Error Messaging in Generative Programming

Danijel Radošević, Ivan Magdalenić, Tihomir Orehovački
Faculty of Organization and Informatics
University of Zagreb
Pavlinska 2, 42000 Varaždin, Croatia
{danijel.radosevic, ivan.magdalenic, tihomir.orehovacki}@foi.hr

Abstract. *Standard programming tools use a system of error messages and warnings to help programmers in finding syntax and logical errors in their programs. Generative programming differentiates the level of generator from the level of generated application. The source code is synthesized at the level of generator from a set of code templates according to application specification and rules defined by generator configuration. Code templates, application specification and generator configuration are mutually dependent and error in any part may result in incorrect source code. This paper deals with possibilities of introducing error messages that are specific at the level of generator. An example of generator is developed and discussed.*

Keywords. generative programming, generator, error messages

1 Introduction

Generative Programming is one of the concepts of Software Product Lines (SPL). SPL provides a means for composing software products that match the requirements of different application scenarios from a single code base and can be developed using a variety of implementation techniques [13]. Other concepts in this area are pre-processor definitions, components, Aspect Oriented Programming, Feature-Oriented Programming (FOP) [8][13], Aspectual Feature C Modules (AFMs) [1] and frames like XVCL [15]. Using Generative Programming helps to increase the software making productivity, by producing it in a way comparable to industrial production. Almost every new technology has its own specific problems. In the case of the generative programming problems arise during the creation of program code templates, building of source code generator and definition of application. The creation of source code requires high-quality error messaging system when using generative programming technique.

Our work is based on the SCT dynamic frames model that is used for source code generation. The SCT model consists of three basic components: Specification (S), which describes the application characteristics, Configuration (C), which describes the rules for building applications, and Templates (T), which refers to application building blocks. The SCT model is described in detail in [9]. It is primarily designed for web application development, but there are no constraints to using the SCT model in development of any kind of a source-code, regardless to problem domain and programming language.

This paper introduces error messaging system to the SCT dynamic frames model. Introduced error-related messages are compared to similar messages in object-oriented programming languages and some generative systems.

The paper is organized as follows: Related work is presented in section 2. The basics of the SCT model are explained in section 3. Possible SCT model inconsistencies are discussed in section 4. Section 5 describes error messaging in generative programming, which is followed by one example in section 6. The conclusion is given in section 7.

2 Related work

The process of writing code in one of the object oriented programming languages [6] consists of four steps: writing the source code, compiling the source code, linking the executable code and finally testing the program. If during the writing of source code a programmer makes syntax or logical mistake, in the remaining three phases following types of error related messages may occur:

- **Compile-time error message** indicates that source code violates syntax or grammatical rules of a programming language. Before it is possible to compile source code into object code, it is necessary to correct all syntax errors. The most common examples of compile-time errors are

[12][14]: Undeclared Variables, Undeclared Functions, Missing Semicolons, Extra Semicolons, Incorrect Number of Braces, Unmatched Parentheses, Unterminated Strings, Left-Hand Side of Assignment does not Contain an L-Value, Value-Returning Function has no Return Statement, Converting Errors (e.g. int* to int**), and Illegal Function Overloading.

- **Link-time error message** prevents the generation of executable code when it is not possible to link object codes. Some examples of frequent logical mistakes that lead to occurrence of link-time error messages are [11]: Uninitialized variables, Setting a variable to an uninitialized value, Using a single equal sign to check equality, Divide by Zero, Forgetting a Break in a Switch Statement, Overstepping Array Boundaries, and Misusing the && and || operators.
- **Run-time error message** occur in the testing phase when the program or any of its parts returns the unexpected results. Most often causes of this kind of messages are following logical errors [14]: Infinite Loop, Misunderstanding of Operator Precedence, Dangling Else, Off-By-One Error, Code inside a Loop that does not Belong There, and Not Using a Compound Statement When One is Required.
- **Warning message** do not interrupts the process of compiling or linking code but often indicates the cause of errors that occur during the testing phase. Using '=' when '==' is Intended, Loop has no Body, and Uninitialized Variable are examples of common syntax warnings [14].

