Challenge, Issues and Comparison of Metamodels Matching

Lamine Lafi[1], Slimane Hammoudi[2]

[1] ISSAT, Institut Supérieur des Sciences Appliquées et de Technologie de Sousse,Tunisia, Amine.Lafi@issatso.rnu.tn

[2] ESEO, Ecole supérieure de l'Ouest Angers, France, slimane.hammoudi@eseo.fr

Abstract. 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 Model Driven Engineering (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 to discuss metamodels matching as a key technique for a semi-automatic transformation process. We review and discuss the main approaches that have been proposed in the state of the art for metamodels matching. We compare two recent algorithms of metamodel matching namely "Similarity Flooding" and SAMT4MDE+ using match quality measures proposed for schema matching in databases. A Plug-in under the Eclipse framework has been developed to support our comparison using three couple of metamodels.

Keywords: Metamodel matching, Model transformation, semi-automatic transformation process

1 Introduction

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 a core development artifact. New significant approaches, mainly Model Driven Architecture (MDA) [1] defined at the OMG (Object Management Group), "Software Factories" proposed by Microsoft [2] and the Eclipse Modeling Framework (EMF) [3] from IBM, are born and have been experimented. In the literature, several issues around MDE have been studied and subjected to intensive research, e.g. modeling languages [4] [5], model transformation [6] [7], mapping between metamodels [8] [9], and design methodologies [10]. Among these issues, model transformation languages occupy a central place and allow the definition of how a set of elements from a source model are analyzed and transformed into a set of elements in a target model. However, these transformations are created manually, often a tedious and error-prone task, and therefore an expensive process. These transformations consist of creating a set of rules involving, and at 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 [9][11]the authors have initiated a first attempt towards this semi-automation. They introduced an approach separating

mapping specifications from transformation definitions, and implemented this approach in a tool called Mapping Modeling Tool (MMT). In this first approach, a mapping specification was created manually to define the relationships between metamodels (i.e. equivalent metamodel elements), with the transformation definition being generated automatically from the mappings.

In [12], the authors have proposed to push the semi-automation process one step further by using matching techniques [13] to semi-automatically generate mappings between two metamodels. The produced mappings could then be adapted and validated by an expert for the automatic derivation of a transformation model, as a set of transformation rules. Thus, matching techniques between metamodels are the centerpieces for a semi-automatic transformation process in MDE and particularly in MDA. In fact, metamodels matching allows discovering mappings between two metamodels and the mappings allow in turn generating transformation rules between two metamodels. However, there has been little research in metamodel matching in contrast to the ontology [14] and database [15] domains, where an intensive research has been conducted. In this paper, we firstly discuss the techniques and foundations of schema matching that has been extensively studied in the database area, and then we review the main different approaches that have been proposed for metamodel matching in the context of MDA/MDE. We compare two recent algorithms of metamodel matching namely "Similarity Flooding" and Extended SAMT4MDE (noted SAMT4MDE+) using match quality measures proposed for schema matching in databases. A Plug-in under the Eclipse framework has been developed to support our comparison using three couples of metamodels.

We will start by stressing the role of matching techniques in the semi-automatic process of model transformation.

This paper is organized as follows: section 2 introduces the main concepts and techniques for a semi-automatic transformation process, presents schema matching techniques and situates them in the context of metamodel matching. Section 3, reviews and compares five approaches that have been proposed for metamodel matching in the context of MDA. Section 4 presents the two recent algorithms for metamodel matching and section 5 discusses an experimental comparison between these two algorithms using three pairs of metamodels. Section 6 presents ourprototype implemented as a plug-in for Eclipse. Finally, section 7 concludes our work and presents some final remarks and perspectives.

2 Metamodels matching for model transformation

2.1 Metamodels matching for model transformation: Overview

It is well recognized today that model transformation is one of the most important operations in MDA [16]. The following definition of model transformation, largely consensual, is proposed in [17]:

"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".

However, we point out two main problems concerning the MDA transformation process.

- The first problem concerns the manual creation of "transformation rules" between metamodels. Generally, this task is tedious and error-prone, and therefore expensive in terms of efficiency [18]. Moreover, writing the transformation rules requires a good mastery of both the transformation language and the source and target metamodels, in order to express the correspondence both from a structural and a semantic point of view.

- The second problem concerns the specification of these "transformation rules", which merge together techniques of mapping and transformation without an 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.

In the MDA context, and according to previous works [9][19], 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. A mapping specification is a definition of the correspondences between metamodels (i.e. a metamodel for building a PIM (Platform Independent Model) and another for building a PSM (Platform Specific Model)). This definition is largely obtained by a matching process between two metamodels, and completed by an expert. Transformation definitions contain an explicit description of how to transform a model into another using a transformation language. Transformation definitions are a set of rules that are obtained automatically from all the mappings between two metamodels. Hence, in our approach the transformation process of a PIM into a PSM can be structured in two stages: a mapping specification obtained by a matching process and completed by an expert, and a transformation definition derived automatically from the mappings.

