Code Generation by Object Observation - an Evaluation*

Sebastian Geiger[1], Sebastian Kerekes[2], Michael Kraxner[3], and Martin Lackner[4]

[1] Favoritenstrasse 9-11, 1040 Wien
sbastig@gmx.net
MatrNr.:
[2] Favoritenstrasse 9-11, 1040 Wien
contact@sebastiankerekes.com
MatrNr.:
[3] Favoritenstrasse 9-11, 1040 Wien
michael.kraxner@gmail.com
MatrNr.: 9925916
[4] Favoritenstrasse 9-11, 1040 Wien
lackner.martin@gmail.com
MatrNr.: 0927551

Abstract. ... This paper evaluates possibilities and limitations of code generation by object observation. ...

* This work has been created in the context of the course "Advanced Model Engineering" (188952) in SS13.

Table of Contents

1 Introduction

In the last decade model driven engineering has devop ...

2 Semantics

Any (model) language is defined by its syntax, in this context a metamodel, and its semantics, the meaning of each element in the metamodel. We can distinguish between the static and the dynamic semantics. The static semantics defines structural properties of model elements in a metamodel (for further reading see [1] [5]), and the dynamic semantics regards to the execution of such models and its behavior [5]. For our work we are interested in the dynamic semantics.

Dynamic Semantics The dynamic or execution semantics defines the behavior under execution of models. This can be done in many ways, e.g., the semantics of UML is defined in OCL and natural language. A more sophisticated way which is seen very often in practice is either to use code generators to generate compilable source code, or to provide an interpreter which can process the models directly. But this approach is only feasible if the semantics is defined formally. Otherwise it would be too error prone, time-consuming and very costly if this would be done by manually studying and exploring the models [5].

We can distinguish between four different "semantic description styles" to define the semantics of a model formally [5] [1]:

Denotational
: This semantic provides a mapping between mathematical functions and model specific language constructs. E.g., it maps the keyword "-" to the subtraction function on natural numbers.

Axiomatic
: This semantic provides a mapping between logical rules and model specific language constructs. E.g., stating that two arguments of a function "-" produces an output with the same type of the two arguments.

Translational
: This semantic provides a mapping between the elements of a original model and model elements of a language where the semantics is already formally defined. The mapping is done by model transformation and can be used if the semantics of the source and the target models are closely correlated. Due to the fact, that the model transformation is done on the syntax level it is not guaranteed that the semantic of the final model behaves as desired. You also have to have an in-depth knowledge of both model languages and use multiple frameworks to achive good results.

Operational
: This semantics describes how model constructs are executed on an abstract machine. It describes the effect of the computation and how the computation is produced.

Another approach to define the semantics is to "weave behavior" into the abstract syntax by a meta-language. On this approach the behavior of an operation is specified inside the body of the operation on the meta level. E.g., Kermata uses this approach by extending its abstract meta-layer with an imperative action language [3].

The last approach we will mention here is to define semantics with "Rewrite Rules". A system consits of rewrite rules and each rewrite rule has a left- and a right-hand side. When executing a model the system takes an model element, looks if such a model element occurs on the left-hand side of a rule and replaces it with the right-hand side if such a configuration is found [3].

3 Code Generation by Observation vs. Compilation

As already mentioned in 2 there are two possible ways to execute models. Either you can generate code for your model, compile and execute it then, or you can make use of a model interpretor (execution engine), or you can combine these two methods. At the second possibility there must exist a kind of virtual machine which executes the model and fires events which can be catched, and therefor code can be generated by observating these events.

Interpretation On interpretation, instance code is written for each model element which has be done once by the developer of the interpretation engine. Whenever a model is executed by the interpretation engine it analyzes the given model element at run-time and chooses the right instance implementation which will be executed then. E.g., if you have an user defined EClass in your model the interpretation engine stores this class in a hashmap with all its properties [4] [2].

Advantages of Model Interpretation

Here are some of the advantages of model interpretation [4].

Enable Debugging

When models are interpreted it is possible for developers to debug thru the models. This can be very helpfull on complex models.

Faster Changes

Models can be faster changed because there is no need to regenerate, rebuild, retest and redeploy the whole model.

Changes at Runtime

Models can be changed during runtime without stopping the running application.

Compilation If you are using code generators you have to write a compiler component which can deal with any model element of your meta-model. You have to generate code for every model element and compile it for a specific model. At run-time the model element logic is inside the instance code. E.g., if you have an user defined EClass in your model you can use a template based code generation approach where for each EClass in your model a java class is generated and filled with the corresponding properties [4] [2].

Advantages of Compilation

Here are some of the advantages of compilation [4].

Additional Checks

Code which will be generated need to be compiled. The compiler will check the code for errors which is not be done by an interpretor.

Architecture Independence

If you are using an interpretor you have always to follow your own architecture. When using code generation you can suite the architecture of the code for the customer needs.

Easier to Understand

Generated code is easier to understand, you can follow the code line by line and understand the behavior exactly. On model interpretation you have to understand the implementation of the interpretor and also the semantics of the model.

Resume It depends on the use case, on the developers skills and of the domain if you are using an interpreator or code generators. There are advantages and disadvantages on both approaches. In the next chapters we will take a closer look on both approaches and discuss the advantages and disadvantages in detail.

4 Related work

recent developments . . . Executable UML,

4.1 fUML

4.2 xMOF

4.3 xtend

XXX really ???

5 Code Generation with xtend

6 Code Generation by Object Observation

7 Evaluation

References

1. A. Cuccuru, C.. Mraidha, F. Terrier, S. Gerard. Enhancing UML Extensions with Operational Semantics. *Lecture Notes in Computer Science Volume 4735* (2007), 271–285.
2. A. Kirshin, D. Moshkovich,, A. Hartman. A UML Simulator Based on a Generic Model Execution Engine. *Lecture Notes in Computer Science Volume 4364* (2007), 324–326.
3. B.R. Bryant, J. Gray, M. Mernik, P.J. Clarke, R.B. France, G. Karsai. Challenges and directions in formalizing the semantics of modeling languages. *Computer Science and Information Systems 2011 Volume 8, Issue 2* (2011), 225–253.
4. Johan den Haan. http://www.theenterprisearchitect.eu/archive/2010/06/28/model-driven-development-code-generation-or-model-interpretation. Accessed: 2013-05-31.
5. S. Andova, M.G.J. van den Brand, L.J.P. Engelen, T. Verhoeff. MDE Basics with a DSL Focus. *Lecture Notes in Computer Science Volume 7320* (2012), 21–57.