

REVIEW OF DRAWBACKS OF EXISTING TOOLS TRANSFORMING PLATFORM SPECIFIC MODELS TO CODE

Anton Shyrokykh

Abstract: *AGILE as way of developing software have great support in development community. Using of software models as abstractions could increase software accuracy and documentation quality, but does not apply with AGILE principles, due to high effort. Model to code transformation via model driven development has a potential to resolve high effort problem and make modeling align better with AGILE principles.*

As reliability and flexibility of code generation tools is very important characteristics, paper is devoted to investigation of code generation approach that can be supported by most widespread used tools that allow to obtain code skeletons from UML class diagrams.

Keywords: *Code generation, Model driven development, Class diagram, Object-Oriented Programming.*

Introduction

According to AGILE manifesto and principles, teams should at regular intervals reflects on how to become more effective, pay continuous attention to technical excellence and good design, deliver working software frequently and welcome changing requirements (Manifesto for Agile Software Development). Developing models as additional abstractions of software help teams communicate better, detect high level errors early and increase quality of software under development. But models as additional artifacts require additional efforts to produce, support and to keep them up to date. This makes this powerful tool

almost not applicable in conditions of AGILE software development process with often requirements changing. Also it is not compatible with principle "Working software over comprehensive documentation".

Model to code automatic or semi-automatic transformation process using model driven development approach helps to solve most of the defined problems. Model to code transformation helps to reduce development effort on coding, as most of the code is generated from the model, and also helps to support connection of models to code, making it simple to keep up to date. Model driven development makes model to code transformation compatible with AGILE manifesto and principles.

There exists several approaches to implement code generation in model driven development. Some of the approaches are: modeling with support of transformation language, direct model manipulation, intermediate representation approach. Different approaches allows to choose and balance some key attributes, as flexibility of code generation, effort to perform transformation, transformation accuracy etc. Each of approaches has representative technology, but the most widespread used tools use visual notation of the model and use direct model manipulation approach to perform transformation. In most cases transformation implementation is either hidden, or not extensible. As a result, produced code needs some effort comparing with manual coding or more accurate but more expensive transformation approaches, as using of transformation language.

Code generation is a key activity in Model-Driven Development approach. The aim of UML diagrams models not to be just models, but to become a code that corresponds to processes or software structure. Nowadays there are some

foundation in papers divided to investigation of analytical Model transformation aspects.

Review of Papers

Model-to model generative approaches work with graph representation of models. In order to obtain resulting models For example, in the paper (Eickhoff C at. el.,2019) it is proposed to use reachability graph computation for Eclipse Model frameworks (EMF). Such models are based on model transformations provided as simple Java lambdas. Thus, it is possible to test and check model and controller synthesis on EMF model using model transformations. Proposed tool enables sharing of common sub-models between multiple states thus providing a memory efficient encoding of large reachability graphs. Model attributes can be modified. It is model to model transformation tool based on graphs. Using `java.lang.reflect` authors could achieve this reflective access for general Java objects can be archived.

Other direction of software development artifacts producing are related to web development sphere.

Paper (Laaz, N. at. el, 2019) presents an approach based on MDA for the user interfaces development of SWAs. A Meta Model for Html5 is defined. Then the transformation engine that allows the automatic generation of the output models is developed. These models represent an input to a Model to Text generator that give an almost complete web pages ready to be deployed, focusing on the graphical aspect of the application on one hand, and the annotations and the user event handling on the other. The main contribution in the proposed approach is the generation of ontologies, as well as annotated web pages from one IFML diagram, besides the abstraction of technical details. By using this models driven method, the user interfaces of semantic web applications can be

easily generated without having to know all the technical specification of the execution platform.

Fundamentals of Model transformation approaches used in AGILE approach are investigated in paper (Chebanyuk & Markov, 2016). Authors consider Model-Driven Engineering promises and analyze research state of art of different Model transformation activities.

Tasks, needed to be automated, for increasing effectiveness of different software development operations are summarized in paper (Shane Sendall and Wojtek Kozaczynski). Authors formulate requirements from software model transformation language, which supports model-driven software development are formulated. This paper makes strong contribution to systematic review of model transformation requirements and also to classification of architectural approaches to transformations.

