Semi-Automatic Generation of Transformation Rules in Model Driven Engineering: The Challenge and First Steps

W. Alouini\textsuperscript{1}, O. Guedhami\textsuperscript{2}, S. Hammoudi\textsuperscript{3}, M. Gammoudi\textsuperscript{4}, D. Lopes\textsuperscript{5}

\textsuperscript{1} FST, Faculté des Sciences de Tunis, Tunis, Tunisia.
\textsuperscript{2} ISIG, Institut Supérieur d’Informatique et de Gestion, Kairouan, Tunisia.
\textsuperscript{3} ESEO, Ecole supérieure de l’Ouest Angers, France.
\textsuperscript{4} ESSAIT, Ecole Supérieure des Statistiques de Tunis, Tunisia.
\textsuperscript{5} Federal University of Maranhão, São Luís, Brazil

\{wajih.alouini, oumaima-guedhami\} @yahoo.fr
slimane.hammoudi@eseo.fr
mohamed.gammoudi@fst.rnu.tn
denivaldo.lopes@gmail.com

Abstract

Research and practice for Model Driven Engineering (MDE) have significantly progressed over the last decade for dealing with the increase of complexity within systems during their development and maintenance processes by raising the level of abstraction using models as information storage. New significant approaches, mainly Model Driven Architecture (MDA) defined at the OMG (Object Management Group), “Software Factories” proposed by Microsoft and the Eclipse Modeling Framework (EMF) from IBM, are born and have been experimented. As models grow in use for developing systems, transformation between models grow in importance. University and industry are seeking for effective and efficient ways to treat transformation as first-class assets in MDE. In order to produce new and more powerful transformations, we argue that the semi-automatic generation of transformation rules is an important challenge in future MDE development to make it easier, faster, and cost-reduced process. In this paper we propose 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. We will introduce and discuss briefly two main operations “adaptation” and “derivation” which we consider as core techniques for a semi-automatic transformation process in MDA, along with the first two main techniques of matching and mapping. We will introduce a still theoretical idea already adopted in schema matching field of combining many techniques of matching taken from the literature in order to improve the quality of the obtained mappings. This combination will raise the semi-automation level and thereby significantly contribute to results of higher quality. Finally, an illustrative example is presented and related works are discussed.

Keywords: model driven architecture, transformation language, mapping metamodel, matching metamodel, semi-automatic transformation, transformation architecture and matching techniques, combining matchers.
1. Introduction

The Model Driven Engineering (MDE) initiative treats models as the available internal information during software development process. Hence, the raising of level of abstraction using models, which are closer to the problem domain, as proper artifacts contributes to the tackling of the increase of complexity within the software and its development process. As a consequence, the programming process is not yet the primary focus of work but treating models goes up to a first interest in building software systems. Academy and industry have provided several MDE based approaches, among which the most well known are MDA [1] by OMG, “Software factories” by Microsoft [2] and the Eclipse Modeling Framework (EMF) from IBM [3]. In the literature, several issues around MDE have been studied and subject of intensive research, e.g. modeling languages [4][5], model transformation languages [6][7], mapping between metamodels [8][9], Scalability and reuse of model transformations [10], Maintenance and evolution of model transformations [11] and Model-driven development methodologies, approaches, and languages with a focus on transformations [12]. A key element of the MDA (Model Driven Architecture) [13] is the notion of model transformation which plays a vital role all over the life cycle of the software development process. A model transformation is a set of transformation rules and techniques operating on a source model to produce another model, the target model.

Several model transformation languages have emerged allowing to specify and execute transformations between source and target metamodels and their corresponding models, respectively. The challenge is to define how a set of elements from a source model are analyzed and transformed into a set of elements of a target model, the output of transformation being a set of rules involving, and in the same time merging mapping and transformation techniques between two metamodels. For achieving this target, these model transformation languages really aim at considering models and transformations as first class artifacts.

In order to make transformation easier, faster, and cost-reduced process, and therefore less expensive, the semi-automatic generation of transformation rules is an important challenge in future MDE development, we have separated between a first task that is done manually which is the mapping specification and a fully automatic transformation definition generation that contains the operational description of the transformation rules between models. We propose an extension for the OMG transformation architecture by using the expertise thus built up and push the semi-automation process one step further by exploiting the potential benefits of matching techniques [14][15], to provide semi-automatic mappings between two metamodels. The produced mappings could be reported to an expert user who adapts and validates them for the automatic derivation of a transformation model, as a set of transformation rules.

