TOWARDS A SEMI-AUTOMATIC TRANSFORMATION PROCESS IN MDA
Architecture and Methodology

Slimane Hammoudi[1], Wajih Alouini[2] and Denivaldo Lopes[3]
[1]*Ecole Supèrieure d'Electronique de l'Ouest(ESEO), 49009 Angers Cedex 01, France*

[2]*ISIG, Institut Supérieur d'Informatique et de Gestion, University of Kairouan, Tunisia*

[3]*Federal University of Maranhão (UFMA), CCET – DEE, São Luís - MA, Brazil*

Keywords: Model Driven Architecture, Transformation Language, Mapping Metamodel, Matching Metamodel, Semi-Automatic Transformation, Transformation Architecture and Transformation Methodology.

Abstract: Recently, Model Driven Engineering (MDE) approaches have been proposed for supporting the development, maintenance and evolution of software systems. Model driven architecture (MDA) from OMG (Object Management Group), "Software Factories" from Microsoft and the Eclipse Modelling Framework (EMF) from IBM are among the most representative MDE approaches. Nowadays, it is well recognized that model transformations are at the heart of these approaches and represent as a consequence one of the most important operations in MDE. However, despite the multitude of model transformation languages proposals emerging from university and industry, these transformations are often created manually. In this paper we propose in the first part an extended architecture that aims to semi-automate the process of transformation in the context of MDA. This architecture introduces mapping and matching as first class entities in the transformation process, represented by models and metamodels. In the second part, our architecture is enforced by a methodology which details the different steps leading to a semi-automatic transformation process. Finally, a classification of these different steps according to two main criteria is presented: how the steps are achieved (manual/automatic), and who is responsible for their achievement (expert, designer or software).

1 INTRODUCTION

The main motivation behind MDE (Bézivin, 2006) is to transfer the focus of work from programming to modeling by treating models as first class entities and consequently the primary artifacts of development. One of the most important aspects of the MDE approach is the explicit specification of business logic through Platform Independent Models (PIMs) and the flexibility to implement them on different target platforms via Platform Specific Models (PSMs). A specific platform can be any technology that supports the execution of these models, either directly or after translation to code. The PIM reflects the functionalities, the structure and the behavior of a system. The PSM is more implementation-oriented and corresponds to a first phase, binding of a given PIM to a given execution platform. In this context, designers and developers have to focus on modeling the problem domain and not on programming one possible (platform-specific) solution. There are nowadays several approaches based on MDE principles, the most well known being MDA (OMG, 2007) by OMG or "Software factories" by Microsoft (Dominguez, 2006). In the literature, several issues around MDE have been studied and subject of intensive research, e.g. modeling languages (Bézivin, 2004-1), model transformation languages (Jouault, 2006) (Bézivin, 2003) (OMG, 2005), mapping between metamodels (Hammoudi, 2005-2) (Lopes, 2005-1), and design methodologies (Almeida, 2006). Among these issues, model transformation languages occupy a central place and allow to define how a set of elements from a source model are analyzed and transformed into a set of elements of a target model.

However, these transformations are created manually, often a fastidious and error-prone task, and therefore an expensive process. These transformations consist of creating a set of rules involving, and in the same time merging mapping and transformation techniques between two metamodels. A semi-automation of the transformation process leads to a real challenge allowing many advantages: It enhances significantly the development time of transformation and decreases the errors that may occur in a manual definition of transformations. In (Hammoudi, 2005-1) (Lopes, 2005-2), we have initiated a first attempt towards this semi-automation. We have introduced an approach separating mapping specification from transformation definition, and have implemented this approach in a tool called MMT (Mapping Modeling Tool). In this first approach, a mapping specification was created manually to define the relationships between metamodels (i.e. equivalent metamodel elements), while transformation definition was generated automatically and contained the operational description of the transformation rules between models. In this paper, we propose to push the semi-automation process one step further by using matching techniques (Kappel, 2007) (Lopes, 2006-1, 2006-2), to provide semi-automatic mappings between two metamodels. The produced mappings could be adapted and validated by an expert for the automatic generation of a transformation model, as a set of transformation rules. In this paper, we present an extended architecture of the transformation process in the context of MDA. This architecture introduces the matching and mapping components as two other important operations in the transformation process. Based on this architecture, a methodology presents the different steps of the semi-automatic transformation process.

This paper is organized as follows: section 2 introduces the MDE/MDA approach and presents its most common scenario of transformation in MDA. Section 3 presents an extended architecture for a semi-automatic transformation process and discusses the matching and mapping metamodels as two important components in this process. Section 4 presents the methodology which in the first part, starts with two metamodels source and target, and details the different steps for generating transformation rules semi-automatically.
The second part of this methodology consists of applying the transformation rules to a PIM model to generate an equivalent PSM on a given specific platform. Section 5 reviews the main steps of the transformation process according to two main criteria: how the steps are achieved and who is responsible for their achievement. Finally, section 6 concludes our work and presents some final remarks and future perspectives.

