From Mapping Specification to Model Transformation in MDA: Conceptualization and Prototyping

Slimane Hammoudi, Denivaldo Lopes

ESEO, Ecole Supérieure d'Electronique de l'Ouest
4, Rue Merlet de la Boulaye
BP 926, 49009 ANGERS cedex 01

Abstract. In this paper, we present in the first part our proposition for a clarification of the concepts of mapping and transformation in the context of Model Driven Architecture (MDA), and our approach for mapping specification and generation of transformation definition. In the second part, we present the application of our approach from UML to C#. We propose a metamodel for mapping specification and its implementation as a plug-in for Eclipse. Once mappings are specified between two metamodels (e.g. UML and C#), transformation definitions are generated automatically using transformation languages such as Atlas Transformation Language (ATL). We have applied this tool to edit mappings between UML and C# metamodels. Afterwards, we aim to use these mappings to generate ATL code to achieve transformations from UML into C#.

1 Introduction

The Object Management Group (OMG) has stimulated and promoted the adoption of the Model Driven Architecture (MDA™) [1] to define an approach to software development based on modeling and automated mapping of models to implementation. In this approach, models become the hub of development, separating platform independent characteristics (i.e. Platform-Independent Model - PIM) from platform dependent characteristics (i.e. Platform-Specific Model - PSM).
The MDA approach promises a number of benefits including business logic is protected against the changes in technologies, systems can evolve for meeting new requirements, old, current and new technologies can be harmonized, legacy systems could be integrated and harmonized with new systems.
In this approach, models are applied in all steps of development up to a target platform, providing source code, files of deployment and configuration, and so on. MDA proposes architecture to address the complexity of software development and maintenance, which has no precedents. It claims that software developers can create and maintain software artifacts with little effort. However, before this becomes a mainstream reality some issues in MDA approach need solutions such as mapping, transformation, handling of semantic distance between metamodels [3], bidirectional mapping [4], and so on.

Hammoudi S. and Lopes D. (2005).
From Mapping Specification to Model Transformation in MDA: Conceptualization and Prototyping.
In *Proceedings of the Joint Workshop on Web Services and Model-Driven Enterprise Information Systems*, pages 132-143
DOI: 10.5220/0002573001320143

In this paper, we use the term mapping as a synonym for correspondence between the elements of two metamodels, while transformation is the activity of transforming a source model into a target model in conformity with the transformation definition. In our approach, a transformation definition is generated from a mapping specification. The distinction between mapping specification and transformation definition is detailed in later sections. The objective of this paper is threefold. First, to provide a precise definition of the concepts of mapping and transformation. Second, to provide a general metamodel for mapping specification between two metamodels (source and target) in the context of MDA. Third, to present a tool based on eclipse enabling the edition of mappings and the generation of transformation definition from mapping specifications. We will apply this tool to C# Platform from UML as PIM.

This paper is organized in the following way. Section 2 is an overview of MDA and the main concepts behind this framework. Section 3 presents our proposition for the clarification of the concepts of mapping and transformation and our metamodel for mapping specification between two metamodels in the context of MDA. Section 4 introduce briefly a formalism for mapping and shows the implementation of our proposed metamodel through a plug-in for eclipse, and its application to C# platform from UML PIM. Section 5 concludes this paper and presents the future directions of our research.

2 Background

The OMG's Model Driven Architecture (MDA) is a new approach to develop large software systems in which the initial efforts aim to cover their functionalities and their behavior. MDA separates the modeling task of the implementation details, without losing the integration of the model and the development in a target platform. The key technologies of MDA are Unified Modeling Language (UML), Meta-Object Facility (MOF), XML Meta-Data Interchange (XMI) and Common Warehouse Metamodel (CWM) [1]. Together, they unify and simplify the modeling, the design, the implementation, and the integration of systems. One of the main ideas of MDA is that each model is based on a specific meta-model. Each meta-model precisely defines a domain specific language. Finally, all meta-models are based on a meta-metamodel. In the MDA technological space, this is the Meta-Object Facility (MOF). There are also standard projections on other technological spaces like XMI for projection on XML and Java Metadata Interface (JMI) for projection on Java. MDA also introduces other important concepts: Platform Independent Model (PIM), Platform-Specific Model (PSM), transformation language, transformation rules and transformation engine. These elements of MDA are depicted in figure 1.

