The Journal of Systems and Software 97 (2014) 1–14

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Search-based metamodel matching with structural and syntactic measures

Marouane Kessentini[a,*], Ali Ouni[a,b], Philip Langer[c], Manuel Wimmer[c], Slim Bechikh[a]

[a] University of Michigan, USA
[b] Université de Montréal, Canada
[c] Vienna University of Technology, Austria

ARTICLE INFO

Article history:
Received 19 March 2013
Received in revised form 21 February 2014
Accepted 17 June 2014
Available online 18 July 2014

Keywords:
Model matching
Search-based software engineering
Simulated annealing

ABSTRACT

The use of different domain-specific modeling languages and diverse versions of the same modeling language often entails the need to translate models between the different languages and language versions. The first step in establishing a transformation between two languages is to find their corresponding concepts, i.e., finding correspondences between their metamodel elements. Although, metamodels use heterogeneous terminologies and structures, they often still describe similar language concepts. In this paper, we propose to combine structural metrics (e.g., number of properties per concept) and syntactic metrics to generate correspondences between metamodels. Because metamodel matching requires to cope with a huge search space of possible element combinations, we adapted a local and a global metaheuristic search algorithm to find the best set of correspondences between metamodels. The efficiency and effectiveness of our proposal is evaluated on different matching scenarios based on existing benchmarks. In addition, we compared our technique to state-of-the-art ontology matching and model matching approaches.

1. Introduction

Model-Driven Engineering (MDE) considers models as first-class artifacts during the software lifecycle (Bézivin, 2005). Available techniques, approaches, and tools for MDE are growing and they support a huge variety of activities, such as model creation, model validation, model simulation, and code generation to name just a few. In this context, model transformations (Mens and Van Gorp, 2006; Sendall and Kozaczynski, 2003) are vital for realizing support for those activities. Furthermore, the use of different domain-specific modeling languages and diverse versions of the same language (Cicchetti et al., 2008) often entails the need for having the possibility of model exchange between languages and their accompanying tools. Model exchange is mostly achieved by model transformations for down-grading/up-grading models to different language versions and for translating them between different modeling languages (Tratt, 2005). The first step in transformation development (Haas, 2007) is to find the corresponding concepts of two modeling languages, i.e., finding similarities among metamodel elements. In recent years, more attention is paid by the MDE community to find manual, semi-automated, and fully automated solutions for this matching problem between metamodels (de Sousa et al., 2009; Del Fabro and Valduriez, 2009; Dolques et al., 2011; Falleri et al., 2009; Garcés et al., 2009; Kappel et al., 2007; Kolovos et al., 2009; Voigt et al., 2010; Voigt, 2010). Please note that correspondences are also needed as crucial input to develop automation support for other model management scenarios, such as model differencing, model merging, and model synchronization.

Metamodel matching in general can be reduced to the graph isomorphism problem between two given graphs (Kolovos et al., 2009). Theoretically, the graph isomorphism problem is NP-hard (Khuller and Raghavachari, 1996) and the majority of existing matching approaches focuses mainly on the use of structural metrics or element identifiers to compute similarity values for metamodel elements (Brun and Pierantonio, 2008; de Sousa et al., 2009; Del Fabro and Valduriez, 2009; Falleri et al., 2009; Garcés et al., 2009). However, especially for many scenarios, this is often insufficient, because element identifiers are not available, the terminology used for naming the elements in the metamodels may be very heterogeneous, and the structure of the metamodels may vary drastically while still reflecting similar modeling concepts.

* Corresponding author. Tel.: +1 313 5836366.
E-mail addresses: marouane@umich.edu, kessentiniglp@yahoo.fr (M. Kessentini), ouniali@iro.umontreal.ca (A. Ouni), langer@big.tuwien.ac.at (P. Langer), wimmer@big.tuwien.ac.at (M. Wimmer), slim@umich.edu (S. Bechikh).

http://dx.doi.org/10.1016/j.jss.2014.06.040
0164-1212/

In this paper, we propose a hybrid approach to combine structural and syntactic similarity metrics for metamodel matching. For structural comparison, we adapted some widely used and well-known quality metrics such as number of subconcepts, cohesion, and coupling (Fenton and Pfleeger, 1997). In addition, we adapted new syntactic measures (Yates and Neto, 1999) (compared to existing work for metamodel matching) that estimate the similarity between names of concepts.

To the best of our knowledge, we present the first work using *optimization techniques* to find the best matching solution that maximizes structural and syntactic measures. Our heuristic search algorithm takes two metamodels as input (named *source* and *target* metamodel) and a list of structural and syntactic similarity metrics, and generates a list of correspondences between the source and target metamodels as output. Our approach generates not only *one-to-one* correspondences, but also *many-to-many* correspondences. The best solution, i.e., set of correspondences, will maximize both structural and syntactic similarity measures. As a huge number of possible metamodel element combinations have to be considered, we use a *hybrid metaheuristic method* to explore the solution space (Bechikh et al., 2008). In particular, we adapted a global search, namely genetic algorithm (GA) (Moscato, 1989), to generate an initial solution and, subsequently, a local search, namely simulated annealing (SA) (Kirkpatrick et al., 1983), to refine the initial solution generated by GA. This hybrid scheme was selected for the metamodel matching problem, because this problem is characterized by a very high dimension search space that requires, at a first glance, the identification of the promising regions, and subsequently, the local examination of the neighborhood(s) to discover a satisfying quasi-optimal solution.

In this paper, we present the following contributions:

- We combine different measures, taking into account the structure and the syntax, to find the best set of correspondences between two metamodels (cf. Section 3).
- We combine local and global metaheuristic search algorithms to deal with the complexity of metamodel matching (cf. Section 4). However, the novelty of our proposals is not related to the combination of global and local search to explore the search space but the adaptation of both algorithms to the metamodel matching problem.
- The effectiveness and efficiency of our proposal is evaluated on a set of different matching scenarios based on existing benchmarks (cf. Section 5). In addition, we compared our approach to four ontology-based approaches (Aumueller et al., 2005; Ehrig and Sure, 2005; Euzenat, 2004; Kalfoglou et al., 2005), as well as to EMF Compare (Brun and Pierantonio, 2008) and Atlas Model Weaver (AMW) (Del Fabro and Valduriez, 2009), the state-of-the-art model matching tools for the Eclipse Modeling Framework (EMF). Our results show that our search-based approach was significantly better than state-of-the-art matching tools over an average of 50 runs.

The remainder of this paper is structured as follows: The next section gives an overview on the challenges of metamodel matching by using a motivating example that is subsequently used as a running example. Section 3 introduces the measures used in the metamodel matching process and Section 4 describes our adaptation of optimization techniques to the metamodel matching problem domain. Section 5 describes obtained results of our metamodel matching experiment for several matching scenarios and further represents the result of comparing our approach to available matching tools. Section 6 summarizes related work, and finally, conclusions and future work are given in Section 7.

2. Metamodel matching issues

In case of one-to-one correspondences, i.e., a correspondence connects exactly two elements, the complexity of a naïve metamodel matching approach is the Cartesian product of the elements of both metamodels to be matched. If more complex correspondences are computed, such as one-to-many or even many-to-many correspondences, the search space becomes even larger (cf. Section 4.3 for a more detailed discussion). Thus matching is typically considered to be a computation intensive task, especially when large artifacts containing many elements have to be matched.

2.1. Metamodel heterogeneities

The main challenges in matching two artifacts are the heterogeneities between them (Kashyap and Sheth, 1996), because they drastically complicate the identification of correspondences. In case two completely equivalent models are matched, every element in one artifact would have an equivalent, clearly corresponding element in the other model. However, heterogeneities are often occurring in practice, making sophisticated matching techniques and one-to-many or even many-to-many correspondences inevitable.

The following heterogeneities may occur between metamodels (Wimmer et al., 2010):

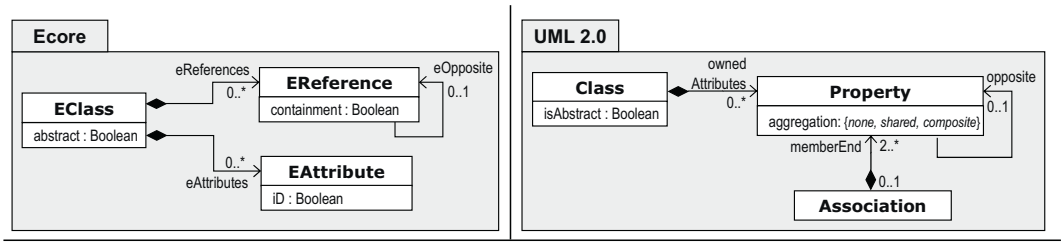
- *Structural heterogeneity*: Different modeling constructs (encodings) are used to represent the same modeling concept. For instance, one metamodel uses two classes connected by one reference, while the other metamodel expresses the same information with one class.
- *Syntactic heterogeneity*: The same modeling concepts are referred by differently named model elements or different modeling concepts are referred by the equally named model elements.
- *Formal language heterogeneity*: Different formal languages (e.g., relational vs. hierarchical vs. graph-based languages) are applied for specifying the metamodels.

In this paper, we are focusing on structural and syntactic heterogeneities, whereas it is assumed that the metamodels to be matched are defined in the same metamodeling language. Thus, formal language heterogeneity is not considered, but may be resolved by a preprocessing step that transforms the metamodels to be matched to a representation that uses a common formal language (Kurtev et al., 2002; Wimmer, 2009).

2.2. Running example

To make the metamodel matching challenge more concrete, we introduce a running example that is used throughout the paper. In particular, we use a simplified excerpt of a frequently occurring model exchange case study, in which two different structural modeling languages UML Class Diagrams (Anon., 2014d) and Ecore (Anon., 2014c) are bridged. This scenario is frequently occurring in practice to bridge traditional UML tools with the Eclipse Modeling Framework, cf. e.g., Vallecillo et al. (2012).