2 MDE: OVERVIEW AND TRANSFORMATION PROCESS

At the beginning of this century, software engineering needs to handle software systems that are becoming larger and more complex than before. Object-oriented and component technology seem insufficient to provide satisfactory solutions to support the development and maintenance of these systems. To adapt to this new context, software engineering has applied an old paradigm, i.e. models, but with a new approach, i.e. Model Driven Engineering. In this new global trend called Model Driven Engineering, MDA is a particular variant.

2.1 Model Driven Architecture (MDA)

MDA is based on standards from the OMG; it proposes an architecture with four layers (OMG, 2001): metametamodel, metamodel, model and information (i.e. an implementation of its model). Figure 1a presents the basic Metamodeling architecture of MDA with the relationships between different levels of models. In this approach, everything is a model or a model element. In level M0, a real system is *representedBy* a model in level M1, and a model in level M1 *conformsTo* a metamodel in level M2. These two important relationships of MDA are discussed in (Bézivin, 2005).

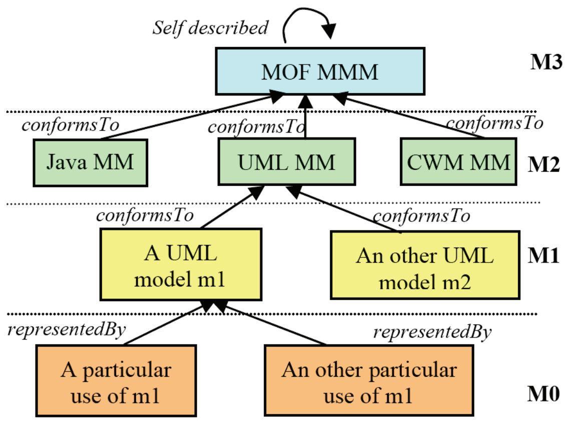


Figure 1a: Architecture with four Meta-layers.

In level M3, a metametamodel is a well-formed specification for creating metamodels such as the Meta Object Facility (MOF), a standard from the OMG. In level M2, a metamodel is a well-formed specification for creating models. In level M1, a model is a well-formed specification for creating software artifacts. In level M0, an operational example of a model is the final representation of a software system.

According to this architecture, we can state the existence of few metametamodels such as MOF and Ecore (Budinsky, 2003; Eclipse, 2004), several metamodels such as UML, UEML (UEML, 2003) and EDOC (OMG, 2003), more models describing real life applications such as a travel agency, and finally infinite information such as the implementation of this travel agency model using Java or C#. This organization is well known in programming languages where a self-representation of EBNF notation could be obtained easily in some lines. This notation allows defining infinity of well-formed grammars. A given grammar, e.g. the grammar of the C language, allows defining the infinity of syntactically correct C program. Several different executions could be realized from a C program. Besides the MDA architecture with four layers, figure 1.b illustrates the primary idea around the development of software systems using MDA. This figure involves two kinds of transformation (represented by arrows): model to model transformation and model to code transformation.

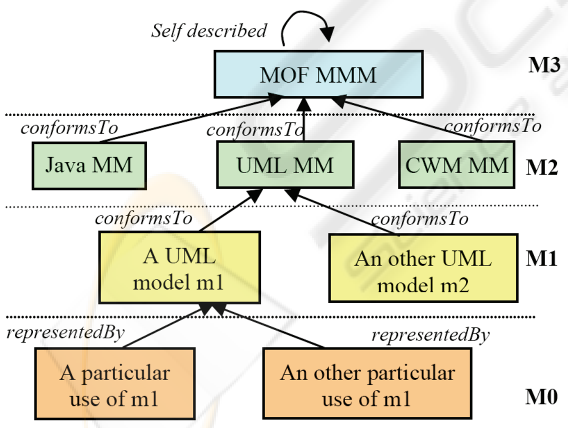


Figure 1b: MDA: Primary Idea.

The development is based on the separation of concerns (e.g. business and technical concerns), which are afterwards transformed between them. So, business concerns are represented using Platform-Independent Model (PIM), and technical concerns are represented using Platform-Specific Model (PSM). According to figure 1.b, PIM (e.g. a UML business model) is transformed into PSM (e.g. based on Web Services), which could be refined in other PSMs (e.g. based on Java and JWSDP[1]), until exported as code, configuration files, and so on. Analyzing each type of model, we can deduce that a PIM and PSM have a different life cycle. PIMs are more stable over time while PSMs are subject to frequent modification. So, this approach preserves the business's logic (i.e. PIMs) against the changes or evolutions of technologies (i.e. PSMs).

2.2 Model Transformation in MDA

It is well recognized today that model transformation is one of the most important operations in MDA. The following definition of model transformation largely shared in the community is provided in (Kleppe, 2003): "*A Transformation is the automatic generation of a target model from a source model, according to a transformation definition. A transformation definition is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language. A transformation rule is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language*".

