



Easing Model Transformation Learning with Automatically Aligned Examples*

Xavier Dolques¹, Aymen Dogui², Jean-Rémy Falleri³, Marianne Huchard⁴,
Clémentine Nebut⁴, and François Pfister⁵

¹ INRIA, Centre Inria Rennes - Bretagne Atlantique, Campus universitaire de Beaulieu, 35042 Rennes, France, xavier.dolques@inria.fr

² Supélec Paris, France, aymen.dogui@supélec.fr

³ Université de Bordeaux, France, falleri@labri.fr

⁴ LIRMM, Université de Montpellier 2 et CNRS, Montpellier, France, first.last@lirmm.fr

⁵ LGI2P, Ecole des Mines d'Alès, Nîmes, France, francois.pfister@mines-ales.fr

Abstract. Model Based Transformation Example (MTBE) is a recent track of research aiming at learning a transformation from examples. In most MTBE processes, a transformation example is given in the form of a source model, a transformed model and links between source elements and the corresponding transformed elements. Building the links is done manually, which is a tedious task, while in many cases, they can be deduced from the examination of the source and transformed models, by using relevant attributes, like names or identifiers. We exploit this characteristic by proposing a semi-automatic matching operation, suitable for discovering matches between the source model and the transformed model. Our technique is inspired by and extends the Anchor-Prompt approach, and is based on the automatic discovery of pairs of anchors (pairs of elements for which there is a strong assumption of matching) to support the whole matching discovery. An implementation of the approach is provided for validation on a case study.

1 Introduction

Model transformations are the operational, often automated, part of Model Driven Engineering (MDE), and several transformation languages have been proposed to introduce useful concepts to develop transformations. The QVT standard [1] has been proposed by the OMG to unify the field.

Writing a transformation requires two important skills: firstly a strong knowledge in transformation languages and metamodeling and secondly a good comprehension of the semantics of the source and target domains. While transformation developers have the first skill, the second one is usually owned by the domain experts. This fact makes the development of a model transformation difficult and time-consuming, because the transformation developers have to interact, on abstract concepts of a specific domain, with the domain experts, so as to obtain a correct transformation.

* This research was partially supported by the european project OPEES

Two kinds of approaches have recently been introduced to assist the development of model transformations. The first kind of approach operates at the metamodel level [2,3] and exploits an alignment between the source and target metamodels. It assumes (thus is efficient when) the source and target metamodels are very similar in their structure and terminology. The second approach, Model Transformation By Example (MTBE), uses metamodels and models. It aims at inferring either the transformation [4,5,6], or the result of a transformation [7], by using a set of transformation examples. In this paper we focus on this second kind of approaches.

Applying MTBE requires to have transformation examples: a source model, a transformed model and the links between source and transformed elements. While having a source and a target model is quite easy (a domain expert can create them), retrieving the links between the elements of these models is tedious and time-consuming, because no mainstream metamodeling environment is capable of creating them when models are manually edited. Therefore, these links are usually manually looked for and added. We believe that the major part of these links can be automatically retrieved. Indeed, when the transformed model is created, the names of the transformed elements are usually equal or very similar to the ones of the source elements, maybe using different naming conventions. Also, the underlying metamodels are different but often neighbors of an element in the source model (understood as the instantiation of the metamodel) are transformed into neighbors of the transformed element.

In this paper, after the context description (Section 2), we propose an approach (Section 3), that combines string similarity and schema matching techniques to automatically retrieve the links going from the elements of a source model to their corresponding elements in the transformed model. This approach helps the transformation developers to gather transformation examples, allowing them to benefit from the MTBE approaches. We describe our tool and case study in Section 4. Related work is discussed in Section 5, and we conclude in Section 6.

2 Problem Statement

The MTBE process aims at inferring a rule-based transformation from transformation examples. A classical version of the process is presented in the l.h.s of Figure 1. The input of the process is a transformation example, defined by a source model, a transformed (target) model and matching links between the two models. It results into transformation rules, deduced from the example, that can transform any model conforming to the source metamodel to a model conforming to the target metamodel. Several proposals for the MTBE engine can be used, *e.g.* [4,5,6].

We illustrate this section and the rest of the paper with a classical example of transformation from UML class model to entity-relationship model. The input is thus an example of a UML model, and the corresponding transformed entity-

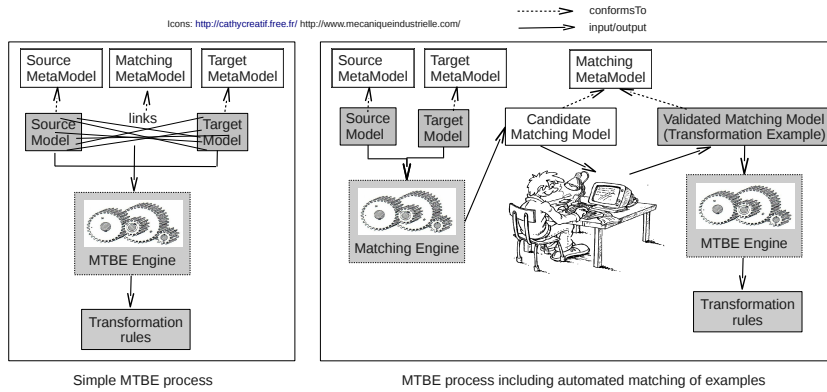


Fig. 1. The MTBE process: a simple view (l.h.s) and including assistance for matching (r.h.s)

relationship model. Figures 2 and 3 give the used metamodels (in ecore format) for UML class diagrams and entity-relationship models.