When considering the example shown in Fig. 1, we can clearly see that on the one hand both languages have a large overlap of modeling concepts. But on the other hand, there are also several heterogeneities that make the automated computation of the correspondences challenging. First, in Ecore all modeling elements have for their names a prefix 'E' or 'e' and there is a slightly different terminology used in UML than in Ecore, e.g., *containment* vs. *aggregation*. Thus, we have syntactic heterogeneity due to different naming conventions. Second, the *attribute* and *reference* concept of Ecore is represented by one concept named *property* in UML.



Ecore		UML 2.0
EClass	↔	Class
abstract	↔	isAbstract
eAttributes, eReferences	↔	ownedAttributes
EAttribute, EReference	↔	Property
eOpposite	↔	opposite
containment	↔	aggregation

Fig. 1. Simplified excerpt of Ecore and UML 2.0 metamodels (top) and correspondences between them (bottom).

This is a typical structural heterogeneity. Although the models can represent the same information in UML as in Ecore, different structures have to be instantiated when these models are built. This also entails the need for many-to-many correspondences, because the *property* concept is representing both *attributes* and *references*. Moreover, not only meta-classes have to be mapped, but also meta-attributes and meta-relationships have to be considered.

3. Structural and syntactic measures for metamodel matching

The matching approach proposed in this paper is based on structural and syntactic similarity metrics for finding the best matching solution. Syntactic and structural similarities are captured by different measures that could be integrated in existing matching approaches. In this section, we describe in detail the different metrics used for metamodel matching.

3.1. Structural metrics

Chidamber and Kemerer (Chidamber and Kemerer, 1994) proposed a first version of structural metrics and later the definition of some of them were improved and presented in (Fenton and Pfleeger, 1997). The latter work as well as (Haohai et al., 2004) focusses on the study of structural metrics that can be applied to the model or metamodel level. All these metrics can be applied to the model and the metamodel level with some adaptation. The structural metrics quantify the properties of the metamodel elements. In general, we describe them using the terminology of concepts and sub-concepts. For example, a class can be considered as a concept composed of sub-concepts such as attributes and methods. However, in contradiction to the code level, coupling and cohesion are not very suitable to metamodel or model level since methods calls exist mainly on the code level. Nevertheless, the number of relationships between metamodel elements may indicate coupling and cohesion. In our experiments, we used the following structural metrics: weighted methods per class (WMC), depth of inheritance tree (DIT), number of children (NOC), number of association links (NAC) and number of attributes (NAT). More information can be found about these metrics in (Chidamber and Kemerer, 1994). However, the selection of structural metrics depends on the source and target metamodeling languages. In general, these metrics can be used as input of our proposal but the designer can include some additional metrics that can be found in (Chidamber and Kemerer, 1994).

3.2. Syntactic metrics

We start from the assumption that the vocabulary that is used for naming the metamodel elements is borrowed from the respective domain terminology and then we determine which part of the domain properties is encoded by an element. Thus, two metamodel elements could be syntactically similar if they use a similar/common vocabulary. The vocabulary of an element includes names of classes, attributes, methods, associations, parameters, etc. This similarity could be interesting to consider when searching for correspondences between metamodels. For example, when matching Ecore elements to UML 2.0 elements (cf. Fig. 2), *EClass* is matched with *Class*; attribute *abstract* to *isAbstract*, etc. Thus, it is important to evaluate the syntactic similarity between the names of metamodel elements. We are using two measures to approximate the syntactic homogeneity between metamodel elements: (1) cosine similarity and (2) edit distance.

3.2.1. Cosine similarity

When comparing the vocabulary of two elements, cosine similarity is used to compare vectors encoding the vocabulary of each element (Yates and Neto, 1999). In fact, approximating domain syntactic with vocabulary is widely used in several areas, e.g., information retrieval, natural language processing, etc. In all these techniques, the syntactic similarity between two entities indicates whether they share many common elements (properties, words, etc.).

Ecore		UML 2.0
EClass	↔	Class
abstract	↔	isAbstract
eAttributes	↔	ownedAttributes, aggregation
eReferences	↔	memberEnd
EAttribute EReference	↔	Property
isID	↔	ownedAttributes
eOpposite	↔	opposite
containment	↔	aggregation

Fig. 2. Solution encoding.

For our proposal, we used the cosine similarity measure which has been well studied to estimate the syntactic similarity between words. As a preliminary step, we tokenize the names of metamodel elements using a Camel Case Splitter (Corazza et al., 2012). For instance, the metamodel element "ownedAttribute" will be split into two terms "owned" and "attribute", and "isAbstract" into "is" and "abstract". After that, we use WordNet (Miller, 1995) as a knowledge base for measuring syntactic similarity. WordNet makes syntactic knowledge available which can be used in overcoming many problems associated to the nature of vocabulary used to name model/metamodel elements. For example, when matching Ecore elements to UML 2.0 elements in Fig. 2, WordNet allows the easy production of synonyms for each (sub)name of metamodel element, e.g., "Attribute" = {*property*, *dimension*, ...} and "Property" = {*attribute*, *dimension*, ...}. As such, each metamodel element could be represented as a set of terms. Metamodels are represented as vectors of terms in n-dimensional space where n is the number of unique terms in all considered metamodels. For each metamodel, a weight is assigned to each dimension (representing a specific term) of the representative vector that corresponds to the term frequency score (TF) in the metamodel. The similarity among metamodels is measured by the cosine of the angle between its representative vectors as a normalized projection of one vector over the other. The cosine measure between a pair of metamodels, A and B, is defined as follows:

$$Sim(A, B) = \cos(\theta) = \frac{\vec{A} \times \vec{B}}{|\vec{A}| \times |\vec{B}|} \quad (1)$$

The model can be analyzed in order to extract the used vocabulary for each element (e.g., class, method, etc.). In general, the vocabulary related to a metamodel element is represented by the names of the classes, attributes, relationships, methods, parameters, type declarations, etc. Then, each element is represented as a vector in n-dimensional space where n is the number of terms in all metamodel elements and its tokens and synonyms generated, respectively, using Camel Case Splitter and WordNet.

After the vocabulary extraction step, the TF (term frequency) score is calculated. TF represents the coefficients of the vector for a given element. Hence, TF can be calculated by suitable static analysis of the elements in the metamodel. Then, the distance between two vectors is considered as an indicator of its similarity. Therefore, using the cosine similarity, the conceptual similarity among a pair of elements $e1$ and e2 is quantified by

$$Sim(e1, e2) = \frac{\sum_{i=0}^{n-1}(tf_{e1}(w_i) \times tf_{e2}(w_i))}{\sqrt{\sum_{i=0}^{n-1}(tf_{e1}(w_i))^2}\sqrt{\sum_{i=0}^{n-1}(tf_{e2}(w_i))^2}} \quad (2)$$

where $tf_{e1}(wi)$ and $tf_{e2}(wi)$ are term frequency values of the term w_i in the representative vectors of elements $e1$ and $e2$.

For example, to calculate the cosine similarity between "eAttribute" and "Property", the first step is to apply Camel Case Splitter and Wordnet. The result is to obtain two vectors corresponding to each element: "eAttribute" = {*e*, *Attribute*, *eAttribute*, *property*, *dimension*, ...} and "Property" = {*Property*, *attribute*, *dimension*, ...}. After calculating TF for each vector the cosine similarity Sim(*eAttribute*, *Property*) = 0.67.

3.2.2. Edit distance

We use the Normalized Levenshtein Edit Distance (Yujian and Bo, 2007) to calculate naming similarity between metamodel elements. Levenshtein distance (Lev) returns the number of edits needed to transform one string into the other, with the allowable edit operations (insertions, deletions, or substitutions) of a single character. To have comparable Levenshtein distances, we use the normalized edit distance. The Normalized Levenshtein (LevNorm) between two metamodel elements $e1$ and $e2$ is obtained by dividing the Levenshtein distance $Lev(e1, e2)$ by the size of the longest string, denoted by length(e) given by (3):

$$LevNorm(e1, e2) = \frac{Lev(e1, e2)}{Max(length(e1), length(e2))} \quad (3)$$

For example, LevNorm(eAttribute, Property) = 0.8, as Lev(eAttribute, Attribute) = 8, length(eAttribute) = 10, length(Property) = 8, and LevNorm(eAttribute, Attribute) = 0.1.

To evaluate the syntactic homogeneity (SynHomog) between two metamodel elements $e1$ and $e2$, we combine Cosine Similarity and Normalized Levenshtein Edit Distance as illustrated in (4). Both metrics are complementary and assess two different aspects of string comparison: Cosine similarity is concerned with the domain properties while Normalized Levenshtein focuses on difference between strings but cannot tell if they have something in common.

$$\text{SynHomog}(e1, e2) = \alpha \times Sim(e1, e2) + \beta \times (1 - LevNorm(e1, e2)) \quad (4)$$

where $\alpha + \beta = 1$.

The next section explains how structural and syntactic metrics can be combined to find suitable matching between elements in the metamodel level.

4. Metamodel matching using optimization techniques

We describe in this section the adaptation of genetic algorithm (GA) (Moscato, 1989) and simulated annealing (SA) (Kirkpatrick et al., 1983) to automate metamodel matching. All applications of GA and SA follow generic patterns that are described in Algorithms 1 and 2. The novelty of our work is not related to the combination of GA and SA algorithms to explore the search space but to their adaptation for the metamodel matching problem. To apply them on a specific problem, one must specify the encoding of solutions, the operators that allow movement in search space, i.e., operators for changing solutions, and the solution quality evaluation, i.e., the fitness function. These three elements are detailed in the next sections.