The working group on model transformation of the Dagstuhl seminar (Bézivin, 2004-2) suggests that this should be generalized, in that a model transformation should also be possible with multiple source models and/or multiple target models. In our discussions here we are concerned with a transformation that takes a platform-independent model and transforms it in to a platform-specific model. In the context of the basic four levels Metamodeling architecture of MDA, various scenarios of model-to-model transformation have been identified. Figure 2 presents the most common scenario of these transformations, which is compatible with the MOF2.0/QVT standard (OMG, 2005). Each element presented in Figure 2 plays an important role in MDA. In our approach, MOF is the well-established metametamodel used to create metamodels. 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

[1] Java Web Service Development Pack.

source into the target metamodel. The transformation rules are based on the transformation language, such as the standard QVT.

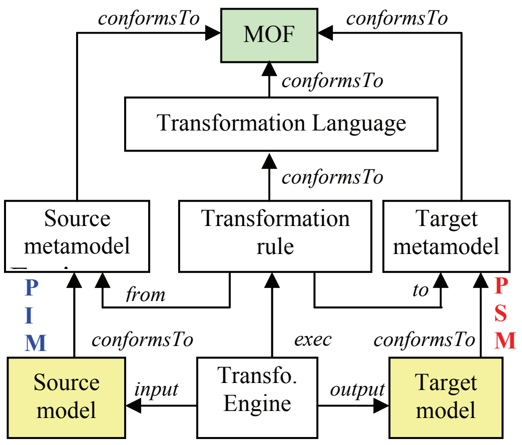


Figure 2: Model Transformation in MDA: from PIMs to PSMs (Lopes, 2005-1; Jouault, 2006)

The transformation engine takes the source model, executes the transformation rules, and produces the target model as output. Using a unique formalism (e.g. MOF) to express all metamodels is very important because this allows the expression of different sorts of relationship between models based on separate metamodels. Transformations are one important example of such a relationship, but there are also others (Bézivin, 2005) like model weaving, model merging, model difference, metamodel alignment, etc. Thus, given ma(s)/Ma and mb(s)/Mb, where ma is a model of a system s created using the metamodel Ma, and mb is a model of the same system s created using the metamodel Mb, then a transformation can be defined as follow:

$$m_a(s)/Ma \rightarrow m_b(s)/Mb.$$

When *Ma* and *Mb* conform to the same metametamodel (e.g. MOF), the transformation may be expressed in a transformation language such as QVT. There are a number of general challenges in the definition of a language for model transformation. Some of these challenges are that it must be expressive and provide complete automation, be unambiguous, and Turing complete for it to be generally applicable. The recent standardization effort by OMG (OMG, 2007) and many industrial and academic efforts in this area will allow advancement on these challenges.

Before introducing our architecture for a semi-automatic transformation process, we would like to recall the two main problems concerning the main scenario of the MDA transformation process illustrated by figure 2 and that have motivated our current work:

- The first problem concerns the creation of “transformation rules” between metamodels which, as mentioned in the introduction, are often created manually, generally a fastidious and error-prone task, and therefore expensive process.
- The second problem concerns the specification of these “transformation rules”, which merge together techniques of mappings and transformations without explicit distinction between them. That is to say, the specification of correspondences between elements of two metamodels and the transformation between them are grouped in the same component at the same level. As we have already discussed in (Hammoudi, 2005-1), an explicit distinction between techniques of mapping and transformation could be very helpful in the whole MDA process of transformation. Moreover, the separation between the mappings and transformations parts is a first step towards a semi-automatic process, since mappings could be automatically generated by a matching process.

3 AN ARCHITECTURE FOR THE TRANSFORMATION PROCESS

Figure 3 illustrates our proposal of an extended architecture for the transformation process in MDA, allowing a semi-automatic generation of transformation rules and the semi-automatic generation of a target model from a source model. The three main operations of our approach are: Matching, Mapping and Transformation. All the components linked to these operations, and their relationships, are presented in figure 3 based on the four level MDA metamodeling architecture.

The matching operation is the process that produces the mappings between two metamodels. Generally, this task implies a search of equivalent or similar elements between two metamodels. In the database domain, this task is called schema matching. In our context, a matching model (*Matching M*) takes two metamodels designed by source and target (representing respectively a PIM and a PSM metamodel), and produces a mapping model (*Mapping M*).