The Fig. 1 illustrates the main concepts and techniques involved in a semi-automatic transformation process. The matching operation 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. 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 (a program: a set of rules), 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. A transformation model (program) takes a source model defined by designers or and produces an equivalent target model on a specific platform.

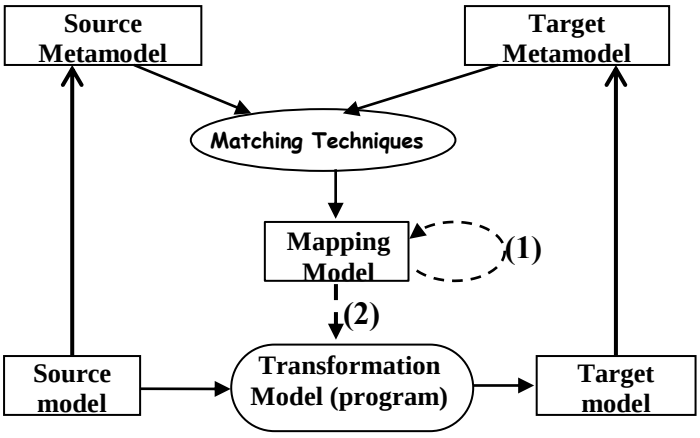

Fig.1. *Semi-automatic transformation process*.

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:

(**1**)

$$\text{Mapping} = \text{Matching} + \text{Adaptation}$$

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:

$$\text{Transformation} = \text{Mapping} + \text{Derivation} \quad (\mathbf{2})$$

2.2 From schema matching to metamodel matching

Matching between metamodels are the centerpieces for a semi-automatic transformation process in MDE, particularly in MDA. Matching techniques have been studied in various research domains, including digital libraries, ontologies, agent matchmaking, schema integration and evolution in databases [14] [19]. In the context of MDE, we can find 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 that we situate these approaches in our context of metamodeling matching.

2.2.1 Classification of schema matching approaches

In the literature, several schema matching approaches have been proposed [14] [19]. Each schema matching approach has its own characteristics that were grouped in a taxonomy discussed below [18] [20]. In addition, each approach has been evaluated through match quality measures discussed in the next section 2.2.2.

- Individual matcher approaches use only one matching criterion. They are classified in:
  - *Schema-only based*, when they consider only metamodels. They can be classified in:
    - ✓ *Element level*, the mapping is realized for each individual element. It can be classified in linguistic and constraint-based. Linguistic are based on name similarity, description, global namespace, while constraint-based are based on type similarity and key properties.
    - ✓ *Structure-level,* the mapping is realized considering the combinations of elements related in a structure. It is only classified in constraint-based that use graph matching.
  - *Instance/contents-based*, when they consider only instances (or models). It can also be classified in element-level. This last can be classified in linguistic and constraint-based. In this case, linguistic is based on word frequencies and key terms present in the element instances, while constraint-based is based on value pattern and ranges of the element instances.
- Combining matchers use multiple matching criteria. They can be classified in:
  - *Hybrid*, they combine multiple approaches to create only one matcher in order to produce a result, i.e. the creation of mapping between elements.
  - *Composite*, they combine many results obtained from different approaches in order to produce the mapping between elements. This combination of results can be manual or automatic.

2.2.2 Matching quality Measure

The interrelationships between metamodels can be organized in sets which can be manually or automatically created. A set created manually can contain all needed matches (i.e. matched elements); while a set created automatically can contain valid and non-valid matches. The first set is denominated real matches, and the later derived matches (cf. Fig.2).

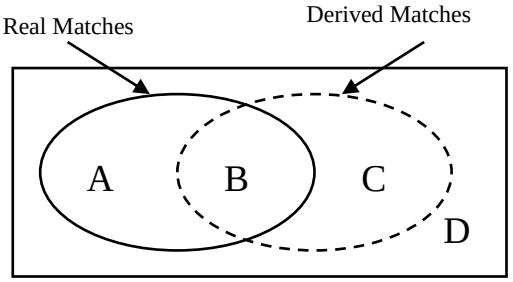

Fig.2. *Comparing real matches and automatically derived matches***.**

In addition, other subsets are defined as follows [14] [20]:

- A (false negatives) are matches needed but not automatically identified.
- B (true positives) are matches which are needed and have also been correctly matched by the automatic match operation.
- C (false positives) are matches falsely proposed by the automatic match operation.
- D (true negatives) are false matches which have also been correctly discarded by the automatic match operation.