Recent study [10] into analysis of students' compilation behavior revealed that "Unused Variable", "Undeclared Variables", "Expected `;' Before" and "Control Reaches End of Non-Void Function" are most frequent error related messages that occur when writing source code. It should be noted that "Unused Variable" and "Control Reaches End of Non-Void Function" are warning messages while "Undeclared Variables" and "Expected `;' Before" are compile-time error messages. Error related messages are not only useful within aforementioned steps of writing programming code, but also during the process of generative application development. There are three mature approaches to generative programming: XVCL, GenVoca and Codeworker.

XVCL (XML-based Variant Configuration Language) is a meta-programming language based on the same concepts as frame technology [2]. It can be used for handling variants in program code or software product lines. To facilitate the implementation of the variations, we can use XVCL commands to mark the variation points in program and thus decompose it into generic and adaptable components called x-frames. GenVoca (a mixture of the names Genesis and Avoca) is a composition methodology aimed for creating system families. It is based on two main ideas [3]: feature modularity which came out of inheritance and programming style called programming-by-difference [4]; and program objectification. In GenVoca, each module is comprised of set layers where each layer specifies an aspect of module. Codeworker [5] is a versatile parsing tool based on generative technique. It is used for generating source code by parsing existing or newly created language. A scripting language provided by Codeworker is configured for the writing of code generation templates and for the description of language grammars.

Although principles of XVCL, GenVoca and Codeworker have been tested in practice, studies related to the error messaging in generative application development were not well explored before.

3 SCT model basics

The SCT generator model [9] is developed on the basis of previous Scripting model of generator (SGM) [7]. The model defines the source code generator from three kinds of elements: Specification (S), Configuration (C) and Templates (T). All three model elements together make the SCT frame (Figure 1):

- **Specification** contains features of generated application in form of attribute-value pairs.
- **Template** contains source code in target programming language together with connections (replacing marks for insertion of variable code parts)
- **Configuration** defines the connection rules between Specification and template.

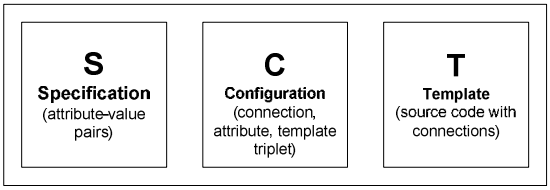

Figure 1. SCT frame

Starting SCT frame contains the whole Specification, the whole Configuration, but only the base template from the set of all Templates. Other SCT frames are produced dynamically, for each connection in template, forming generation tree (Figure 2). So, SCT generator model is generator model based on dynamic frames, unlike some other frames-based generator models, like XVCL [2].

It's important that SCT based generator is fully configurable, so the whole generation process is defined in Configuration, with no need to change the code of generator itself.

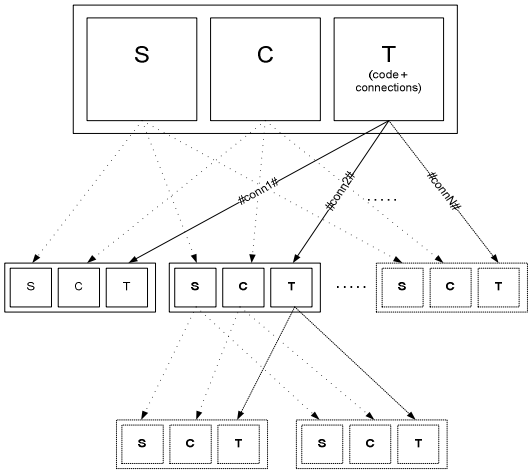

Figure 2. The generation tree

As shown in Figure 2, the process of code generation is recursive, where each level behaves as a whole generator.

4 Possible SCT model inconsistencies

There are several possible SCT model inconsistencies that can occur during development of source code generator: the existence of all needed files, correct syntax according to SCT model definition, insufficient Specification and mutual dependencies of SCT elements.