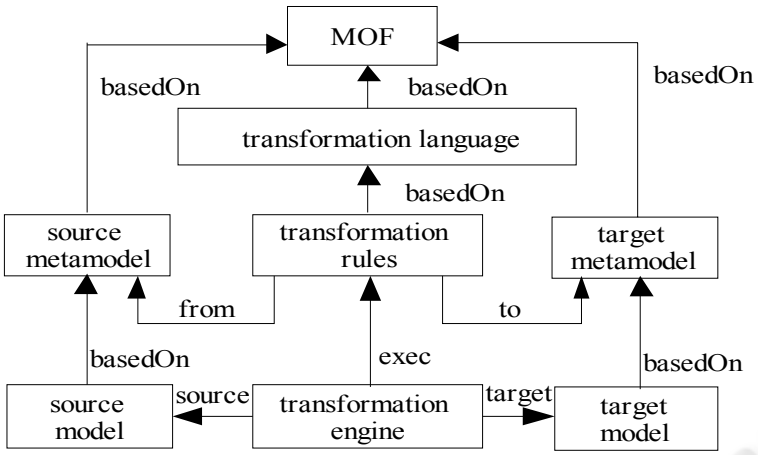

Fig. 1. Transformation in MDA

Each element presented in Figure 1 plays an important role in MDA. In our approach, MOF is the well-established meta-meta-model used to build meta-models. The PIM reflects the functionalities, the structure and the behavior of a system. The PSM is more implementation-oriented and corresponds to a first binding phase of a given PIM to a given execution platform. The PSM is not the final implementation, but has enough information to generate interface files, a programming language code, an interface definition language, configuration files and other details of implementation. Mapping from PIM to PSM determines the equivalent elements between two meta-models. Two or more elements of different meta-models are equivalents if they are compatible and they cannot contradict each other. Model transformation is realized by a transformation engine that executes transformation rules. Transformation rules specify how to generate a target model (i.e. PSM) from a source model (i.e. PIM).

To transform a given model into another model, the transformation rules map the source into the target meta-model. The transformation engine takes the source model, executes the transformation rules, and gives the target model as output. Using one unique formalism (e.g. MOF) to express all meta models is very important because this allows the expression of all sorts of correspondence between models based on separate meta-models. Transformations are one important example of such correspondence, but there are also others like traceability, etc. In other words, given m1(s)/Ma and m2(s)/Mb, where m1 is a model of a system s created using the meta-model Ma, and m2 is a model of the same system s created using the meta-model Mb, then a transformation can be de- fined as m1(s)/Ma → m2(s)/Mb. When Ma and Mb are based on the same meta-meta-model, the transformation may be expressed in a unique transformation language (i.e. a language independent of the meta-model). Furthermore, the transformation language itself may be considered as a domain-specific language. This has many interesting consequences. One of these is that a transformation program corresponds to an MDA model. We may thus easily consider higher-order transformations, i.e. transformations having other transformations as input and/or producing other transformations as output. One of the most popular meta-models is UML, but there are plenty of other meta-models being defined. For