Based on the cardinalities of these sets, the following match quality measures are provided as parameters for benchmarks:

$$Precision = \frac{|B|}{|B| + |C|} \quad (3)$$

reflects the share of real correspondences among all found ones.

$$Recall = \frac{|B|}{|A| + |B|} \quad (4)$$

specifies the share of real correspondences that are found.

$$F\text{-}Measure = 2 * \frac{\mathrm{Pr}ecision * \mathrm{Re}call}{\mathrm{Pr}ecision + \mathrm{Re}call} \quad (5)$$

$$Overall = Recall * (1 - \frac{1}{\mathrm{Pr}ecision}) \quad (6)$$

All these measures were developed specifically in the schema matching context [18] [21]. We can notice that F-Measure represents the harmonic mean of Precision and Recall. The main underlying idea of Overall is to quantify the post-match effort needed for adding missed matches and removing false ones.

2.2.3 Metamodel Matching versus schema matching

Our aim is not to compare schema matching approaches with those of metamodel matching. The main reason is that the technological spaces of both approaches are not the same. A technological space [22] is a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities. It is often associated to a given user community with shared know-how, educational support, common literature and even workshop and conference regular meetings. Although it is difficult to give a precise definition, some TSs can be easily identified, e.g. the XML TS, the DBMS TS, the abstract syntax TS, the meta-model (OMG/MDA) TS, etc. In one

case, in schema matching efforts have been made mainly in the context of databases and involve ER schemas. In the other case, metamodel matching focuses on UML OMG standard which is structurally and semantically richer than ER schemas. Moreover, the level of abstraction of metamodels and schemas is not the same. However, techniques known and used to find semantic correspondences between the elements of two schemas are to be applied to the metamodel matching problems since the aim "finding semantic correspondences" is still the same. We can say that metamodel matching techniques would probably subsume schema matching techniques from the fact that a technological space of metamodels includes the technological space of schemas in database area. Thus metamodel matching techniques will probably become a generic solution to many problems of matching; that we can call the X-matching.

3 A brief review of Metamodel Matching Techniques

Table 1 gives a summary of an evaluation of five approaches of metamodels matching. These approaches are*: Similarity Flooding* (SF), *ModelCVS* project, Semi-Automatic Matching Tool for MDE (SAMT4MDE), Generic Model Weaver (AMW) and an extended SAMT4MDE (SAMT4MDE+). On the left hand side of table 1 we can see all the features used for the evaluation.

A detailed discussion concerning this evaluation is presented in [13]. According to the comparative study presented in Table 1 we have drawn the following conclusions:

- Only two approaches namely: Similarity Flooding and SAMT4MDE+ are exclusively concerned with the problem of metamodels matching in the context of MDA (the other three approaches are based on matching patterns in databases and ontology), and they are the most recent algorithms,

- These two approaches give the most effective quality measures,

- They operate semi-automatically despite the fact the expert intervenes to validate the matched elements.

- Similarity Flooding and SAMT4MDE+ use the MDA technological space (UML, MOF) for metamodels of any size.

We have also deduced the following conclusions:

- The *Similarity Flooding* algorithm is more suitable usually with metamodels of small size,

- We also note that the quality measures will be very weak if the correspondence is between two metamodels of largely different size.

All these details, remarks and findings lead us to carry an experimental comparative survey between the two approaches of *Similarity Flooding* and *SAMT4MDE+* using the same pairs of metamodels. In order to achieve this task, we rely on quality measures presented in section 2.2.2.

Table 1. Summary of the evaluations.

	SF	ModelCVS	SAMT4MDE	AMW	SAMT4MDE+
References	[21]&[23]	[20]	[19]	[18]	[24]
Test problems					
Tested Schema types	XML/SQL DDL Relational/RDF/ UML/ Ecore	UML1.4/UML 2.0	Object-oriented model: UML, Java, and C-Sharp	SQL DDL Relational/ UML	UML 2.0/ Ecore
Schema / Schema tasks	18/9	-	2/1	-	-
Min /Max / Avg schema size	5/22/12	Large schema		Very large schema	-
Min /Max / Avg schema similarity	0.46/0.94/0.75	-	Schema similarity=0.79	-	Schema similarity=0.68
Match result representation					
Matches	Element-level correspondences between similarity value in [0.1]		With discrete value {0,1,-1}		Similarity value in [0.1]
Element Repr	Node	Models/Metamodels	Node	Models or Metamodels Data	Class
Local/Global Cardinality	1:1/m:n	1:1/m:n	1 :n	-	1:1/m:n
Quality of measure And test methodology					
Employed Quality Measures	Overall	Precision, Recall, F-measure	Precision, Recall, Overall, F-measure	Element similarity*, Link filtering, Link rewriting.	Precision, Recall, Overall, F-measure
Subjectivity	7 user/ 6 configurations	10 scenarios and 13 tools settings	02 Fragments of UML and C# metamodels	A set of input metamodels	1 class and its neighbor classes
Pre-match effort	None	None	None	None	None
Studies Impact on match Quality	Filters, fix-point formulas, randomizing initial similarity	F-Measure: IF>0.5: positive benefit IF<0.5: negative Benefit			Threshold
Best average Match quality					
Prec/ Recall	**	0.63/0.58	0.79/0.68	-	0.84/0.90
F-measure		0.61	0.73	-	0.87
Overall	˜0.6	-	0.49	-	0.73
Evaluation high light					
	User subjectivity, no pre-match effort	no pre-match effort	Representative metamodels for developing information system	no pre-match effort	no pre-match effort
Application Area	Database and ontologies	Database and Structural Modeling languages	Model transformation	Model transformation	Model Transformation
Manual work of user	Choice of metamodels to align, Evaluate matching suggestions produced by algorithm.	Including ontology creation tools, query tools, matching tools, and Reasoning tools.			Evaluate matching suggestions produced by algorithm
Match granularity	Element level Mapping	Schema-level Matching	Schema-level Matching	-	-

