Model-Driven Engineering Support for Building C# Applications

Anna Derezińska, Przemysław Ołtarzewski
Institute of Computer Science, Warsaw University of Technology
A.Derezinska@ii.pw.edu.pl

Abstract – **Realization of Model-Driven Engineering (MDE) vision of software development requires a comprehensive and user-friendly tool support. This paper presents a UML-based approach for building trustful C# applications. UML models are refined using profiles for assigning class model elements to C# concepts and to elements of implementation project. Stereotyped elements are verified on life and during model to code transformation in order to prevent creation of an incorrect code. The Transform OCL Fragments into C# system (T.O.F.I.C.) was created as a feature of the Eclipse environment. The system extends the IBM Rational Software Architect tool.**

Keywords: MDA, UML, OCL, C#, model transformation, code generation, Eclipse, IBM Rational Software Architect

I. Introduction

Modeling at various levels of abstraction is viewed as an essential part of model-driven engineering approaches (MDE) [1]. The availability of good modeling tools can help to achieve this goal.

Model transformations are becoming more and more important in software development, particularly as a part of MDE. There are different kinds of model transformations [2,3]. According to the taxonomy given by Mens and van Gorp in [3] code generation can be treated as a vertical, exogenous model transformation. In a vertical transformation the source and target models reside at different abstraction levels. Exogenous transformations are transformations between models using different notations.

However, inconsistencies in models could make automatic code generation hardly possible negating MDE idea [4]. Moreover, only 15.6% of different solutions for UML model consistency management are suitably integrated with a CASE tool [4]. Therefore, developing a model-to-code transformation, we should assist a user in precise mapping between concepts of two abstraction levels and in verification of the model refinement and code generation process.

To solve these problems a Transform OCL Fragments into C# system (T.O.F.I.C.) was designed and implemented [5]. The approach is developed in accordance to Model Driven Architecture (MDA) [6,7], which is the leading MDE proposal. A UML model can be refined towards a model consistent to a given programming language, and then transformed into a target code. Code specific features are encapsulated in UML profiles [8].

End users may also want to enhance their models with more precise specifications, like class invariants, operations pre- and post-conditions. The T.O.F.I.C. system supports this features via transformation of a class model combined with OCL queries [9].

The system was implemented in full integration with an existing CASE tool - IBM Rational Software Architect [10]. The extension mechanisms of RSA were used as well as extensions of the standard development platform Eclipse Modeling Framework (EMF) [11,12] on that RSA is based. The current T.O.F.I.C. system version facilitates C# code generation from a refined UML class model and OCL constraints, and is a working prototype.

The rest of the paper is structured as follows: Section 2 describes briefly background and principles of the approach. Section 3 presents tool requirements. Technology and system architecture are discussed in Section 4. In Section 5 we present the basic steps of model-driven development using the tool. Finally, we make conclusions and suggest some future work.

II. MDA vision at work

This section describes basic MDA models and their relations (A) and other background of the approach (B). Next, the main principles of realization of MDA ideas are presented (C).

A. *MDA basic concepts*

The Object Management Group (OMG) developed a variety of standards focused on object-oriented systems and system modeling. OMG standards provide a foundation for Model Driven Architecture (MDA) [6]. MDA is an architectural framework that stimulates a software development approach by different detailed specifications. The key features are separation of different abstraction levels extracting conceptual models and descriptions based on virtual or specific technological platforms, as well as, integration of different solutions.

MDA recognizes several fundamental abstract levels, so called viewpoints, with corresponding models. Computation Independent Viewpoint deals with algorithmic issues and system requirements, regardless of structures and system processing. At the next abstract levels Platform Independent Models (PIMs) are developed. Their specification is relative to a set of features of different platforms and should be these features independent. Characteristics of a technological platform can be summarized building a Platform Description Model (PDM). Merging of the appropriate PIM with one of available PDMs gives a Platform Specific Model (PSM). It expresses the PIM model in terms of details of the target platform. The models can be further transformed to lower abstract levels and merged with another platform depended details. Relations between models are shown in Fig. 1.

B. *Background*

There are two main directions of research dealing with execution of models. In the first one, a model is run directly using a kind of virtual machine supporting abstracts of a modeling notation. The OMG developed Foundation Subset for Executable UML Models (FUML) [13] that defines a basic virtual machine for the subset of the Unified Modeling Language. The second direction tends to transformation of models into executable languages, mostly the known general purpose languages. In the later case, we can use the existing environments supporting further program development.