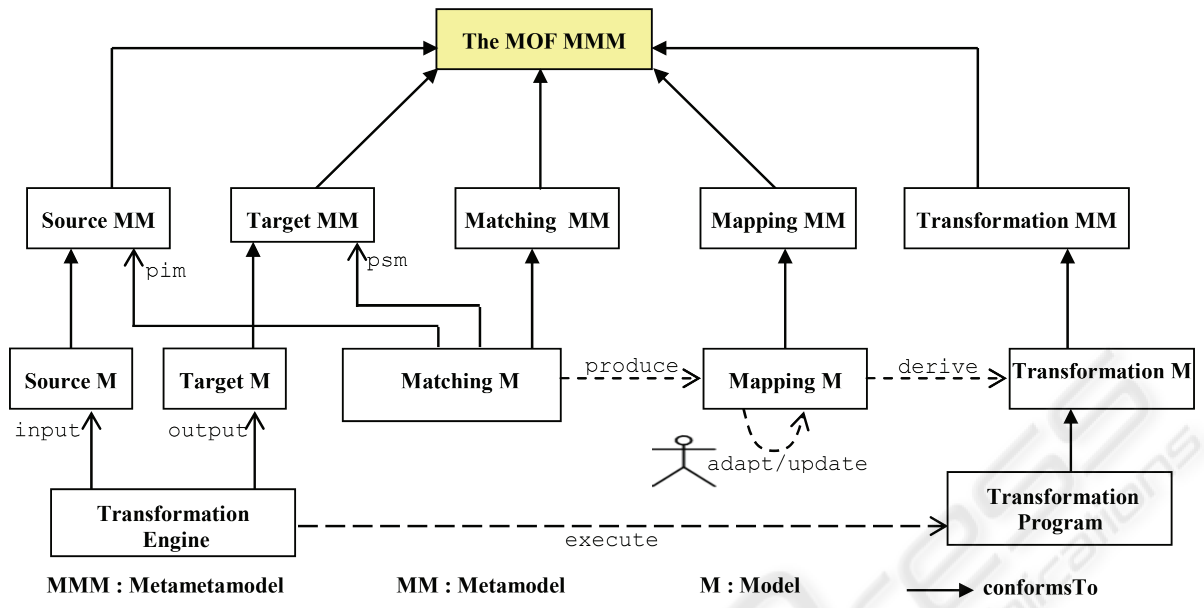


Figure 3: Architecture for a semi-automatic transformation process in MDA.

The matching model conforms to a metamodel of matching *(Matching MM)* which implements techniques that consist of finding semantically equivalent modeling concepts between two metamodels. Thus, different kinds of relationships between metamodel elements are discovered using the metamodel of matching. The relationships between metamodel elements are saved in a mapping model which conforms to a mapping metamodel *(Mapping MM*). This metamodel defines the different kinds of links (relationships) that could be generated by the matching model. Each kind of link corresponds to one transformation pattern specified in the transformation model described hereafter. Given that no generic matching solution exists for different metamodels and application domains, it is recommended to give the human expert the possibility to check the obtained mappings, and, if necessary, update or adapt it. This is the only step in the whole process, in which the expert intervenes to complete and/or validate the obtained results. Finally, a transformation model *(Transformation M),* in conformance to its transformation metamodel (*Transformation MM*), is derived automatically from a mapping model. A transformation model is basically represented by a set of rules that states how elements from source metamodel are transformed into elements of target metamodel. These rules are expressed in a transformation language based on MDA standards (OCL, MOF). This language, such as the standard QVT is described by a metamodel as a general formalism and abstract syntax for model transformation in MDA. Frequently, the transformation model is completed by some information such as those concerning the execution environment, and produces a transformation program ready for the execution. This last part is often achieved by a designer (or software engineer) who implements a business model in a specific platform. Finally, a transformation engine takes a source model as input, and executes the transformation program to transform this source model into the target model.

According to our approach and architecture, the matching and transformation components are executable programs that take models or metamodels as parameters, while the mapping component is a set of relationships between elements of source and target metamodels. Concerning the mapping component, we have proposed in a previous work a generic metamodel and implemented it in a tool called MMT (Lopes, 2005-1). In this first approach, the mapping model between two metamodels, was supposed to be defined manually by an expert. From this mapping model, a transformation model represented by a set of rules is generated automatically.