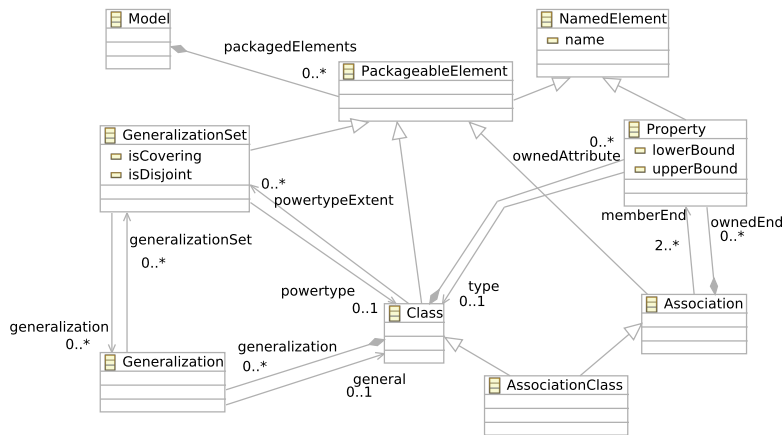


Fig. 2. A metamodel for UML class diagrams (drastic simplification of the UML metamodel)

The chosen example models literary texts (novels or poetry), written by (and with a foreword from) authors. Each text has one or several styles. The examples are given with concrete syntax in Figures 4 and 5, and an excerpt is given with abstract syntax in Figures 6 and 7 (in the form of an instance of the metamodels). Though less readable, the abstract syntax is the one actually

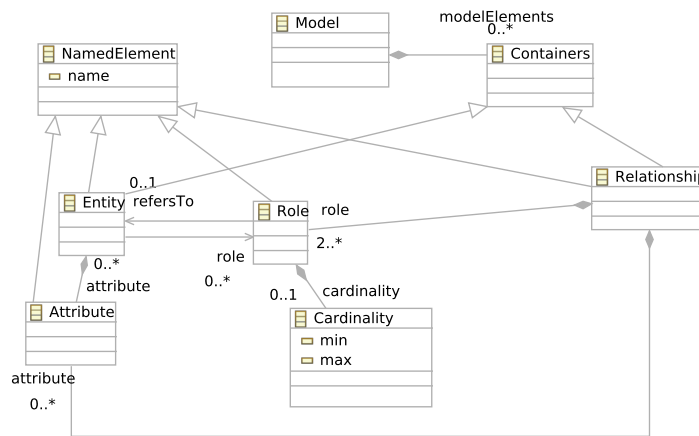


Fig. 3. A metamodel for Entity-Relationship models

handled by the tools. The presented excerpts show the authors writing texts but hide the poetry, the style, and the fact that authors write a foreword for texts.

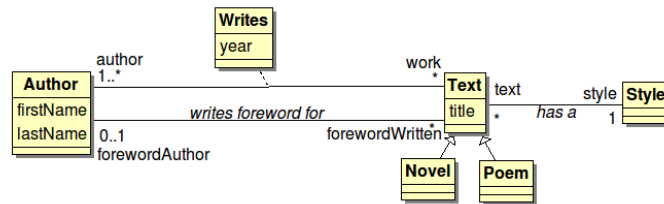


Fig. 4. An example UML model for the UML2ER transformation

The examples are quite easy to build, moreover they will be useful for testing or documentation purposes. However, to feed the MTBE process, the transformation links have to be given. For example, one has to specify that the **Author** class from Figure 4 has to be transformed into the **Author** entity from Figure 5, and that the inheritance link from **Novel** to **Text** has to be transformed into an **is_a** relationship (and in fact using the abstract syntax shown in Figures 6 and 7: the generalization element has to be transformed into the **is_a** relationship and the two linked roles). This is a tedious task, and only dedicated to the MTBE process. Our purpose is to use string similarity and alignment methods to generate part of those links. The resulting architecture for the MTBE process is presented in the r.h.s of Figure 1. The links from the source model to the transformed model are partly generated by a matching engine. The generated

