Algorithm source codes generation for ensuring N-version software diversity

D V Gruzenkin¹, A S Kuznetsov¹, R Y Tsarev¹, T N Yamskikh¹, A N Knyazkov¹, A N Pupkov¹, Leong Yee Rock ²

¹ Siberian Federal University, 79, Svobodny pr., Krasnoyarsk, 660041, Russia
² VYROX International Sdn Bhd (MSC Status), Unit 16-05, Level 16, Tower A, Vertical Business Suite, Avenue 3, Bangsar South, No. 8, Jalan Kerinchi, www.vyrox.com, Kuala Lumpur, 59200, Malaysia

E-mail: gruzenkin.denis@good-look.su, alex@vyrox.com

Abstract. Diversity is one of the fundamental principles for N-version software development. The development of redundant versions can significantly improve the reliability of software, but it is more time and labour consuming. Business and manufacturing require a higher level of software reliability. The task of reducing the lead-time and cost effectiveness becomes more important when developing N-version software for such industries. It is proposed to use automatic source code generation to create N-version software. The use of domain-specific language for the intermediate representation of the algorithm simplifies code generation in the target programming language. The authors discuss a new conceptual model of two-step code translation for N-version software. The proposed approach will simplify the process of N-version software development and reduce the number of required resources. The paper contains an example of C code generation as a target programming language, using flowcharts as a graphical representation of the source algorithm. The described approach is intended for use in safety-critical industries.

1. Introduction

Nowadays software engineers and designers use visual programming languages to create software, including N-version one, by manipulating graphical elements [1-3]. Special development environments, debuggers, and translators have been developed for visual programming languages [4-7].

At present, a great number of practical tasks in the field of software have been efficiently solved. Moreover, the problems are often solved with the help of the existing well-known algorithms that need only some modification to be applied for a specific task. Moreover the demand for high quality software solutions satisfying the users’ needs with minimum lead-time and cost effectiveness is increasing with the ever-growing pace of life and development of new technologies.

Modern software engineering practices the development of domain-specific programming languages to solve specific practical problems and describe the domain objects. To develop complex domain-specific languages, specialized tools are used [8, 9]. Visual languages such as flowcharts and UML activity diagrams have been developed and widely used to describe software algorithms [10, 11]. At the same time, there is no full-fledged unified application-oriented symbolic (text) language for the description of program algorithms.
With the language intended for practical application it will be possible to introduce a uniform system of software algorithms created with different visual languages, using their various notations. It will be possible to use a single code generator tool for the algorithm described in the target programming language, instead of selecting a specific code-generator tool for a particular diagram. In turn, this will significantly speed up the development of software, N-version software in particular, since considerable time and resources are spent to develop various versions of software units, which should only differ in the way of implementation, for example, the algorithm used to solve a very specific problem.

2. Domain-specific programming language designed to describe algorithms

At the first stage of development we chose flowcharts to record the program algorithm as a visual language used to generate the code in a domain-specific language. This choice was due to the use of technical standardization of graphical symbols, which is important to eliminate the ambiguity in understanding and drawing the elements of the flowchart.

The following requirements are imposed to the intermediate representation of the algorithm in a domain-oriented language:

1) it should be easy to understand for people;
2) it should be well formalized;
3) it should be well-defined;
4) it should be easy to create an algorithm based on graphical representation.

Since the representation of data in XML format meets all the requirements above we chose it as an intermediate representation of the algorithm.

Each symbol is assigned to an element (tag) with its name translated into English. Depending on the properties of the graphic element each tag may have a certain set of attributes and / or a certain number of descendants (tags attached). Based on these provisions we created an object-oriented language for the intermediate representation of XML-based software algorithms.

It is difficult to use the intermediate representation of the software algorithm created with special flowchart-based software directly to generate code in the target language. Therefore, the XML representation is broken down into a sequence of strings, or tokens. We split the set of tokens into disjoint subsets (token classes) [4].

The domain-specific language translator uses the traditional scheme of interaction between the syntax parser and lexical scanner, when the parser asks for the next token from the scanner. The task of parsing is to build the syntactic structure of an XML document for the subsequent generation of program code in the target programming language. The syntactic structure of the document is built by defining the derivations (if any) of a specific language construct. The given grammar is used for this purpose. More clearly the process of code generation in the target programming language from a flowchart is shown in Figure 1. For simplicity Figure 1 does not show the interaction between the translator and its environment.

3. An example of code generation in C

As an example we consider the generation of C program code for finding an extremum with one of the optimization methods - the Uniform Search method. The flowchart for code generation is shown in Figure 2. It should be noted that Figure 2 shows the enlarged flowchart of the algorithm.

