UDC 004.4’22

O. Marchenko, M. Olenin, A. Shyrokykh
(National Aviation University, Ukraine)

Overview of using Microsoft .NET Compiler platform SDK in model transformation activities within Model-Driven Development

In modern software development world Model Driven Development (MDD) approach considered as good practice in software development. MDD helps to solve distinct software engineering problems, such as reusing and connecting artifacts on different stages of development life cycle, increasing accuracy and interconnection.

Introduction

The model-driven approach has a potential to increase development productivity and quality by describing important aspects of a solution with more human-friendly abstractions and by generating common application fragments with templates. [2]. The essential goal of MDA is to derive value from models and modeling that helps us deal with the complexity and interdependence of complex systems. Deriving artifacts and implementations from models may be fully or partially automated. Automation reduces the time and cost of realizing a design, reduces the time and cost for changes and maintenance and produces results that ensure consistency across all of the derived artifacts. Transformation deals with producing different models, viewpoints, or artifacts from a model based on a transformation pattern. In general, transformation can be used to produce one representation from another, or to cross levels of abstraction or architectural layers. [3].

MDA specifies three default models of a system. These models can perhaps more accurately be described as layers of abstraction, since within each of these three layers a set of models can be constructed, each one corresponding to a more focused viewpoint of the system (user interface, information, engineering, architecture, etc.).

A Computation Independent Model (CIM) is also often referred to as a business or domain model because it uses a vocabulary that is familiar to the subject matter experts (SMEs). It presents exactly what the system is expected to do, but hides all information technology related specifications to remain independent of how that system will be (or currently is) implemented.

A Platform Independent Model (PIM) exhibits a sufficient degree of independence so as to enable its mapping to one or more platforms. This is commonly achieved by defining a set of services in a way that abstracts out technical details. Other models then specify a realization of these services in a platform specific manner.

A Platform Specific Model (PSM) combines the specifications in the PIM with the details required to stipulate how a system uses a particular type of platform. If the PSM does not include all of the details necessary to produce an implementation of that platform it is considered abstract (meaning that it relies on other explicit or implicit models which do contain the necessary details). [4].

MDD approach with using common standards as UML for models representation, one of transformation language standards provided as QVT, and a modeling tool with support of above standards require continuous efforts to introduce, adopt and support in the organization [8].

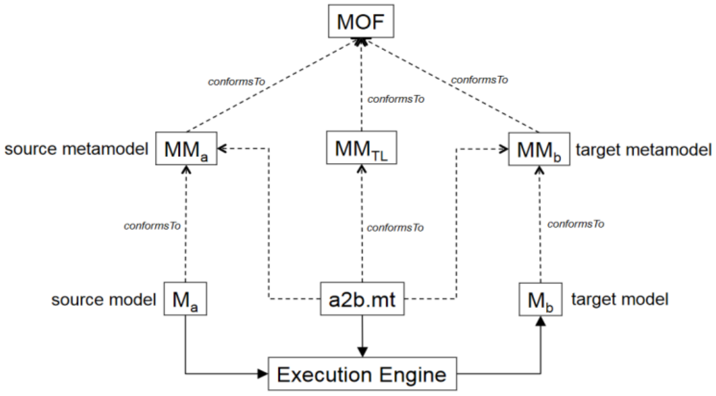

Fig. 1. Common model-to-model transformation pattern

Efforts for introducing MDD includes:

- High qualified stuff to work with MDD, as architects, special engineers with standards knowledge

- Knowledge base and experience on successful MDD practices, templates, domain model component libraries, pattern models etc.

- Software tools with support of chosen modeling and transformation standards. Also, variety of transformation operations is limited by supported features of chosen practical tools.[1].

- Changing organization structure to more fit to MDD, as some efforts will be moved from development software life cycle stage to design and modeling. Organization structure changes were described at [5].

The above could also limit the contribution of architecture and modeling teams to model-to-model and model-to-artifact transformation process.