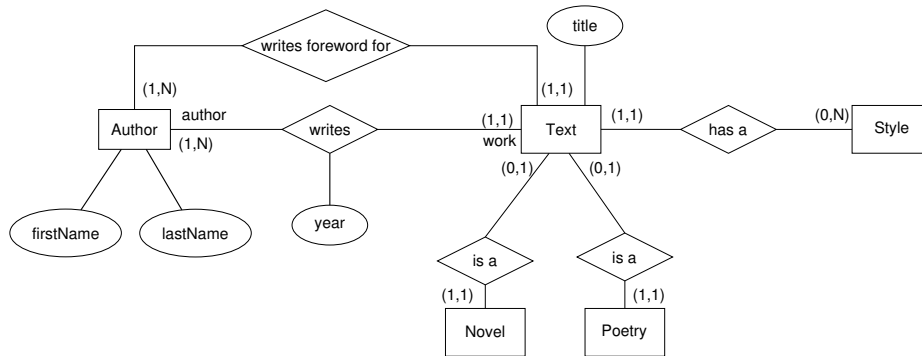


Fig. 5. An example of Entity-relation model for the UML2ER transformation

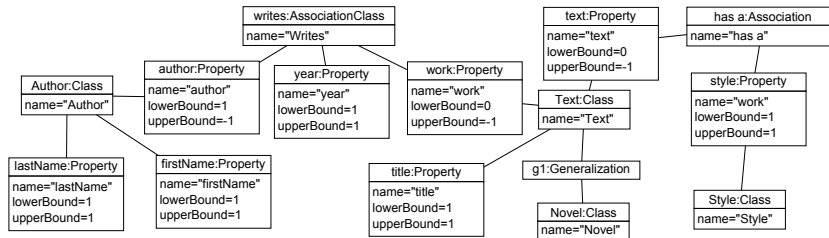


Fig. 6. Excerpt of the UML example with abstract syntax

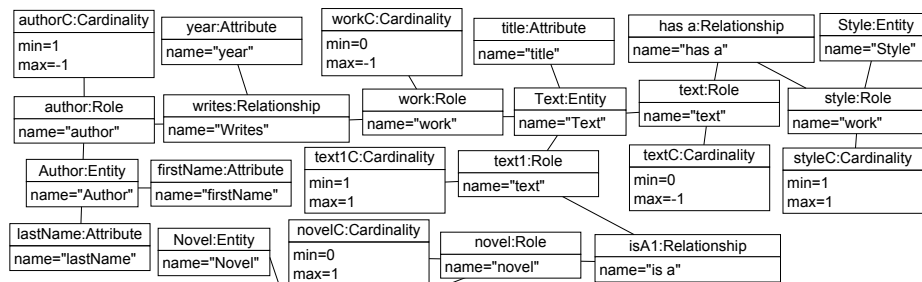


Fig. 7. Excerpt of the ER example with abstract syntax

links are checked by an expert. The *matching engine* takes source models and target (transformed) models, and provides a candidate matching model (matching links). This candidate matching model is proposed to an expert who validates and fixes the matching model, producing the validated matching model which composes (together with the source and target model) the transformation example. Our aim is to assist the domain expert as far as possible, providing him with an initial matching solution where nearly obvious matches are included. For example, with the example of the literary texts, we discover all the “obvious” matchings (the class `Text` maps to the entity `Text`, the `firstName` property

maps to the `firstName` attribute, ...) and also more sophisticated mappings such that: the generalization from `Novel` to `Text` maps to an `is_a` relationship and the two roles `text` and `novel`. Even if the complete mappings are not discovered (as explained in the case study, we have for the UML2ER example a precision of 1.0 and a recall of 0.7), the generation of a partial mapping is a valuable help in a MTBE process.

To sum up, we study a matching problem that considers two models that on one hand come from two different metamodels (maybe very divergent), with relations differently named and organized; and on the other hand contain a large set of common underlying entities and a large set of similarly named entities, due to the common underlying natural semantics.

3 The model matching Approach

The literature offers several approaches to build a matching between two structures [8,9]. Due to the specificities of our problem, we propose a tool inspired by the Anchor-Prompt approach [10]. The original approach is a two-step process designed to match ontologies. The first step is the discovery of matches with a high confidence rate (anchors), while the second one propagates those anchors so as to discover other matchings. Our approach follows the same two steps and improves the second, they are described below.

3.1 Anchor discovery

This first step consists in finding pairs of anchors, i.e. initial matchings. The original Anchor-Prompt approach does not specify a process to discover pairs of anchors. In our case, the target model is the result of the transformation of the source model, and the entities and their values, are very close. Although the source and target metamodels are different, it is common that model entities have an identifying attribute (such as the name) and that this attribute value does not change much during the transformation. In values, we can have slight variations due to naming conventions: prefixes or suffixes can often be added, but it is mostly improbable that both a prefix and a suffix is added so we assume that they are not very different. A high confidence rate is needed for this subset since the next step strongly depends on the quality of those pairs of elements. Those matchings cannot be detected using types, as the two models may be instances of different metamodels, thus we need to rely on some attribute values of those elements, *e.g.* the attribute `name` of the UML metaclass `NamedElement`, that we assume to remain nearly unchanged after transformation.

Let Att_{src} and Att_{tgt} be the sets of all the attributes of all the elements respectively in the source and target models. Let $P = Att_{src} \times Att_{tgt}$. We want to extract $M \subset P$, a set of attribute pairs validating a matching test. From this set M we generate a set of pairs of anchors A by replacing each attribute value in the pairs of M by the entity containing this value. A general algorithm for the anchor discovery is given in Listing 8.