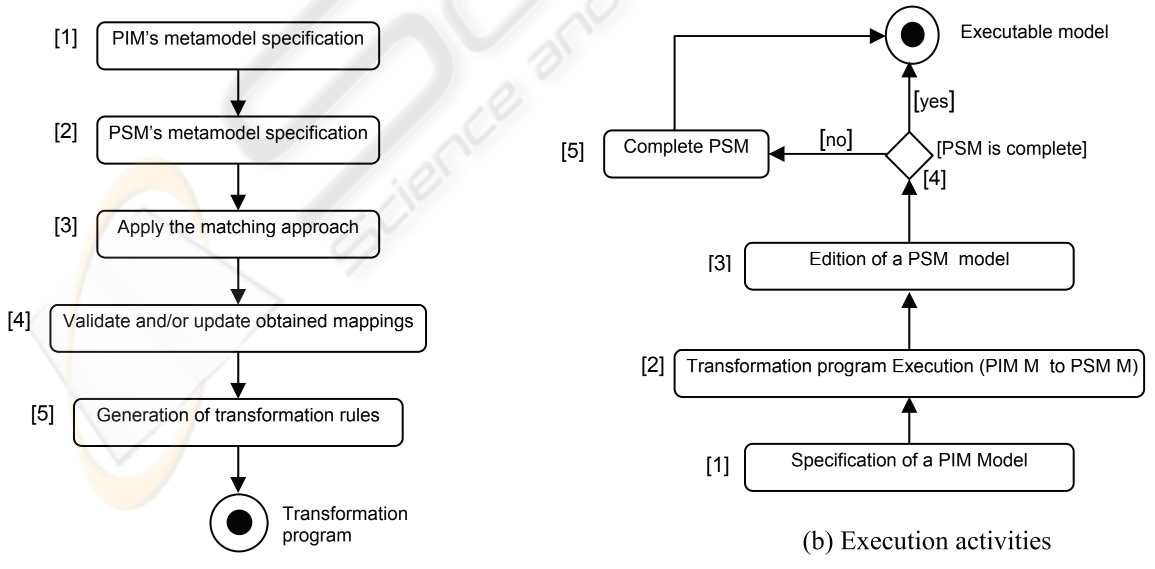
4 A METHODOLOGY FOR A SEMI-AUTOMATIC TRANSFORMATION PROCESS

We intend through our methodology to enforce the new architecture for the transformation process presented above and to discuss the implementation of the main steps. Figure 4 illustrates the main steps of a methodology allowing a semi-automatic transformation process. These steps are represented by two activities diagrams. The first activity diagram (a), on the left side, shows the steps followed by a certain domain expert who starts with the specification of the two metamodels source and target and follows the process until the generation of transformation rules and an executable transformation program. The second diagram (b), on the right side, illustrates the steps of a designer who specifies a business model of a given application based on a PIM metamodel and generates automatically, by using a transformation program, an implementation of this business model on a given specific platform. This distinction between expert user and designer is discussed in (Gavras, 2004) where a classification of MDA technology users is presented. Moreover, in any MDA based project, the distinction between preparation activities and execution activities is essential. The first activities are performed by the expert, while the second activities are mainly performed by designers or software engineers.

The first goal within such a methodology is to introduce the matching process into the OMG's Model-Driven Architecture (MDA) approach in order to increase the degree of automation of the transformation process. This requires the reduction of human expert manual tasks by the rational choice among the plethora of existing works on matching algorithms. These algorithms have a high applicability to the problem of useful automatic mapping production.

For this purpose, we propose a methodology based on MDA standards. All the metamodels, source and target, as well as transformations, are based on the same metametamodel "Meta Object Facility" ("MOF 2.0). As noted previously, it is clear that using the same metametamodel, i.e. MOF, will generally facilitate mapping discovery between metamodels. However, it should be noted that we do not claim that this methodology is exhaustive, but it traces accurately the essential phases of the transformation process including the matching process.

In the next section, we outline the content for each phase of the methodology and describe what is to achieve in every step. In accordance to figure 4, we describe the steps of a domain expert leading to a generation of a transformation program in section 4.1 (preparation activities), while section 4.2 presents the steps followed by a designer (execution activities) implementing its business model into a specific platform


[1] PIM's metamodel specification
[2] PSM's metamodel specification
[3] Apply the matching approach
[4] Validate and/or update obtained mappings
[5] Generation of transformation rules
Transformation program
Executable model
[yes]
[no]
[5] Complete PSM
[PSM is complete]
[4]
[3] Edition of a PSM model
[2] Transformation program Execution (PIM M to PSM M)
[1] Specification of a PIM Model

(a) Preparation

(b) Execution activities

Figure 4: Methodology for a semi-automatic transformation process.

4.1 Generation of a Transformation Program

This first part of the transformation process groups five steps leading to an executable transformation program. There follows, the presentation of each step in this first part:

4.1.1 PIM's Metamodel Specification

This phase aims to define the appropriate PIM metamodel for a given application domain. Since MDA allows the separation of concerns between the business aspect and the technology, the PIM metamodel represents business logic without taking care of technological features. Thus, the models in conformance to PIM metamodel ignore all the platform specific details. PIMs metamodels are represented completely or partly by MDA metamodels standards such as UML or EDOC. Basically, there are two possibilities to create new metamodels in the context of MDA:

- Using the concept of profiles to extend the UML metamodel in order to take into account particular semantics of a given system.
- Creating new metamodels based directly on the MOF or Ecore metamodel

4.1.2 PSM's Metamodel Specification

This phase aims to define and specify the appropriate PSM metamodel. PSM are lower levels than PIMs as they must adhere to specific constraints imposed by the target platform, i.e. the platform in where the application will be implemented. The OMG afford compliant tools to specify PSM metamodels. In this way, UML profiles, as official OMG specifications, could also serve to precise specific constraints for a given platform. As samples of PSM, we quote the Web Service Description Language (WSDL) metamodel for the web service technology and the relational metamodel for the relational databases. Once these two first phases are accomplished, the matching process can be applied.

