UDC 004.43

O.I. Marchenko, V.V. Pogorelov

TO THE QUESTION OF CODE GENERATION FOR TEMPLATES OF MODELING LANGUAGES

Abstract. The research covers translation of modeling language templates to constructs of object-oriented programming languages. The main feature of modeling language templates is ability to define signature constraints on type parameter. Type constraints on template parameter in a target language are taken into account if present. The paper provides a brief overview of general techniques of templates translation and illustrates features of their implementation for translation of templates with signature constraints. Comparative analysis of the techniques was made and cases, when using each of the techniques is reasonable, are presented.

Keywords: template, modeling language, signature constraint, type constrain, object-oriented language.

Introduction

Nowadays, construction of complicated software is often based on Model-Driven Development (MDD) approach. The first stage of the approach is creation of a model that addresses the problem, the second stage is the model verification and the third one is code generation to a High-Level General-Purpose Programming Language (HLGPPL), usually to an Object-Oriented Language (OOL) [1]. As a rule the primary model description is implemented in modeling languages. Those languages have high expressive power and usually provide high-level abstract Template Data Types (TDTs) with specific features that are not common in HLGPPLs.

The experience in development of HLGPPL compilers gives us different techniques of templates translation. These techniques are well studied and show good results. Some of them could be adopted in order to translate modeling language TDTs.

Reworking the techniques, we should take into account some new criteria for evaluating the translation process that were not important in development of HLGPPL compilers. These criteria are code readability and maintainability (simplicity of further support).

Customers set often requirements to the developed according to MDD models to be supported both in the source modeling language and in the target HLGPPL. Thus, readability and maintainability play an important role in modern approaches of software development.

Analysis of recent achievements

Translation process of TDTs is well described for OOLs. The focus of the previous research is put on two main approaches of code generation for templates. The first approach implies OOL to OOL transformations that removes templates from the code. Such approach is used in some compilers of C++ programming language [2]. The other approach is based on translation from OOL to an intermediate language. Such approach is used in Java programming language compilers [3]. The slightly different idea of the second approach involves usage of type objects that are used to keep type information in run-time. This idea is implemented in most modern compilers of C# programming language [4].

During the past few decades, the concept of readability [5] and maintainability [6] measurements was developed and is suitable for evaluation of TDTs translation for modeling languages.

Purpose of the research

Research of the paper has the several objectives described below.

1. Analyzing the existing techniques of OOL templates translation. Adopting the techniques to generate OOL code for modeling language templates. The main feature of modeling language templates is ability to define signature constraints on a type parameter.

2. Developing combined techniques of translation that will allow to translate certain TDT of modeling language more efficiently.

3. Comparative analysis of the techniques and determination of the cases when using each of the technique is reasonable.

Main part

During the research, templates translation techniques of OOL were analyzed and 3 techniques for templates transformation of modeling languages are proposed. These techniques are:

1) full substitution of templates;

2) direct translation of templates;

3) combined translation of templates.

Before going into details of the techniques some characteristics of modeling language templates should be illustrated.

 ISSN 1562-9945

Signature constraints

A distinctive feature of modeling languages used in the research is ability to define signature constrains on a template parameter. The semantics of signature constraints is described below.

If signature constraint *C* is defined on a formal type parameter *T* of template *class TC <template T :? C>* then each actual type parameter of *TC* should contain a method with the prototype corresponding to *C*, otherwise it will be a compile error.

Figure 1 illustrates the semantics described above.

Signature constraints do not influence on the logic of the model. They used in order to perform automatic model verification and should be preserved in an OOL after translation if possible.

```
SIG_CONSTRAINT C = f2(int)-> double
CLASS TC <TEMPLATE T :? C> {
    v1 : T;
    m1 () -> double {
      return v1.f2(10);
    }
}
CLASS UT1 {
        f2(int i)-> double {
            return i*9.81;
        }
}
CLASS UT2 {
}
...
var1 : TC <UT1>;    <--- OK
var2 : TC <UT2>;    <--- ERROR
```

Fig. 1 – Semantics of signature constraints

Full substitution of templates

The idea of the technique is the following: if a translator encounters specialization of the template with actual parameter then new non-template data type is created. Each usage of a formal type parameter is replaced by an actual type parameter in all places where the formal type parameter was used. Thus, the transformation is performed on the modeling language layer.

Signature constrains on a type parameter are omitted after the transformation. They are only used for TDTs verification.

We might say that the model containing TDTs are replaced by equivalent model without TDTs. Figure 2 illustrates this technique.