We tested several matching operations and we present here the most relevant:

```

proc Anchor-Discovery (In: AttSrc set of Attribute values ,
                      AttTgt set of Attribute values ,
                      Out: A set of Entity pairs)
M := AttSrc x AttTgt;
P := empty set;
A := empty set;
for pair in M do
  if match(pair.source , pair.target)
  then P.add(pair);
for pair in P do
  A.add( (pair.source.entity , pair.target.entity) );

```

Fig. 8. General process of anchor discovery

- **equality:** the most obvious matching operation is the equality. If two elements share exactly the same value, then they are likely to be matched. But this test is worthless if we do not check the occurrence frequency of the values matched. Indeed it appears in our tests that some values are not relevant, such as stereotypes or cardinalities in class diagrams. Thus, another condition for two attributes to be matched is that their value appears once and only once in the source and target models. This matching operation appears to be reliable as it brings a precision of 1 in most of our tests.
- **substring:** the drawback of the previous operation is that it may pass through simple renaming transformations, that may add or delete a prefix or a suffix. To tackle this issue after the equality test we check, if the values are character strings, if one value is a substring of the other. As with the previous operation, we must be cautious on the obtained results, and check if the substring exists as an attribute value in the model that contains the longest string value of the couple. This method is once again reliable in most of the cases, and in our tests it always gave a precision of 1.

We also experimented with other matching operations that use the longest common substring or the Levenshtein distance, but our context implies that values in the target model remain really close to values from the source model, even capital or lowercase letters are important. To find highly reliable matchings we cannot afford to use distance methods that may lower the precision of the matchings.

At the end of this step, for our example, *A* includes, among others, the pairs of anchors (*Text:Class, Text:Entity*) or (*has a:Association, has a:Relationship*).

3.2 Anchor propagation

Considering the anchors as a nearly correct match, we propagate this information on paths outgoing from an anchor and leading to another close anchor to discover other potential matches. Indeed, we assume that on a path between two anchors, even if the metamodels are different, when an entity *e* is close to another entity *f* in the source model, it is likely that the entity which results from the transformation of *e* is close to the entity which results from the transformation

of f . Due to the differences between the metamodels, the path between the two entities is likely to be differently labeled. The process cannot be correct in all the cases, because during the transformation some elements can be removed or added, but it is likely to produce many correct matches.

Source and target models may be seen as two labeled graphs G_{src} and G_{tgt} , in which a node represents an instance of a class from the metamodel, and an edge represents an association between class instances (cf. graphs in Figures 6 and 7). We enumerate from the two graphs all the paths connecting two anchors and whose length is less than a constant α .

We align the nodes from a path between two anchors a_1 and a_2 of G_{src} with the nodes from a path between the anchors a'_1 and a'_2 if $(a_1, a'_1) \in A$ and $(a_2, a'_2) \in A$. For example we will align a path between `Text:Class` and `has a:Association` with a path between `Text:Entity` and `has a:Relationship`. One difference from the original Anchor-Prompt approach is in the alignment of paths with different lengths, for which Anchor-Prompt only aligns pairs of paths of identical length. This way the original approach leads to match elements that are on the same position on the path. More generally, in our approach, when aligning two paths, we consider each pair of nodes as shown in Figure 9, but not with the same weight: we are giving the maximum weight to pairs of nodes that are in the same position relatively to each node's path length.

Let X and Y be two lists of nodes, respectively from G_{src} and G_{tgt} and representing two paths to be aligned. X and Y are starting by two anchors that are matched together. Let $x \in X$ and $y \in Y$. $index(x)$ and $index(y)$ are the position of the nodes in the list starting from 1. The weight of the pair (x, y) is defined by:

$$W(x, y) = 1 - \left| \frac{index(x)}{length(X) + 1} - \frac{index(y)}{length(Y) + 1} \right|$$

For instance, $W(x_1, y_1) = 1 - \left| \frac{1}{6} - \frac{1}{4} \right| = 0.92$ and $W(x_1, y_2) = 1 - \left| \frac{1}{6} - \frac{2}{4} \right| = 0.67$, showing that x_1 is more likely to match with y_1 rather than with y_2 .

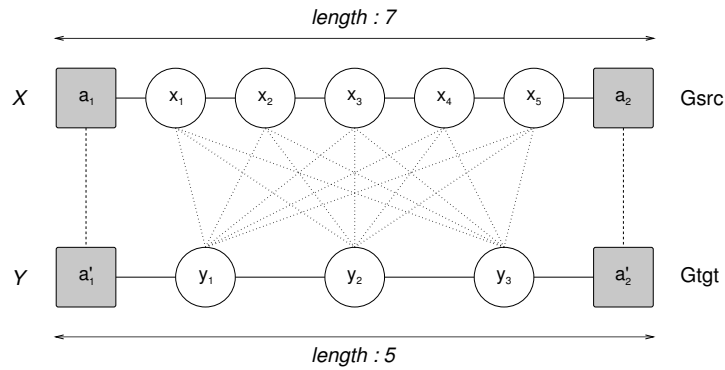


Fig. 9. Aligning two paths

We align each pair of paths whose extremities are anchors, incrementing the similarity coefficient of a pair of elements each time the elements appear in two paths to be aligned. The increment is computed depending on the weight of the pair.