example, there could be a meta-model of the Java language. Based on these two meta-models, it is possible to express a transformation from UML 1.5 class diagrams to Java 1.4.2 code. In fact, models have been used for a long time, but they remained disconnected from the implementation process.
The automatic generation of code to a specific language from a UML class diagram is not new either; some CASE tools give this support such as Poseidon for UML (http://www.gentleware.com). However developers still have to write all the application codes by hand. Moreover, when the application has evolved to acquire new capabilities or adapt to new technologies, these tools cannot help the developer, and the model is used only as documentation. An Integrated Development Environment (IDE) provides a set of tools integrated on a single user interface that often comprises a sophisticated text editor, a graphical editor for GUI, an editor to database tables, a compiler and a debugger, e.g. IBM's WebSphere Studio and Microsoft's Visual Studio .NET. An IDE can aid the software development, but only in the programming phase (i.e. it is based on code-centric approach). A tool powered with MDA will be enabled to support the system development throughout its life cycle. The development of large software systems can take some suggested benefits (some benefits are still not proven) from MDA:

- The same PIM can be used many times to generate models on different platforms (PSMs) [8];
- Many views of the same system, e.g. many abstraction levels or details of implementation. We de-fine abstraction levels as the possibility to see a system (e.g. applications and business process) fragmented in many different and interlinked levels, each level detaching important characteristics of the same system;
- Enhancement of the portability and of the interoperability of systems in the level of models;
- Preservation of the business's logic against the changes or evolution of technology ;
- An uniform manner for business models and for technologies to evolve together;
- Prevention against error-prone factors linked to manual design of systems [7];
- Increase the return on technology investments;
- Enhancement of the reengineering, i.e. it assists the recuperation of business's logic from source codes or from implementation environments;
- Enhancement of the interaction and of the migration between different technological spaces.

Apart from these benefits, the approach using models forces the architects to think about the architecture and the model behind the system in development, whereas a code centric approach makes architects concentrated on the code, so they consequently forget the main properties of the system. Other case studies have shown some benefits of the MDA Approach. In [11], the authors have demonstrated that the development of a case study (i.e. J2EE PetStore) using a MDA tool is 35% faster than the development using a code centric approach (i.e. using a non powered-MDA tool).

3 Mapping and Transformation

Nowadays, MDA suffers from a lack of agreement on terminology, especially concerning the concepts of mapping and transformation. In MOF QVT [6], mapping is defined as *specification of a mechanism for transforming the elements of a model conforming to a particular metamodel into elements of another model that conforms to another (possibly the same) metamodel*. In MDA distilled book [13], mapping is defined as *the application or execution of a mapping function in order to transform one model to another*, and mapping function is defined as *a collection of mapping rules that defines how a particular mapping works*. In both references and others discussed in [7], the concept of mapping and transformation are not so clear, since these terms can refer to many different concepts. Moreover, they are usually defined without explicit distinction between them. The table 1 presented in [7], and extended briefly here, illustrates in an obvious manner that the terminology related with the transformation and mapping concepts is really immature.

Table 1: Equivalencies between terms according to the Transformation Pattern

	Transf. Instance	**Tranf. Function**	**Transf. Model**	**Transf. Program**	**Transf. Progr. Lang**	**Transf. Interp**
MDA Guide [2]	*Transfo*	*Mapping Instance*	*Mapping*		*Mapping Language*	
MDA Distilled [13]	*Mapping*	*Mapping rule*	*Mapping function*			
MDA Explained[10]	*Transfo. Mapping*	*Mapping/Transf rule*	*Transfo. Definition*			*Transfo. tool*
QVT DSTC [6]	*Tracking*	*Transfo. rule*	*Transfo.*			*Transfo. Engine*
QVT Partners [14]			*Transfo. Relation*	*Mapping Relation*		
[17]		*Mapping*	*Model of Mapping*		*Mapping Formalism*	
[18]	*Transfo.*		*Transfo. Pattern*			
[19]				*Transfo. Spec*		*Transformer*
[20]	*Transfo. Process*		*Transfo. Descr*			