4.1. Global search: genetic algorithm overview

Genetic algorithm (GA) (Moscato, 1989) is a population-based metaheuristic inspired by Darwinian Theory. The basic idea is to explore the search space by evolving a population of solutions for a pre-specified number of generations. Algorithm 1 gives the pseudocode of a canonic elitist GA. The Elitism concept consists that a sub-optimal solution could not be favored for survival over a better one. Its basic iteration is as follows. A parent population P is generated (by environmental selection except for the initialization step where it is produced randomly) and each of its individuals is evaluated. Once the evaluation step is performed, we fulfill the mating pool by selecting parents from P. These selected parents are then subject to genetic operators (crossover and mutation) in order to generate an offspring population Q. Once offspring individuals are evaluated, P and Q are merged to form the population U. We perform now environmental selection on U by selecting fittest individuals and thereby we generate the parent population for the next generation. Elitism is then ensured by saving best individuals coming from parent population P and offspring population Q in each generation of the algorithm. Once the termination criterion is met, the GA returns the best (fittest) individual from P.

Algorithm 1. GA algorithm

```
GA pseudo-code
1: 1: P0 ← random_initialization()
2: P0 ← evaluate(P0)
3: t ← 0;
4: while (NOT termination_condition) do
5:    Parents ← parent_selection(Pt)
6:    Qt ← genetic_operators(Parents)
7:    Ut ← merge (Pt, Qt)
8:    Pt+1 ← environmental_selection(Ut)
9:    t ← t+1;
10: end
11: s ← fittest(Pt)
12: return s
```

4.2. Local search: simulated annealing overview

A global search optimizer is a search method that is able to locate the *global optimum* if we give it enough time to ensure optimality. For example, this capability is ensured by accepting worse solutions that degrades the objective or cost function in Tabu Search and by diversity promotion techniques (such as mutation) in GA. A simulated annealing (SA) optimizer is a search method that is able to locate *the first local optimum* encountered. In fact, the behavior of the local search consists of examining the local neighborhood of the current solution and performing replacement if the neighbor is better than the current solution. In this way, once a local optimum is found, the SA method stagnates within this point and is unable to move away from it. It is important to note the encountered local optimum may be the global one. In the memetic algorithm framework, neighborhood-based metaheuristics are seen/considered as SA operators (Knowles and Corne, 2004). In fact, although Tabu Search and SA are global search optimizers, their search processes are based on *local examination of the neighborhood* of the current solution. In this way, neighborhood-based metaheuristics can be used as local search operators when hybridized with GAs. Since, in this paper, we use GA as global search optimizer and SA as local search operator, we provide a brief description of SA.

Simulated annealing (SA) (Kirkpatrick et al., 1983) is a local search algorithm that gradually transforms a solution following the annealing principle used in metallurgy. The generic behavior of this heuristic is shown in Algorithm 2. After defining an initial solution, the algorithm performs the following three steps in each iteration:

(1) Determine a new neighboring solution.
(2) Evaluate the fitness of the new solution.
(3) Decide on whether to accept the new solution in place of the current one based on the fitness gain/lost.

Algorithm 2. SA algorithm

```
SA algorithm
1: current_solution ← intial_solution
2: current_cost ← evaluate (current_solution)
3: T ← Tinitial
4: while (T > Tfinal) do
5:     for i = 1 to iterations (T) do
6:         new_solution ← move (current_solution)
7:         new_cost ← evaluate(new_solution)
8:         Δcost ← new_cost – current_cost
9:         if (Δcost ≤ 0 or e^(−Δcost/T) < random())
10:            current_solution ← new_solution
11:            current_cost ← new_cost
12:        end if
13:    end for
14:    T ← next_temp(T)
15: end while
16: return current_solution
```

Delta-cost corresponds to the difference between the performance (fitness function value) of the new generated solution and the current one. When delta-cost < 0, the new solution has lower cost than the current solution and it is accepted. For delta-cost > 0 the new solution has higher cost. In that case, the new solution is accepted with a probability $e^{-\Delta \text{cost}/T}$. The introduction of a stochastic element in the decision process avoids being trapped in a local optimum. Parameter T, called temperature, controls the acceptation probability of an inferior solution. T begins with a high value, resulting in a high probability of accepting a solution during the early iterations. Then, T decreases gradually (cooling phase) to lower the acceptation probability as we advance in the search process. At each temperature value, the three steps are repeated for a fixed number of iterations. One attractive feature of the simulated annealing algorithm is that it is problem-independent and can be applied to most combinatorial optimization problems (Kirkpatrick et al., 1983). However, SA is usually slow to converge to a solution.

To conclude, the SA algorithm starts with an initial solution generated randomly. A fitness function measures the quality of the solution at the end of iteration. The generation of a neighboring solution is obtained by randomly changing two dimensions with new ones. To decrease the temperature, we use a geometric cooling schedule (Kirkpatrick et al., 1983). The temperature is reduced using:

$$T_{i+1} = \alpha \times T_i \tag{5}$$

where α is a constant less than 1.

4.3. Global and local search for metamodel matching

In this section, we describe our adaptation of SA and GA to the metamodel matching problem. We start by giving an overview of our approach. Then, we detail our GA and SA adaptation including: (1) solution encoding, (2) change operators, and (3) fitness function.

The search-based process takes the source and target metamodels and a set of structural and syntactic metrics as inputs. As output, our approach generates a set of correspondences between source and target metamodels. Our proposal can generate both one-to-one and many-to-many correspondences (many-to-many in the general case) between two metamodels. A solution consists of a set of random correspondences between source and target metamodels that should maximize structural and syntactic metric values.

Then the process of generating a solution can be viewed as the mechanism that finds the best correspondences between source and target metamodel elements. This matching should maximize structural and syntactic similarity between the metamodel elements. Thus, the correspondence set quality is the sum of the qualities of all single correspondences with respect to structural and syntactic similarity of their referred elements. Consequently, finding a good correspondence is equivalent to finding the combination of metamodel elements that maximizes the global quality of the correspondence set. As the number of combinations may be very large because of multiple mapping possibilities, it may become difficult, if not impossible, to evaluate them exhaustively. As stated in the previous section, heuristic search offers a good alternative in this case. The search space dimensions are the metamodel elements and the possible coordinates in these dimensions are the possible correspondences (source metamodel element(s) ↔ target metamodel(s)). A solution consists then in choosing a correspondence for each metamodel element.

In order to demonstrate the difficulty related to the metamodel matching problem, we characterize the dimensionality of the search space. This dimensionality is directly related to the adopted solution structure. A particular solution in our problem consists of a set of correspondences. We consider all types of correspondences: one-to-one, one-to-many, and many-to-many. Assuming n and m to be the numbers of elements, respectively, of the source and target metamodel elements, the number of possible

correspondences C that have to be explored exhaustively in order to find the exact optimum is expressed as follows:

$$C = \sum_{1}^{n} \frac{i}{n} \times \sum_{1}^{m} \frac{i}{m} \quad (6)$$

According to Eq. (6), the search space dimensionality can be huge due to different types of mapping that we are considering. For example, if $n = m = 20$ then $C = 1.09951E + 12$, and then, we have to explore all possible matching solutions starting from the solutions having only one correspondence up to the model having all possible correspondences included. This fact pushes us to use heuristic search in an attempt to find near optimal solution(s) in a reasonable time and with reasonable computational effort.

Most existing hybrid search schema combines GA with SA to explore huge search space (Holland, 1975; Knowles and Corne, 2004; Talbi, 2002). Such a combination is called memetic algorithm (MA) (Moscato, 1989). In fact, population-based metaheuristics such as GA are known by their global search ability in well-exploring the search space and identifying promising areas that may contain the local/global optima. Nevertheless, GAs have some weaknesses. In fact, their exploitation ability is not sufficient to well-identify the local/global optima (Bechikh et al., 2008; Knowles and Corne, 2004; Talbi, 2002). Contrariwise, SA algorithms have a good exploitation ability allowing them to focus on local optima. However, their exploration process is not sufficient to well-examine the overall search space. From these observations, we remark that GA and SA methods are *complementary*. The basic idea is to build a search algorithm that has highly effective search ability not only from the exploration viewpoint but also from the exploitation one. In fact, it is well-known that GAs are not well-suited for fine-tuning structures which are very close to optimal solutions. Instead, the strength of GAs is in quickly locating the high performance regions of vast and complex search spaces. Once those regions are located, it may be useful to apply local search methods to the high performance structures evolved by the GA.

In the following, we give an overview of our GA and SA adaptation. To illustrate the adaptation process, we use the matching example between Ecore and UML 2.0 introduced by Fig. 1 in Section 2.

4.3.1. Encoding of solutions

One key issue when applying a search-based technique is to find a suitable mapping between the problem to solve (i.e., metamodels matching) and the search-based techniques to use.

In this section, we discuss how we encode correspondences between a source and a target model as a search solution. As stated previously, we propose to define an n-dimensional search space, whereas each dimension corresponds to one of the n elements of the models to be matched. Each dimension has a finite set of target metamodel elements. For instance, the transformation of the Ecore metamodel shown in Fig. 1 will generate an 8-dimensional space that accounts for the different elements as described in Fig. 2.

To define a particular solution, we associate with each dimension (source metamodel element) one or many target metamodel elements. Each correspondence defines a coordinate in the corresponding dimension, and the resulting n-tuple of correspondences then defines a vector position in the n-dimensional space. For instance, the solution shown in Fig. 2 suggests that *EClass* corresponds to *Class*, *eAttributes* to two target elements (*ownedAttributes* and *aggregation*), etc. Thus concretely, a solution is implemented as a vector where dimensions are the source metamodel elements affected to target ones. Of course, many-to-many correspondences are possible as also illustrated by the example shown in Fig. 2. Both algorithms GA and SA used the same solution encoding. To generate an initial population for GA, a random matching is executed to provide a possible mapping between source and target metamodels elements. The maximum and minimum lengths of the solutions are specified as an input of GA and SA. GA generates a population of solutions however SA is using as input the best solution generated by GA after a number of iterations as input.

4.3.2. Solution evaluation

The fitness function quantifies the quality of the proposed matching. The goal is to define an efficient and simple fitness function in order to reduce the computational complexity. The proposed function is based on two main objectives:

(1) Maximizing structural similarities between source and target metamodel elements.
(2) Maximizing the syntactic similarities between source and target metamodel elements.