Because the SCT model assumes the use of several files, the first step is to check the existence of these files. The next step is checking the syntax of Specification and Configuration. The check of the syntax of programming templates cannot be done at this point since it can be done only on final source code. If some necessary attributes and their values are not specified, some connections in templates will be unused and the generated source code will be invalid. This could be detected regardless of compiling: the necessary Specification attribute could be found in Configuration according to the unused connection in the generated code.

Specification and Configuration are mutually dependent. All attributes in Specification have to be defined in Configuration. Configuration and Templates are also mutually dependent. All connections that appear in program templates have to be defined in Configuration. The check of existence of all needed templates cannot be done because SCT model allows selection of templates dynamically based on values from the specification.

There are some issues related to syntactic correctness of the generated code that are as a consequence of the process of generating source code. These errors are hard to detect because they depend on target programming language. Some of them are: usage of unsafe names in Templates, calls of functions prior to their declarations, and breaking program restrictions.

The usage of unsafe names in Templates (variables, functions, classes etc.) is potential cause of syntactic incorrectness because the attribute values from Specification could collide with the names in Templates. Using names with prefixes/suffixes could reduce the risk. Some programming languages require that functions are to be defined prior to their calls. The order of Specification attributes could lead to the breach of that rule. This could be solved by providing function declarations prior to their use (which should be included in Templates or generated). Breaking program restrictions can be done be exceeding the size limit and other restrictions caused by Specification values.

Generally, the issues can be avoided/solved by the appropriate generative application development process, where building generators and generated applications are closely connected processes. The error messaging could significantly help in generative application development process.

5 Error related messages

The current error related messages introduced in the SCT generator model include *errors*, as obligatory kind of messages, and *warnings* that could be ignored in some cases. That is similar to error related messages in standard programming languages (e.g. structural and object-oriented). The whole list of error related messages in the SCT generator model is given in Table 1:

Error or Warning	Message/Explanation
Error 01	Specification attribute **<attr. name>** is not used in Configuration.
	Incomplete Configuration or wrong specification attribute.
Error 02	No template file <**file name**>
	Template that is specified in Configuration was not found.
Error 03	Connection <**conn.**> was not found in Configuration.
	Connection in '#'-es that is used in template was not specified in Configuration.
Error 04	Can't write into file: <**file name**>
	Generated code can't be written in a file due to file/folder protection, or file in use.
Error 05	Can't open Specification file: <**file name**>
	Missing Specification file or file can't be read due to file/folder protection, or file in use.
Error 06	Can't open Configuration file: <**file name**>
	Missing Configuration file or file can't be read due to file/folder protection, or file in use.
Error 07	Output types are not specified. Use **OUTPUT** keyword.
	At least one output type has to be specified in Specification.
Warning 01	Attribute **<attr. name>** is used in Configuration, but not specified in Specification.
	Some attributes, and their values, should not to be specified in all cases, so this message sometimes could be ignored.
Warning 02	Possibly incorrect output type <**output type**> for <**attr. value**>.
	It is possible explanation of Error 01, for cases where <**attr. value**> is a file name.

Table 1: Error-related messages

All error messages are related to SCT model inconsistencies and help in building of SCT based generators in a way similar to error messages in standard programming languages. Decomposition of error related messages among SCT model elements is shown in Figure 3.

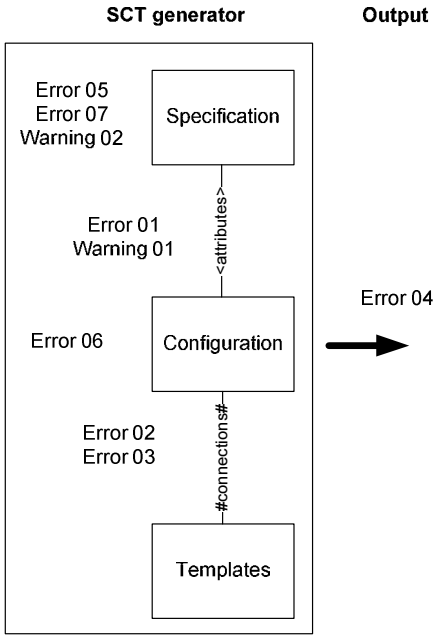

Figure 3. Decomposition of error-related messages

Some errors related messages are placed in the interconnection of two model elements because Specification and Configuration share Specification attributes, while Configuration and Templates share connections (Figure 3). The example or Error report is given in Figure 4.