Furthermore, we will present in this paper the matching and mapping components as two other important operations in the transformation process. We will introduce and discuss briefly two main operations “adaptation” and “derivation” which we consider as core techniques for a semi-automatic transformation process in MDA.

On the other hand, we will introduce the idea of integration of metamodel matching algorithms taken from schema matching or ontology matching fields in a common framework. Our target is to capture the fundamental characteristics of each approach and to take profit of the composition of the best characteristics of each individual algorithm. For this purpose, three techniques of integration of these algorithms are used : sequential, parallel and combining techniques.

This paper is organized as follows: section 2 introduces the core concepts of MDA and model transformation in MDA and point out the main problems of the
transformation process. Section 3 presents our extension of OMG transformation architecture for a semi-automatic transformation process. This extension gives emphasis to the separation between mapping specification and transformation definition and discusses the matching and mapping metamodels as two important components in this process. Section 4 is purely theoretical; it proposes the idea of combining different matching techniques for improving the quality matching results. For this purpose, we identify three different manners of matching topologies: Parallel combination, Sequential combination and combination matches. In Section 5 an illustrative example is presented. Section 6 discusses related works and situates them in relation to our work but covers the related work in schema matching not in metamodel matching since in our knowledge no similar research work is yet done in MDE. However, we only mention the works which are the closest to our approach, i.e. they are based on a composition/ aggregation approach. Finally, section 7 concludes our paper and presents some final remarks and perspectives.

2. MDE: Overview and transformation process

At the beginning of this century, software engineering needed to handle software systems that became larger and more complex than before. The object-oriented and component technology seems 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 (MDE), MDA is a particular variant that preaches reliability, reusability and maintainability of complex systems. It aims at realizing that by using automation, of the most error prone parts of the software development process.

Figure 1a. Architecture with four layers

Figure 1b. MDA Transformation: Primary Idea
2.1 Model Driven Architecture (MDA)

MDA is based on architecture with four layers [1]: metamodel, 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 represented by a model in level M1, and a model in level M1 conforms to a metamodel in level M2. These two important relationships of MDA are discussed in [16] [17].

In level M3, a metametamodel is a well-formed specification for creating metamodels such as the Meta Object Facility (MOF), a standard from 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 [16] [12], several metamodels such as UML, UEML and EDOC [18], 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 infinity of syntactically correct C programs. Several different executions could be realized from a C program.

Figure 1.b illustrates the primary idea around the development of software systems using MDA. 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), until exported as code, config files, and so on. Analyzing each type of models, we can deduce that PIM and PSM have different life cycles. PIM is more stable over time while PSM is subject to frequent modification. So, this approach preserves a business’s logic (i.e. PIM) against the changes or evolution of technologies (i.e. PSM).

2.2 Model Transformations: Core concepts and main problems

It is well recognized today that model transformation is one of the most important operations in MDA. The working group on model transformation of the Dagstuhl seminar 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 by a transformation that take a platform-independent model and transforms it in 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 MOF2.0/QVT standard [7].
Each element presented in Figure 2 plays an important role in MDA. In our approach, MOF is the well-established metamodel used to create metamodels. Mapping from PIM to PSM determines the equivalent elements between two metamodels. Two or more elements of different metamodels are equivalent if they are compatible and they cannot contradict each other. A transformation engine that executes transformation rules realizes model transformation. 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 metamodel. The transformation engine takes the source model, executes the transformation rules, and gives 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 all sorts of relationship between models based on separate metamodels. Transformations are one important example of such a relationship, but there are also others [19][13] like model weaving, model merging, model difference, model metrication (establishing measures on models), 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:

\[ ma(s)/Ma \rightarrow mb(s)/Mb. \]

When Ma and Mb are based on the same metamodel (e.g. MOF), the transformation may be expressed in a transformation language such as QVT. There are many general challenges in the definition of a language for model transformation among which the necessity to be expressive and provide complete automation, to be unambiguous, and Turing complete for the target to be generally applicable. The recent standardization effort by OMG [7] 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 previous [8][17] and current work:

- The first problem concerns the creation of “transformation rules” between metamodels which, as mentioned in the introduction, are often created manually using a transformation language, generally a fastidious and error-prone task, and therefore expensive process [13].
- 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 discussed in [14][9], 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 discovered and generated by a matching process.