* Contains element to element similarity and structural similarity

** Change from configuration to another

4 Similarity Flooding versus SAMT4MDE+

4.1 Similarity Flooding

Similarity Flooding in [21] is a generic alignment algorithm that allows calculating the correspondences between the nodes of two labeled graphs. This algorithm is based on the following intuition: if two nodes stemming from two graphs have been determined as similar, therefore, there is strong probability for the neighboring nodes to be similar, too. More precisely, SF applies five successive phases on the labeled graphs which have been provided at the input phase. This algorithm is applied after the transformation phase that consists in transforming the MMsource and the MMtarget to the directed labeled graphs Gsource and Gtarget. Along this phase a set of six strategies to encode the metamodel into such a graph has been used. Each of these strategies has got its proper techniques to transform these two models into a graph and they are explained in [21].

In this paper we consider only three tree strategies of encoding namely: Standard, Saturated and Flattened, despite the fact that they have given the best quality measures [21].

4.1.1 The Standard Configuration

According to [21] the Standard configuration extends the Basic configuration to obtain similarities not only inspecting the names of the elements, but also their types, and main attributes (abstract for the EClass, lowerBound and upperBound from EAttribute and EReference, and containment from EReference). In that purpose, a node representing an element E is linked by an arc labeled kind to a node N representing the type of the element E. N is labeled according to the type of the element E, removing the prefix "E" for all the elements of Ecore, and adding the suffix "Element" (e.g. EClass is turned into ClassElement). To deal with the main attributes, when an element E has for type a meta-class with an attribute A, then the node corresponding to E is linked with an arc labeled A to a node whose label is the value of the attribute A for E. For example, to represent an abstract EClass myEClass, the node representing myEClass is linked with an arc labeled abstract to a node labeled true. The left side of Fig. 5 in [21] shows an excerpt of the graph exGsource generated using the Standard configuration.

4.1.2 The Flattened Configuration

In [21] the flattened configuration is based on the Standard one, but with flattened inheritance. The nodes representing abstract EClasses and the arcs labelled supertype are deleted. Instead, arcs labelled *own* (resp. *ref*) connect nodes representing an EClass Ecl to nodes representing the EAttribute (resp. EReference) defined by Ecl and all its superclasses. Moreover, when an EReference Eref is typed by an abstract EClass Ecl, a type arc is created from the node corresponding to Eref to each non-abstract sub-class of Ecl.

4.1.3 The Saturated Configuration

The Saturated configuration according to [21] is still based on the Standard one. Here, the transitive relations are saturated, like in [25]. EClass nodes are now connected by a supertype arc to the nodes representing all the super-classes of this EClass. EClass nodes are also connected by own (resp. ref) arcs to the nodes representing the EAttribute (resp. EReference) introduced and inherited by the EClass. Finally, for a node representing an EReference, type arcs are created to the node representing the EClass that types the EReference as well as all the nodes representing the super-classes of the EClass.

4.2 SAMT4MDE+

In [24] a new metamodel matching algorithm of that uses structural comparison between a class and its neighbouring classes in order to select the equal or similar classes from source and target metamodels. The proposed algorithm for metamodel matching called SAMT4MDE+ is an extension and enhancement of the algorithm presented in [26] and it is implemented in the Semi-Automatic Matching Tool for MDE (SAMT4MDE) which is capable of semi-automatically creating mapping specifications and making matching suggestions that can be evaluated by users. This provides more reliability to the system because mapping becomes less error-prone. The algorithm proposed can identify structural similarities between metamodel elements. However, sometimes elements are matched by their structures but they do not share the same semantic. The lack of semantic analysis leads the tool to find false positives matches, i.e. derived correspondences that are not true.