In [2] architectural approaches to model transformations were classified and direct model manipulation could be considered as an option to solve some of the drawbacks of standard MDD approach along with better reusing development resources through the organization.

The contribution of this paper is describing of an option of implementing MDD with platform-specific development tools, aligning with target PSM platform for leveraging PIM to PSM transformations through direct model manipulation approach with using Microsoft Roslyn compiler platform and Roslyn API for Microsoft .NET Platform.

Related papers

Tasks, needed to be automated, for increasing effectiveness of different software development operations are summarized in paper [2]. 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.

Test Driven Development definition and principles were defined and described in book [9]. This books made a great contribution into collecting and generalizing knowledge about unit and integration testing and TDD approach.

Tasks and challenges

Task: provide an overview of possibility to use Microsoft Roslyn in PIM to PSM transformation for applying direct model manipulation approach. Provide general overview of what drawbacks in standard MDD approach could be solved and what are the benefits. Overview the correspondence of proposed model transformation language to requirements, stated by [2]. Explore and describe additional benefits of using target platform tools and languages in model transformation for overall transformation and development process.

Model to artifact transformation applying direct model manipulation using .NET Compiler platform SDK

Standard MDD approach including some well known drawbacks. As the primary one there is a high efforts (costs, resources, people) on introducing, using and support especially on the starting point of applying MDD. As was described in [6], modeling requires additional training and tools, a corresponding investment in time, money and effort is needed—not at the time of toil, but early in a project’s development life cycle. Many will argue that the resistance to modeling software is more cultural than anything else. Also the side effect of the MDD and moving the most of the effort from writing code to modeling is that the development team is less involved in modeling activities/

In order to represent code transformation using object oriented programming language and apply direct model manipulation approach, it is required to represent source model (let it be PIM represented in UML component diagram) in object oriented form, target model (let it be source code) via object oriented form, set of objects and API for operations with them and transformations itself via programming language constructions, utility classes and methods, which will describe transformation process itself.

Taking UML diagrams as source model means that they can be stored in XMI format (via XML messages), which is consumable via .NET Framework default serialization classes. C# development environment allows to generate object-oriented representation and classes from XML automatically, via “Special Paste” option in Visual studio IDE. Representing UML as objects allow manipulating them in a manner, well known by .NET developers.

Compilers build a detailed model of application code as they validate the syntax and semantics of that code. They use this model to build the executable output from the source code [7].

For PIM to PSM transformations purpose only syntax model of the code is needed. The .NET Compiler Platform SDK provides access to this model. .NET Compiler Platform SDK exposes object models for each stage of compiler work, as well as syntax tree object model to represent the source code. The set of classes exposed by Roslyn API provide a full set of instruments for representing any C# of VB written code, so is, Roslyn API could perform a role of meta model for PSM. Runtime snapshot of Roslyn API objects could represent a PSM itself. Also worth noting that Roslyn model has 1:1 correspondence with C# source code and has transformation operations already included in SDK. As an additional benefit of using C# and Roslyn as transformation framework is a strongly typed nature of C# and Roslyn library, which gives compile-time validation checking of defined transformation rules, saving from generating invalid output, and also validation extension ability of using and writing Roslyn based code analyzers for transformation code.

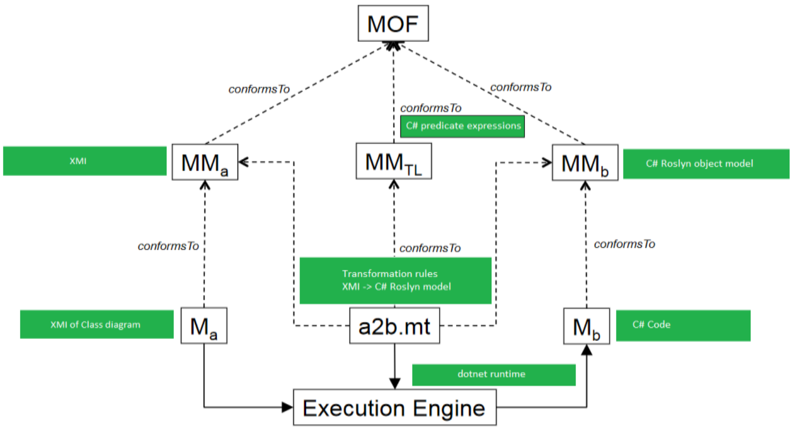

Fig. 2. Adapted common model-to-model transformation pattern with using C# as transformation language and Roslyn as target metamodel

As a transformation operations language for .NET platform C# and VB could be considered.