The structural and syntactic similarity approximations are calculated using the different metrics defined in Section 3. In this context, we define the fitness function to maximize as:

$$f = \frac{\sum_{i=1}^{n} StructureSim(e_i) + SyntacticSim(e_i)}{2 \times n} \quad (7)$$

where n is the number of source metamodel elements. We treat both components of the fitness function with an equal importance and we normalized it in the range of [0,1].

The structural similarity is calculated using the metrics defined in Section 3, whereas the goal is to minimize the difference between the metric values of matched source and target elements. Thus, this similarity is defined as follows:

$$StructureSim(e_i) = 1 - \frac{\sum_{j=1}^{t} |sqm_{i,j} - tqm_{i,j}|}{\sum_{j=1}^{t} Max(sqm_{i,j}, tqm_{i,j})} \quad (8)$$

where t is the number of target metamodel elements matched to source metamodel elements e_i; *sqm* (for e_i) and *tqm* (for the matched element(s)) are the average quality metrics values used to characterize the structure (e.g., number of subconcepts, etc.).

The syntactic similarity $SyntacticSim(e_i)$ of a source metamodel element e_i corresponds to the weighted sum of each vocabulary used to calculate the similarity between e_i and the target metamodel elements matched to e_i. Hence, the syntactic similarity of a solution corresponds to the average of syntactic coherence for each source metamodel:

$$SyntacticSim(e_i) = \frac{\sum_{k=1}^{t} SynHomog(e_i, e_k)}{t} \quad (9)$$

where $SynHomog(e_i,e_k)$ is the average of syntactic measures applied between the source metamodel element e_i and the matched metamodel element e_k.

4.3.3. Change operators

A change operator is a modification applicable to solutions for producing new ones. SA considers change as movement in the search space driven by random coordinate modifications. For SA, the change operator involves randomly choosing l dimensions ($l \leq n$) and replacing their assigned target metamodel elements by randomly selected new ones. For instance, Fig. 3 shows a new solution derived from the one of Fig. 2. The dimensions 3 and 5 are selected to be changed. They get new target metamodel elements assigned in addition to the already existing ones. For instance, *eAttributes* is now mapped in addition to *opposite*. All other dimensions keep their correspondences as for the previous iteration. The number of change frequency is a parameter of SA (two in this example).

In other words, a change consists of proposing new correspondence alternatives to one or more metamodel elements.

Ecore		UML 2.0
EClass	↔	Class
abstract	↔	isAbstract
eAttributes	↔	ownedAttributes, aggregation, **opposite**
eReferences	↔	memberEnd
EAttribute EReference	↔	Property, **Class**
isID	↔	ownedAttributes
eOpposite	↔	opposite
containment	↔	aggregation

Fig. 3. Change operator in SA.

The above-described mutation operator is also used by GA. In addition, GA uses a cross-over operator where two parent individuals are selected, and a sub-vector is picked on each one. Then, the crossover operator swaps some dimensions from one parent to the other. Each child thus combines information from both parents. To select the individuals that will undergo the crossover and mutation operators, we use stochastic universal sampling (SUS) (Moscato, 1989), in which the probability to select an individual is directly proportional to its relative fitness in the population. For each iteration, we used SUS to select *population_size/2* individuals from population p to form population $p+1$. These (*population_size/2*) selected individuals will "give birth" to another (*population_size/2*) new individuals using crossover operator.

5. Metamodel matching experiments

To validate our approach, we performed an experiment using several matching cases. Furthermore, we have done a comparative study with four existing approaches from the field of ontology and schema matching (Aumueller et al., 2005; Ehrig and Sure, 2005; Euzenat, 2004; Kalfoglou et al., 2005) and two prominent approaches from the field of model matching (Brun and Pierantonio, 2008; Del Fabro and Valduriez, 2009). We start by describing the test set used in our experiments. Subsequently, we describe the measures used for the evaluation. Finally, we discuss the obtained results by our approach and compare them to the results gained by existing approaches.

5.1. Matching test set

The metamodels used in our experiments can be found in (Anon., 2014a). We start by describing the metamodels used to validate our proposal for the *model exchange* case and, subsequently, for the *metamodel evolution* case. For both cases, we rely on metamodel test sets that have been already used in previous studies (Kappel et al., 2007; Langer et al., 2013).

5.1.1. Model exchange case test set

For this case, we used a test set which consists of five structural modeling languages: UML 2.0 (Anon., 2014d) (class diagram part), UML 1.4 (Anon., 2014d) (class diagram part), Ecore (Anon., 2014c), WebML (Anon., 2014b) and EER (Chen, 1976). Table 1 gives

Table 1
The used metamodels for the model exchange case.

Metamodels	#Model elements	Terminology
UML 2.0 CD	158	OO
UML 1.4 CD	143	OO
Ecore	83	OO
WebML	53	DB
EER	23	DB

Table 2
The used metamodels for the metamodel evolution case.

Metamodels	#Model elements (min, max)
GMF Map	(367, 428)
GMF Graph	(277, 310)
GMF Gen	(883, 1295)

an overview on some meta-information of the metamodels such as the number of metamodel elements. The five metamodels can be classified as small-sized through medium-sized to large-sized. The UML and Ecore metamodels use object-oriented (OO) vocabulary, while WebML and EER use database (DB) terminology. In our experiments, ten different matching scenarios are used where each scenario matches two different metamodels. For evaluating the quality of the match result, manual correspondences for each scenario are reused from previous studies (Kappel et al., 2007; Langer et al., 2013) and are provided at our website using the INRIA alignment format.

The test set includes a comparative study with four existing tools: Alignment API (Euzenat, 2004), COMA++ (Aumueller et al., 2005), CROSI (Kalfoglou et al., 2005), and FOAM (Ehrig and Sure, 2005). These four tools are coming from the field of schema and ontology matching, but due to the fact that metamodels are transformable to ontologies (Kappel et al., 2006; Parreiras et al., 2007), these tools can be re-used for matching metamodels as well. For more information on how we mapped Ecore to ontologies expressed in OWL, we kindly refer the interested reader to (Kappel et al., 2006). The metamodels in OWL format can also be found at our website. Additionally, we extend the comparative study with two prominent tools from the MDE field, namely EMF Compare (Brun and Pierantonio, 2008) and ATLAS Model Weaver (AMW) (Del Fabro and Valduriez, 2009). With this set of tools, we cover mature matching tools from the schema/ontology field and emerging tools from the MDE field as well.

5.1.2. Metamodel matching test set

For this case, we chose to analyze the extensive evolution of three Ecore metamodels coming from the Graphical Modeling Framework (GMF),[1] an open source project for generating graphical modeling editors. In our case study, we considered the evolution from GMF's release 1.0 over 2.0 to release 2.1 covering a period of two years. For achieving a broad data basis, we analyzed the revisions of three models, namely the *Graphical Definition Metamodel* (GMF Graph for short), the *Generator Metamodel* (GMF Gen for short), and the *Mappings Metamodel* (GMF Map for short). Therefore, the respective metamodel versions had to be extracted from GMF's version control system and, subsequently, manually analyzed to determine the actually applied changes between successive metamodel versions. Additionally, we had to specify all correspondences manually that exist across all metamodel versions. Table 2 describes the minimum/maximum number of model elements over the different versions.

5.2. Measures

To assess the accuracy of our approach, we compute the measures *precision* and *recall* originally stemming from the area of information retrieval. When applying precision and recall in the context of our study, the precision denotes the fraction of correctly matched elements among the set of all proposed matchings. The recall indicates the fraction of correctly matched elements among the set of all expected ones (i.e., how many matchings have not

[1] http://www.eclipse.org/modeling/gmp/.

been missed). Thus, both values may range from 0 to 1, whereas a higher value is better than a lower one. We considered also another metric to evaluate our results which is *F-measure*. *F*-measure takes both precision and recall into account to overcome some over- or underestimations of the two measures. Formally the *F*-measure is in our case the equally weighted average of the precision and recall measure.

To ensure the stability of our results and a fair comparison, our GA and SA algorithms are executed over 50 runs and we calculated an average of precision, recall, and *F*-measure.

5.3. Results and discussions

5.3.1. Results for the model exchange case test set

Fig. 4 illustrates the results at a glance obtained with our proposal (called GAMMA for short: Generic Algorithms for Metamodel Matching) and with the state-of-the-art ontology and model matching approaches as a star glyph for the matching experiment. Each axis of the glyph represents a matching task where two metamodels are matched. The three quantitative variables represent precision (blue), recall (red) and *F*-measure (green). Some of the tested techniques did not deliver any results for some matching tasks, thus some axis of the star glyph are empty. We have also tested our approach and the six other techniques with different settings which may impact the matching results. The complete results can be found in our website (Anon., 2014a). We present in Fig. 4 the best results that we obtained for each approach in average for the set of matching tasks. In addition, we present a table containing all the raw data about precision, recall, and *F*-measure values for each matching tool/matching task combination and the average values over all matching tasks in Appendix A.

Overall, our search-based approach performs better than the four ontology-based and two model-based approaches for the 10 scenarios. In fact, we obtained the highest precision and recall matching scores. The precision scores obtained by our proposal are between 0.48 and 1 for all the scenarios. Also, the recall scores were between 0.59 and 0.89. The highest precision value, using our GAMMA approach, was achieved for UML1.4–UML2.0 with precision = 1; recall = 0.67; and *F*-measure = 0.8. The best recall value was achieved for Ecore to UML2.0 with precision = 0.64; recall = 0.89, and *F*-measure = 0.75.

For the four Ontology matching techniques, the highest precision value was achieved with Alignment API for UML1.4–UML2.0 (precision = 0.96; recall = 0.40; *F*-measure = 0.57). The best recall and *F*-measure value was achieved with COMA++ for UML1.4–UML2.0 (precision = 0.63; recall = 0.58, *F*-measure = 0.61).

Concerning the model matching techniques, EMF Compare achieved the best results also for UML1.4–UML2.0 (precision = 0.44, recall = 0.55, *F*-measure = 0.49) as did AMW (precision = 0.56, recall = 0.60, *F*-measure = 0.58).