Numerous tools address the problem of model to code transformation for the commonly used languages, like Java, C++, C#, e.g. [10]. They mainly take only class models as a source, but code generation from behavioral models, especially statecharts, is also supported [14]. However, the most of the model to code transformations deal with general, may be incomplete, class models. Therefore the models and their transformations cannot be precisely verified, and an ambiguous or incorrect code can be generated. Even restricting to a class model, modeling concepts are comprehend and transformed in various ways, as for example an association [15]. PIM models that are the targets of an automatic transformation process [16] require also further user-controlled refinement in order to conform to platform dependant requirements.

One of MDE-based approaches is a code generation process that introduces an intermediate model level between UML models and code generations [17]. The simplified meta-model of the intermediate level is intended to simplify further transformation to a desired programming language.

A solution that provides the closest functionality to the proposed approach is Rational Modeling Extension for .NET [18]. However its usage has several limitations, including possibility of full modeling C# concepts or modification capabilities. The main drawback is weakness of validation facilities that are the crucial requirement for refinement of inconsistent models and generation of a trustful code.

C. *Principles of the approach*

The approach presented in this paper is realized according to MDA concepts (Fig. 1). It is based on gradual model refinement and model to code transformation. The main idea was the enhancement of general purpose UML models with features targeted to a selected programming language (C# in the discussed case). A refined UML model (Fig. 2) was a source for further model to code transformation. Therefore the model can be verified before and during transformation in order to preserve consistency between input model and the generated code and to check correctness of the output.

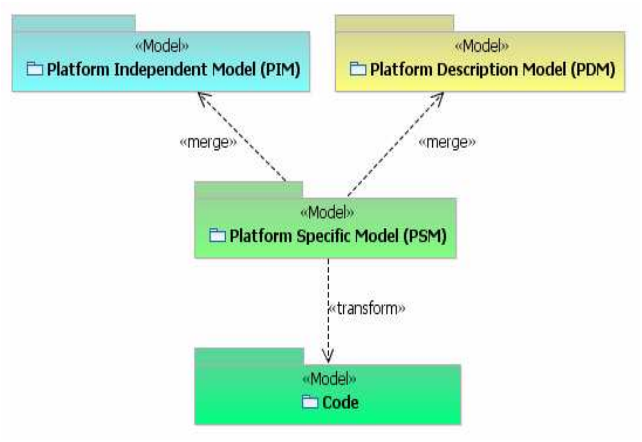
«Model»
Platform Independent Model (PIM)
«Model»
Platform Description Model (PDM)
«merge»
«merge»
«Model»
Platform Specific Model (PSM)
«transform»
«Model»
Code

Fig. 1. Model relations in OMG MDA

Two kinds of refined models were distinguished: a code model and a mapping model. A code model consists of UML elements corresponding to elements of the target programming language. Mapping model depicts a structure of directories and files of compilation units. It defines also relations between compilation units from a mapping model and corresponding elements of a code model. Code and mapping models constitute input data for code generation.

These models can be introduced in UML using profiles. Profiles are the primary mechanism for defining domain-specific UML variant [1]. A UML profile describes how UML model elements are extended to support usage in a particular domain. UML model elements are extended using stereotypes and tagged values that define additional properties associated with the elements. However, the UML profile definition does not provide a means for precisely defining semantics associated with extension. To cope with this problem stereotypes were defined with their correctness constraints. The constraints help to verify whether a given stereotype can be related to a particular UML model element in a certain model context.

Two profiles were created. The first profile - *CSharpCodeProfile* is responsible for mapping of UML meta-model elements (in the current system version – only UML class elements) to corresponding constructions of the target language (currently C#). Apart from classes, methods and fields, appropriate stereotypes can specify that given model

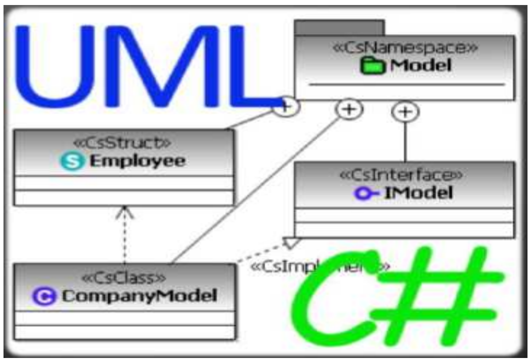

Fig. 2. UML model refined for C # target

elements should be recognized as C# properties, indexers, delegates, namespaces etc. For example, a stereotype «CsNamespace» can be added to a package defining a naming scope in terms of C#.

