyatorov multisintaksicheskikh yazykov programmirovaniya multiversionnykh sistem* [Generation of multisyntax programming languages compiler for N-version systems]. Software products and systems, 2008, no. 4, pp. 101–103.

6. Sukhov A.O. *Sravnenie sistem razrabotki vizualnyh predmetno-orientirovannyh jazykov* [Comparison of the development systems for visual object-oriented languages]. Mathematics of software systems, Perm state national research institute, 2012, no. 9, pp. 84–111.

7. Tsarev R.Yu. *Metodologija mnogoatributivnogo formirovanija multiversionnogo programmnogo obespechenija slozhnyh sistem upravlenija i obrabotki informacii* [Methodology of multiple attribute formation of multiversion software for complex control and information processing systems]. Krasnojarsk: Krasnoyarsk State Agrarian University, 2011, 210 p.

8. Aho A.V., Lam M.S., Sethi R., Ullman J.D. Compilers: Principles, Techniques and Tools Addison Wesley; 2nd edition 2006, 1000 p.

9. Beazley D.M. PLY (Python Lex-Yacc). Home page. Available at: http://www.dabeaz.com/ply/ (accessed 03.12. 2015).

10. Dia Home page. Avaiable at: http://wiki.gnome.org/Apps/Dia (accessed 03.12.2015).