3. An architecture for the transformation process

Transformation definition contains a detailed description of how 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. In the MDA context, and according to previous works [11][17], the concepts of mapping and transformation should be explicitly distinguished, and together could be involved in the same process that we call transformation process. In fact, in the transformation process, the mapping specification precedes the transformation definition. Transformation definitions contain an explicit description of how to transform a model into another using a transformation language.

Matching between metamodels are the centerpieces for a semi-automatic transformation process in MDE and MDA in particular. Matching techniques have been studied in various research domains, including digital libraries, ontologies, agent matchmaking, schema integration and evolution in databases [20][21][15]. In the context of MDE, we can find very few works in the literature that address the problem of metamodels matching. Schemas in the context of databases and metamodels in our context of MDE are closely related, hence, we propose to review the different approaches of schema matching, and after we situate these approaches in our context of metamodeling matching.

3.1 From schema matching to metamodel matching

In our MDE context with respect to our extended architecture of figure 4, metamodel matching results in a mapping model that must be conform to a mapping metamodel. 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. We have adapted this operator as follows: given two metamodels Ma and Mb, and C Ma →Mb, the mapping model (a set of correspondences) which conforms to the mapping metamodel Mc. The operator match could be defined formally as:

\[ \text{Match} (\text{M}_a, \text{M}_b) = C_{\text{M}_a \rightarrow \text{M}_b}/\text{M}_c. \]

In general, metamodels are created with a specific purpose and by different groups of persons. Each purpose is determined in function of the domain, and each group of persons models a system in different ways. In the modeling task, each group abstracts, classifies and generalizes the reality based on its own knowledge. Consequently, metamodels that were created in the same context and by different groups may have different structure and terminology causing the semantic distance among them [23].

According to our approach, a model can be transformed into another model, only if the metamodel of the former can be mapped into the metamodel of the latter. In order to map metamodels, the equivalent or similar elements must be identified, and the semantic distance should be minimized. The notion of semantic distance was developed to cover the notion of “how close is close enough”. A dual for “semantic distance” is schema similarity that is defined as “the ratio between the number of matching elements and the number of all elements from both input schemas” [24]:
(SS = Nm/Nt, where SS is the schema similarity, Nm is the number of matching elements and Nt is the number of all elements). Semantic distance can also be quantified as a numeral value (like schema similarity) or as a subset of a metamodel. By the way, according to the MDA manifesto [19], “one of the primary purposes of automation in MDA is to bridge the semantic gap between domain concepts and implementation technology by explicitly modeling both domain and technology choices in frameworks and then exploiting the knowledge built into a particular application framework”.

Moreover, automation, which is the main objective of this work in the context of model transformation, is one of the basic tenets of MDA manifesto [19].

### 3.2 The transformation definition

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 given transformation definition metamodel.

Several research projects have studied the mapping specification between metamodels [18][13][25]. However, the ideas around mapping specification are not sufficiently developed to create efficient tools to enable automatic mappings. Nowadays, transformation languages have been largely studied and raised to a standard transformation language called QVT defined by OMG [7]. 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 easy tasks. Moreover, tools to enable the automatic creation of mapping specification and automatic generation of transformation definition are still under development.

### 3.3 For an automatic mapping generation

A mapping specification is a definition of the correspondences between metamodels (i.e. a metamodel for building a PIM and another for building a PSM). This definition is largely obtained by a matching process between two metamodels, and completed by an
expert. Our first proposition for extending [17] the OMG model-transformation architecture has emphasized the separation between mapping and transformation parts. However, in this first proposal, the mapping is manually specified by an expert, while the transformation is generated automatically. The next step is to push the semi-automation process one step further by exploiting the potential benefits of matching techniques [14][15], to provide semi-automatic mappings between two metamodels. Figure 3 illustrates the different concepts of MDA according to our vision where mapping specification is a mapping model (Mapping M), and transformation definition is a transformation model (Transformation M). 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 is generated from a mapping model. A transformation engine (tool) takes a source model as input, and executes the transformation program to transform this source model into the target model.

This figure 3 illustrates our proposal of an extended architecture for the transformation process in MDA, allowing a semi-automatic generation of transformation rules between two metamodels, and the semi-automatic generation of a target model from a source model. The first 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 [26][27] is the process that produces the potential 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). 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 one of the steps in the whole process, in which the expert intervenes to complete and 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. The derivation process is explained in the next sub-section. 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.