So representing source model, target model and transformation operations via .NET Compiler Platform and C# language is possible. Also worth noting that using object oriented language opens great abilities to form transformation frameworks, class libraries of predefined domain classes, extend them via inheritance and polymorphic mechanisms, available in C#. This could partially compensate a drawback of direct model manipulation approach, described in [2], that since the programming languages are “general-purpose”, they lack suitable high-level abstractions for specifying transformations.

As for requirements to transformation language, described in [2], usage of C# and .NET Compiler platform SDK and C# as transformation language doesn’t match requirement for representing transformation in graphical form.

Compliant recommendations are:

1. Proposed transformation language is executable either as standalone command line application, or as a part of more complex programs.

2. It is implementable in efficient way, clear and understandable from the developers perspective, because of reusing development environment and tools.

3. Clearly differentiate the description of source model selection rules from the rules producing target model, as the rules are described using different set of classes.

4. It can provide a means to define the conditions, under which the transformation is allowed to execute, for example valid XMI as input.

5. As for transformations combinations, conditional selection and transformation repetition requirements, proposed transformation process representation contains several extension points to provide additional transformations and filtering. After XMI is deserialized, it's object representation could be additionally transformed into intermediary representation, supporting two-way elements navigation or model cleanup. After Roslyn object model was created, it is available for modifications and refactoring by means of Roslyn-based code fixes. Roslyn object model could also be complemented from other models or rules, for example code files attributions with license info.

Conclusions

Provided overview of using direct model manipulation approach with .NET compiler platform SDK shows that this approach could compete with those proposed by dedicated modeling tools and transformation languages, solving the problem of involving development team into modeling activities and potentially decreasing the efforts of introducing/supporting MDD in production. Among advantages of using target platform C# language into model transformation activities are:

1. Involving development team resources to model transformation activities
2. High level of flexibility of direct model transformation approach
3. Reducing variety of tools, technologies and specialists involved in project
4. Adoption of modern software development techniques as TDD, unit testing and refactoring

Further work

Develop Roslyn-based transformation PIM to PSM framework, including tooling set for interpreting UML class, component and package diagrams via XML, class library with common domain models and predefined PIM to PSM transformations, framework extension points. Investigate additional possibilities to extend or reuse Roslyn-based framework, including auto-generating unit tests C# projects based on PIM, code analyzers, code-fixes and solution item templates.

References

1. Chebanyuk O. Designing of Software Model to Model Transformation Language. - International Journal of Computers. Volume 3, 2018.

2. S.Sendall, W. Kozaczynski. Model Transformation – the Heart and Soul of Model-Driven Software Development. - IEEE Software October 2003

3. Object Management Group Model Driven Architecture (MDA) MDA Guide rev. 2.0

4. The Fast Guid to Model Driven Architecture. The basics of Model Driven Architecture. - Cephas Consulting Corp January 2006

5. Anthony J. Lattanze. The Architecture Centric Development Method. February 2005

6. IBM The Value of Modeling. A technical discussion of software modeling. June 2004

7. Understand the .NET Compiler Platform SDK model. https://docs.microsoft.com/en-us/dotnet/csharp/roslyn-sdk

8. Model-Driven Software Engineering In Practice. Morgan & Claypool September 2012, 182 pages

9. Vladimir Khorikov. Unit testing. Principles, Practices and Patterns. Manning Publications, 2019. — 288 p.