At the end of the process we have a set of node pairs with similarity coefficients. The similarity coefficients have no meaning if compared globally. If a node appears only in one path between two anchors, then all the pairs containing this node will have a similarity coefficient that may be lower than ones with nodes appearing in many paths between two anchors, making difficult to decide which pairs are important. However, comparing the similarity coefficients of all the pairs containing one node is more meaningful. Indeed, the pair with the highest similarity coefficient is more likely to be a matching, so all the similarity coefficients of this node should be compared relative to it.

Figure 10 shows in the case of the object of name “text” and type *Property* in our example how it is deduced that its matching element in the target model is the object of name “text” of type *Role* attached to the *Relationship* named “hasa”. We see that for all the matching links containing the *Property* “text”, the highest similarity coefficient is obtained with the *Role* “text”, and none of the other matching links pass over a threshold that, after some experiments, we fixed at 80% of the highest value for an object. The same principle can be used symmetrically for the *Role* “text” that validates the choice of this matching.

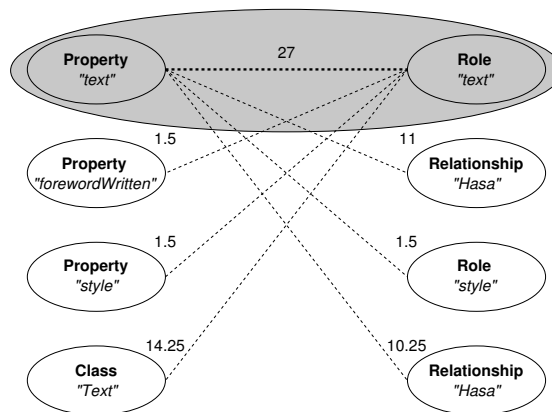


Fig. 10. Filtering of the similarity coefficient

4 Case study

This section presents the experimental results obtained with our approach. We present the implementation used to run the experimentation, then the experimental protocol and data, and the obtained results with their interpretation.

4.1 Tool implementation

A tool called MANDARINE⁶ has been developed based on the approach described above, with the objective of improving MTBE processes. It has been designed to be efficiently integrated with the approach described in [11] by using a part of its metamodel but it can also be used as a standalone tool for an integration with another approach. It is based on the Eclipse Modeling Framework, as it is a modeling facility widely adopted by the MDE community, and has been implemented in JAVA.

The tool takes as input the source and target example models of the transformation and returns the computed matchings between the models as a model conforming to *Matching Model*, the metamodel described in Figure 12. This metamodel is also used to describe the input of the MTBE process from [11]. Technically, the only requirement for the input models is that they must be recognized by EMF as instances of an Ecore Model.

An informal representation of the architecture of the tool is provided in Figure 11. The process of matching is split in two distinct steps: the first one implements the anchor discovery process from section 3.1 with the ability to choose the matching operation between attributes. This step returns a Matching Model as a result. The second part implements our adaptation of Anchor-Prompt from section 3.2, where the maximum length of the considered paths and the threshold for filtering the similarity coefficients may be passed as input with a Matching Model. Although the two processes are designed to be launched one after another, they are independently implemented to allow flexibility of use and further evolution.

The *Matching Evaluation* tool is the infrastructure to evaluate the discovered matchings against an expert matchings, it will be discussed in Section 4.

4.2 Testing protocol and metrics

As an extension of the tool previously described, we designed a testing platform presented on the right hand side of Figure 11. This platform takes as input two matching models, one created by an expert that gives a reference result and that we will refer to as A_{expert} , and another one automatically computed that will be called A_{auto} and of which we want to measure the quality. Those two models are then automatically compared according to several metrics. Model matching being similar with schema matching or ontology matching, we propose here to use metrics from those last two domains. We will especially refer to the metrics described in [12]: precision, recall and overall. In the following we will use $A_{positives} = A_{auto} \cap A_{expert}$ as the set of matching links that are present in both expert and automatically obtained matchings (see Figure 13).

Precision The precision calculates the ratio of correct matchings in A_{auto} over the size of A_{auto} . Therefore this metric depends on the quantity of bad matchings

