Mapping Specification in MDA: From Theory to Practice *

Denivaldo Lopes[1,2], Slimane Hammoudi[1],
Jean Bézivin[2], and Frédéric Jouault[2,3]

[1] ESEO, France
[2] Atlas Group, INRIA and LINA, University of Nantes, France
[3] TNI-Valiosys, France
{dlopes, shammoudi}@eseo.fr
{Jean.Bezivin, Frederic.Jouault}@lina.univ-nantes.fr

Abstract. In this paper, we present a metamodel for supporting the mapping specification between two metamodels. A mapping model based on this proposed metamodel defines correspondences between elements from two metamodels. It can then be used to generate a transformation definition, e.g. using Atlas Transformation Language (ATL). This metamodel is based on the Eclipse Modeling Framework (EMF). A plug-in for Eclipse implements this metamodel for mapping with the aim of providing a tool for supporting mapping specification and for generating the transformation definition.

1 Introduction

Recently, the Object Management Group (OMG) has stimulated and promoted the adoption of the Model Driven Architecture (MDA^{TM})[4] [1] approach for developing large and complex software systems. 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 aims to provide an architecture with which complex software systems (such as B2B applications on the Internet) can evolve for meeting new requirements or new technologies, business logic is protected against the changes in technologies, and legacy systems are integrated and harmonized with new technologies. However, before that becomes a mainstream reality, several issues in MDA approach need solutions, such as mapping, transformation [2], handling of *semantic distance* between metamodels [3], bidirectional mapping [4], and so on.

In this paper, we limit our objectives to providing some insights into mapping specification and transformation definition. We propose a metamodel for

* This work has been done in the context of the INTEROP European Network of Excellence.

[4] MDA^{TM} is a trademark of the Object Management Group (OMG).

mapping and a tool for editing mapping models based on this metamodel. Afterwards, we can use this tool to generate transformation definition. Our approach is implemented through a tool using a plug-in for Eclipse [5].

This paper is organized in the following way. Section 2 is an overview of MDA. Section 3 presents our approach for mapping two metamodels within the context of MDA. Section 4 shows the implementation of our proposed metamodel for mapping through a plug-in for eclipse. Section 5 contains conclusions and presents the future directions of our research.

2 Overview

Some areas, such as civil and electrical engineering, have assimilated the importance of models. In these areas, the model predates the building. Moreover, in some countries, it is impossible to start a building without its models (i.e. an electrical and civil engineering project). Models are largely used during the building, and then for maintenance, for modification (i.e. evolution) and for criminal investigation (such as police investigation in the case of a building that crumbles). In fact, we have much to learn from good practice in other areas. At times, we need to develop more the discipline of model engineering in computer science and its use in software enterprises.

For a long time, modeling languages, such as UML, were only used for documenting software systems, but nowadays models become the impetus for software development thanks to the MDA approach.

MDA is based on an architecture with four meta-layers: metametamodel, metamodel, model and information (i.e. an implementation of its model). In this approach, everything is a model or a model element, and a high level of abstraction and comprehension about a problem and its solution are provided.

The MDA approach has been accepted by companies, universities, governments and organizations as a potential solution for the problem of complexity in the development and maintenance of software systems. The period of skepticism about MDA seems to be over. Several case studies have demonstrated some benefits of the MDA approach, such as gain in productivity and protection against error-prone factors linked with manual programming [6]. Other possible benefits of MDA include:

- an increase in the return on investments in technology.
- a uniform approach for business models and for technologies to evolve together.
- an architecture ready for handling the problems of yesterday, today and tomorrow; and for integrating old, current and new technologies used in software development [7].

Several potential benefits need time to be demonstrated. However, the MDA approach has been used with success in some projects such as the case study presented at [6].

Before reaching a stage comparable to civil and electrical engineering, the MDA approach needs to be more developed and experimented. Recently, some initiatives have contributed to enhancing MDA, such as some response [8] [9] to the RFP-QVT [2] and the creation of tools for implementing the concepts around MDA [10] [11] [12] [13]. Among these initiatives, we call attention to the eclipse projects based on the concepts of MDA, such as Eclipse Modeling Framework (EMF) [10] developed under an open source philosophy. This project provided the basis for researchers to develop and implement their ideas and contributions on software engineering. Our aim is to contribute to this project providing an approach for *mapping specification* and transformation definition.