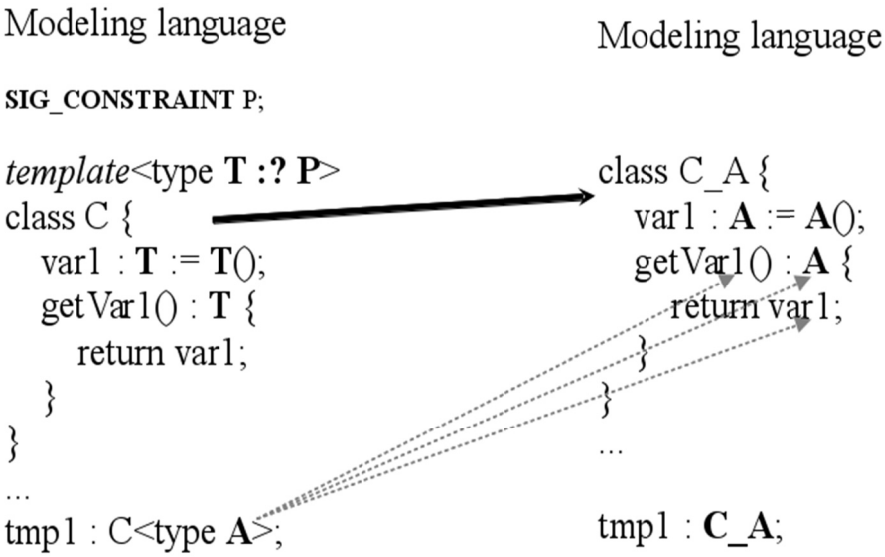

Fig. 2 – Full substitution of templates

This translation technique is faster than the others but leads to code blow-up. Consequently future modifications of the generated code is more difficult and compilation takes more time.

The technique is independent from a target language. This is useful in cases when one modeling language is translated into several target languages.

Direct translation of templates

According to this technique, all TDTs of a modeling language are translated into TDTs of an OOL with preservation of their semantics. Suchwise, semantics of the modeling language templates is implemented by means of OOL templates and other constructions presented in the OOL. Figure 3 illustrates the technique for translation from a modeling language to the OOL Java.

In order to implement semantics of the modeling language the signature constraints are translated into an interface that contains methods defined by the signature. Each class that is used as an actual template parameter implements the interface. Type constraint is put on a formal template parameter. Thus, semantics of templates of the modeling language is implemented for translation into Java.