⁶ Model AligNement Disseminating AttRIBUTE INstances Equivalences

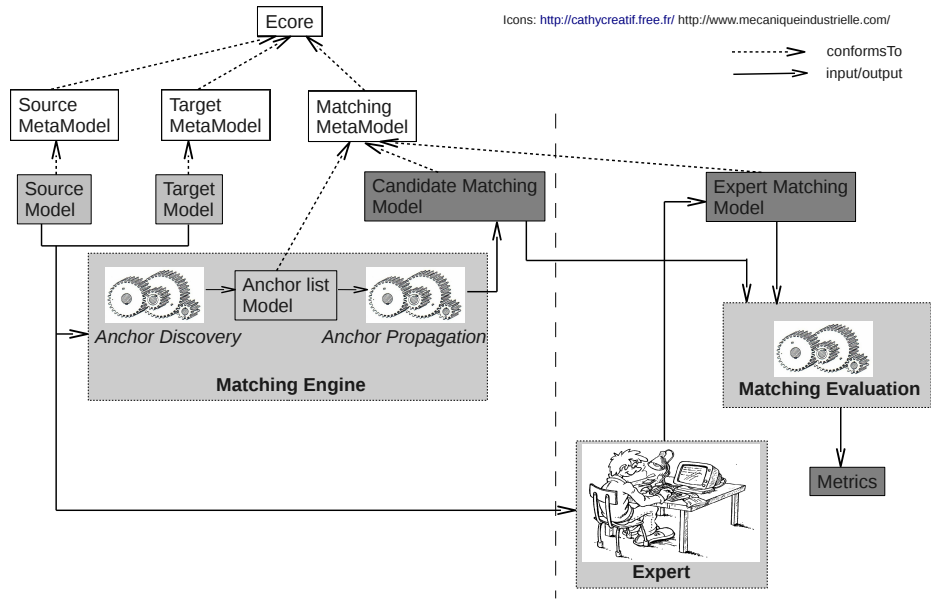


Fig. 11. An architecture for the anchor-based matching tool

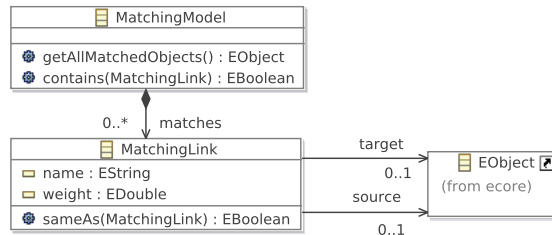


Fig. 12. Metamodel describing the matching used in our tool

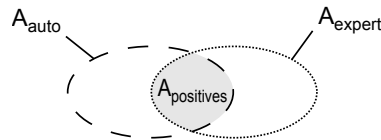


Fig. 13. Schematic illustration of matching comparison

introduced by the approach. It is a rational number going from 0 (if all the obtained matchings are wrong) to 1 (if there are no wrong matchings at all *i.e.* $A_{auto} \subseteq A_{expert}$). It is calculated by the formula: $precision = \frac{|A_{positives}|}{|A_{auto}|}$

Recall The recall calculates the ratio of correct matchings in A_{auto} over the number of correct matchings, *i.e.* the size of A_{expert} . Therefore this metric depends

on the quantity of matchings from A_{expert} missed by the approach. It is a rational number going from 0 (if there are no matchings from A_{expert} in A_{auto}) to 1 (if all the matchings from A_{expert} have been discovered, *i.e* if $A_{expert} \subseteq A_{auto}$). It is calculated with the following formula: $recall = \frac{|A_{positives}|}{|A_{expert}|}$

Overall The overall combines precision and recall to quantify the needed effort to go from A_{auto} to A_{expert} , relatively to the size of the expert model. It is a rational number bounded between $-\infty$ and 1. Overall is 1 if $precision = recall$, and 0 if the number of wrong matchings in A_{auto} added to the number of missing matchings is equal to the size of A_{expert} . If this number is greater than the size of A_{expert} then overall is a negative number. It is calculated with the following formula: $overall = 1 - \frac{(|A_{auto}| - |A_{positives}|) + (|A_{expert}| - |A_{positives}|)}{|A_{expert}|}$

4.3 Data

We propose here to validate our approach by applying it on 22 model transformations⁷. The data used for this case study comes from several sources: home made transformations, UML refactorings [13] and transformations from the ATL zoo of transformations [14]. In the latter case, the models used as examples are given with the transformation.