In fact, the transformation of a PIM into a PSM is a process [7]. Here, we extend this process, separating mapping specification and transformation definition. This new proposed process involves several entities: a metametamodel (such as MOF and Ecore), a source and target metamodel (such as UML), a source and target model, a mapping model, a transformation language model (such as a model based on the ATL [14] and YATL [15] metamodel), and a transformation engine. Mapping model specifies correspondences between the source and target metamodel. A transformation model is generated from a mapping model. A transformation program is based on its transformation model. The transformation is achieved by a transformation engine that executes a transformation program. The transformation engine takes a source model, executes the transformation program, and provides a target model as output. Figure 1 depicts the transformation process, relating each involved entity.

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 element into a target element in conformity with the *transformation definition*. Following these two concepts, the *mapping specification* precedes the *transformation definition*. Figure 2 presents three categories of mapping: one-to-one, one-to-many and many-to-one. This classification is based on the concept of similar structure and semantics between the elements of metamodels. Two or more elements from two metamodels present similar structure and semantics, if the target element(s) can represent the same information contained in the source element(s).

A one-to-one mapping is characterized by one element from a target metamodel that may represent the similar structure and semantics of one element from a source metamodel. A one-to-many mapping is characterized by a non-empty and non-unitary set of elements from a target metamodel that presents similar semantics to one element from a source metamodel. A many-to-one mapping is characterized by an element from a target metamodel that presents similar semantics to a non-empty and non-unitary set of elements from a source metamodel. This last mapping is not directly implemented in some languages such as ATL, but it can be simulated using a transformation rule that takes only one element from the source metamodel and determines the other elements based on the relationships between the first element with the others.

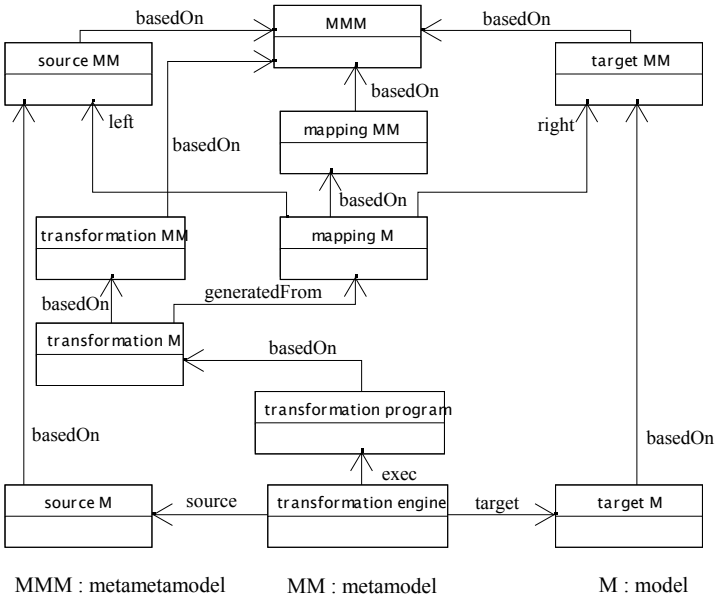

Fig. 1. Transformation Process in MDA

Several research projects have studied the specification of mapping between metamodels [16] [17] [18]. However, the ideas around mapping are not sufficiently developed to create efficient tools to enable automatic mappings.

3 Mapping

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 [19]. 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 [18] (for identifying structural and naming similarities between models) and on machine learning (for learning mappings) [20]. Other propositions enabling transformation definition have been based on graph theory [21]. 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 [7].