In order to answer to automatic code producing challenge many commercial and noncommercial tools support such function as codegeneration. From the other hand, one can prove, that namely such Model-Driven Development foundations are popular and used often, which can be implemented by many developers to fell concepts of Code Generation.

Investigation of Codegeration Features of tools

Most widespread used tools use visual notation of the model and use direct model manipulation approach to perform transformation. Visual notation in most cases corresponds to UML Class diagram, which is considered to be a standard notation.

The general drawback of that approach, that transformation implementation is either hidden, or not extensible. As a result, produced code needs some effort comparing with manual coding or more accurate but more expensive transformation approaches, as using of transformation language. In addition each particular tool can have own drawbacks and limitations.

Several widespread used codegeneration tools from software models were chosen to investigate their codegeneration drawbacks.

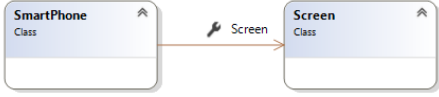
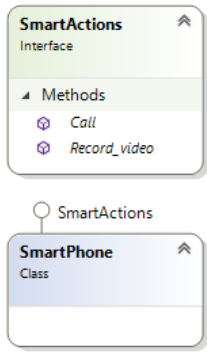
Visual Studio Class Designer

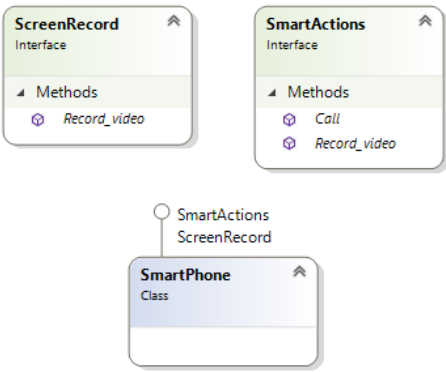
Class Designer is a part of Microsoft Visual Studio IDE. Source code of IDE Class Designer is not public and does not support extensions.

Class Designer has the next features:

- 1 Design: Edit your project's code by editing the class diagram. Add new elements and delete unwanted ones. Your changes are reflected in code.
- 2 Visualize: Understand your project's structure by viewing the classes in your project on a diagram. Customize your diagram so that you can focus on the project details that you care about the most. Save your diagram to use later for demonstration or documentation.
- 3 Refactor: Override methods, rename identifiers, refactor parameters, and implement interfaces and abstract classes (Microsoft, 2019).

Table 1. Visual studio drawbacks

Bug code	Drawing or explanation	Textual description
VS-B1		<p>There are two classes on class diagram and inside of one class declare a property named as other class is declared. You obtain an error message from Visual Studio environment that this data type already is used. But, according to design rule it is expected the generation of composition relationship between two classes. You cannot do it, because Visual Studio Environment do not contain composition relation.</p>
VS-B2		<p>You have an interface and a class. You design inheritance relationship between interface and class inside of the code generated class it is expected to see interface methods with public modifier ready for overloading. Really such a situation is not happened. Only inheritance is appeared.</p>

Bug code	Drawing or explanation	Textual description
VS-B3	 <p>The diagram shows three UML elements: two interfaces and one class. 'ScreenRecord' is an interface with a 'Record_video' method. 'SmartActions' is another interface with 'Call' and 'Record_video' methods. 'SmartPhone' is a class that inherits from both 'SmartActions' and 'ScreenRecord', as indicated by the solid line with hollow triangle heads pointing to the interfaces.</p>	<p>Then it is advisable when several interfaces have method with the same signature and when they are inherited by the same class to point explicit inheritance starting from the name of the interface.</p>
VS-B4	<p>Designer can add to class diagram as many components (classes and interfaces) as he considers that it is needed.</p>	<p>There is no mechanism of verification class diagram to correspondence of cognitive design principles</p>

Eclipse codegeneration tool

The EMF project is a modeling framework and code generation facility for building tools and other applications based on a structured data model. From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, along with a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor (Eclipse, 2018).

The core EMF framework includes a meta model (Ecore) for describing models and runtime support for the models including change notification, persistence support with default XMI serialization, and a very efficient reflective API for manipulating EMF objects generically.