In addition to the values presented in Fig. 5, we calculate an average value of precision, recall, and *F*-measure for each approach. The average precision and recall of our approach are, respectively, 0.78 and 0.74. However, for the remaining techniques, the best average precision, recall, and *F*-measure was achieved by AMW (precision = 0.45; recall = 0.43; *F*-measure = 0.41). Thus, we can clearly conclude that our GAMMA approach performs much better, in average, than ontology-based and model-based state-of-the-art matching tools.

To better understand the reasons why better precision and recall values are obtained by our approach, we selected randomly a set of matching/mapping errors generated by our approach and AMW and we classified manually those mapping errors using different criteria: one-to-one mapping, many-to-one or one-to-many mapping, and many-to-many mapping between different element names, mapping between elements having the same name. Fig. 5 shows the number of errors in each category. It is clear that both approaches provide very good results with one-to-one mappings however our proposal performs much better than AMW when the mapping concerns elements having different names. This can be explained by the fact that AMW heavily relies on structural information when comparing between source and target metamodels. In addition, the use of different scenarios (solutions) at each iteration helps our algorithm select the best alternative between the different type of mappings (many-to-many, many-to-one, etc.). This is one of the main reason that explain the performance of our proposal since a population of solutions is evaluated at each iteration and ranked using a fitness function based on both structural and syntactic information. It is infeasible to explore this huge number of alternatives using a deterministic approach.

5.3.2. Results for the metamodel evolution case test set

Fig. 6 summarizes our findings regarding the performance of our search-based software engineering approach for the metamodel evolution matching tasks at a glance (cf. Appendix B for a table containing all precision, recall, and *F*-measure values for the matching scenarios). The results are illustrated as a star glyph in the same manner as before for the model exchange matching results in Fig. 4. Each axis of the glyph represents a mapping of two model versions. The quantitative variables again represent precision, recall, and *F*-measure. Overall, using our approach, we were able to generate correct correspondences with an average precision of 86% and a recall of 89% for all the six versions studied. At the same time, it is worth noting that the evolution history of these three models is very different. GMF Graph was extensively modified but most of the correspondences were detected using our technique with a good recall of 94% for the different versions. GMF Gen was subject to a huge number of revisions. Thus, the evolution of this model is a very representative mixture of different scenarios for the matching between initial and revised models leading to a precision average of 83% and a recall average of 86%. The evolution of the third model under consideration, GMF Map, contained also many revisions during more than two years. Using our approach, we could find the suitable matching between the different versions correctly with 91% and 86% as average of precision and recall.

5.3.3. Discussion

An important observation in our experiments is that the best correctness values are obtained when different versions of the same language are matched. This is also highlighted by the fact that the matching case UML 1.4–UML 2.0 provided the best results for the model exchange matching cases. Perhaps this can be explained by the high degree of name similarities, although, the UML 1.4 metamodel has not been revised to end up with the UML 2.0 metamodel—on the contrary, the UML 2.0 metamodel has been built from scratch. Another observation is that in some cases, especially in model exchange scenarios, metamodel elements have similar names, but completely different semantic. Thus, they should not be mapped. In this case, the structural difference used by our approach helped detect dissimilarities even if the elements have the same name.

Finally, since we viewed the matching problem as a combinatorial problem addressed with heuristic search, it is important to contrast the correctness results with the execution time. We executed our algorithm on a standard desktop computer (Pentium CPU running at 3 GHz with 4GB of RAM). The execution time is shown in Fig. 7. The figure reveals that greater execution times may be obtained with large metamodels than small ones. However, even for large metamodels we obtained a very reasonable execution time (a maximum of 33 min). The execution time of all the remaining considered tools in our experiments is up to 13 min. In any case, our approach is meant to apply to situations where the execution time

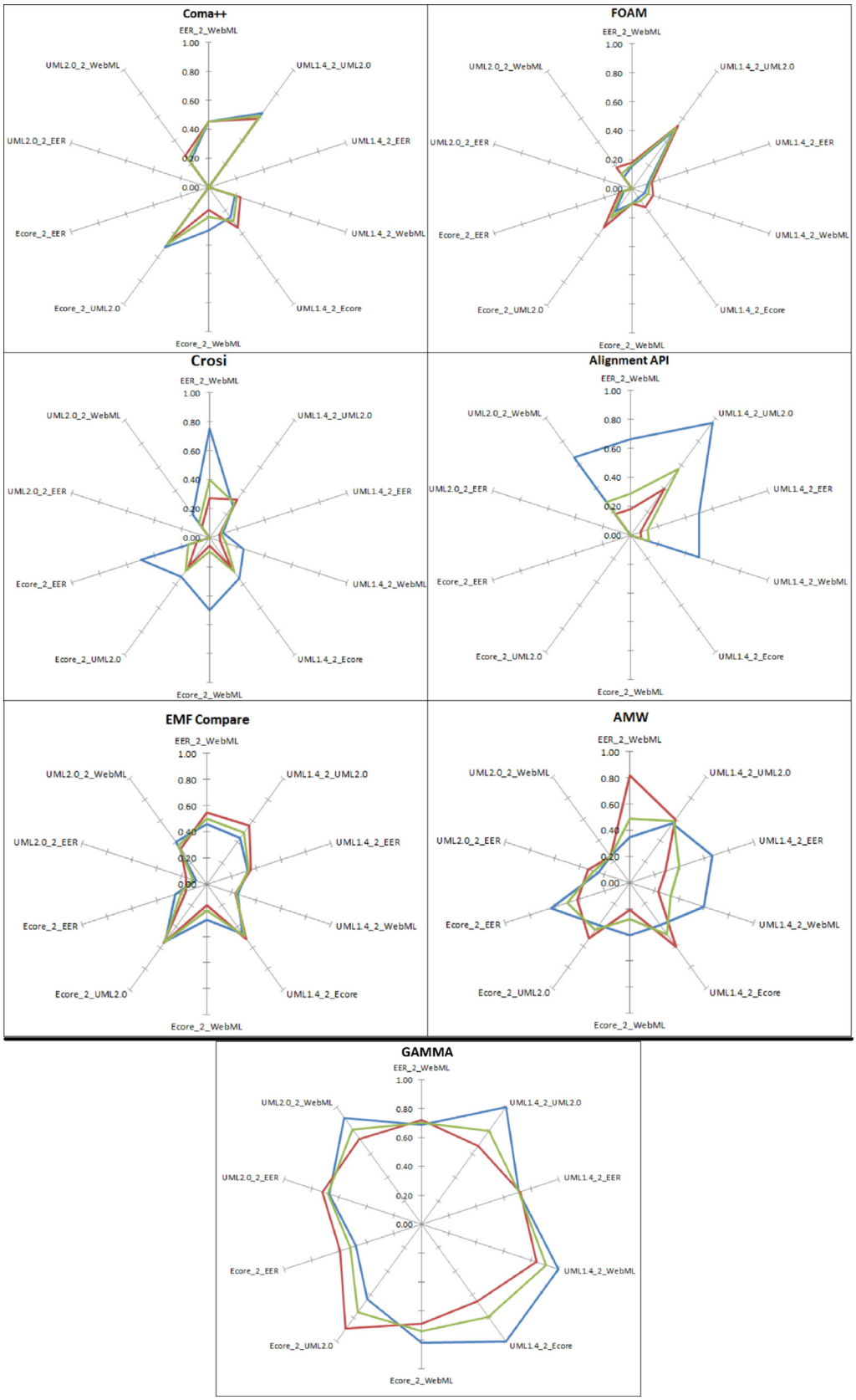


Fig. 4. Results comparison for model exchange case (Precision in blue, Recall in red, *F*-measure in green). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of the article.)

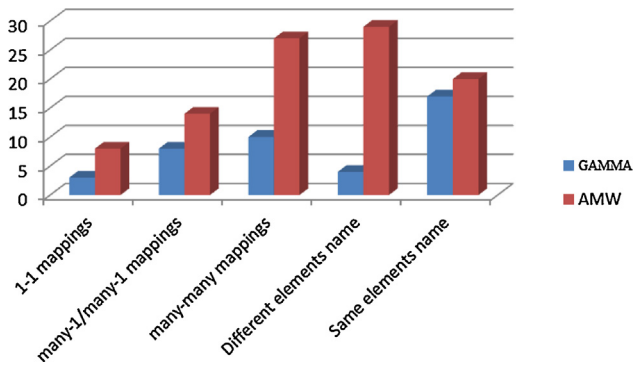


Fig. 5. Types of matching errors.

is not the primary concern, e.g., when mining model repositories or automatically generating transformations.

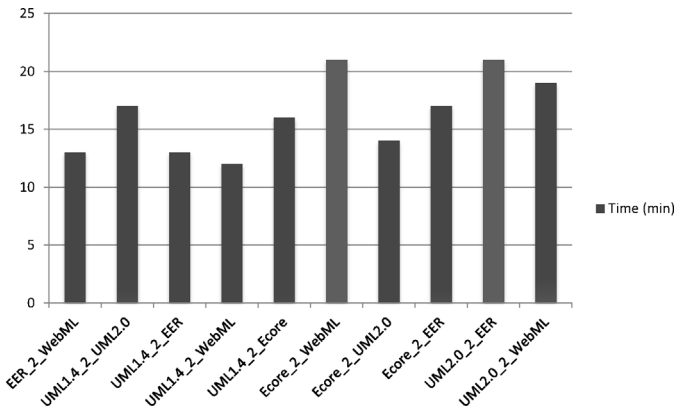
5.4. Threats to validity

We explore in this section the factors that can bias our empirical study. These factors can be classified in four categories: conclusion, construct, internal, and external validity. We consider each of these in the following paragraphs.