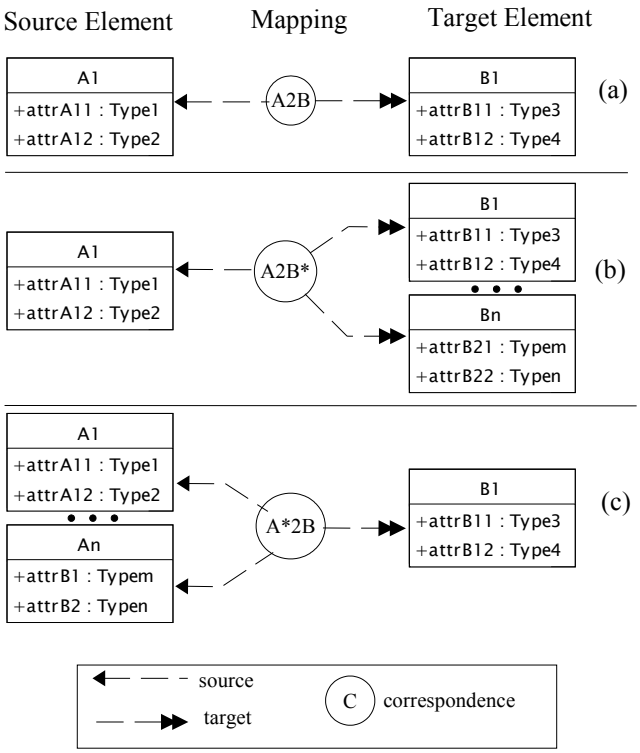

Fig. 2. Categories of mappings: (a)one-to-one, (b)one-to-many and (c) many-to-one

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, e.g. using Atlas Transformation Language (ATL) [14].

3.1 Foundation for mapping

Given $M_1(s)/M_a$, $M_2(s)/M_b$, and $C_{M_a \to M_b}/M_c$, where M_1 is a model of a system s created using the metamodel M_a, M_2 is a model of the same system s created using the metamodel M_b, and $C_{M_a \to M_b}$ is the mapping between M_a and M_b created using the metamodel M_c, then a transformation can be defined as the function $Transf(M_1(s)/M_a, C_{M_a \to M_b}/M_c) \to M_2(s)/M_b$. In this section, we aim to detail $C_{M_a \to M_b}/M_c$. In general, M_a, M_b and M_c are based on the same metametamodel which simplifies the mapping specification. For now, we can define $C_{M_a \to M_b} \supseteq \{M_a \cap M_b\}$, where $\cap$ is a binary operator that returns the elements of M_a and M_b which have equivalent structure and semantics.

The process of identifying and characterizing inter-relationships between metamodels is denominated *schema matching* [18]. In fact, *mapping* describes how two metamodels [5] are related to each other. So, *schema matching* results in a *mapping*. According to *model management algebra* [22], a *mapping* is generated using an operator called *match* which takes two metamodels as input and returns a *mapping* between them.

The identification of inter-relationships between metamodels is generally based on the structure of the metamodels. The structure of a metamodel is a consequence of relationships between its elements. These relationships relate two metamodel elements and have some characteristics such as kind. Generally, five relationship kinds can relate a model element to another model element [17]: *Association*, *Contains*, *Has-a*, *Is-a*, *Type-of*.

These relationship kinds can be formalized as follow:

- **Association**: $A(a, b)$ means a is associated with b.
- **Contains**: $C(c, d)$ means container c contains d.
- **Has-a**: $H(e, f)$ means e has an f.
- **Is-a**: $I(g, h)$ means g is an h.
- **Type-of**: $T(i, j)$ means i is a type of j.

In [17], the authors propose the five cross-kind-implications:

- if $T(q, r)$ and $I(r, s)$ then $T(q, s)$.
- if $I(p, q)$ and $H(q, r)$ then $H(p, r)$.
- if $I(p, q)$ and $C(q, r)$ then $C(p, r)$.
- if $C(p, q)$ and $I(q, r)$ then $C(p, r)$.
- if $C(p, q)$ and $I(q, r)$ then $H(p, r)$.

In [17], two models are defined equivalents *"if they are identical after all implied relationships are added to each of them until a fix point is reached"*. Applying these relationship kinds and cross-kind-implications to metamodels, they can also be simplified and compared between them to find equivalences and similarities.