According to our vision, the concepts of mapping and transformation should be explicitly distinguished, and together could be involved in the same process that we denominate transformation process. In fact, in the transformation process, the mapping specification precedes the transformation definition. A mapping specification is a definition (the most declarative as possible) of the correspondences between metamodels (i.e. a metamodel for building a PIM and another for building a PSM). Transformation definition contains a minute description to transform a model

into another using a hypothetic or concrete transformation language. Hence, in our approach the transformation process of a PIM into a PSM can be structured in two stages: mapping specification and transformation definition. Finally, we define the term transformation as the manual or automatic generation of a target model from a source model, according to a transformation definition. From a conceptual point of view, the explicit distinction between mapping specification and transformation definition remains in agreement with the MDA philosophy, i.e. the separation of concerns. Moreover, a mapping specification could be associated with different transformation definitions, where each transformation definition is based on a giving transformation definition metamodel. Figure 2 illustrates the different concepts of MDA according to our vision where mapping specification is a mapping model, and transformation definition is a transformation model. In this figure, a mapping model is based on its metamodel, and it relates two metamodels (left and right). A transformation model is based on its transformation metamodel, and it is generated from a mapping model. A transformation engine takes a source model as input, and it executes the transformation program to transform this source model into the target model.

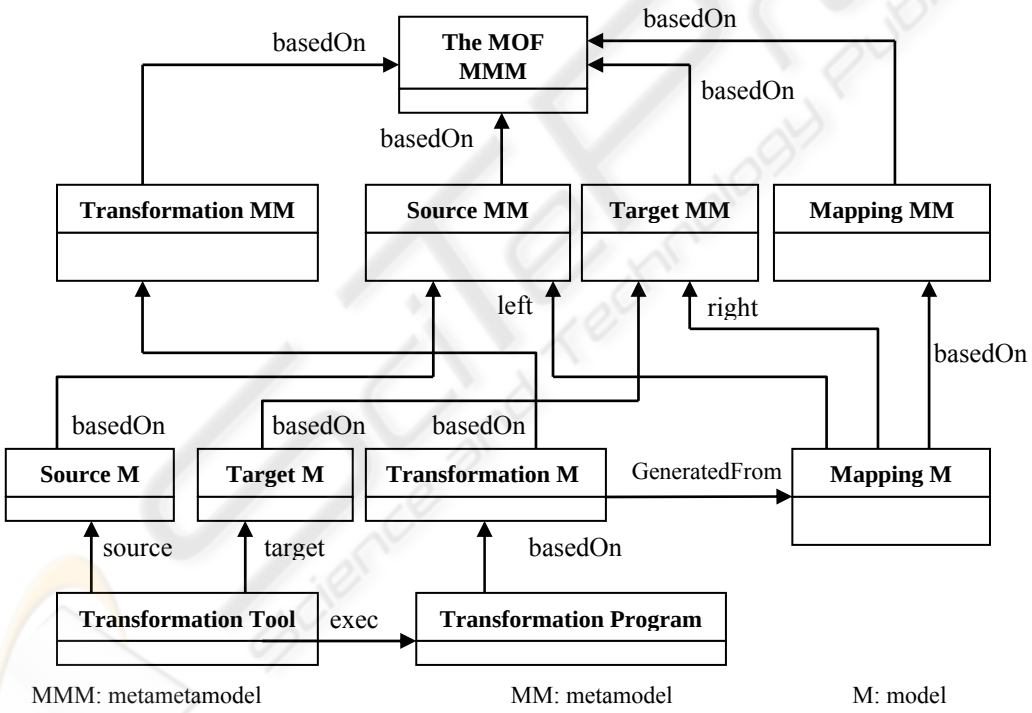

Fig. 2: Transformation Process within MDA: from Mapping to Transformation

Several research projects have studied the mapping specification between metamodels [9] [12] [18]. However, the ideas around mapping specification are not sufficiently developed to create efficient tools to enable automatic mappings.