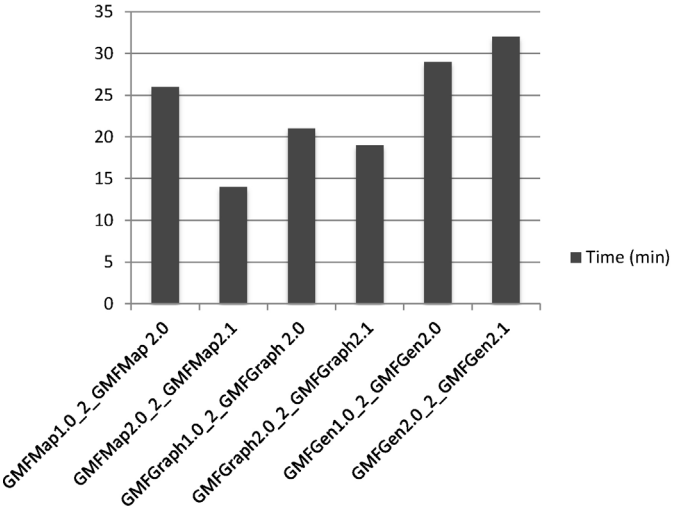
Conclusion validity is concerned with the statistical relationship between the treatment and the outcome. We used the Wilcoxon rank sum test on 50 runs with a 95% ($\alpha < 0.05$) confidence level to test if significant differences existed between the measurements for different treatments. This test makes no assumption that the data is normally distributed and is suitable for ordinal data, so we can be confident that the statistical relationships we observed are significant.

Internal validity is concerned with the causal relationship between the treatment and the outcome. The parameter tuning of our optimization algorithm is important. In fact, different results can be obtained with different parameter settings such number of iterations, stopping criteria, etc. We need to evaluate in our future work the impact of different parameters on the quality of the results (parameters sensitivity).

Construct validity is concerned with the relationship between theory and what is observed. Most of what we measure in our experiments are standard metrics such as precision and recall that are widely accepted as good proxies for quality evaluation. The



(a) Model exchange matching scenarios.



(b) Metamodel evolution matching scenarios.

Fig. 7. Matching performance for the different scenarios.

metrics used for structural and syntactic similarities can be extended by the use of additional ones. Additional experiments are required in future work to evaluate the impact of number of used metrics on the quality of the results. Another limitation is related to the use of Wordnet to find synonyms of the name of model elements

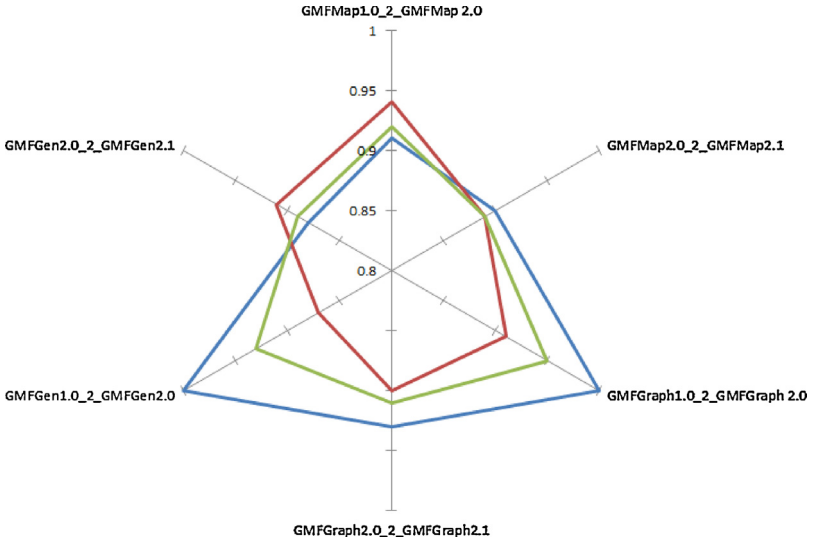


Fig. 6. Matching results of GAMMA for metamodel evolution cases (Precision in blue, Recall in red, *F*-measure in green). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of the article.)

which is not usually feasible due to the technical words considered in Wordnet. We will investigate in our future work the adaptation of new Wordnet versions that are more adapted to the software development context (Yang and Tan, 2012, 2013). However, the vocabulary used in the meta-model level is not rich compared to the source code vocabulary where a huge list of technical words can be used.

External validity refers to the generalizability of our findings. In this study, we performed our experiments on different widely used modeling languages belonging to different domains and with different sizes. However, we cannot assert that our results can be generalized to industrial applications and other languages. Future replications of this study are necessary to confirm our findings.

6. Related work

In the last decades, a lot of contributions related to matching artifacts in general have been presented in the field of schema integration going back to Batini et al. (1986), Kashyap and Sheth (1996), and Parent and Spaccapietra (1998). Based on those works, many schema matching approaches for automating integration tasks for databases and ontologies have been developed (cf. Euzenat and Shvaiko, 2007; Rahm and Bernstein, 2001; Shvaiko and Euzenat, 2005 for surveys).

According to the dimensions introduced in Rahm and Bernstein (2001), our approach may be classified as schema-based, the match granularity is element-level, i.e., classes, attributes, and references, and we use name-based and structure-based matching techniques. Concerning the match cardinality, we support one-to-one, one-to-many, many-to-one, and many-to-many correspondences.

One of the most flexible matching systems is COMA++ (Aumueller et al., 2005) which allows to combine the results of several matchers to compute a common match result based on different weights for the single matcher results. Several systems have been proposed which incorporate machine learning techniques in order to exploit previously gathered information on the schema as well as on the data level (see Euzenat and Shvaiko, 2007 for an overview). For instance, LSD, Autoplex, and Automatch use Naive Bayes over data instances, SemInt is based on neural networks, and iMap analyses the description of elements found in both sources by the establishment of a similarity matrix. The eTuner (Lee et al., 2007) approach automatically tunes the configurations of matching systems by creating synthetic schemas for which mappings are known. Hence, in contrast to our approach which tunes the search process itself, eTuner adjusts other matching systems externally in order to increase the accuracy of the found correspondences. Of course, our approach is in the spirit of schema matching systems, but has to take care of peculiarities of metamodels. Furthermore, we are the first that apply search-based techniques for solving the metamodel matching problem that is also a novel contribution in the broader field of schema matching.

In the field of MDE, there are several generic, but at the same time adaptable frameworks and tools for matching models, such as EMFCompare, DSMDiff, and SiDiff (see Kolovos et al., 2009 for a survey). Besides these generic frameworks, there is the Epsilon Comparison Language (ECL) (Kolovos et al., 2011), which is a domain-specific language for manually developing model comparison rules that are tailored to specific modeling languages and matching scenarios. Another approach for matching models, with the purpose to have valid training instances for model transformation generation approaches, is presented in Dolques et al. (2011). The authors reuse and adapt the Anchor-Prompt approach—previously only available for ontology matching—also for model matching. The topic of matching metamodels has been mostly discussed for generating model transformations automatically based on computed correspondences. For instance, in Kappel et al. (2007), we have evaluated the accuracy of ontology matching tools for matching metamodels by transforming them to ontologies. This work has been also the basis for the first part of the evaluation of the approach presented in this paper. Furthermore, some matching approaches dedicated for metamodels have been proposed in the last years. While some of these approaches (Del Fabro and Valduriez, 2009; Voigt and Heinze, 2010) reuse techniques, such as similarity flooding, that have already been applied for database schema matching and graph matching, other approaches are based on matching rules specifically developed for metamodels (de Sousa et al., 2009; Falleri et al., 2009; Garcés et al., 2009; Voigt et al., 2010; Voigt, 2010), and some approaches combine both (Del Fabro and Valduriez, 2009). Related to this line of research, we have presented a framework in Wimmer et al. (2009) for tuning correspondences by translating test model instances based on the correspondences and using the results to improve the correspondences for the next iteration.

An orthogonal approach to reason about commonalities and differences of two model versions is to compare their instance sets. Thus, the goal is not to compute correspondences between (meta-)model elements using different metrics related to their names and structure as is done in our approach and other syntactic matching approaches, but model comparison operators are used whose input consists of two models and whose output is a set of so-called witnesses instances of the first model that are not instances of the second one (Maoz et al., 2011a, 2011b, 2012). This approach is referred to as semantic model differencing, because the semantics of a model may be interpreted as the set of valid model instances. In our approach, we are currently neglecting this view in the matching process and consider the names of the model elements as a dimension of semantics that is referred often referred as real-world semantics in the information integration field (Batini et al., 1986; Kashyap and Sheth, 1996; Parent and Spaccapietra, 1998) by using syntactic measures for establishing correspondences between model elements based on naming similarities. Consequently, we are not describing the correspondences on the instance level as semantic differencing does, but purely on the type level. While it is evident that structural/syntactic and semantic model differencing have different pros and cons, their combination is considered as an open challenge (Maoz et al., 2012).

To the best of our knowledge, there is no equivalent approach—neither in the classical fields of schema/ontology matching nor in the newer field of model matching—to the presented approach of this paper with respect to the application of search-based techniques. In particular, to find optimal solutions by using GA and SA for matching problems represents a novel contribution to the field of model matching. The effectiveness of the approach against state-of-the-art metamodel matching approaches has been elaborated in Section 5 by an experiment concerning model exchange matching cases.

Widely related to metamodel matching is the field of Model Transformation By-Example (MTBE). In MTBE, transformations are derived from input/output model pairs. MTBE has been introduced by Balogh and Varró (2009), Varró (2006) and Wimmer et al. (2007). MTBE frequently requires the metamodel-level correspondences (derived by your approach) as input. In Dolques et al. (2010), a MTBE approach based on Relational Concept Analysis has been proposed which has been extended in Saada et al. (2011) for generating Jess rules to transform models represented by Jess facts. There is a subarea in MTBE, called model transformation by demonstration (Brosch et al., 2009; Sun et al., 2009), where not only the states of the input and output model pairs are considered, but more important, the operations applied on a particular model are recorded or reconstructed, and subsequently, generalized to be re-playable also on other models. This kind of MTBE approach is heavily used for developing in-place transformations (Kappel et al., 2012), but not

limited to this scenario, e.g., see Langer et al. (2010) for out-place transformations.