3.2 Eclipse Modeling Framework (EMF)

Eclipse Modeling Framework (EMF) is a modeling framework and code generation facility for supporting the creation of tools and applications driven by models [10]. 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.

EMF is a framework which meets the requirements of Domain-Specific Languages (DSL) [3]. DSLs are defined by Steve Cook as *"languages that instead of*

[5] In our approach, we prefer employ the term metamodel in the definition of the term mapping.

being focused on a particular technological problem such as programming, data interchange or configuration, are designed so that they can more directly represent the problem domain which is being addressed" [3]. DSL presumes the existence of many metamodels, despite UML which is a general-purpose metamodel. MOF can also be used for creating DSLs, but OMG has focused its efforts on the adoption of UML and UML profiles, to the detriment[6] of specific metamodels.

Within the eclipse's philosophy, EMF is one more plug-in that can be used such as it is or extended by other plug-ins. In the context of Eclipse, plug-in is the mechanism for extending the basic architecture of Eclipse with things that have common or different purposes, where they share a common environment [23].

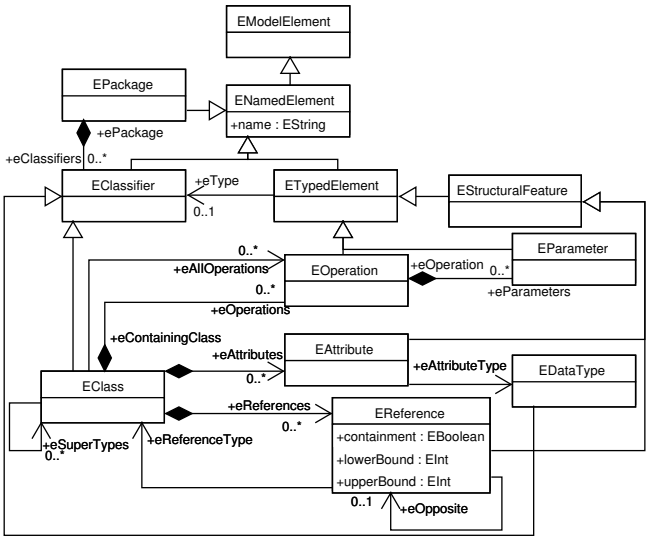

Fig. 3. Ecore metamodel (fragment)

Figure 3 presents a fragment of Ecore metametamodel [10]. Ecore has some points in common with MOF, but the former is more simple than the latter[7]. In addition, the serialization of a metamodel based on Ecore is more clear and simple than the similar metamodel based on MOF.

3.3 A metamodel for mappings

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

[6] The OMG has provided a limited number of metamodels, such as UML and Enterprise Distributed Object Computing (EDOC).

[7] This comparison is made using MOF 1.4. However, MOF 2.0 (i.e. Essential MOF - EMOF) is closer to Ecore.

– identification of what elements have similar structures and semantics to be mapped.
– explanation of the evolution in time of the choices taken for mapping one element into another element.
– bidirectional mapping. It is desirable, but is often complex [4].
– independence of model transformation language.
– navigation between the mapped elements.

Figure 4 presents a metamodel for mapping specification.