SCT generating process report:
Error 03: Connection [**#attributes#**] in **class.metascript** was not found in Configuration. Template line number = 5
Error 03: Connection [**#attributes#**] in **class.metascript** was not found in Configuration. Template line number = 5
Output file: **output/linked_list.cpp** .
Generating process failed.
Errors = 2
Warnings = 0

Figure 4. Example of Error report

As shown in Figure 4, each error-related message occurs as many times as it appears in generation process.

6 An Example

Example is given for C++ linked lists generator. Generator[1] uses Specification, Configuration and set of Templates to produce C++ source code.

[1] Example is available at http://generators.foi.hr/SCT_error_messages_example/index.html

Specification of the example generator includes one output type (*main*), and two lists (*Students* and *Courses*) with their attributes:

```
OUTPUT:main
main:output/linked_list.cpp        (generated file)
list:Students                          (list name)
+key:student_id1              (subordinated attributes)
+field_int:student_id1
+field_char:surname
+field_char:name
list:Courses                           (list name)
+key:course_id                (subordinated attributes)
+field_int:course_id
+field_char:course_name
```

Possible erroneous situations that target Specification are as follows:

- Specification file can't be opened (Error 05)
- No one output type specified (Error 07)
- Incorrect output type (Warning 2; occurs if filename specified for attribute that is not an output type)

Configuration defines one basic template file (*main.metascript*) that is connected to first output type in Specification, and a list of configuration rules:

```
#1#,,main.metascript
#class_headings#,list,
       class_heading.metascript
#list#,list
#classes#,list,class.metascript
#attributes#,field_*,field_*.metascript
#field#,field_*
#attributes_entry#,field_*,
       attributes_entry.metascript
#attribute#,field_*
#attributes_print#,field_*,
       attributes_print.metascript
#head_allocation#,list,
       head_allocation.metasc ript
#menu#,list,menu.metascript
#cases#,list,cases.metascript
#key#,key
```

As could be seen in example, each Configuration rule consists from connection (in '#'-es), attribute name (as defined in Specification) and, optionally, a lower level template (meaning that connection should be replaced by whole template). Most of the erroneous in SCT based generators are connected with Configuration:

- Configuration file can't be opened (Error 06)
- No appropriate Configuration rule for Specification attribute (Error 01)
- Attribute defined in Configuration is not used in Specification (Warning 01; could be ignored if attribute is optional)
- Template that is specified in Configuration was not found or can't be opened (Error 02)
- Connection from Templates is not specified in Configuration (Error 03)

Templates consist from 14 textual files containing code templates. Main template is defined

in Configuration (*main.metascript*) giving the base structure of program code to be generated:

```
// C++ linked lists
// SCT generated example
#include <iostream>
using namespace std;
#class_headings#                    (class headings)
#classes#                           (class bodies)
int main(){
    int choice,arg;
    int counter1;
#head_allocation#       (allocation of lists headings)
    do{
        counter1 = 1;
#menu#                              (user options)
        cout << "\n  0. Exit";
        cout << "\n--------------------";
        cout << "\n-> Your choice: ";
        cin >> choice;
        counter1 = 1;
#cases#                             (function calls)
    }while(choice != 0);
    return 1;
}
```

Connections #class_headings#, #classes#, #head_allocation#, #menu# and #cases# have to be defined in Configuration (otherwise, Error 03 occurs).