4.1.3 Application of the Matching Process

This phase aims to select the relevant match algorithms or matchers in order to generate matches (mappings). The Matching process could be represented by a model management operation called *match* (Bernstein, 2003). This operation takes two metamodels M1 and M2 as input and produces Mapping model Mm as output. M1 and M2, respectively, conform to the corresponding metamodels MM1 and MM2. Mm conforms to the metamodel of mapping MMm. The "→" operator represents the "conformsTo" relationship.

```
Mm → MMm = match (M1 → MM1, M2 → MM2)
```

Ideally, the mapping model is represented graphically. In fact, the graphical mapping model provides a higher level view than the lexical counterpart in order to present clear correspondences and to make it easier to check that a correspondence is valid.

The use of multiple matchers may be required (Dimitris, 2006). The choice of the appropriate matching approach to use is based on the examination of the PIM and PSM metamodels characteristics. This decision requires the specification and the definition of the appropriate criteria, essentially the application domain and modeling language. In the literature, several schema matching approaches have been proposed (Sun, 2003). Each schema matching approach has its own characteristics that were grouped by taxonomy. In addition, each approach has been evaluated through match quality measures (Lopes, 2006). Finally, It is ambitious to be certain that the obtained mappings from the matching process are exhaustive and faultless, thus presenting the need of the human expert to intervene as presented in the next phase.

4.1.4 Validate and Update obtained Mappings

The human expert is in charge of this phase. Given that it is extremely optimistic to assert that all the mappings are obtained as a result of the previous phase or that all the matching techniques exist and are utterly effective, it is fairly rational to provide the human expert with interactive tools in performing the required correcting task. These tools, allow the expert to accept, discard or modify the obtained mappings, furthermore, to specify correspondences which the matcher was unable to find. A graphical interface tool is able to present mappings between the source metamodel PIM and the target metamodel PSM and to define all the dependencies between mappings. This phase is of a great consequence for the transformation of mappings into transformation program such as QVT rules. Thus, the expert has to be as accurate as possible. Once the mappings are validated by the expert, the next phases enchain by an actual generation of the transformation rules for which automation is entirely possible.

4.1.5 Generation of Transformation Rules

This phase aims to generate automatically transformation rules from mappings and formatting them into a transformation model in order to be used by the transformation program which transforms the PIM model into the PSM model. The mapping model obtained in the previous step should be sufficiently defined to allow an automatic generation of transformation model. This transformation model, which consists of a set of transformation rules, is a model structured in conformance to the OMG's standard MOF2.0-QVT. These QVT transformation rules are either correspondence rules (declarative approach) or construction rules (imperative approach). A hybrid approach, containing both correspondence rules, and construction rules is also possible (Blanc, 2005). The automatic generation of transformation rules in our approach is due to the explicit distinction between mapping and transformation components. This distinction is stated more in ontologies field where they claim the importance to distinguish between ontology translation and ontology mapping (Dou, 2003):

"It's important to distinguish ontology translation from ontology mapping, which is the process of finding correspondence (mappings) between the concepts of two ontologies. If two concepts correspond, they mean the same thing, or closely related things. The mappings should be expressed by some mapping rules, which explain how those concepts correspond. Obviously, ontology translation needs to know the mappings of two ontologies first, then it can use the mapping rules."

4.2 Generation of a Platform Specific Model

This second part in the transformation process groups also five steps leading to an executable platform specific model. This part is mainly achieved by a designer who, after defining his business model would like to implement it on a specific platform.

4.2.1 Specification of a PIM Model

Using a PIM metamodel, a designer defines his business model focusing only on the business logic without taking into account implementation considerations. In this step the designer may use, for example, different UML diagrams which will lead to a final class diagram ready for the implementation on a given platform. Several tools such as Poseidon may help the designer during this step. This step corresponds typically to the definition of a conceptual model in the context of database design.

4.2.2 Transformation Program Execution (from PIM Model to PSM Model)

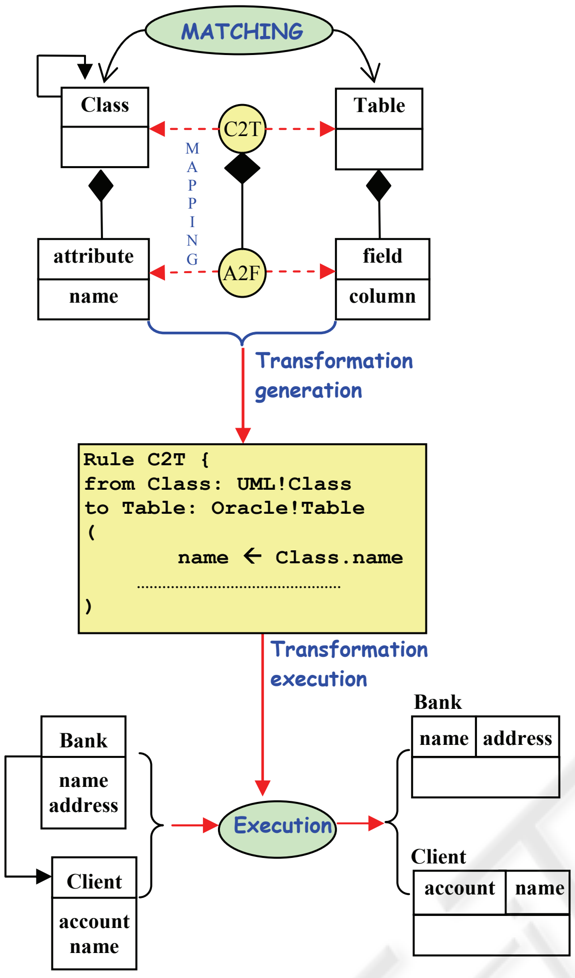
The transformation program obtained in the first part is used here. It takes a PIM model as input and produces the equivalent PSM model as output. The transformation engine, which implements the transformation program, reads the source model, applies the rules to the source model and produces the corresponding target model. All the input and produced models are often expressed in XMI format.