The goal of MTBE and metamodel matching is similar when the computed correspondences are used for generating model transformations. However, the important difference between MTBE and metamodel matching is that the latter is not depending on a set of input/output models. Having the metamodels is sufficient. However, one point for future work we plan to consider is to compare the accuracy of metamodel matching and MTBE.

Our work can be classified in the search-based software engineering (SBSE) area (Harman, 2007) that uses search-based approaches to solve optimization problems in software engineering. Once a software engineering task is framed as a search problem, by defining it in terms of solution representation, fitness function, and solution change operators, there are many search algorithms that can be applied to solve that problem. The closer work in SBSE to our contribution is our recent work (Kessentini et al., 2012) about the use of heuristic search techniques for model transformation. In fact, we proposed MOTOE (MOdel Transformation as Optimization by Example), an approach to automate model transformation using heuristic search. MOTOE uses a set of transformation examples to derive a target model from a source model. The transformation is seen as an optimization problem where different transformation possibilities are evaluated and a quality associated to each one depending on its conformance with the examples at hand.

7. Conclusion

In this paper, we proposed a novel approach that considers metamodel matching as an optimization problem. In particular, we adapted a global search, namely genetic algorithm, to generate an initial solution and, subsequently, a local search, namely simulated annealing, to refine the initial solution generated by the genetic algorithm. The approach starts by generating randomly a set of possible matching solutions between source and target metamodels. Then, these solutions are evaluated using a fitness function based on structural and syntactic measures. The results seem promising for matching scenarios: our case studies clearly indicate that our approach achieves better results, in average, than state-of-the-art matching tools where more than 90% of precision and recall scores were obtained. In fact, our proposal obtained better precision and recall scores when we compared the generated mappings with expected ones using existing benchmarks. Different threats and limitations of our proposal are discussed in Section 5.3.

In future work, we aim for comparing metamodel matching with model transformation by example as well as considering also model instances for matching metamodels. Furthermore, we are aiming at building a comprehensive metamodel matching benchmark, because currently several heterogeneous test sets are available, but a more comprehensive benchmark fulfilling certain characteristics (Rosoiu et al., 2011) is still missing in the field of metamodel matching.

Appendix A. Details of the matching tool comparison (model exchange case)

		Coma++	FOAM	Crosi	Alignment API	EMF compare	AMW	SBSE
EER	Precision	0.45	0.15	0.75	0.67	0.46	0.35	0.69
2	Recall	0.45	0.18	0.27	0.18	0.55	0.82	0.72
WebML	*F*-measure	0.45	0.17	0.40	0.29	0.50	0.49	0.70
UML1.4	Precision	0.63	0.50	0.27	0.96	0.44	0.56	1.00
2	Recall	0.58	0.54	0.32	0.40	0.55	0.60	0.67
UML2.0	*F*-measure	0.61	0.52	0.30	0.57	0.49	0.58	0.80
UML1.4	Precision	0.00	0.12	0.10	0.50	0.33	0.67	0.71
2	Recall	0.00	0.14	0.07	0.07	0.36	0.29	0.72
EER	*F*-measure	0.00	0.13	0.08	0.13	0.34	0.40	0.71
UML1.4	Precision	0.19	0.10	0.25	0.50	0.25	0.60	1.00
2	Recall	0.23	0.15	0.08	0.08	0.23	0.23	0.84
WebML	*F*-measure	0.21	0.12	0.12	0.13	0.24	0.33	0.91
UML1.4	Precision	0.26	0.08	0.35	0.00	0.47	0.40	1.00
2	Recall	0.34	0.16	0.25	0.00	0.52	0.61	0.66
Ecore	*F*-measure	0.29	0.11	0.29	0.00	0.49	0.49	0.79
Ecore	Precision	0.30	0.11	0.50	0.00	0.27	0.40	0.82
2	Recall	0.16	0.11	0.05	0.00	0.16	0.21	0.69
WebML	*F*-measure	0.21	0.11	0.10	0.00	0.20	0.28	0.74
Ecore	Precision	0.52	0.20	0.33	0.00	0.54	0.40	0.64
2	Recall	0.47	0.33	0.25	0.00	0.56	0.53	0.89
UML2.0	*F*-measure	0.49	0.25	0.29	0.00	0.55	0.45	0.75
Ecore	Precision	0.00	0.06	0.50	0.00	0.25	0.63	0.48
2	Recall	0.00	0.09	0.09	0.00	0.17	0.42	0.59
EER	*F*-measure	0.00	0.07	0.15	0.00	0.20	0.50	0.52
UML2.0	Precision	0.00	0.00	0.00	0.00	0.08	0.25	0.67
2	Recall	0.00	0.00	0.00	0.00	0.17	0.33	0.72
EER	*F*-measure	0.00	0.00	0.00	0.00	0.11	0.29	0.68
UML2.0	Precision	0.21	0.10	0.20	0.67	0.40	0.25	0.91
2	Recall	0.27	0.18	0.09	0.18	0.33	0.25	0.73
WebML	*F*-measure	0.24	0.13	0.13	0.29	0.36	0.25	0.81
	Precision	0.26	0.14	0.33	0.33	0.35	0.45	0.79
AVG	Recall	0.25	0.19	0.15	0.09	0.36	0.43	0.72
	F-measure	0.25	0.16	0.18	0.14	0.35	0.41	0.74

Appendix B. Details of the matching experiment (metamodel evolution case)

	GMF M ap 1.0_2 → 2.0	GMF Map 2.0_2 → 2.1	GMF Graph 1.0_2 → 2.0	GMF Graph 2.0_2 → 2.1	GMF Gen 1.0_2 → 2.0	GMF Gen 2.0_2 → 2.1
Precision	0.91	0.90	1.00	0.93	1.00	0.88
Recall	0.94	0.89	0.91	0.90	0.87	0.91
F-measure	0.92	0.89	0.95	0.91	0.93	0.89