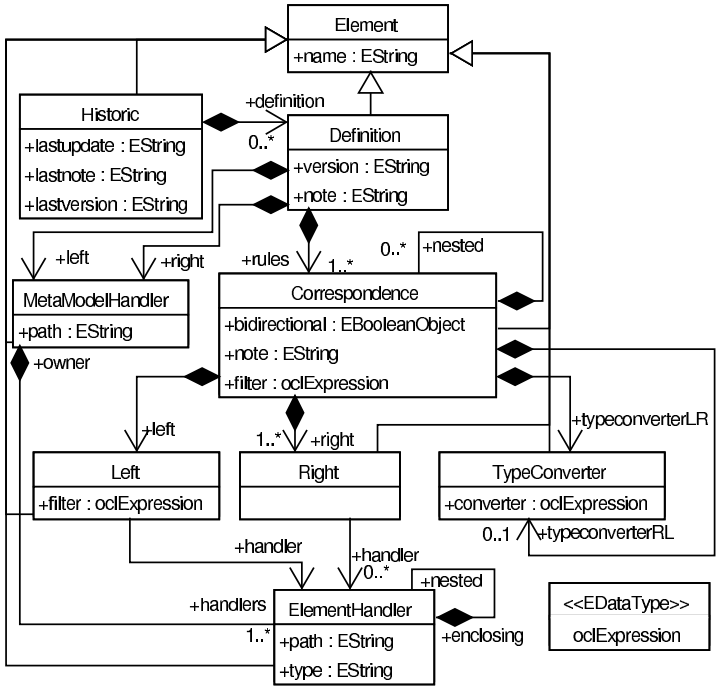

Fig. 4. Metamodel for Mapping Specification

In this metamodel, we consider that a mapping can be unidirectional or bidirectional. In unidirectional mapping, a metamodel is mapped into another metamodel. In bidirectional mapping, the mapping is specified in both directions. Thus, we prefer to denominate two metamodels in a mapping as left or right metamodels.

This metamodel presents the following elements:

– `Element` is a generalization for the other elements.
– `Historic` enables the explanation of the different choices taken for making the mapping. It has the date of the last update, a note, and the number of the last version, and a collection of `Definition`s.

- `Definition` is the main element and it contains all correspondences (i.e. `Correspondence`) between two metamodels (i.e. each correspondence has one `left` element and many `right` elements).
- `Correspondence` 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 `TypeConverter`s 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`.
- `Left` is used to identify the left element of a mapping.
- `Right` is used to identify the right element of a mapping.
- `MetaModelHandler` is used to navigate into a metamodel. It has the information necessary for accessing a metamodel participating in a mapping. A mapping is itself a model, and it must not interfere with the two metamodels being mapped.
- `ElementHandler` enables access to the elements being mapped without changing them.
- `TypeConverter` enables the type casting between a left and a right element. If one element of a left metamodel and another element of a right metamodel are equals, then the mapping is simple and direct. However, if one element of a left metamodel and another element of a right metamodel are similar, then the mapping is complex and it is achieved using type converter, i.e. a complex expression to adapt a left element to a right element.

4 A Plug-in for Eclipse

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, then it can also be transformed into another model. For example, a mapping model can be transformed into a transformation model.

4.1 An illustrative example

Figure 5 presents our proposed tool to support mapping specification. This tool is denominated Atlas Model Weaver. In fact, mapping is a form of weaver [24]. It was implemented as a plug-in for Eclipse that uses our proposed metamodel for mapping (see sections 3.1 and 3.3).