Nowadays, transformation languages are not yet very well explored to make choices about a standard transformation language such as desired by OMG [2]. In the next few years, the submitted propositions [6] [14] in response to QVT RFP might converge to a standard language, which will provide a new step forward in the evolution of MDA. However, wisdom tells us that one problem can be resolved using different solutions, but one solution for all problems does not exist. Thus, it is clear that this standard language will not provide a sufficient solution for all types of model transformations around MDA. However, this will not be a limitation for applying MDA, because a transformation language is also a model, thus one transformation language can also be transformed into another transformation language. A priori, transformations between transformation languages seem unnecessary and unproductive. However, several examples such as Structured Query Language (SQL) (i.e. a standard query language for manipulating databases) have demonstrated that a standard is beneficial, because it establishes a unique and well-known formalism for understanding a problem and its solution. On the one hand, SQL provides a universal language for manipulating databases. On the other hand, SQL can be transformed into a proprietary language for execution into a database engine. A transformation from SQL into a proprietary language provides some benefits such as improved performance, reduction of memory-use, and so on. Making an analogy between SQL and a standard transformation language, we can expect that a standard transformation language can provide some benefits without imposing severe limitations. Mapping and transformation have been studied for a long time ago in the database domain. However, they have taken another dimension with the sprouting of MDA. This not means that they are well studied and done to be applied in the MDA context. In fact, mapping specification and transformation definition are not yet an easy task. Moreover, tools to enable the automatic creation of mapping specification and automatic generation of transformation definition are still under development.
In the next section, we start briefly presenting a foundation for mapping and afterwards we discuss our proposition for specifying mappings (i.e. correspondences between metamodels). This approach for mapping is based on a metamodel and implemented as a tool on Eclipse. This tool provides mapping support that is a preliminary step before the generation of a transformation definition.

4 Foundations and Prototyping

In this section, we present our proposition for specifying mappings (i.e. correspondences between metamodels). This approach for mapping is based on a metamodel and implemented as a tool on Eclipse. This tool provides support for mapping, which is a preliminary step before the creation of a transformation definition, using ATL. We have applied this tool for the different cases presented previously.
The creation of mapping specification and transformation definition is not an easy task. In addition, the manual creation of mapping specification and transformation definition is a labor-intensive and error-prone process [12]. Thus the creation of an automatic process and tools for enabling them is an important issue. Some

propositions enabling the mapping specification have been based on heuristics [12] (for identifying structural and naming similarities between models) and on machine learning (for learning mappings). Other propositions enabling transformation definition have been based on graph theory . Mapping specification is not a new issue in computer science.

For a long time, the database domain has applied mappings between models (i.e. schema) and transformation from different conceptual models, e.g. entity-relationship (ER), into logical or physical models (relational-tables and SQL schema). However, these same issues have taken a new dimension with the sprouting of MDA, because models become the basis to generate software artifacts (including code) and in order to transform one model into another model, mapping specification is required. So, both mapping specification and transformation definition have been recognized as important issues in MDA.

4.1 A metamodel for mapping

In order to define a mapping, we need a metamodel, which enables:

- Identification of what elements has similar structures and semantics to be mapped.
- Explanation of the evolution in time of the choices taken for mapping one element into another element.
- Bi-directional mapping. It is desirable, but is often complex [10].
- Independence of model transformation language.
- Navigation between the mapped elements.

Figure 3 presents our proposition of a metamodel for mapping specification. A complete definition of this metamodel is presented in [5].

In this metamodel, we consider that a mapping can be unidirectional or bi-directional. In unidirectional mapping, a metamodel is mapped into another metamodel. In bi-directional mapping, the mapping is specified in both directions. Thus, as presented previously, we prefer refer to the two metamodels in a mapping as left or right metamodels.