4.4 Results

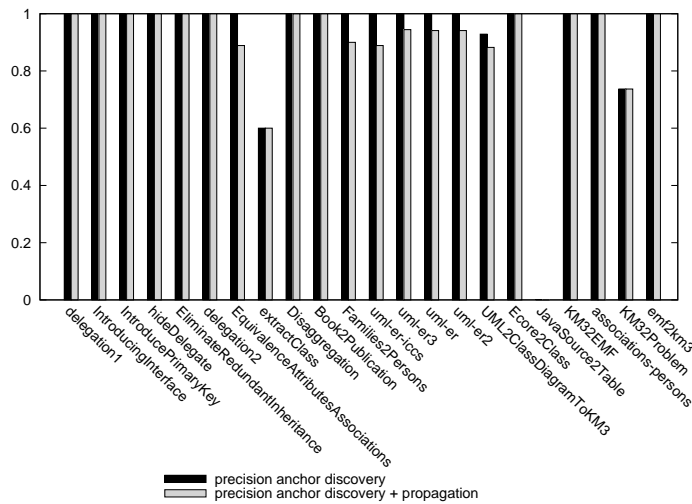


Fig. 14. Precision measured on the case study

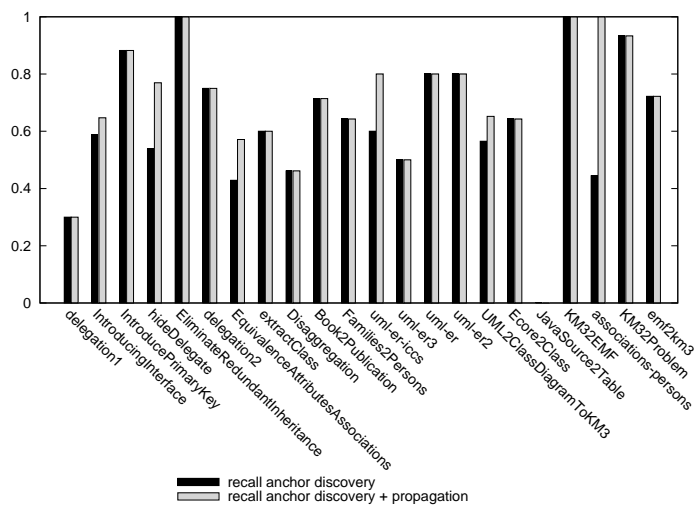


Fig. 15. Recall measured on the case study

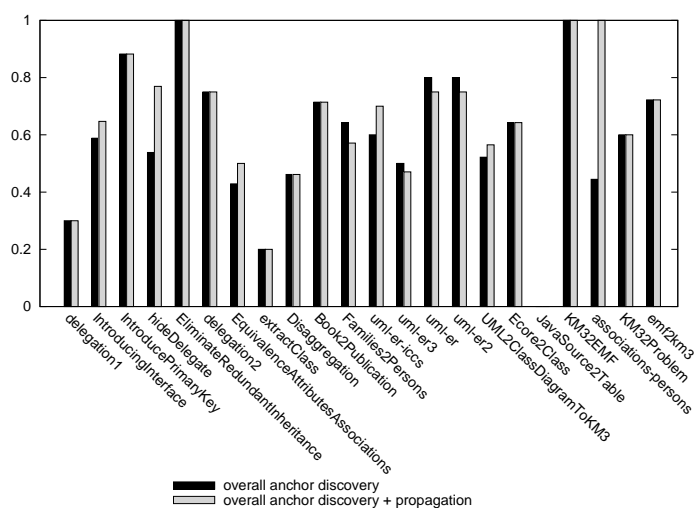


Fig. 16. Overall measured on the case study

The evaluation program has been applied twice for each transformation, first to evaluate the anchor discovery result and then the whole process result. Precision and recall have been measured in each case, and the overall has been calculated from them. Figures 14, 15, and 16 present the obtained results. For the anchor discovery step, we only show the results obtained with the substring

⁷ The detail of all the transformations is given at:

<http://www.lirmm.fr/%7Enebut/Publications/ArticleSupplements/ECMFA2011/examples-casestudy.html>.

similarity metrics, since it is the one giving the best results. We can see in Figure 14 that the precision obtained is good, with a value of 1 in many cases, especially for the anchor discovery process. It enforces our hypothesis that the anchor discovery result can be considered as reliable. On the other hand, the recall values are not as good, but this was expected as our approach intends to assist the matching creation and not to completely automate it. It sometimes occurs that for a same transformation, the propagation of the anchors degrades the precision, while not increasing the recall. That shows that this step could be enhanced; ways to enhance this step are given in conclusion. It can also be seen that one of the transformation is giving no results at all. This transformation, named *JavaSource2Table* is an extraction of statistics from a Java program and therefore does not keep the structure of the source model at all during the transformation.

The overall values are all positives, meaning that if we consider adding or removing a match as an atomic operation with the same cost, then in each case our approach is decreasing the cost of matching two models completely by hand, in many cases by half or more. However, with the good results obtained for the precision, the correction operations of the matching model are mainly adding operations.

5 Related work

Automating the discovery of mappings between database and XML schemas, ontologies, or (meta)models has been thoroughly investigated. We encourage the reader to refer to [8] or [9] for an in-depth description of the existing work. Most of the main approaches such as [15,16,17,18,19] make the assumption that the relations between the two models being compared are identical. The basic idea exploited by these approaches is to compute first a similarity between the elements using their names, and to compute then a similarity using the structure. To compute this second kind of similarity, they assume that the relations between the elements have the same kind in the two compared models. In the MDE terminology, it could be translated as: *the models being compared have the same meta-model*. In our case the models being compared conform to two different meta-models. Therefore, a straightforward use of one of these techniques may not exploit all the potential of those approaches. In [20], they use (among other similarity calculations) the similarity flooding ([15]) and they construct adequate propagation models that capture the semantics of the relationships. These propagation models are nevertheless specific to the studied meta-models (and their meta-meta-model(s)) and are designed by an expert.

Moreover, our alignment problem has two characteristics to exploit: since the source and target models are supposed to be written by the same person, the identifiers and names are very likely to be nearly identical. Second, since the target model results from a transformation from the source model (this is a main difference with approaches that study the meta-model matching), the structures of the two models are supposed not to be radically different one from the other.

That is why we adapted another kind of approach [10]. Indeed, due to the first characteristic of the problem, the anchors should be easy to detect, and due to the second characteristic, the mapping algorithm exploiting the anchors should be efficient.

Concerning the context of application of our matching approach (model transformation by example), several proposals [21,5,6,7] aim at inferring the rules of a transformation or its result using a set of examples. In all these approaches, an example consists at least of a source model, a transformed model, and the links between the elements of these two models. None of those approaches include an assistance to build those last links, and all of them would benefit from the approach proposed in this paper.

In [22], a By-Demonstration approach is proposed to generate model transformations. It consists in building step by step examples of transformations, following strong naming constraints. Thus examples are incrementally built, and both the increments and the naming constraints allow links to be deduced and rules to be inferred.