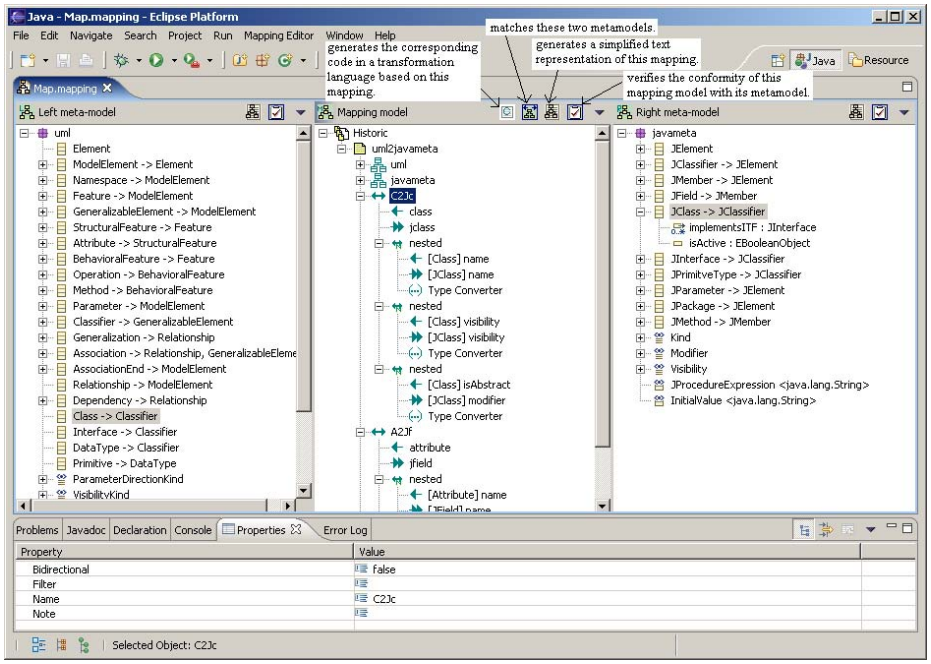

Fig. 5. Atlas Model Weaver (AMW): support for mappings

This tool presents a first metamodel on the left side, a mapping model in the center, and a second metamodel on the right. In this case, a fragment of the UML metamodel is mapped into a Java metamodel. In 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.

According to figure 5, `C2JC` maps `UML Class` into `Java JClass`, mapping also the attributes such as `name` and `visibility` from `Class` into the corresponding attributes of `JClass`.

This tool simplifies the creation of mapping specification and provide some other features such as:

- verification of the conformity between a mapping model and our proposed metamodel for mapping.
- transformation of mappings and metamodels into a simplified text representation which makes them more understandable.

– transformation of mappings specification into transformation definition.

This tool can export a mapping model as transformation definition. For the moment, we have implemented a generator for ATL[14], but we envisage creating generators to other model transformation languages such as YATL [15], in order to evaluate the power of our proposed metamodel for mapping. The resulting code in ATL of the mapping between UML (fragment) and this Java metamodel is presented in figure 6. This figure shows the ATL code fragment as `module uml2java;...`, the rules `C2JC` and `A2Jf`. The last rule transforms an `UML Attribute` into `Java JField`.

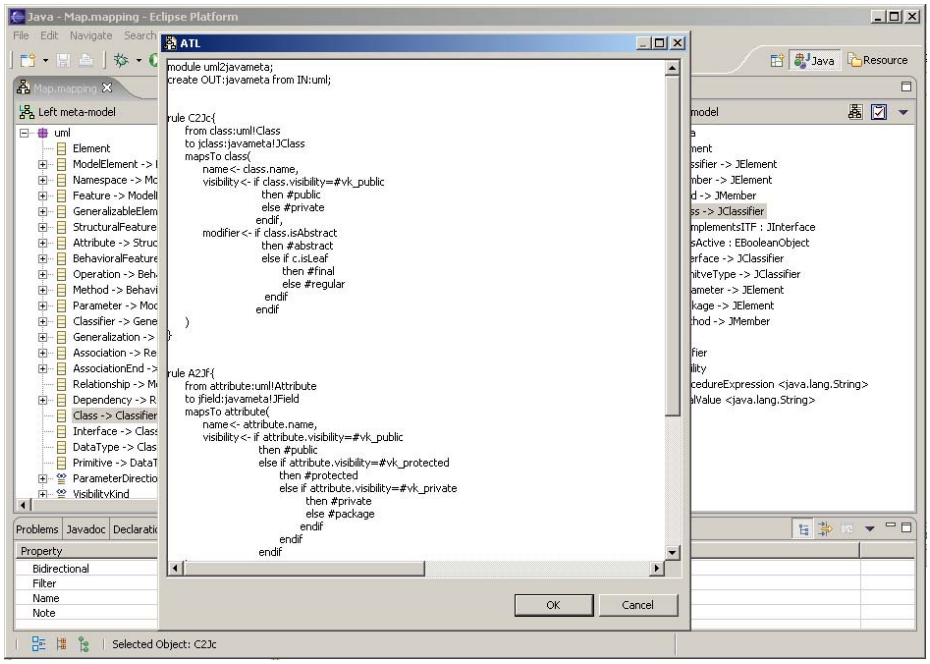

Fig. 6. The generated ATL code using the plug-in for mapping

The code in ATL is generated based on the mapping model. This tool leave the developer free to think only in the mapping between two metamodels, helping him to specify how the metamodels can be inter-related. Afterwards, it can generate the transform definition.