The second profile - *CSharpMapProfile* is used for modeling a file structure of an implementation project. It includes stereotypes identifying a main project directory, secondary directories, implementation units, etc. A main project directory creates a root element of the map model. A compilation unit can be mapped to an element of a code model element using «CsMap» stereotype.

Finally, CSharpPrimitiveTypesLibrary includes primitive types of C# language. The types can be associated to model elements (attributes, operation return types, parameters). They can be also used in OCL expressions. The library together with developed profiles constitutes a PDM in the MDA sense (Fig. 1).

In Fig. 3 and 4 exemplary class diagrams are presented. The first model, without stereotypes, can be treated as a PIM. The next one presents a code model. It is the same model after refinement using stereotypes from CSharpCodeProfile and types from CSharpPrimitiveTypesLibrary. The refined model represents PSM according to MDA.

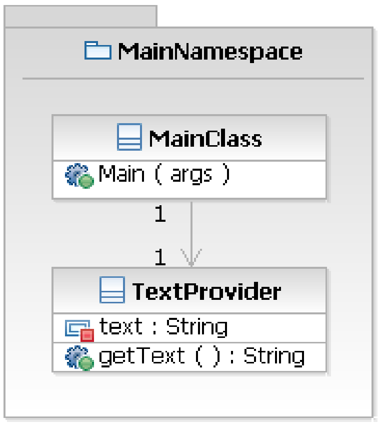

Fig. 3. PIM - class model

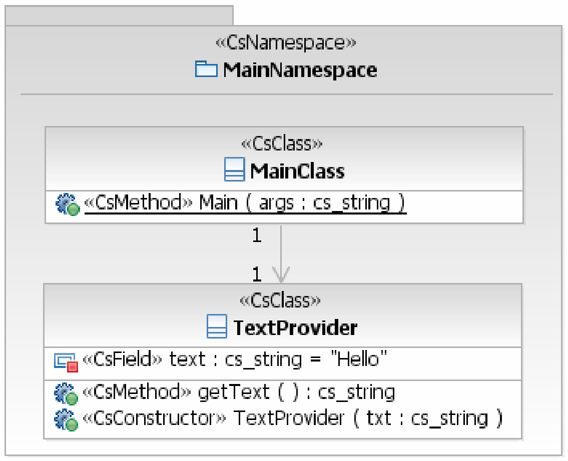
«CsNamespace»
MainNamespace
«CsClass»
MainClass
«CsMethod» Main (args : cs_string)
1
1
«CsClass»
TextProvider
«CsField» text : cs_string = "Hello"
«CsMethod» getText ( ) : cs_string
«CsConstructor» TextProvider (txt : cs_string)

Fig. 4. PSM - C# code model

http://www.springerlink.com/content/q552536748871642
The original publication is available at www.springerlink.com

An exemplary structure of an implementation project is defined by a mapping model shown in Fig. 5. Two implementation units are mapped with «CsMap» stereotype to corresponding classes from the code model.

III. System requirements

The main motivation for creation of the T.O.F.I.C. system was supporting of MDE ideas. This project was also motivated by:

- modeling with C# target, supporting convenient refinement towards C# applications,
- verification of a model refined for a target programming language, a facility scarcely supported by available environments,
- secure generation of a documented code allowing developers to concentrate on problem solving,
- utilization of the extension mechanisms offered by a CASE tool.