4.2.3 From PSM Model to an Executable Model

In this section we group the last three steps of the second part, starting with a first binding of a PSM model and leading to an executable model on a given platform. The PSM model produced from the previous step represents a first version of a platform specific model which usually should be completed by information very specific to the target platform to produce a final executable model. So, the completeness of the PSM obtained is to be verified. In the case of effective completeness the transformation task is successfully accomplished, otherwise, the designer will complete it manually.

4.3 Illustrative Example

Figure 5 illustrates the whole transformation process discussed here. This figure presents a simple example involving the main concepts of matching, mapping and transformation according to our point of view. In this example a fragment of a UML metamodel is matched with a fragment of a relational database metamodel. The result of this matching is a mapping model, defined here using a graphical formalism that we have introduced (Lopes, 2005-1), to specify mappings between elements of two metamodels, which are MOF compliant. This graphical formalism is very useful to specify mappings in a declarative manner and at a high level of abstraction. However, it is clear that this formalism is not sufficient to express complex mappings. Thus, a textual language must sometimes be used to complete it. OCL (Object Constraint Language) has been used in several experimentations of our approach (Lopes, 2005-1). From a mapping model (validated by an expert), a transformation program represented by a set of rules is generated automatically. This program takes a source UML model and produces an implementation of this model as a relational database.



```
Rule C2T {
from Class: UML!Class
to Table: Oracle!Table
(
        name ← Class.name
        ..........................................
)
```

Figure 5: Transformation Process: from metamodel matching to model transformation.

5 DISCUSSION

Table 1 resumes the main steps in the process of transformation focalizing on two important aspects: how they are achieved (manual or automatic) and what is responsible for their achievement (expert, designer, software). Concerning the software aspect, the transformation process involves three main programs which are at the heart of the semi-automatic transformation process:

- *Matching program*: implements the matching metamodel and produces a first version of a mapping model between two metamodels source and target.
- *Generation program*: takes a mapping model validated (updated) by an expert, and generates automatically a transformation program as a set of rules.
- *Transformation program*: takes a source model defined by a designer and produces an equivalent target model on a specific platform.

Table 1: Main steps and actors in the semi-automatic transformation process.

	How	**Who**
PIM & PSM metamodels	manual (graphical	expert user
Matching process	automatic	matching program (expert user)
Validation/update mappings	manual (graphical editor)	expert user
Transformation rules generation	automatic	generation program (expert user)
PIM model	manual (graphical	designer
Transformation rules execution	automatic	transformation program
Final PSM	manual (textual editor)	designer

6 CONCLUSIONS AND FUTURE WORK

In this paper, we have presented our approach for a semi automatic transformation process in MDA through an architecture and a methodology. We argue that a semi-automatic transformation process will be a great challenge in MDA approach as there is not yet a complete solution that automates the development of model transformation. A semi-automatic process will bring many advantages: it accelerates the development time of transformations; it reduces the errors that may occur in manual coding; it increases the quality of final transformation code. The key principle for this process is to consider mapping and matching metamodels as first class entities in MDA. In our previous work (Lopes, 2006-1; Lopes, 2006-2), we have proposed a first algorithm for metamodel matching based on set theory. In future work, we will study and implement other metamodel matching algorithms, e.g. algorithms based on machine learning and heuristics. An evaluation between different metamodel matching algorithms can help to capture the fundamental characteristics of each approach. Thus, a hybrid algorithm can be proposed as the composition of the best characteristics of each individual algorithm.

REFERENCES

Almeida, A.J.P., 2006. Model-driven design of distributed applications. PhD thesis, University of Twente. ISBN 90-75176-422

Bernstein, P.A, 2003. Applying Model Management to Classical Meta Data Problems. In *CIDR'03, Proceedings of the Conference on Innovative Data Systems Research.* CIDR.