Based on the flowchart (Figure 2), an intermediate XML representation is created in a domain-specific language, as shown in Listing 1. Listing 2 shows the program code in the target C programming language.

Thus, we can see that the intermediate code in a domain-oriented language is simple and clear both for people and automated parsing tools due to its formalization and statefulness.
The source code of the program in the target programming language, in this case in the C language, is provided in a readable form, which makes its modification possible if necessary.
<?xml version="1.0" encoding="UTF-8"?>
<version id="0">
  <docAuthor id="1">Gruzenkin Denis et al.</docAuthor>
  <algorithm id="2">
    <algorithm_name id="3">Uniform search algorithm</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">
        <modified_commands id="11">
          <modified_command id="12">i=1</modified_command>
        </modified_commands>
      </condition>
    </start_loop_process>
    <condition id="13">
      <modified_commands id="15">
        <modified_command id="16">i=i+1</modified_command>
      </modified_commands>
    </condition>
    <condition id="17">i&lt;N</condition>
  </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">
      <modified_commands id="22">
        <modified_command id="23">i=1</modified_command>
      </modified_commands>
    </condition>
    <condition id="24">
      <modified_commands id="26">
        <modified_command id="27">i=i+1</modified_command>
      </modified_commands>
    </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>
</algorithm>
Listing 1. XML representation of the Uniform search algorithm.

```xml
<modified_commands>
  <preparation_process href="function.c" id="29">
    <operations id="30">
      <operation id="31">fxk = min(f(xk))</operation>
    </operations>
  </preparation_process>
  <predefined_process id="4">
    <modified_commands id="5">
      <modified_command id="7">x_ = fxk</modified_command>
    </modified_commands>
  </preparation_process>
</modified_commands>

Listing 2. The program code of the generic algorithm in the target language C.

```c
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);
}
```

4. Conclusion
Currently, there are a number of tools for code generation in various target programming languages, both from flowcharts and UML diagrams of various types. A distinctive feature of the approach proposed in this paper is the concept of two-step code generation in the target programming language using intermediate representation in the domain-specific language to describe algorithms. With such approach, it is possible to control the code generation process and to generate the code using several algorithms simultaneously in several target programming languages. This concept can be applied for the creation of N-version software in order to diversify unit versions at the level of algorithms and the languages of implementation.
References

[1] Assem M, Keshk A, Ismail N and Nassar H 2007 Specification-driven automated testing of java swing GUIs using XML ITI 5th International Conference on Information and Communications Technology (ICICT 2007) art 4475622 83-87

[2] Cabeda M and Santos P 2018 Automated test generation based on a visual language applicational model Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) (Lisbon: Portugal/IEEE Computer Society) art 8506528 289-290

[3] Pratomo A B and Perdana R S 2018 Arduviz, a visual programming IDE for arduino Proceedings of the International Conference on Data and Software Engineering (ICoDSE) (Palembang: Indonesia/IEEE Inc.) 1-6

[4] Aho A V, Lam M S, Sethi R and Ullman J D 2006 Compilers: Principles, Techniques and Tools (Boston/Addison Wesley) p 1009

[5] Ganesan L, Scaffidi C and Dove A 2017 Support for learning while debugging in a distributed visual programming language Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) (Raleigh: United States/IEEE Computer Society) art 8103477 267-271

[6] Hsu S-Y, Lou Y-F, Jung S-Y and Sun C-T 2017 Shelves: A user-defined block management tool for visual programming languages Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 10515 LNCS 335-344

[7] Ragusa G and Henriques H 2018 Code review tool for visual programming languages Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) (Lisbon: Portugal/IEEE Computer Society) art 8506527 287-288

[8] Morgan R, Grossmann G, Schrefl M and Stumptner M 2018 Using VizDSL for Modelling Visualization Processes Proceedings - IEEE International Enterprise Distributed Object Computing Workshop (EDOCW) (Stockholm: Sweden/IEEE Inc.) art 8536125 212-215

[9] Ünel G and Sekmen S 2018 CutLang: A Particle Physics Analysis Description Language and Runtime Interpreter Computer Physics Communications 233 215-236

[10] Schumacher F, Holzer M, Greiner T and Rosenstiel W 2011 Modeling and code generation of recursive algorithms with extended UML Activity Diagrams Proceedings of 21st International Conference, Radioelektronika (Brno: Czech Republic) art 5936437 405-408

[11] Wang L, Wang G, Zhou M, Chu Y, Chen K and Chen P 2012 Research on and implementation of the algorithm from the program flowchart to the code Xi’an Dianzi Keji Daxue Xuebao/Journal of Xidian University 39 (6) 70-77