Generated code includes code templates together with the Specification values:

```
// C++ linked lists
// SMG generated example
#include <iostream>
#include <string.h>
using namespace std;

class Students;                     (class headings)
class Courses;
class cStudents{                        (class body)
   public:
   cStudents *next;         (pointer to next element)

int student_id;                          (attributes)
char surname[40];
char name[40];
   . . . .
```

Generated code still can contain some syntax and logical errors, regardless to correct SCT model. In the example, these errors can be caused by following:

- insufficient Specification, e.g. attributes were not specified,
- unsafe names were used in Templates, e.g. variable names are same as Specification values and
- calls of functions prior to their declarations (caused by order in Specification).

7 Conclusion

Generative programming is a relative new approach in automatic program generation, and there are no many studies about error messaging systems in this approach.

The system of error-related messages, that is part of our SCT generation model, is presented in this paper. Unlike object oriented programming languages, errors in our system are related to the consistency of the SCT model. Warnings refer to possible inconsistencies, and developer should decide about their relevance. The use of such error-related messages systems makes building of source code generators easier.

Developed system of error-related messages was tested on the example of SCT based generator which produces program code in C++ that deals with linked lists.

References

[1] Apel S, Leich T, Saake G: **Aspectual Feature Modules**, IEEE Transactions on Software Engineering (TSE), 34(2), 2008, pp. 162-180.

[2] Bassett P: **Framing Software Reuse - Lessons From Real World**, Yourdon Press, Prentice Hall, 1997.

[3] Blair J, Batory D. **A Comparison of Generative Approaches: XVCL and GenVoca**, Technical Report, ftp://ftp.cs.utexas.edu/pub/predator/xvcl-compare.pdf

[4] Johnson RE, Foote B: **Designing Reusable Classes**, Journal of Object-Oriented Programming, 1(2), 1988, pp. 22-35.

[5] Lemaire C: **CodeWorker: A universal parsing tool & a source code generator**, http://codeworker.free.fr/

[6] Lovrenčić A, Konecki M, Orehovački T: **1957-2007: 50 Years of Higher Order Programming Languages**, Journal of Information and Organizational Sciences, 33(1), 2009, pp. 79-150.

[7] Magdalenić I, Radošević D, Skočir Z: **Dynamic Generation of Web Services for Data Retrieval Using Ontology**, Informatica, 20(3), 2009, pp. 397-416.

[8] Prehofer C: **Feature-Oriented Programming: A Fresh Look at Objects**. Lecture Notes in Computer Science, Springer-Verlag, Berlin, Germany, 1241, 1997. pp. 419-443.

[9] Radošević D, Magdalenić I: **Source Code Generator Based on Dynamic Frames**, Journal

of Information and Organizational Sciences, 2011, in press.

[10] Radošević D, Orehovački T: **An Analysis of Novice Compilation Behavior using Verificator**, Proceedings of the 33rd International Conference on Information Technology Interfaces, 27th – 30th June, Cavtat, Croatia, 2011, in press.

[11] Rhodes G: **Common Beginner C++ Programming Mistakes**. Valencia Community College, http://fd.valenciacc.edu/file/grhodes4/CommonBeginnerMistakes.pdf [20/04/2011]

[12] Rinker B: **Error Messages and Debugging in C++**. University of Idaho, Computer Science Department, 2002. http://www2.cs.uidaho.edu/~rinker/cs113/errors.pdf [20/04/2011]

[13] Rosenmüller M, Siegmund N, Saake G, Apel S: **Code Generation to Support Static and Dynamic Composition of Software Product Lines**, Proceedings of the 7th International Conference on Generative Programming and Component Engineering, 19th - 23th October, Nashville, Tennessee, USA, 2008, pp. 3-12.

[14] Teorey TJ, Ford AR: **Practical Debugging in C++**, Prentice Hall, 2001.

[15] Zhang H, Jarzabek S: **XVCL: a mechanism for handling variants in software product lines**, Science of Computer Programming, Elsevier, The Netherlands, 53(3), 2004, pp. 381-407.