A central element in this metamodel is the element `Correspondence.` This element is used to specify the correspondence between two or more elements, i.e. left and right element. The correspondence has a filter that is an OCL expression. When bidirectional is false, a mapping is unidirectional (i.e. left to right), and when it is false it is bidirectional (i.e. in both directions). It has two `TypeConverters` identified by `typeconverterRL` and `typeconverterLR`. `typeconverterRL` enables the conversion of the elements from a right metamodel into the elements from a left metamodel. `typeconverterLR` enables the conversion of the elements from a left metamodel into the elements from a right metamodel. We need often specify only `the typeconverterLR`.

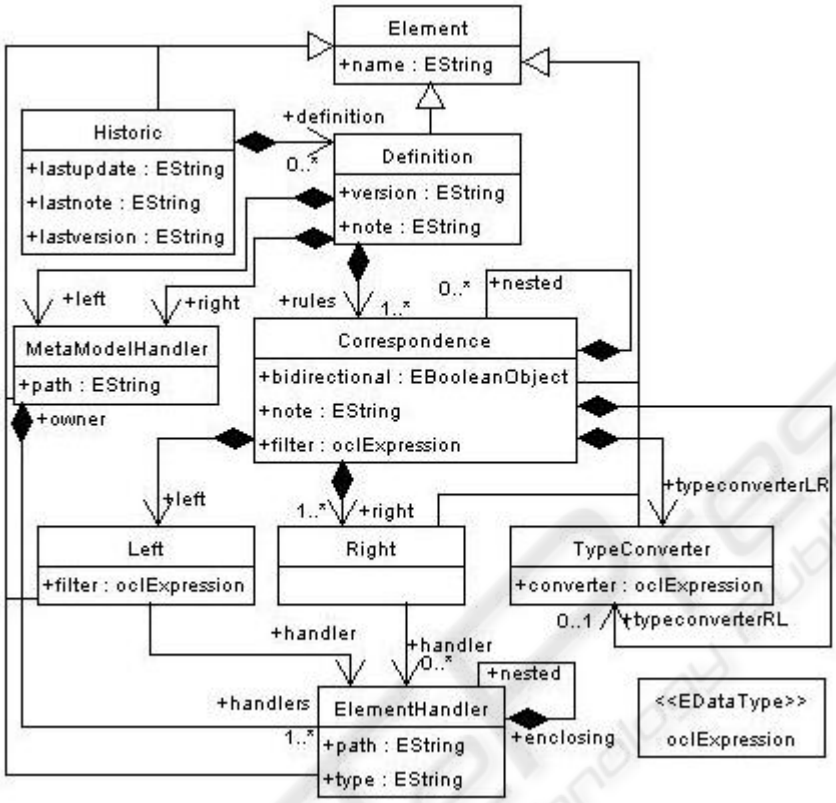

Fig. 3. Metamodel for Mapping Specification

4.2 A Plug-in for Eclipse

Eclipse Modeling Framework (EMF) is a modeling framework and code generation facility for supporting the creation of tools and applications driven by models [15]. EMF represents the efforts of Eclipse Tools Project to take into account the driven model approach. In fact, MDA and EMF are similar approaches for developing software systems, but each one has different technologies. MDA was first designed by OMG using MOF and UML, while EMF is based on Ecore and stimulates the creation of specific metamodels.

A tool supporting our proposed metamodel for mapping should provide:

- Simplification for visualizing mappings. In order to specify a mapping, two metamodels are necessary. From experience, metamodels have generally a considerable number of elements and associations. So the visualization becomes complex, putting two metamodels so large side by side and the mapping in the center. A tool should allow the creation of views, navigation and encapsulation

of details unnecessary for each mapping in order to facilitate the visualization and comprehension of the mapping without modifying the involved metamodels.

- Creation of transformation definition from mapping specification. A mapping specification is a model itself, and then it can also be transformed into another model. For example, a mapping model can be transformed into a transformation model.

Our proposed tool supports all these characteristics, except the *semi-automatic matching* which is the next step for its improvement.