Three levels of code generation are supported:

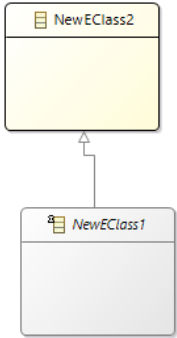
Model - provides Java interfaces and implementation classes for all the classes in the model, plus a factory and package (meta data) implementation class.

Adapters - generates implementation classes (called ItemProviders) that adapt the model classes for editing and display.

Editor - produces a properly structured editor that conforms to the recommended style for Eclipse EMF model editors and serves as a starting point from which to start customizing.

All generators support regeneration of code while preserving user modifications. The generators can be invoked either through the GUI or headless from a command line.

Table 2. Eclipse drawbacks

Bug code	Drawing or explanation	Textual description
E-B1	Repeats VS-B2 behavior	
E-B2	Repeats VS-B3 behavior	
E-B3	Repeats VS-B4 behavior	
E-B4	 <pre> classDiagram class NewEClass1 class NewEClass2 NewEClass1 -- > NewEClass2 </pre>	Generated code shows that interface (namely abstract class) inherits NewClass2

NClass

NClass is a free open source tool to create UML class diagrams with C# and Java language support. The user interface is designed to be simple and user-friendly for easy and fast development. Initially the project was developed by Balasz Tihanyi on sourceforge.

Existing features:

- 4 C# and Java support with many language specific elements;
- 5 Simple and easy to use user interface;
- 6 Inline class editors with syntactic parsers for easy and fast editing;
- 7 Source code generation;
- 8 Reverse engineering from .NET assemblies;
- 9 Printing / saving to image;

Table 3. NClass drawbacks

Bug code	Drawing or explanation	Textual description
NC-B1	Repeats VS-B1 behavior	
NC-B2	Feature is absent	Design limitation – to establish a generalization relation between interface and a class
NC-B3	Repeats E-B4 behavior	

Codegeneration efforts overview

Despite all reviewed drawbacks of chosen approach, codegeneration saves much effort in programming. But auto generated code is not ready for use and needs to be completed manually. Significant effort could be taken to perform correspondence of code structures with complex relations between classes, for example to design patterns.

Consider class diagram of design pattern "Proxy" and estimate efforts needed to be taken while completing codegeneration results.

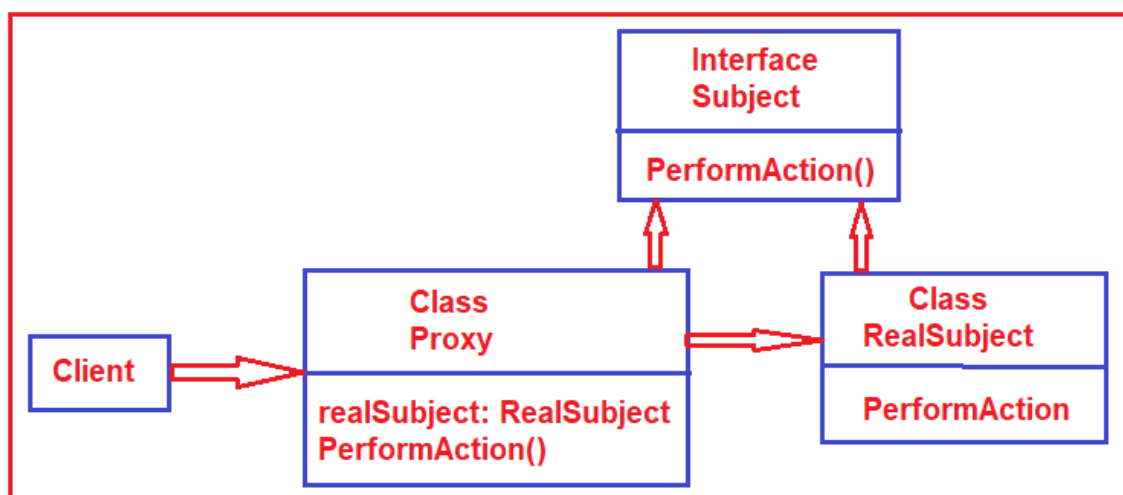


Figure 1. Class diagram of design pattern "Proxy"

Figure is taken from <https://dotnettutorials.net/lesson/proxy-design-pattern/>

Two variants of code, namely code generated by Visual Studio automatically and code designed by software developer are given in the table 4.