Analysis of the main system features and investigation of existing tools finished with conclusions that a high quality tool should meet the following goals:

1) UML modeling – creating of UML class diagrams with OCL constraints associated to model elements,

2) C# modeling – modeling elements of C# language and code structure using UML notation,

3) Modeling of file and directory structures – modeling of file trees and C# compilation items using refined elements of UML class model,

4) Model refinement – step by step gradual model refinement towards a final source code generation,

5) Verification of refined UML class model elements – checking if selected model extracts with associated OCL constraints can be a source for correct C# code generation,

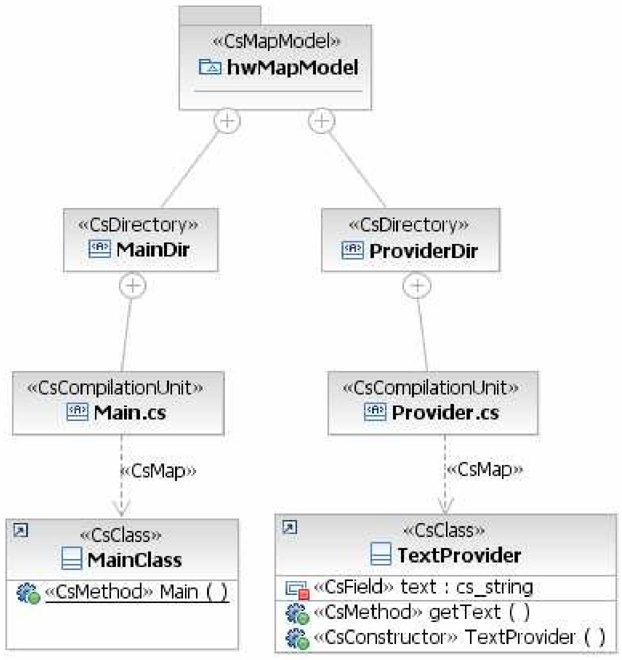

Fig.5. Mapping model.

6) Verification of files and directories – checking if a structure of created file trees and compilation items is correct,

7) Verification of OCL constraints – checking whether input OCL expressions (like class invariants, operation pre- and post-conditions) are syntactically and semantically correct,

8) Code generation – C# code generation based on dedicated UML class model elements and associated OCL constraints,

9) Selective code generation – code generation based only on selected UML model elements,

10) Complete code generation – all permitted C# constructions can be generated based on a refined UML model,

11) Correctness of generated code – a target C# code generated should be correct,

12) Error detection – a user is informed about detected errors in UML models, OCL constraints and during C# code generation,

13) Easy tool comprehension – understandable principles of tool exploitation,

14) Consistency with OMG specifications – support of current UML and OCL specifications,

15) Code readability – the appropriate structure of the generated code,

16) Intuitive user interface – ergonomic and easy to learn user interface, similar to other tools,

17) Compatibility and integration – integration with a given CASE tool, utilization and extension of its functionality.

IV. T.O.F.I.C. SYSTEM

In this section we present the main technology used in the system (A) and its architecture (B).

A. Technology

Extension mechanism of IBM Rational Software Architect [10] is based on capabilities supplied by the Eclipse platform [11]. A net of plug-ins interconnected to a central loading kernel and to each other constitutes the platform. There are two mechanisms that can be used when a plug-in calls functionality of another one: an extension point and a runtime dependency. T.O.F.I.C uses extension points delivered by plug-ins of RSA.

The basic Eclipse component used in RSA is Eclipse Modeling Framework (EMF) [12]. It supports creation of models using Ecore metamodel, which is an implementation of EMOF – a part of MOF OMG standard [19]. The component is responsible also for serialization of models to XMI format (XML Metadata Interchange) and generation of Java code. EMF Technology (EMFT) is an extension of EMF and consists of Eclipse features. EMFT supports following four technologies devoted to processing of models:

EMFT OCL – association of OCL constraints to model elements, parsing of OCL expressions, creation of Abstract Syntax Trees (AST), verification of OCL syntax and semantics,

EMFT Query – formulating queries about features of model elements in an SQL-like language or OCL, modification of model elements that satisfy conditions of the queries,

EMFT Transformation – processing of transactions in EMF model, parallel execution of various threads realizing read and write operations on the same model,

EMFT Validation – definition of constraints consistent with EMF, live and batch validation of constraints, support for a creator of parsers for constraints languages, declaration of an order in which model elements are verified.

Basing on EMF component, an Eclipse UML component was created in order to simplify model manipulation at higher level. Therefore a developer can create tools operating on UML models preserving consistency with the internal Ecore representation. Creation of a user interface is supported by two Eclipse components Graphical Editing framework GEF and Graphical Modeling Framework GMF.