Figure 4 shows our plug-in for Eclipse. This tool is denominated mapping Modeling Tool (MMT). The tool presents a first metamodel on the left side, a mapping model in the center, and a second metamodel on the right. In this figure, the UML metamodel (fragment) is mapped into a C# metamodel (fragment). At the bottom, the property editor of mapping model is shown. A developer can use this property editor to set the properties of a mapping model. Before specifying mapping using our tool, we need create metamodels based on Ecore. Some tools support the editing of a metamodel based on Ecore such as Omondo [15] or the eCore editor provided with EMF. The application of our tool using UML and C# metamodel can be explained in the following steps:

1. We created a project in eclipse and we imported the UML and C# metamodel into this project.
2. We used a wizard to create a mapping model. In this step, we chose the name for the mapping model, the encoding of the mapping file (e.g. Unicode and UTF- 8), the metamodels files in the format XMI.
3. The UML and C# metamodels are loaded from the XMI files, and the mapping model is initially created, containing the elements Historic, Definition, and left and right MetamodelHandlers. For each MetamodelHandler is also created ElementHandlers that are references to the elements of the corresponding metamodel.
4. We edit the mapping model. First, we define the inter-relationships between the metamodels creating correspondences between their elements. Second, we create for each correspondence nested correspondence. Third, for each nested correspondence, we create one Left and one or more Right elements. In addition, each Left and Right element has a ElementHandler. If it is necessary, the TypeConverter is created to explicit the casting between two mapped elements.
5. The mapping model can be validated according to its metamodel, and it can be used to generate a transformation definition (e.g. using ATL language).

This tool can export a mapping model as transformation definition. For the moment, we have implemented a generator for ATL [8], but we envisage creating generators to other model transformation languages such as YATL [4], in order to evaluate the power of our proposed metamodel for mapping.