Table 4. Differences between generated code and such a one that is designed by developer

<p>Code skeleton created by developer</p> <p>Code is taken from</p> <p>https://dotnettutorials.net/lesson/proxy-design-pattern/</p>	<p>Code skeleton, created by visual studio codegeneration tool</p>
<pre> public class Employee { } public interface ISharedFolder { void PerformRWOperations(); } public class SharedFolder : ISharedFolder { public void PerformRWOperations() { } } class SharedFolderProxy : ISharedFolder </pre>	<pre> public interface Subject { void PerformOperation(); } public class Proxy : Subject { public RealSubject RealSubject { get => default(RealSubject); set { } } } public class RealSubject : Subject { </pre>

```
{  
    private ISharedFolder folder;  
    private Employee employee;  
  
    public void PerformRWOperations()  
    {  
        ...  
        folder = new SharedFolder();  
        folder.PerformRWOperations();  
        ...  
    }  
}
```

As result of automatic code generation, interface and classes were created. Work items to complete manually includes:

1. create private RealSubject field in Proxy class and complete implementation of getter and setter
2. define PerformOperation method from Subject interface in RealSubject class and complete implementation
3. define PerformOperation method from Subject interface in Proxy class and complete implementation with private RealSubject instance call
4. complete PerformOperation method in Proxy class with additional Proxy logic

Conclusion

Paper proposes investigation of codegeneration tools. Ideas of investigation is the next: in AGILE approach codegeneration – is operation that must reduce of developer effort. Different software development tools are better to use in different stack of technologies. Number of efforts, spent to refine obtained skeleton of code, influences to general codegeneration time. Practically, all codegeneration tools require additional efforts after codegeneration. These efforts are spent to perform a correspondence of code structure with complex relations between classes, for example to design patterns.

Further research

Such an investigation becomes a start point to perform further investigations, that consists from the next steps:

Grounding of analytical approach or fundamentals to describe transformation rules (model to code transformation) removing limitation of existing codegeneration techniques

Representation of architectural schemas of transformation tool

Analyzing bug tables (table 1-3) representing a set of transformation rules in terms of chosen analytical approach

Designing an architectural solution of newly designed transformation tool

Bibliography

Chebanyuk E. & Markov Kr. (2016) Model of problem domain “Model-driven architecture formal methods and approaches”. International Journal “Information Content and Processing”, Vol. 22, Number 4, 2016, 202-222

(GitHub, 2015) <https://github.com/gbaychev/NClass>

(Eickhoff, C. et al., 2019) Eickhoff, C., Lange, M., Raesch, S. and Zündorf, A. EMFeR: Model Checking for Object Oriented (EMF) Models. DOI: 10.5220/0007681605110518 In Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2019), pages 511-518 ISBN: 978-989-758-358-2

(Shane Sendall and Wojtek Kozaczynski) Model Transformation – the Heart and Soul of Model-Driven Software Development

(Manifesto for Agile Software Development) <https://agilemanifesto.org/iso/en/manifesto.html>

(Eclipse, 2018) <https://www.eclipse.org/modeling/emf/>

(Laaz N., 2019) Laaz, N. and Mbarki, S. OntoIFML: Automatic Generation of Annotated Web Pages from IFML and Ontologies using the MDA Approach: A Case Study of an EMR Management Application. DOI: 10.5220/0007402203530361 In Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2019), pages 353-361 ISBN: 978-989-758-358-2

(Microsoft, 2019) <https://docs.microsoft.com/en-us/visualstudio/ide/class-designer/designing-and-viewing-classes-and-types?view=vs-2019>

Authors' Information

Anton Shyrokykh – National Aviation University, Faculty of Cybersecurity, computer and software engineering, graduate student. Kiev, Ukraine; e-mail: anton.black777@gmail.com

Major Fields of Scientific Research: Model-Driven Development, Distributed long-living transactions