IBM RSA uses EMF, EMFT, GEF and GMF components of Eclipse in its features and makes available for developers via its application interfaces. A developer can use several components to extend a system:

- Modeling API,
- Patterns framework,
- Transformation framework,
- Compare and merge framework,
- Reusable Asset Specification API,
- Plugets API.

The T.O.F.I.C. system operates on a model representation created by IBM RSA and uses extension points and packages delivered by two of these components. Modeling API assists creating UML models and UML profiles, which comprise stereotypes for model elements refinement using attributes and constraints. Operations on a model can be processed in a transitional way. The second component applied in the system is Transformation framework. It is responsible for model to model and model to text transformation.

B. System Architecture

T.O.F.I.C. was developed as an extension of IBM Rational Software Architect. It extends the system functionality with C# modeling and C# code generation capabilities.

T.O.F.I.C. was implemented as a feature of Eclipse platform available via an update site. A general structure of the T.O.F.I.C. feature is shown in Fig. 6. The feature consists of the following main components:

- Eclipse feature grouping all T.O.F.I.C. plug-ins,
- Update site - adding the feature to RSA using configuration manager,
- Branding plug-in - identification of the T.O.F.I.C. feature,
- C# transformation plug-in - transformation of a refined UML model into C# code,
- ConstraintVisitor - a class included in C# transformation plug-in, generation of C# code from OCL constraints associated with refined UML models,
- Primitive C# Types Library Plug-in - introduction of primitive C# types into a refined UML model,
- C# Profiles Plug-in - refinement of UML elements to model C# code items, and a structure of a file tree and a C# project catalogue,

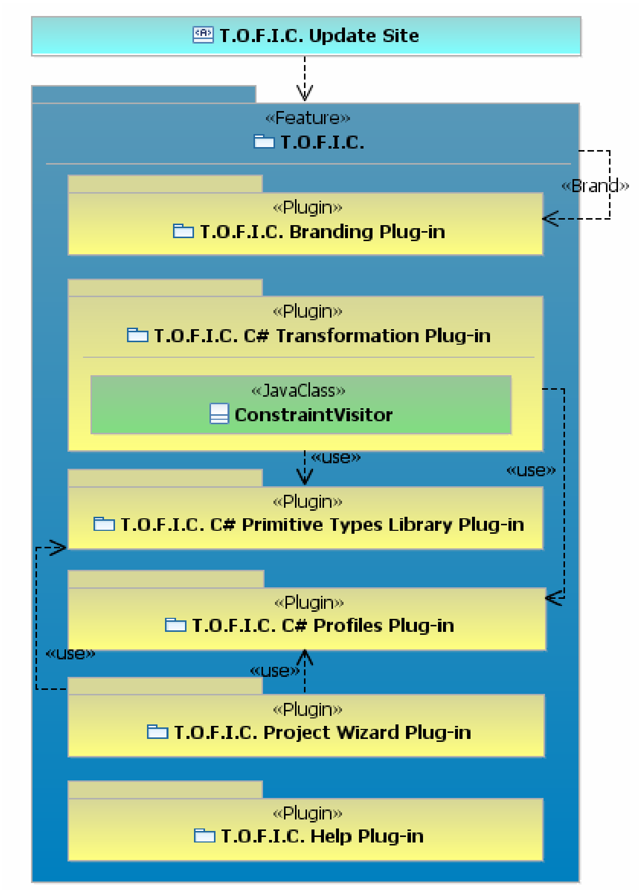
T.O.F.I.C. Update Site
«Feature»
T.O.F.I.C.
«Brand»
«Plugin»
T.O.F.I.C. Branding Plug-in
«Plugin»
T.O.F.I.C. C# Transformation Plug-in
«JavaClass»
ConstraintVisitor
«use»
«use»
«Plugin»
T.O.F.I.C. C# Primitive Types Library Plug-in
«Plugin»
T.O.F.I.C. C# Profiles Plug-in
«use»
«use»
«Plugin»
T.O.F.I.C. Project Wizard Plug-in
«Plugin»
T.O.F.I.C. Help Plug-in

Fig. 6. System architecture.

- Project Wizard Plug-in - creation of an UML project adapted for C# modeling,
- Help Plug-in - extension of the Eclipse help system with T.O.F.I.C. users guide.

V. Application of the System

This section describes how a user can work with T.O.F.I.C.