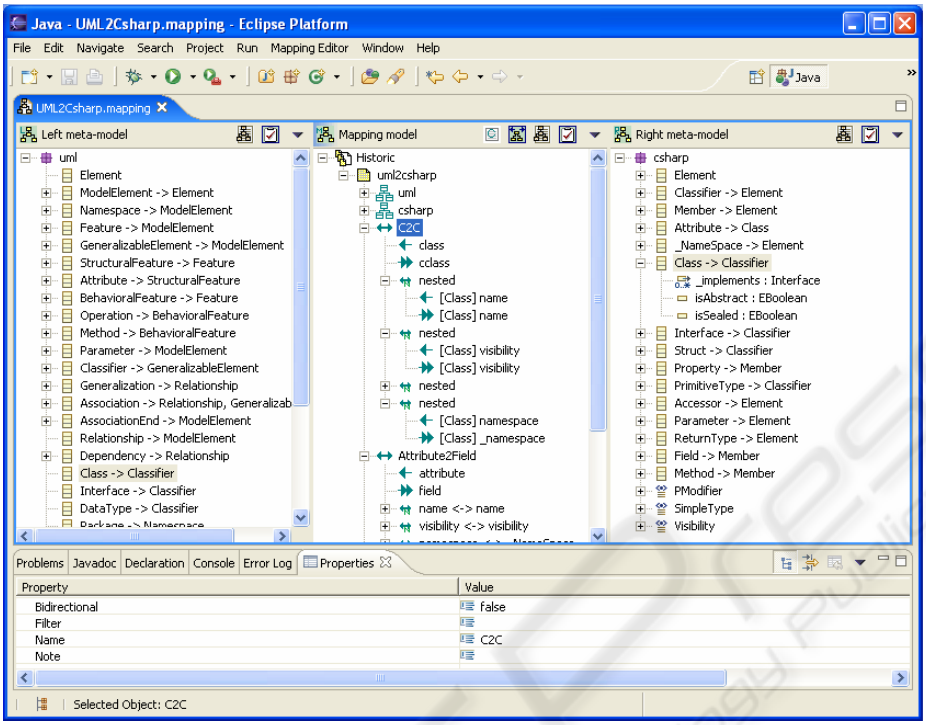

Fig. 4. Mapping Modelling Tool (MMT): From UML to C# metamodel

5 Conclusion

In this paper, we have discussed the MDA approach providing a detailed description of transformation process, distinguishing mapping and transformation. If transformation is the heart and soul of MDA [16], and transforming a PIM into a PSM requires finding correspondences between metamodels, then mapping specification is also another important issue within MDA context. We have proposed a metamodel for mapping and a tool (i.e. MMT) to support mappings. To illustrate our tool, we have specified mappings between UML as PIM and C# as PSM. According to model management algebra, a mapping is generated using an operator called match, which takes two metamodels as input and returns a mapping between them. The schema matching was not yet integrated in our plugin, because, at this stage, we are more interested in addressing the creation of mappings driven by models.

References

1. OMG: Model Driven Architecture (MDA)- document number ormsc/2001-07-01. (2001)
2. OMG, « MDA Guide Version 1.0.1 », OMG/2003-06-01, June 2003.
3. Kent, S., Smith, R.: The Bidirectional Mapping Problem. Electronic Notes in Theoretical Computer Sciences 82 (2003)
4. Patrascoiu, O.: Mapping EDOC to Web Services using YATL. 8th IEEE Enterprise Distributed Object Computing Conference (EDOC 2004) (2004)
5. Lopes, D., Hammoudi, S. Bézivin, J, Jouault, F.: Mapping Specification in MDA: From Theory to Practice. INTEROP-ESA'2005 Conference (February 23-25, 2005)
6. DSTC, IBM, and CBOP. *MOF Query / Views / Transformations Second Revised Submission*, January 2004. ad/2004-01-06.
7. J. M. Favre. Towards a Basic Theory to Model Driven Engineering. *UML 2004 - Workshop in Software Model Engineering (WISME 2004)*, 2004.
8. Bézivin, J., Dupre, G., Jouault, F., Pitette, G., Rougui, J.E.: First Experiments with the ATL Model Transformation Language: Transforming XSLT into XQuery. 2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture (2003)
9. Jan Hendrick Hausmann, S.K.: Visualizing Model Mappings in UML. ACM 2003 Symposium on Software Visualization (SOFTVIS 03) (2003) 169–178
10. A. Kleppe, J. Warmer, and W. Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley, August 2003.
11. Middleware Company: Model Driven Development for J2EE Utilizing a (MDA) Approach.www.middleware-company.com/casestudy.
12. Madhavan, J., Bernstein, P.A., Domingos, P., Halevy, A.Y.: Representing and Reasoning about Mappings between Domain Models. Eighteenth National Conference on Artificial intelligence (AAAI'02) (2002) 80–86
13. S. J. Mellor, K. Scott, A. Uhl, and D. Weise. *MDA Distilled: Principles of Model-Driven Architecture*. Addison-Wesley, 1st edition, March 2004.
14. QVT-Merge Group. *Revised submission for MOF 2.0 Query/Views/Transformations RFP (ad/2002-04-10)*, April 2004. Available at http://www.omg.org/docs/ad/04-04- 01.pdf.
15. Omondo. *Omondo Eclipse UML*, October 2004. Available at http://www.omondo.com.
16. Sendall S, Kozaczynski W : Model Transformation – the Heart and Soul of Model Driven Software Development.
17. G. Caplat and J. L. Sourrouille. Model Mapping in MDA. *Workshop in Software Model Engineering (WISME2002)*, 2002.
18. S.R.Judson, R.B.France, D.L.Carver. "Specifying Model Transformation at the Metamodel Level", WISME 2003.
19. I Kurtev, K.Van den Berg. A synthesis based approach to Transformation in an MDA Software Development Process.
20. M. Peltier. Techniques de Transformation de Modèles Basées sur la méta-modélisation. Thèse de Doctorat, Université de Nantes, 2003.