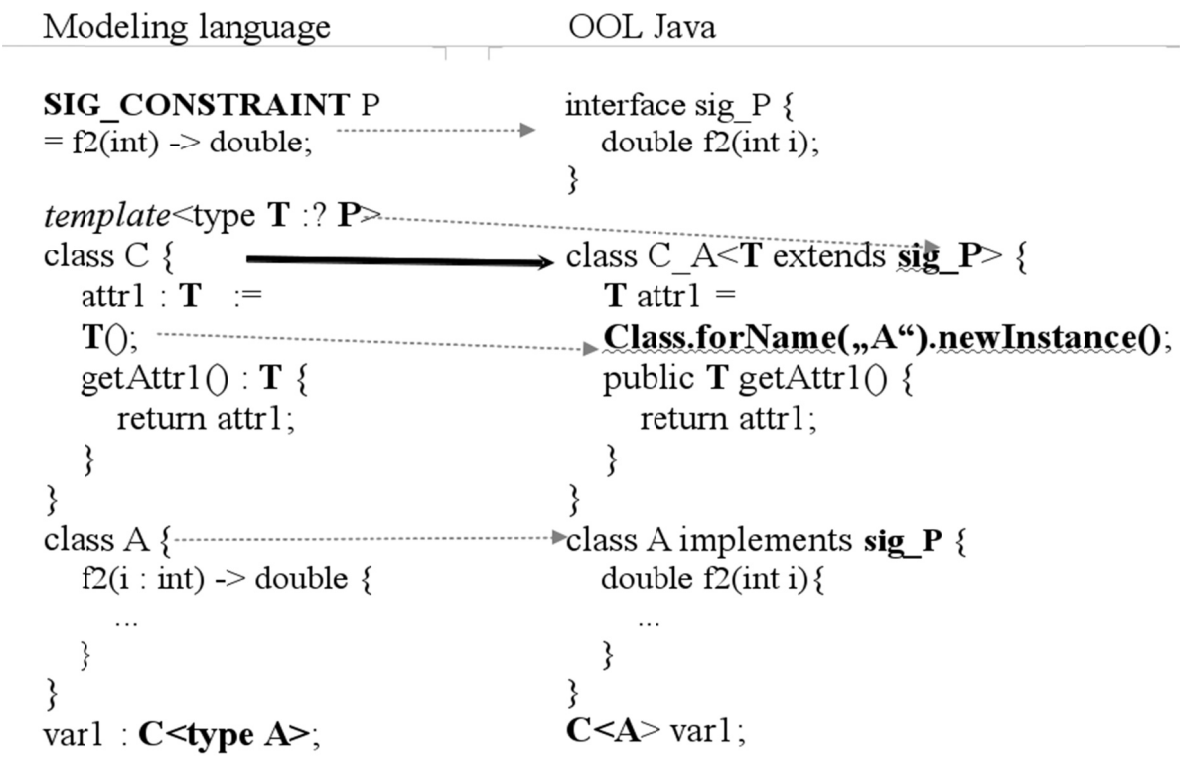

Fig. 3 – Direct translation

In most cases the technique is slower than previously discussed one, but allows generating less code with usage of constructs that are widely-used among programmers in a certain target language. This feature of the technique increases readability of the generated code and simplifies its further support.

Combined translation of templates

Combined techniques of templates translation are based on the idea that some templates are translated according to the technique of full submission and the others are translated according to the technique of direct translation. Thus, partial substitution of the modeling language templates is performed. The key point of the combined translation is determination of translation technique (substitution or direct translation) for certain TDT of a modeling language.

In order to determine translation of an arbitrary TDT of a modeling language the following combined technique is proposed:

1) if an actual template type parameter is a primitive type (int, double, char, etc.) it is reasonable to use full substitution of templates;

2) if an actual template type parameter is a heap-allocated data type, it is reasonable to use direct translation of templates;

3) if signature constraints are defined on a formal template type parameter then any of the techniques (full substitution and direct translation) can be used because they show practically the same results.

Comparative analysis of the translation techniques

In order to compare the transformation techniques the following evaluating critera were taken into account:

1) productivity;
2) compilation time;
3) size of generated code;
4) readability and maintainability.

Tab. 1

Results of comparative analysis of the translation techniques

Translation technique	**Productivity**	**Compilation time**	**Size of generated code**	**Readability and maintainability**
Full substitution	high	long	large	low
Direct translation	low	short	small	high
Combined translation	middle	middle	middle	Middle

As it can be noticed in the table 1, the first two techniques are opposite to each other. As it happens often, in this case «low of the lever» is applied: the more we gain in productivity the less we gain in other characteristics and otherwise. The proposed combined approach has middle values between the two others for all characteristics.

Choice of the exact translation technique depends on the problem that is solved by the developer. Flexible modeling language should provide an option for the user to choose what way is used to translate TDTs. As the default technique it is reasonable to use the proposed combined translation of templates that shows average results.

Conclusions

Analysis of OOL templates translation techniques that was done for the research allowed defining three similar techniques of modeling language templates translation. These techniques are studied and compared for modeling language that has ability to define signature constraints on template type parameter.

 ISSN 1562-9945

A developer of a translator for modeling language should take into account not only common characteristics as productivity, compilation time, size of generated code but also readability and maintainability.

The actual problem for further research is creating techniques for translation TDTs of modeling language into constructs of procedural programing languages like C.

REFERENCES

1. France R. Model-driven Development of Complex Software: A research Roadmap / R. France, B. Rumpe // FOSE '07 2007 Future of Software Engineering. – pp. 37-54.
2. Vandevoorde D., Josuttis N. – C++ Templates: The Complete Guide. – Addison-Wesley Professional, – pp. 175-200.
3. Java generics [electronic resource] / The Java TM Tutorials. Access mode: http://docs.oracle.com/javase/tutorial/extra/generics/index.html
4. Generics [electronic resource] / C# Programming Guide. Access mode: http://msdn.microsoft.com/en-us/library/512aeb7t.aspx
5. Jotgensen A.H. A Methodology for Measuring the Readability and Modifiability of Computer Programs / A.H. Jotgensen // BIT Numerical Mathematics. – Volume 20, Issue 4. – pp. 393-405.
6. Banker D.R. Software Complexity and Maintainability / R.D. Banker, S.M. Datar, D. Zweig // ICIS '89 Proceedings of the tenth international conference on Information Systems. – pp. 247-255.