5 Conclusion

In this paper, we have presented our approach to determine mappings (i.e. correspondences) driven by models. A tool for mapping was also provided. However,

the *schema matching* [18], i.e. finding correspondences between models, was not sufficiently covered here, because in the current stage of our research, we concentrated our efforts on taking into account mapping driven by models, defining a metamodel and tool for mapping.

If transformation is the heart of MDA, and to transform a PIM into a PSM, it is necessary to find correspondences between metamodels, then *mapping specification* is also another important issue within MDA context.

MDA simplifies the development of software, but this is not obtained without effort. In fact, MDA allows developers to develop, maintain and evolve their systems encapsulating the complexity in models, model transformation, mapping, and so on. Thus, what a developer must know in order to develop software artifacts with MDA is only the tip of the iceberg, i.e. models.

In future research, we will develop more fully the *schema matching* and the *weaving technique* in order to integrate them also into our plug-in for Eclipse.

References

1. OMG: Model Driven Architecture (MDA)- document number ormsc/2001-07-01. (2001)
2. OMG: Request for Proposal: MOF 2.0 Query/Views/Transformations RFP. (2002) Available at http://www.omg.org/docs/ad/02-04-10.pdf.
3. Cook, S.: Domain-Specific Modeling and Model Driven Architecture. MDA Journal (2004) 1–10
4. Kent, S., Smith, R.: The Bidirectional Mapping Problem. Electronic Notes in Theoretical Computer Sciences **82** (2003)
5. Eclipse Project: Eclipse (2004) Available at http://www.eclipse.org.
6. Middleware Company: Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach. Technical report, The Middleware Company (2003) Available at www.middleware-company.com/casestudy.
7. Bézivin, J., Hammoudi, S., Lopes, D., Jouault, F.: Applying MDA Approach for Web Service Platform. 8th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004) (2004)
8. Interactive Objects Software GmbH and Project Technology, Inc.: 2nd Revised Submission to MOF Query / View / Transformation RFP. (2004)
9. QVT-Merge Group: Revised submission for MOF 2.0 Query/Views/Transformations RFP (ad/2002-04-10). (2004)
10. Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., Grose, T.J.: Eclipse Modeling Framework: A Developer's Guide. 1st edn. Addison-Wesley Pub Co (2003)
11. GMT: Generative Model Transformer Project (2004) Available at http://dev.eclipse.org/viewcvs/indextech.cgi/ checkout /gmt-home/description.html.
12. Java Community Process: Java^{TM}Metadata Interface(JMI) Specification, Version 1.0. (2002)
13. netBeans.org: Metadata Repository (MDR) (2004) Available at http://mdr.netbeans.org.
14. Bézivin, J., Dupé, 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)

15. Patrascoiu, O.: Mapping EDOC to Web Services using YATL. 8th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004) (2004)
16. Jan Hendrick Hausmann, S.K.: Visualizing Model Mappings in UML. Proceedings ACM 2003 Symposium on Software Visualization (SOFTVIS 2003) (2003) 169–178
17. Pottinger, R.A., Bernstein, P.A.: Merging Models Based on Given Correspondences. Proceedings of the 29th VLDB Conference (2003)
18. Rahm, E., Bernstein, P.A.: A Survey of Approaches to Automatic Schema Matching. VLDB Journal **10** (2001) 334–350
19. 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
20. Martin S. Lacher, G.G.: Facilitating the exchange of explicit knowledge through ontology mappings. 14th International FLAIRS conference (2001) 21–23
21. Agrawal, A., Levendovszky, T., Sprinkle, J., Shi, F., Karsai, G.: Generative Programming via Graph Transformation in the Model-Driven Architecture. OOPSLA 2002 Workshop on Generative Techniques in the Context of Model Driven Architecture (2002)
22. Bernstein, P.A.: Applying Model Management to Classical Meta Data Problems. Proceedings of the 2003 CIDR (2003)
23. Gamma, E., Beck, K.: Contributing to Eclipse: Principles, Patterns, and Plug-Ins. 1st edn. Addison-Wesley Pub Co (2003)
24. Bézivin, J., Jouault, F., Valduriez, P.: First Experiments with a ModelWeaver. OOPSLA 2004 - Workshop on Best Practices for Model Driven Software Development (2004)