6 Conclusion

Model transformation by example (MTBE) is a promising approach to ease the development of model transformations. Several proposals have been developed to make MTBE feasible. Those approaches take as input examples of a transformation, that is: source models, target (transformed) models, and the transformation links from source model elements to target model elements. While designing the source and target models is a simple and valuable task (those models can later on be used for documentation or testing purpose), making explicit the transformation links is a tedious and error-prone task. In this paper, we detailed an approach and a tool using text analysis and alignment techniques to partly generate those links. Such a mapping engine provides valuable help to the expert in charge of designing an example for an MTBE process. In order to validate the proposed approach, we performed experiments on a set of model transformations, and compared, based on metrics such as precision and recall, the matchings generated by our matching tool to reference (manually built) matchings. The results obtained are promising: we obtain very good precision results and fairly good recall results. The experiments also show that the propagation step could be more efficient (loss of precision, for sometimes no gain in recall), while the step of discovery of the anchors is sufficiently efficient (precision of 1, and sufficient recall). Future work will consist in enhancing the propagation step, first integrating in it the similarity metrics between the values of elements computed in the first step, and secondly taking into account attributes that are not of String type, like cardinalities.

References

1. OMG: MOF QVT Final Adopted Specification. Object Modeling Group. (2005)

2. Lopes, D., Hammoudi, S., Bézivin, J., Jouault, F.: Generating transformation definition from mapping specification: Application to web service platform. In: CAISE'05, LNCS 3520. (2005) 309–325
3. Falleri, J.R., Huchard, M., Lafourcade, M., Nebut, C.: Meta-model Matching for Automatic Model Transformation Generation. In: MODELS'08, LNCS 5301, Springer (2008) 326–340
4. Wimmer, M., Strommer, M., Kargl, H., Kramler, G.: Towards model transformation generation by-example. In: HICSS '07: Proc. of the 40th Annual Hawaii International Conf. on System Sciences, IEEE Computer Society (2007) 285b
5. Balogh, Z., Varró, D.: Model transformation by example using inductive logic programming. *Software and Systems Modeling* (2008) Appeared online.
6. Dolques, X., Huchard, M., Nebut, C.: From transformation traces to transformation rules: Assisting model driven engineering approach with formal concept analysis. In: Supplementary Proceedings of ICCS'09. (2009) 15–29
7. Kessentini, M., Sahraoui, H., Boukadoum, M.: Model Transformation as an Optimization Problem. In: MODELS'08, LNCS 5301, Springer (2008) 159–173
8. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. *VLDB J.* **10**(4) (2001) 334–350
9. Shvaiko, P., Euzenat, J.: A survey of schema-based matching approaches. In: *J. Data Semantics IV*, Volume 3730 of LNCS. (2005) 146–171
10. Noy, N.F., Musen, M.A.: Anchor-prompt: Using non-local context for semantic matching. In: Proc. of the Workshop on Ontologies and Information Sharing at IJCAI-2001, Seattle (USA) (2001) 63–70
11. Dolques, X., Huchard, M., Nebut, C., Reitz, P.: Learning transformation rules from transformation examples: An approach based on relational concept analysis. In: 14th IEEE International Enterprise Distributed Object Computing Conference Workshops of EDOC'10, IEEE Computer Society Press (2010) 27–32
12. Do, H.H., Melnik, S., Rahm, E.: Comparison of schema matching evaluations. In: *Web, Web-Services, and Database Systems*. (2002) 221–237
13. Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D.: *Refactoring: Improving the Design of Existing Code*. Addison-Wesley (2000)
14. ATL transformation zoo: <http://www.eclipse.org/m2m/at1/at1Transformations/>
15. Melnik, S., Garcia-Molina, H., Rahm, E.: Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In: ICDE, IEEE Computer Society (2002) 117–128
16. Do, H.H., Rahm, E.: Coma - a system for flexible combination of schema matching approaches. In: VLDB, Morgan Kaufmann (2002) 610–621
17. Madhavan, J., Bernstein, P.A., Rahm, E.: Generic schema matching with cupid. In: VLDB, Morgan Kaufmann (2001) 49–58
18. Ehrig, M., Staab, S.: Qom - quick ontology mapping. In: *International Semantic Web Conference*. LNCS 3298, Springer (2004) 683–697
19. Euzenat, J., Loup, D., Touzani, M., Valtchev, P.: Ontology Alignment with OLA. In: Proc. of the 3rd EON Workshop, 3rd Int. Semantic Web Conf. (2004) 333–337
20. Fabro, M.D.D., Valduriez, P.: Towards the efficient development of model transformations using model weaving and matching transformations. *Software and System Modeling* **8**(3) (2009) 305–324
21. Wimmer, M., Strommer, M., Kargl, H., Kramler, G.: Towards model transformation generation by-example. In: HICSS, IEEE Computer Society (2007) 285
22. Langer, P., Wimmer, M., Kappel, G.: Model-to-model transformations by demonstration. In Tratt, L., Gogolla, M., eds: *ICMT*. Volume 6142 of *Lecture Notes in Computer Science*, Springer (2010) 153–167