The function similarity (c1,c2) is the weighted mean which has as parameters basicSim (c1,c2) and structSim (c1,c2) with the weights coefBase and coefStruct, respectively. It returns continuous values representing the similarity level between c1 and c2.

The weights are *coefBase* and *coefStruct* that added result in 100%, or 1(one).

For example, if *coefBase* is equal to 0.3, then *coefStruct* must be equal to 0.7. The value of *similarity (c1,c2)* is compared to a *threshold* value in the range [0,1]. If a similarity is greater than threshold value, the classes c1 and c2 are correspondent, otherwise they don't correspond. Thus, threshold is an important measure to take a decision: if the value is low, in general in the range [0, 0.5], many elements will be considered correspondent in a wrong way (*false positive*), if the value is high, in general in the range [0.8, 1], many classes will not be considered as correspondent (*false negative*).

According to [24] the similarity function between two classes c1 and c2 is given by:

$$similarity(c1,c2) = basicSim(c1,c2) * coefBase + structSim(c1,c2) * coefStruct \quad (7)$$

where 0 <= coefBase<=1, 0<=coefStruct<=1, and coefBase + coefStruct = 1.

5 Comparative study and First Experiments

In order to achieve a comparative study between the two approaches for metamodels matching (SF and SAMT4MDE+), we have used three pairs of metamodels among the following list: *Ecore, Minjava, UML, Webml, and er_ODM*. These metamodels are presented in details in [27].Three alignments have been considered for our comparison:

- Ecore 2 Minjava
- Ecore 2 UML
- Webml 2 er_ODM

To achieve an experimental survey, we developed a java plug-in in under eclipse. This plug-in allows to help the user to choose the algorithm to execute. Every algorithm will be evaluated for each of the couples of the three couples of metamodels: Ecore2Minjava, Ecore2UML2.0, and ER-ODM2WebML.

The results of the experimentation of the two algorithms Similarity Flooding and SAMT4MDE+ are given by Fig. 3 and Fig. 4. The assessment of the first algorithm is done according to the three configurations: Standard, Flattened, and Saturated (since they gave good results in [21]) on the three couples of metamodels Ecore2Minjava, Ecore2UML2.0 and, ER-ODM2WebML. The second algorithm is evaluated on the same pairs of metamodels.

According to Fig 3 and Fig 4 we note that the values of the quality measures vary according to the size of the metamodels and configurations used in Similarity Flooding. This diversity of the values justifies and validates the conclusions that we achieved according to the presented theoretical survey in [13]. Indeed, for the algorithm Similarity Flooding the first two pairs of metamodels Ecore2Minjava and Ecore2UML2.0, the values of precision, recall and F-Measure are weak for each of the three chosen configurations. This is due to the fact that Ecore, Minjava and UML are of large size. On the contrary for the third couple, we note that the results are distinctly better for each of the three configurations. This is justified by their reduced sizes with respect to the two other pairs of metamodels.

We also note that the values of the quality measures vary from a pair of metamodels to another. Indeed, the best values of precision, Recall and F-Measure are given by the pair of metamodel Ecore2Minjava which is of a large size. These results contradict the assessment given in [21].

The third couple of metamodel ER_ODM2WebML has given good measures of quality for the first algorithm thanks to its small size. These measures are very close to those sent back by the couple Ecore2Minjava with SAMT4MDE**+.** This proves that this last keeps its performance for the metamodels on a large scale.

Fig. 3 represents the assessment of the S*imilarity Flooding* algorithm. The assessment of the algorithm *SAMT4MDE+* is presented by Fig. 4.

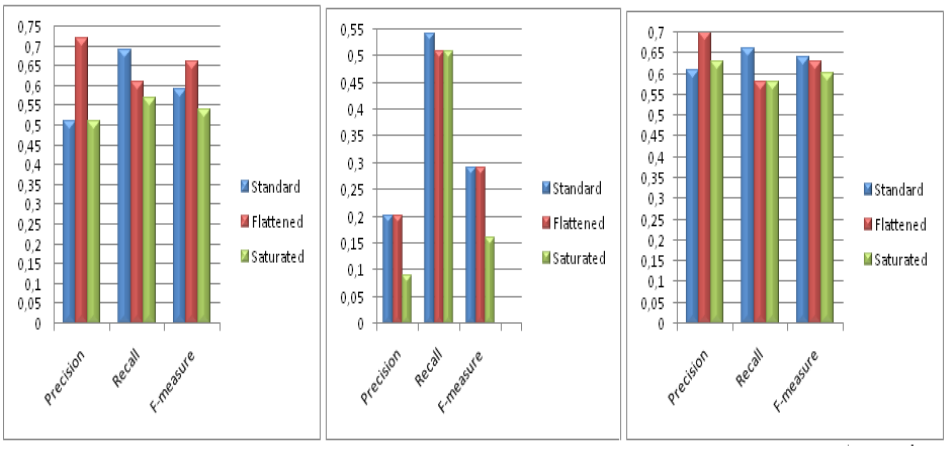

Fig. 3.*The quality measures produced byEcore2Minjava (left), Ecore2UML2.0 (center) and ER_ODM2WebML (right) according to the algorithm Similarity Flooding.*