A. *Basic activity flow*

The system can be used for developing a model, refining it and building a C# application. The whole process consists of the following steps:

1) Project creation

A user creates a C# model project using the project creator. The names for a project, a code model and a map model are provided. T.O.F.I.C. generates the project using appropriate UML profiles and libraries. The project includes two models with «CsCodeModel» and «CsMapModel» stereotypes, and primitive types of UML and CSharp. In Fig. 7 the structure of a project is shown at the left hand side.

2) UML modeling

The user models a code structure using IBM RSA diagram editors. The UML diagrams are created. Class elements can be annotated with OCL constraints.

3) C# implementation structure creation

The user creates a structure of implementation project using UML class diagrams.

3) C# modeling

Available stereotypes and primitive types are used for gradual refinement of model elements. Relations between a map model and a code model are built (Fig. 5).

Steps 2) 3) and 4) can be fulfilled in sequel or in parallel. A general UML model can be created, further accompanied with stereotypes. On the other hand, stereotyped elements can be also created at the initial step, according to the user needs.

4) Transformation configuration

Having a completed model, the user creates an empty Eclipse project. A new configuration is build using the transformation management menu. The root element of the map model constitutes a source element of the transformation. The Eclipse project states as a transformation target element.

5) Code generation

Next, the user run the configured transformation and the corresponding C# code structures are created (Fig. 7).

The help system can assist in solving problems at any stage of project development.

B. *Model verification*

Proposed profiles include stereotypes with their correctness constraints. The constraints check dependencies between elements of a code model in accordance to the specification of C# language [20]. They verify also a correct structure of file and directory trees in a mapping model. Verification rules of each constraint are implemented in a dedicated Java class. A constraint has an associated key of an error message generated in case the constraint is broken. Error messages are stored in an appropriate file of the profile properties.

Preservation of constraints can be verified on user demand. Selected model elements are validated, as well as all elements included in the selected ones. Some stereotype constraints are also checked automatically after modification of a model. Model verification is also performed before model to code transformation.

VI. Conclusions

The paper presented the environment for building C# applications using MDA paradigm. The T.O.F.I.C. system is an extension of IBM Rational Software Architect. It can be used for modeling of C# code and project structure. The gradual refinement of meaning of model elements is made by utilization of dedicated UML profiles and a library of primitive types. A refined model is transformed into C# code. The system is easy to use thanks to full integration with the CASE tool, its modeling policies and interfaces, which are known to a user.

The main advantage of the system is its capability to precise modeling of detailed concepts of the desired programming language. The refinement process can be verified beginning from the early stages of system modeling.