3.4 The aspects of automation with the architecture

The first goal with this architecture is to introduce the matching process into the OMG’s MDA approach in order to increase the degree of automation of the transformation process. This leads to a reduction in manual human tasks often fastidious and error-prone, by the rational choice among the plethora of existing works on matching techniques. These techniques are suitable for the problem of automatic mapping production. Thus, from a software point of view, the transformation process involves three main programs which are at the heart of a semi-automatic development:

Matching program: implements the matching metamodel and produces a first version of a mapping model between two metamodels source and target. This mapping model is adapted and validated by an expert user.

Generation program: takes a mapping model validated by an expert, and derives automatically a transformation model (program) as a set of rules.

Transformation program: takes a source model defined by designers or engineers and produces an equivalent target model on a specific platform.

Figure 3 Architecture for a semi-automatic transformation process in MDA.
Two important operations (dashed arrows) adaptation (1) and derivation (2) allow linking and completing the two main operations (matching and transformation) in the whole process of transformation. Adaptation is the responsibility of the expert user who should accept, discard or modify the obtained mappings, furthermore, to specify the correspondences which the matcher was unable to find. Loosely speaking, the mapping and matching techniques (models) could be defined with the following intuitive formula:

\[
\text{Mapping} = \text{Matching} + \text{Adaptation}
\]

The mapping model obtained in the previous step after adaptation by the expert user should be completely defined allowing an automatic generation of transformation model. This operation is called derivation and, in the same way as above, transformation and mapping models can be defined with the following intuitive formula:

\[
\text{Transformation} = \text{Mapping} + \text{Derivation}
\]

4. Matching techniques combination

Matching algorithms, so called matchers, are developed trying to automate the metamodel matching task partly. They compute correspondences between elements of models based on syntactical, linguistic and structural aspects and instance information and provide the user with the most likely mapping candidates [15][21]. Many systems combine the results of a number of these matchers to achieve better mapping results [28]; [29]. The idea is to combine complementary strengths of different matchers for different sorts of models. This balances problems and weaknesses of individual matchers, so that better mapping results can be achieved. There is need to have a good quality in order to avoid extra work for correcting wrongly identified correspondences. Unfortunately, most of the proposed techniques are built for individual matchers or are hard wired within specific matching processes.

Most promoted matching systems use a combination of different matching techniques for improving the quality matching results, we have never seen the same principle in metamodel matching field. The topology of the matching system has a major impact on the performance
and the quality of a matching task. We identify three different types of matching topologies: (1) Parallel combination, (2) Sequential combination and (3) combination matches

4.1 Parallel matches

Parallel combination implies that all matchers are executed independently, typically on the whole cross product of source and target metamodels elements.

![Parallel matcher combination](image)

**Figure 5.** *Parallel matcher combination*

Figure 5 shows the concept. It receives metamodels as an input and produces mapping proposals as an output. The processing of the given metamodels is done by applying two different matchers each leading to a specific matching result.

This approach allows for a combination of matching results from different approaches and even grants a possibility for importing matching results from other tools.

4.2 Sequential matches

As shown in figure 6, sequential combination systems rely on a sequence of matchers to narrow down a set of mapping candidates. This sequencing can improve performance since the search space is reduced by the first matchers. However, the performance improvement is achieved without the risk of losing possible mapping results since the sequence is done both in the two sides (matcher1/matcher2 then matcher2/matcher1). In each side, we put a mapping result as input to a matcher that can then refine the found correspondences.

![Sequential matcher combination](image)

**Figure 6.** *Sequential matcher combination*

4.3 Combination matches

We propose an approach on metamodel matching-based on combining different matching techniques for an improved matching result. These techniques differ from existing ones in taking additional information into account, like instances, existing mappings, graph structures etc., thus allowing for an increased match quality for an automatic approach.
Combined approaches synthesize two or more functions extracted from the existing matchers and may be organized in a logical chain. Each step in the chain allows refining results of previous steps.

For example, firstly specifications are matched in a semantic manner to identify semantically corresponding specifications, after which those with high semantic similarity may be scrutinized even further by performing a structural matching algorithm. This way of matching assumes that several matching algorithms are executed concurrently thus facilitates multi-criteria analysis of specifications.