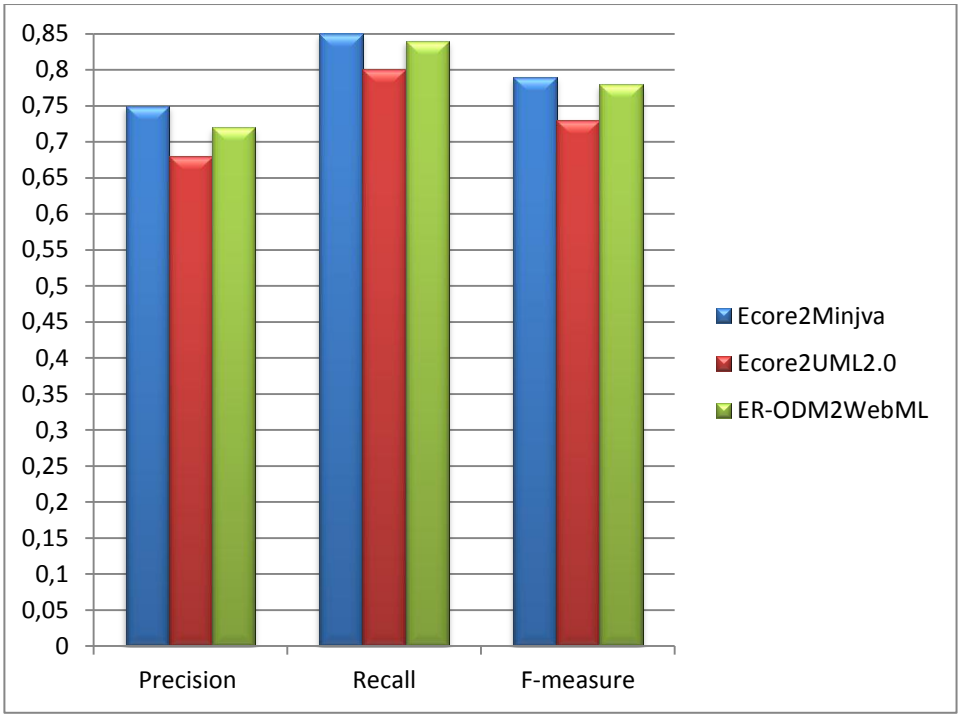

Fig.4. *The quality measures produced by Ecore2Minjava (left), Ecore2UML2.0 (center) and ER_ODM2WebML (right) according to the algorithm SAMT4MDE+.*

According to this experimental survey concerning the two algorithms for metamodel matching, *Similarity Flooding and SAMT4MDE+*, the main drawn findings are the following:

- In order to get the quality measures, the mapping model in SAMT4MDE+ must be validated by an Expert user, whereas the multi-mappings in Similarity Flooding are validated automatically on the basis of the models of mappings validated by experts or manually aligned,
- SAMT4MDE+ gives very good quality measures of matching no matter the size of the metamodel to correspond, but this cannot be confirmed only after having tested this algorithm on the basis of the mappings validated by experts, or by many different couples of metamodels aligned manually.

We can conclude that SAMT4MDE+ is more effective than Similarity Flooding because of two main reasons:

- SAMT4MDE+ is more suitable for the metamodels of large size, whereas SF is suitable only for metamodels of small size,
- The user's intervention to validate the suggestions of mappings determined by the algorithm will have very positive consequences on the performance of the quality measures. This is due to the fact that the expert user tried to choose all the mappings that appears correct during the validation phase.

6 Prototyping

We illustrate this section with two figures (Fig.5 and Fig.6) which are screenshots of our Plug-in developed to evaluate the two approaches of metamodels matching. Every figure illustrates one of the two following steps of our approach:

- The algorithm selection for the matching process (Fig. 5)
- The validation and adaptation of the obtained mappings by an expert user (Fig. 6)

Fig. 5 presents on the left hand side, the Ecore metamodel and in the right the UML metamodel. Fig. 5 presents both metamodels in the form of trees. The steps for using our plug-in tool for metamodel matching with SAMT4MDE+ are as follows:

- Import source and target metamodels: the tool loads the UML and Ecore metamodels.
- Select and run a metamodel matching algorithm: in our case SAMT4MDE+ is chosen to propose matched elements between the two metamodels.
- Validate the pairs of matched elements: the user can validate or refuse the pairs of the matched elements (cf. Fig. 6).
- Generate the quality measures: the tool display the main quality measures, *Precision, Recall, F-Measure.*