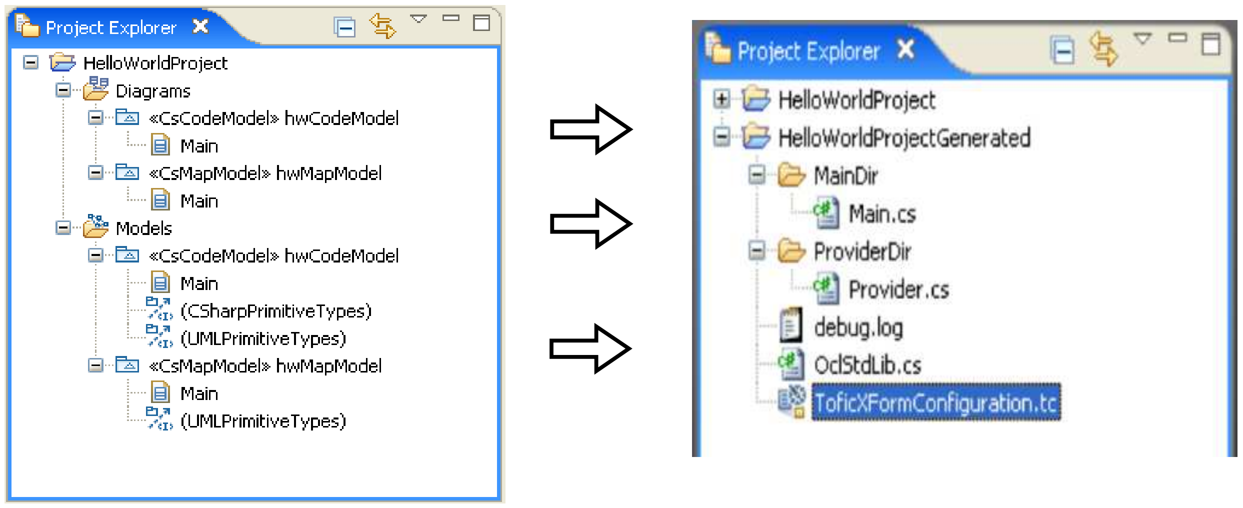

Fig. 7. Structure of a T.O.F.I.C. project and a generated C# project

The T.O.F.I.C. system is still under development. The working prototype will be extended with new functionality. The CSharpCode profile will comprise comprehensive set of C# language grammar constraints. The code will be generated for more language constructions expressed by appropriate stereotypes. Further development of the T.O.F.I.C system should take into account new versions of RSA environment and C# compiler. Another direction of T.O.F.I.C. development is extension of the OCL processing.

References

[1] R. France, B. Rumpe, “Model-driven Development of complex software: A research roadmap”, *Future of Software Engineering at ICSE'07*, IEEE Soc., 2007, pp. 37-54.

[2] K. Czarnecki, S. Helsen, "Feature-based survey of model transformation approaches", *IBM System Journal*, Vol. 45, No 3, 2006, pp. 621-645.

[3] T. Mens, P. van Gorp, "Taxonomy of model transformation", *Proceedings of the International Workshop on Graph and Model Transformation (GraMoT 2005),* Tallin, Estonia, Sept. 2005, *ENTCS,* Vol. 152 , March 2006, Elsevier, 2006, pp. 125-142.

[4] F. J. Lucas, F. Molina, A. Toval, "A systematic review of UML model consistency management", *Journal of Object Technology*, 2009, in press.

[5] P. Ołtarzewski, “T.O.F.I.C. Extending IBM Rational Software Architect with UML model refinement and C# code generation”, Bach. Thesis. Inst. of Comp. Science, Warsaw Univ. of Technology 2008 (in polish).

[6] MDA home page, http://www.omg.org/mda/

[7] S. Frankel, *Model Driven Architecture: Appling MDA to enterprise computing*, Wiley Press, Hoboken, NJ, 2003.

[8] Unified Modeling Language Superstructure v. 2.1.2 (2007). OMG Document formal/2007-11-02, http://www.uml.org

[9] A. G. Kleppe, J. Warmer, *The Object Constraint Language: Getting your models ready for MDA*, Addison-Wesley 2nd ed, Boston MA, 2003.

[10] IBM Rational Software Architect, http://www-306.ibm.com/software/rational

[11] Eclipse Open Source Community, http://www.eclipse.org

[12] Eclipse Modeling Framework (EMF), http://www.eclipse.org/modeling/emf

[13] Semantics of a Foundation subset for executable UML models (FUML) 2008, http://www.uml.org

[14] R. Pilitowski, A. Derezinska, “Code Generation and Execution Framework for UML 2.0 Classes and State Machines”, T. Sobh (Ed.) *Innovations and Advanced Techniques in Computer and Information Sciences and Engineering*, Springer, 2007, pp. 421-427.

[15] G. Gonzola, C. R. del Castillo, J. Llorens, "Mapping UML associations into Java code", *Journal of Object Technology*, vol. 2 , no 5, September-October 2003, pp135-162.

[16] O. Nikiforowa, N. Pavlova, "Development of the tool for generation of UML class diagram from two-hemisphere model", *Proc. of Inter. Conf. on Soft. Eng. Advances*, ICSEA, Oct. 26-21 Malta, 2008, pp. 105-112.

[17] T. Haubold, G. Beier, W. Golubski, “A pragmatic UML-based meta model for object-oriented code generation”, *Proc. of 21st Inter. Conf. on Soft. Eng. & Knowledge Eng. SEKE’09* , 2009, pp. 733-738.

[18] L. K. Kishore, D. Saini, “IBM Rational Modeling Extension for Microsoft .NET”, http://www.ibm.com/developerworks/rational/library/07/0306_kishore_saini/

[19] Meta Object Facility (MOF), OMG specification, http://www.omg.uml

[20] C# 2.0 specification http://msdn.microsoft.com/en-us/library/ms228593(VS.80).aspx