5. An illustrative example

To illustrate our approach for a semi-automatic transformation process, figure 5 partly inspired from [26], shows fragments of the UML2.0 and Ecore metamodels, on the one hand in MOF notation (cf. upper half of Figure 7) and on the other hand as a tree form with mappings between the two metamodels (cf. lower half of Figure 7). The mappings are represented graphically using a language that we have defined in our previous works [17][18]. A brief description of this language is presented hereafter. We assume here that our techniques of matching have provided these mappings in a first step.

Our graphical notation for mapping uses the following concepts:
- connection (source and target),
- association (could be composition) and
- mapping element.

A connection links one or more metamodel elements to a mapping element. A connection may be a main connection (links between the main entities in each metamodel) represented by full lines, or secondary connection (links between the own properties of these entities) represented by dotted lines. Secondary connections link for example attributes of two classes such as the name of a class, the property isAbstract of a class…

![Figure 7. Transformation process from UML 2.0 to Ecore (fragments)](image-url)

Class
- isAbstract : Boolean [1..1]
- ownedAttributes:Property[0..*] aggregation : {none, shared, composite}[1..1]
  opposite : Property[0..1]
Association
- memberEnd(Property)[2..*]
The association expresses a relationship between mapping elements. The composition shows a tight relation between mapping element (i.e. composite mappings). The mapping element allows linking elements from a source metamodel to elements of a target metamodel. According to Figure 7 (in the tree-based view), the C2EC element maps the element Class (C) of UML2.0 to (2) the element EClass (EC) of Ecore.

In UML 2.0 classes can be abstract or concrete (cf. attribute isAbstract) and own a collection of Properties. While isAbstract represents a class attribute where the value is common for all the instances of a class, Properties defines the values owned by each instance of a class. Moreover, Properties can also be owned by Association, and in this case it represents association ends also called roles (cf. memberEnd in). Depending on the attribute aggregation of the contained properties, Associations may be of type simple, aggregation or composition. The xor-constraint between ownedAttributes and memberEnd ensures that a property is either an attribute or an association end but not both at the same time. This xor-constraint leads to the duplication of the class property in the tree-based view. Similarly, in Ecore, Eclasses can be abstract or concrete (cf. attribute abstract) and own a collection of EAttributes and EReferences. EReferences represents the relationships between EClasses, the attribute containment states if this relationship, called also association, is of a composition type.

The mapping element C2EC in Figure 7 represents the ideal situation of one-to-one correspondences. The main connection specifies the equivalence between Class of UML and EClass of Ecore. The secondary connection defines the equivalence of the attribute isAbstract of Class to the attribute abstract of EClass; moreover data types and multiplicities of the two attributes match exactly. The mapping element oA2eA maps through the main connection the reference ownedAttributes of UML Class to the reference eAttributes of Ecore EClass. However, secondary connections are impossible since the attributes of the classes Property and EAttribute do not match at all. These attributes express semantics that are completely different. In the one case (Property in UML), we are interested in the semantic of the link (shared or composite) and the possibility of mutual reference (opposite), while in the other case (EAttribute in Ecore), we are interested in the “Unique identifier” property of the attribute such as the primary Key constraint in relational database. In this situation of semantic incompatibility, adaptation techniques should be used by an expert to allow a transformation of an UML model to equivalent model in Ecore. The expert should for example define rules in a language such as OCL, in order to map UML properties to equivalent Ecore properties. The last mapping element mE2eR maps through the main connection the reference memberEnd of UML Association to the reference eReferences of Ecore EClass. The first problem here concerns the UML concept of class association which does not have an equivalent concept in Ecore. However, the association could be easily modeled through bidirectional references in Ecore, as a pair of references linked in both directions.

Finally, the first secondary connection of the mapping element mE2eR between the attribute aggregation of property and the attribute containment of EReference, needs a “small adaptation”, since the type enumeration with the values none, shared and composite should be mapped to the type Boolean. Clearly, the adaptation could be defined by a rule, expressing how the three enumeration literals are mapped to the Boolean values true and false.

This mapping example between UML 2.0 and Ecore illustrates the importance of adaptation mechanism since structural and semantic incompatibilities are common between metamodels. This mechanism allows to define a complete mapping model between two metamodels. Each property in the source metamodel should have an equivalent property in
the target metamodel. As we discussed previously, a complete mapping model allows through derivation techniques an automatically transformation rules generation. These rules, transform automatically a source model into an equivalent a target model. The produced target model, could even be not conform to the associated metamodel since the priority here is the transformation of a source model and not the conformity of the target model to the associated metamodel.