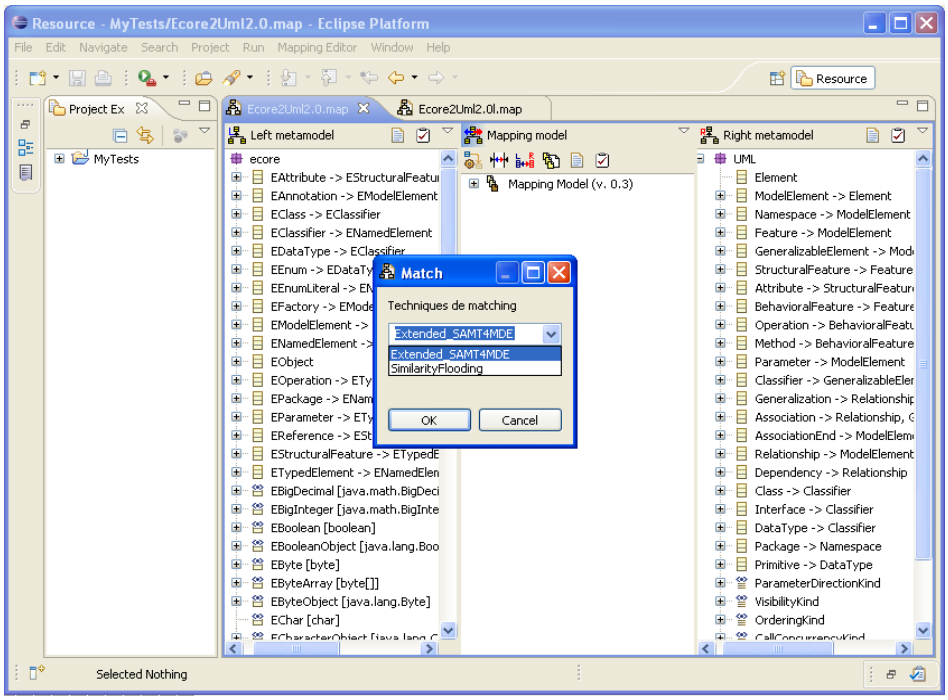

Fig.5.*Algorithm selection for the matching of source and target metamodels*

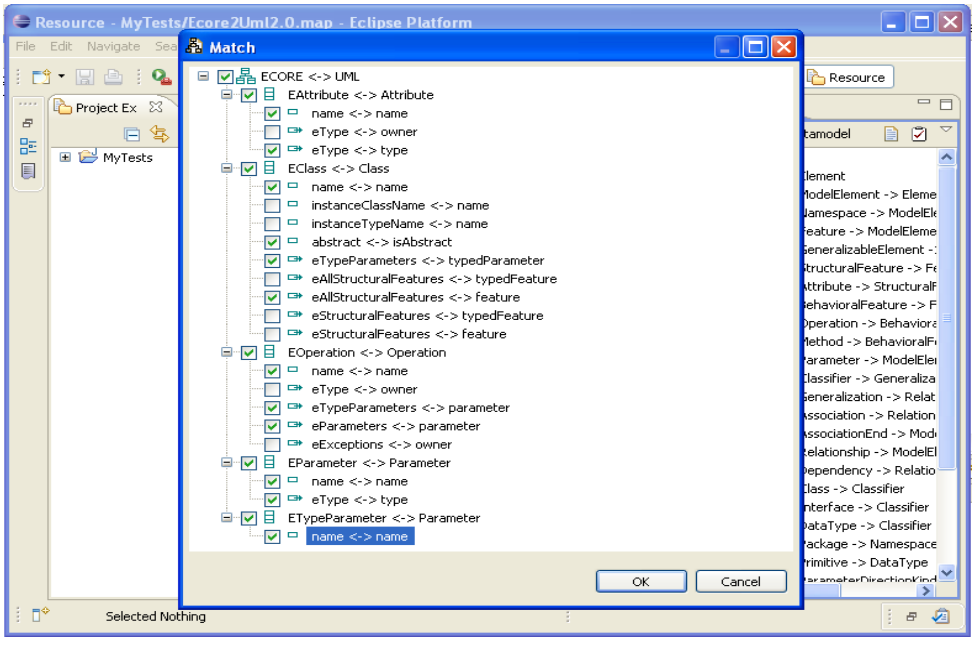

Fig.6.*Validation step of the matched elements*

7 Related work and Conclusion

A semi-automation of the transformation process in MDE/MDA 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. Matching techniques between metamodels are the centerpieces for a semi-automatic transformation process in MDE/MDA. The contribution of this work is twofold: First, we presented the main techniques and artifacts involved in the semi-automatic transformation process. Second, we reviewed five main approaches that have been proposed in the literature for metamodel matching, and, then we have studied from an experimental point of view the two most recent techniques of metamodels matching Similarity Flooding and SAMT4MDE+. This experimental comparison allowed us to get different values of matching quality measures using different couples of metamodels. We have noticed that the algorithm SAMT4MDE+ gave more effective results than those given by the algorithm Similarity Flooding.