Bézivin, J., 2005. On the Unification Power of Models. In *Software and Systems Modeling*, 4(2):171-188,

Bézivin, J., Dupé, G., Jouault, F., Pitette, G., Rougui, J.E., 2003. First Experiments with the ATL Model Transformation Language: Transforming XSLT into Xquery. In *2nd OOPSLA*, *Workshop on Generative Techniques in the context of Model Driven Architecture*.

Bézivin, J., Hammoudi, S., Lopes, D., Jouault, F., 2004. Applying MDA Approach for Web Service Platform. In *EDOC 2004, 8th IEEE International Enterprise Distributed Object Computing Conference.*

Bézivin J., Heckel, R., 2004. Language Engineering for Model-driven Software Development. In *Dagstuhl Seminar*.

Bézivin, J., Lammel, R., Saraiva, J. Visser, J., 2006. Model Driven Engineering: An Emerging Technical Space. *In GTTSE 2006, Generative and Transformational Techniques in Software Engineering.*

Blanc, X., 2005. *MDA en action, Ingénierie logicielle guidée par les modèles*, EYROLLES. Paris, 1st edition

Budinsky, F., Steinberg, D., Merks, E. , Ellersick, R., Grose, T. J., 2003. *Eclipse Modeling Framework: A Developer's Guide,* Addison-Wesley Pub Co, 1st édition.

Dimitris, M., Dimitris, P., 2006. A Tool for Semi-Automated Semantic Schema Mapping: Design and Implementation. In *Caise'06*, *International Workshop Data Integration and the Semantic Web*.

Dominguez, K., Pérez, P., Mendoza, L., Grimán, A., 2006. Quality in Development Process for Software Factories According to ISO 15504 , In *CLEI electronic journal, [http://www.clei.cl, Vol. 9 Num. 1 Pap. 3: June 2006]*.

Dou D, McDermott D, and Peishen Qi 2003 Ontology Translation on the Semantic Web. In Proc. Int'l *Conf. on Ontologies, Databases and Applications of Semantics (ODBASE2003). LNCS **2888.***

Eclipse Tools Project. Eclipse Modeling Framework (EMF) version 2.0, 2004. Available on http://www.eclipse.org/emf.

Gavras. A, Belaunde.M, Pires L.F, Almeida.J.P, "Towards an MDA-based Development Methodology for Distributed Applications" , *Proceedings of the 1st European Workshop on Model-Driven Architecture with Emphasis on Industrial Applications (MDA-IA 2004)*

Hammoudi, S., Janvier, J., Lopes, D., 2005. Mapping Versus Transformation in MDA: Generating Transformation Definition from Mapping Specification, *In VORTE 2005, 9th IEEE International Enterprise Distributed Object Computing Conference.*

Hammoudi, S., Lopes, D., 2005. From Mapping Specification to Model Transformation in MDA: Conceptualization and Prototyping. In *MDEIS'2005, First International Workshop*.

Jouault, F., 2006. *Contribution à l'étude des langages de transformation de modèles*, Ph.D. thesis, University of Nantes.

Kappel, G., Kargl, H., Kramler, G., Schauerhuber, A., Seidel, M., Strommer, M., Wimmer, M., 2007. Matching Metamodels with Semantic Systems – An Experience Report. In *BTW 2007, Datenbanksysteme in Business, Technologie und Web.*

Kleppe, A., Warmer, J., Bast, W., MDA Explained: *The Model Driven Architecture: Practice and Promise. Addison-Wesley*, 1st édition, August 2003.

Lopes, D., 2005. *Study and Applications of the MDA Approach in Web Service Platforms*, Ph.D. thesis (written in French), University of Nantes.

Lopes, D., Hammoudi, S., Bézivin, S., Jouault, F., 2005. Generating Transformation Definition from Mapping Specification : Application to Web Service Platform. In *CAISE'05, Proceedings of the 17th Conference on Advanced Information Systems Engineering*.

Lopes, D., Hammoudi, S., Abdelouahab, Z., Schema Matching in the context of Model Driven Engineering: From Theory to Practice. *Editores Tarek Sobh and Khaled Elleithy, Advances and Innovations in Systems, Computing Sciences and Software Engineering, Springer, 2006*

Lopes, D., Hammoudi, S., De Souza, J., Bontempo, A., 2006. Metamodel matching : Experiments and comparison. In *ICSEA'06, Proceedings of the International Conference on Software Engineering Advances*.

OMG, 2001. Model Driven Architecture (MDA)- document number ormsc/2001-07-01. (2001).

OMG, 2003. UML Profile for Enterprise Distributed Object Computing Specification. OMG Adopted Specification ptc/02-02-05.

OMG, 2005. MOF QVT Final Adopted Specification, OMG/2005-11-01.

OMG, 2007. MDA Specifications. Available in http://www.omg.org/mda/specs.htm#MDAGuide.

Sun, X. L. and Rose, E.. Automated Schema Matching Techniques: An Exploratory Study. Research Letters in the Information

Unified Enterprise Modeling Language (UEML), 2003. Available on http://www.ueml.org.