6. Related Work

Matching techniques, which are the basis of a semi-automatic transformation process, have been studied intensively in various fields. The schema matching in database [20] was a precursor whereas, today, with the web and the semantic Web, research on techniques of ontology matching is very active [19][15]. A prosperous bibliography on the topic of matching and research around this theme is maintained on the web [30].

Ontology matching [19] is developing quickly as a way to find the solution to the semantic heterogeneity problem which is one among many problems in the matching process between two metamodels. In fact, for getting similarity between entities used in current ontology matching systems, many strategies are used (e.g., string similarity, synonyms, structure similarity and based on instances) [14].

Among schema matching approaches, the authors of [31] propose an approach consisting on three phases that automatically detects mappings between two metamodels and uses them to generate an alignment between those metamodels. The approach begins by the generation of directed labelled graphs representing the meta-models to match, then, follows with the application of the Similarity Flooding Algorithm [32] on the graphs produced. Those graphs are used in a fix-point computation process whose result tells what nodes in the first graph are similar to nodes in the second graph. The final step consists of the generation of an alignment between the two metamodels. The paper evaluates different configurations of similarity flooding on several metamodels.

In the same order of idea aiming to facilitate semi-automatic mapping generation and improving model matching for model transformation development, [33] proposes to integrate existing matching approaches in a common framework and to reuse matching results by applying different metamodel specific matchers, namely an instance matcher, a graph matcher, an annotation matcher, a data type matcher, a frequency matcher, a pattern matcher, a transformation reuse matcher, and a matcher configuration based on model transformation type classification. This novel approach is implemented in a prototype combining the instance matcher and data type matcher in an existing framework. [34] uses a combination of metamodels and ontologies in order to “annotate” model instances but in their approach the ontology concepts are created corresponding to the same metamodel that is also used to create model instances.

Other studies have been interested in semi-automating the transformation process in MDA context, or more generally in the MDE context, by focusing on the separation between mapping and transformation processes. References [13][25][22][24] have addressed this issue. In [23][25] we have proposed an approach separating mapping specification and transformation definition, and, an initial foundation for performing metamodel matching is discussed. In [13], they propose various heuristics for the creation of the matching between two metamodels. These heuristics are implemented by a matching metamodel to generate a correspondence model (Weaving model) from properties of the source and target metamodels taken as input. In [25], they propose an approach called lifting, to transform a source metamodel expressed in MOF into an equivalent target metamodel expressed in ontologies.
Matching techniques largely developed in the areas of ontologies [19] and database schemas [20] are then applied. Their approach has been implemented through a prototype available on the site of the project, called Model CVS\(^1\). In this second approach, the aspect of automatic generation of transformation rules from correspondence is not treated.

7. **Conclusion and Future Work**

In this paper, we have presented our approach for a semi-automatic transformation process in MDA using an extended architecture. We argue that a semi-automatic transformation process will be a great challenge in MDA 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; and it increases the quality of the final transformation code.

The key principle for this process is to consider mapping and matching metamodels as first class entities in MDA. These matching techniques will push semi-automation of transformation process to a new level and thereby significantly contribute to results of higher quality.

The main goal of our idea is to lower the effort in model transformation development and to reduce possible errors. Our proposal of using a combining matcher framework increases the quality of matching results. For this purpose, we have introduced an approach based on executing several matchers in parallel, in sequential or in combined way. We believe that it may improve the performance significantly by early pruning many irrelevant element combinations. Finally, an illustrative example using UML and Ecore metamodels is presented and related works are discussed.

In future work, we will validate our approach by giving another illustrative example working with heterogeneous metamodels making profit of matching techniques already used in ontology development. This is to evaluate the completeness of the transformation rules generated and to investigate in the case of incompleteness of the obtained mappings, the adaptation and derivation techniques discussed in this paper. We also will implement many algorithms to work together on the same framework and evaluate the efficiency of combining many matching algorithms.

**Reference**


\(^1\) [http://www.modelcvs.org/](http://www.modelcvs.org/)
[13.] M. D. Del Fabro, “Semi-automatic Model Integration using matching transformation and weaving models”. In SAC ’07, ACM. 2007
[30.] Web: http://www.ontologymatching.org/