In the future work, we will concentrate on how to combine different approaches to enhance the matching process. In addition, we will consider studying the optimization of mapping models which seems to be another important issue in MDE.

References

1. OMG, 2001. Model Driven Architecture (MDA)- document number ormsc/2001-07-01. (2001).
2. 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.
3. 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.
4. Bézivin, J., Hammoudi, S., Lopes, D., & Jouault, F. (2004). Applying MDA Approach for Web Service Platform.8th IEEE International Conference on EDOC. pages58-70.
5. Blanc, X. (Ed. 1). (2005). MDA en action, Ingénierie logicielle guidée par les modèles. Paris, France: EYROLLES.
6. Jouault, F., 2006. Contribution à l'étude des langages de transformation de modèles, Ph.D. thesis (written in French), University of Nantes.
7. OMG, 2005. MOF QVT Final Adopted Specification, OMG/2005-11-01.
8. Lopes, D. (2005a).Study and Applications of the MDA Approach in Web Service Platforms, Ph.D. thesis (written in French), University of Nantes, France.
9. Hammoudi, S., Janvier, J., Jouault, F., 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.
10. Almeida, A.J.P., 2006. Model-driven design of distributed applications. PhD thesis, University of Twente. ISBN 90-75176-422.

11. Hammoudi, S., Lopes, D., 2005. From Mapping Specification to Model Transformation in MDA: Conceptualization and Prototyping. In MDEIS'2005, *First International WorkshopOn Model Driven Development, Miami, USA.*pages15-23.
12. Slimane Hammoudi, Wajih Alouini, Denivaldo Lopes, Marianne Huchard: Towards A Semi-Automatic Transformation Process in MDA : Architecture, Methodology and First Experiments.
13. Lamine Lafi, Waji Alouini, Slimane Hammoudi, Mohamed Gammoudi: Metamodels Matching: Issue, techniques and comparison 2nd Intenational Workshop FTMMD, Joint to International Conference ICEIS. June 2010, Portugal.
14. Feiyu, L. State of the Art: Automatic Ontology Matching, Research Report, School Of Engineering, Jonkoping, Sweden, 2007.
15. Do, H.H., Melnik, S., Rahm, E.: Comparison of schema matching evaluations. In: Web,Web-Services, and Database Systems, Volume 2593 of LNCS, Springer (2002) 221-237
16. Sendall, S., Kozaczynski, W. 2003. Model Transformation – the Heart and Soul of Model Driven Software Development. IEEE Software, Special Issue on Model Driven Software Development, pp42-45, Sept /Oct 2003.
17. Kleppe, A., Warmer, J., Bast, W., 2003. MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley, 1st edition.
18. Del Fabro, M. D., 2007. Semi-automatic Model Integration using matching transformation and weaving models. In SAC'07, ACM.
19. 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.
20. 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 and Web.
21. Falleri, J.R. , Huchard, M. Lafourcade, M. Nebut, C. Metamodel matching for automatic model transformation generation. In: Proceedings of MoDELS '08, (2008) 326–340.
22. Kurtev, J. Bézivin, and M. Aksit. Technological spaces: An initial appraisal. In Int. Federated Conf. (DOA,ODBASE, CoopIS), Industrial track, Los Angeles, (2002)
23. Melnik, S. , Garcia-Molina, H. Rahm, E. Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching. In Proceedings of the 18th international Conference on Data Engineering (February 26-2002). ICDE. IEEE Computer Society, Washington, Pages 117-128.
24. Jose de Sousa Jr, Denivaldo lopes, Daniela Barreiro Claro, and Zair Abdelouahab. A Step Forward in Semi-automatic Metamodel Matching: Algorithms and Tool.
25. Pottinger, R., Bernstein, P.A.: Merging models based on given correspondences.In: VLDB. (2003) 826-873.
26. Chukmol, U., Rifaiem, R., Benharkat, N.: EXSMAL: EDI/XML Semi-Automatic Schema Matching ALgorithm, Proceedings of the Seventh IEEE International Conference on ECommerce Technology, IEEE Computer Society, 422-425, (2005).
27. Jean Remy Falleri : Contributions à l'IDM : reconstruction et alignement de modèles de classes. Ph.D. thesis (written in French), University of Montpillier 2.
28. Eclipse Project. (2006). http://www.eclipse.org.