References

http://web.mst.edu/~marouanek/jss/matching/
http://www.big.tuwien.ac.at/projects/webML
http://www.eclipse.org/emf
http://www.omg.org/technology/documents/modelingspeccatalog.htm#UML
Aumueller, D., Do, H.-H., Massmann, S., Rahm, E., 2005. Schema and ontology matching with COMA++. In: Proceedings of ICMD.
Balogh, Z., Varró, D., 2009. Model transformation by example using inductive logic programming. Softw. Syst. Model. 8 (3), 347–364.
Batini, C., Lenzerini, M., Navathe, S.B., 1986. A comparative analysis of methodologies for database schema integration. ACM Comput. Surv. 18 (4), 323–364.
Bechikh, S., Said, L.B., Ghédira, K.,2008. A novel multi-objective memetic algorithm for continuous optimization. In: Proceedings of ICTAI. IEEE, pp. 180–189.
Bézivin, J., 2005. On the unification power of models. Softw. Syst. Model. 4 (2), 171–188.
Brosch, P., Langer, P., Seidl, M., Wieland, K., Wimmer, M., Kappel, G., Retschitzegger, W., Schwinger, W., 2009. An example is worth a thousand words: composite operation modeling by-example. In: Proceedings of MoDELS.
Brun, C., Pierantonio, A., 2008. Model differences in the Eclipse Modeling Framework. UPGRADE. Eur. J. Inform. Prof. 9 (2), 29–34.
Chen, P.P-S., 1976. The entity-relationship model-toward a unified view of data. ACM Trans. Database Syst. 1, 9–36.
Chidamber, S.R., Kemerer, C.F., 1994. A metrics suite for object-oriented design. IEEE Trans. Softw. Eng. 20 (6), 293–318.
Cicchetti, A., Di Ruscio, D., Eramo, R., Pierantonio, A., 2008. Automating co-evolution in model-driven engineering. In: Proceedings of EDOC.
Corazza, A., Martino, S.D., Maggio, V.,2012. LINSEN: an efficient approach to split identifiers and expand abbreviations. In: Proceedings of ICSM. IEEE, pp. 233–242.
de Sousa, J., Lopes, D., Barreiro Claro, D., Abdelouahab, Z., 2009. A step forward in semi-automatic metamodel matching: algorithms and tool. In: Proceedings of ICEIS, pp. 137–148.
Del Fabro, M.D., Valduriez, P., 2009. Towards the efficient development of model transformations using model weaving and matching transformations. Softw. Syst. Model. 8 (3), 305–324.
Dolques, X., Huchard, M., Nebut, C., Reitz, P., 2010. Learning transformation rules from transformation examples: an approach based on relational concept analysis. In: Proceedings of EDOCW.
Dolques, X., Dogui, A., Falleri, J.-R., Huchard, M., Nebut, C., Pfister, F., 2011. Easing model transformation learning with automatically aligned examples. In: Proceedings of ECMFA, pp. 189–204.
Ehrig, M., Sure, Y., 2005. FOAM – framework for ontology alignment and mapping – results of the ontology alignment evaluation initiative. In: Integrating Ontologies.
Euzenat, J., 2004. An API for ontology alignment. In: Proceedings of ISWC.
Euzenat, J., Shvaiko, P., 2007. Ontology Matching. Springer, London, UK.
Falleri, J.-R., Huchard, M., Lafourcade, M., Nebut, C., 2009. Metamodel matching for automatic model transformation generation. In: Proceedings of MoDELS, pp. 326–340.
Fenton, N., Pfleeger, S.L., 1997. Software Metrics: A Rigorous and Practical Approach. International Thomson Computer Press, London, UK.
Garcés, K., Jouault, F., Cointe, P., Bézivin, J., 2009. Managing model adaptation by precise detection of metamodel changes. In: Proceedings of ECMDA-FA, pp. 34–49.
Haas, L.M., 2007. Beauty and the beast: the theory and practice of information integration. In: Proceedings of ICDT, pp. 28–43.
Haohai, M., Shao, W., Zhang, L., Jiang, Y., 2004. Applying OO metrics to assess UML meta-models. In: Proceedings of UML.
Harman, M., 2007. The current state and future of search based software engineering. In: Proceedings of ICSE.
Holland, J.H., 1975. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, USA.
Kalfoglou, Y., Hu, B., Reynolds, D., Shadbolt, N., 2005. CROSI project, final report. Technical Report 11717. University of Southampton, UK.
Kappel, G., Kapsammer, E., Kargl, H., Kramler, G., Reiter, T., Retschitzegger, W., Schwinger, W., Wimmer, M., 2006. Lifting metamodels to ontologies: a step to the semantic integration of modeling languages. In: Proceedings of MoDELS.
Kappel, G., Kargl, H., Kramler, G., Schauerhuber, A., Seidl, M., Strommer, M., Wimmer, M., 2007. Matching metamodels with semantic systems – an experience report. In: Proceedings of BTW Workshops, pp. 38–52.
Kappel, G., Langer, P., Retschitzegger, W., Schwinger, W., Wimmer, M., 2012. Model transformation by-example: a survey of the first wave. In: Conceptual Modelling and its Theoretical Foundations.
Kashyap, V., Sheth, A.P., 1996. Semantic and schematic similarities between database objects: a context-based approach. VLDB J. 5 (4), 276–304.
Kessentini, M., Sahraoui, H.A., Boukadoum, M., Omar, O.B., 2012. Search-based model transformation by example. Softw. Syst. Model. 11 (2), 209–226.
Khuller, S., Raghavachari, B., 1996. Graph and network algorithms. ACM Comput. Surv. 28 (1), 43–45.
Kirkpatrick, S., Gelatt Jr., C.D., Vecchi, M.P., 1983. Optimization by simulated annealing. Sciences 220 (4598), 671–680.
Knowles, J., Corne, D., 2004. Memetic algorithms for multiobjective optimization: issues, methods and prospects. In: Recent Advances in Memetic Algorithms. Springer, Washington DC, US, pp. 313–352.
Kolovos, D., Ruscio, D., Pierantonio, A., Paige, R., 2009. Different models for model matching: an analysis of approaches to support model differencing. In: Proceedings of CVSM.
Kolovos, D., Rose, L., Paige, R., 2011. The Epsilon Book. http://www.eclipse.org/gmt/epsilon/doc/book/
Kurtev, I., Bézivin, J., Akşit, M., 2002. Technological spaces: an initial appraisal. In: Proceedings of CoopIS.
Langer, P., Wimmer, M., Kappel, G., 2010. Model-to-model transformations by demonstration. In: Proceedings of ICMT.
Langer, P., Wimmer, M., Brosch, P., Herrmannsdörfer, M., Seidl, M., Wieland, K., Kappel, B., 2013. A posteriori operation detection in evolving software models. J. Syst. Softw. 86 (2), 551–566.
Lee, Y., Sayyadian, M., Doan, A., Rosenthal, A., 2007. eTuner: tuning schema matching software using synthetic scenarios. VLDB J. 16 (1), 97–122.
Maoz, S., Ringert, J.O., Rumpe, B., 2011a. ADDiff: semantic differencing for activity diagrams. In: Proceedings of FSE, pp. 179–189.
Maoz, S., Ringert, J.O., Rumpe, B., 2011b. CDDiff: semantic differencing for class diagrams. In: Proceedings of ECOOP, pp. 230–254.
Maoz, S., Ringert, J.O., Rumpe, B., 2012. An interim summary on semantic model differencing. Softwaretechnik-Trends 32 (4).
Mens, T., Van Gorp, P., 2006. A taxonomy of model transformation. Electr. Notes Theor. Comput. Sci. 152, 125–142.
Miller, G.A., 1995. WordNet: a lexical database for English. Commun. ACM 38 (11), 39–41.
Moscato, P., 1989. On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms. Caltech Concurrent Computation Program, C3P Report 826. California Institute of Technology.
Parent, C., Spaccapietra, S., 1998. Issues and approaches of database integration. Commun. ACM 41, 166–178.
Parreiras, F.S., Staab, S., Winter, A., 2007. On marrying ontological and metamodeling technical spaces. In: Proceedings of ESEC/FSE, pp. 439–448.
Rahm, E., Bernstein, P.A., 2001. A survey of approaches to automatic schema matching. VLDB J. 10 (4), 334–350.
Rosoiu, M.-E., Trojahn dos Santos, C., Euzenat, J., 2011. Ontology matching benchmarks: generation and evaluation. In: Proceedings of Ontology Matching Workshop.
Saada, H., Dolques, X., Huchard, M., Nebut, C., Sahraoui, H.A., 2011. Generation of operational transformation rules from examples of model transformations. In: Proceedings of MoDELS, pp. 546–561.
Sendall, S., Kozaczynski, W., 2003. Model transformation: the heart and soul of model-driven software development. IEEE Softw. 20 (5), 42–45.
Shvaiko, P., Euzenat, J., 2005. A survey of schema-based matching approaches. J. Data Semant. IV, 146–171.
Sun, Y., White, J., Gray, J., 2009. Model transformation by demonstration. In: Proceedings of MoDELS.
Talbi, E.-G., 2002. A taxonomy of hybrid metaheuristics. J. Heuristics 8 (5), 541–564.
Tratt, L., 2005. Model transformations and tool integration. Softw. Syst. Model. 4 (2), 112–122.
Vallecillo, A., Gogolla, M., Burgueño, L., Wimmer, M., Hamann, L., 2012. Formal specification and testing of model transformations. In: Proceedings of SFM, pp. 399–437.
Varró, D., 2006. Model transformation by example. In: Proceedings of MoDELS.
Voigt, K., 2010. Semi-automatic matching of heterogeneous model-based specifications. In: Proceedings of Software Engineering Workshops.
Voigt, K., Heinze, T., 2010. Metamodel matching based on planar graph edit distance. In: Proceedings of ICMT.
Voigt, K., Ivanov, P., Rummler, A., 2010. MatchBox: combined meta-model matching for semi-automatic mapping generation. In: Proceedings of SAC.
Wimmer, M., 2009. A meta-framework for generating ontologies from legacy schemas. In: Proceedings of DEXA Workshops.
Wimmer, M., Strommer, M., Kargl, H., Kramler, G., 2007. Towards model transformation generation by-example. In: Proceedings of HICSS.

Wimmer, M., Seidl, M., Brosch, P., Kargl, H., Kappel, G., 2009. On realizing a framework for self-tuning mappings. In: Proceedings of TOOLS.
Wimmer, M., Kappel, G., Kusel, A., Retschitzegger, W., Schoenboeck, J., Schwinger, W., 2010. From the heterogeneity jungle to systematic benchmarking. In: Proceedings of MoDELS Workshops.
Yang, J., Tan, L., 2012. Inferring semantically related words from software context. In: Proceedings of MSR, pp. 161–170.
Yang, J., Tan, L., 2013. SWordNet: inferring semantically related words from software context. Empirical Softw. Eng., 1–31.
Yates, R.B., Neto, B.R., 1999. Modern Information Retrieval. Addison-Wesley, New York, US.
Yujian, L., Bo, L., 2007. A normalized Levenshtein distance metric. IEEE Trans. Pattern Anal. Mach. Intell. 29, 1091–1095.

Marouane Kessentini is a tenure-track assistant professor at University of Michigan. He is the founder of the research group: Search-based Software Engineering@Michigan. He holds a PhD in Computer Science, University of Montreal (Canada), 2011. His research interests include the application of artificial intelligence techniques to software engineering (search-based software engineering), software testing, model-driven engineering, software quality, and re-engineering. He has published around 50 papers in conferences, workshops, books, and journals including three best paper awards. He has served as program-committee/organization member in several conferences and journals. He served as the co-chair of the SBSE track at GECCO2014 and the founder of the north American search based software engineering symposium (NasBASE).

Ali Ouni is currently a PhD student in computer science under the supervision of Pr. Marouane Kessentini (University of Michigan) and Pr. Houari Sahraoui (University of Montreal). He is a member of the SBSE research laboratory, University of Michigan, USA and member of the GEODES software engineering laboratory, University of Montreal, Canada. He received his bachelor degree (BSc.) and his Master Degree Diploma (MSc.) in computer science from the Higher Institute of Applied Sciences and Technology (ISSAT), University of Sousse, Tunisia, respectively in 2008, and 2010. His research interests include the application of artificial intelligence techniques to software engineering (search-based software engineering). Since 2011, he published several papers in well-ranked journals and conferences. He serves as program committee member in several conferences and journals such as GECCO'14, CMSEBA'14, and NasBASE'15.

Philip Langer is postdoctoral researcher in the Business Informatics Group at the Vienna University of Technology. Before that, he was researcher at the Department of Telecooperation at the Johannes Kepler University Linz and received a PhD degree in computer science from the Vienna University of Technology in 2011 for his thesis on model versioning and model transformation by demonstration. His current research is focused on model evolution, model transformations, and model execution in the context of model-driven engineering.

Manuel Wimmer is a postdoctoral researcher in the Business Informatics Group (BIG). He received his Ms and PhD degrees in business informatics from the Vienna University of Technology in 2005 and 2008, respectively. Furthermore, he is one of the prize winners of the Award of Excellence 2008 assigned by the Austrian Federal Ministry of Science and Research for his PhD thesis.In 2011/2012 he was on leave as research associate at the University of Málaga (working with Prof. Antonio Vallecillo). He was funded by an Erwin Schrödinger scholarship granted by the Austrian FWF (FondszurFörderung der WissenschaftlichenForschung). His current research interests comprise Web engineering, model engineering, and reverse engineering. He is/was involved in several national and international projects dealing with the application of model engineering techniques for domains such as tool interoperability, versioning, social Web, and Cloud computing.

Slim Bechikh received the BSc degree in computer science applied to management and the MSc degree in modeling from the University of Tunis, Tunisia, in 2006 and 2008, respectively. He also obtained the PhD degree in computer science applied to management from University of Tunis in January 2013. He worked, for four years, as an attached researcher within the Optimization Strategies and Intelligent Computing lab (SOIE), Tunisia. Now, he is a postdoctoral researcher at the SBSE@MST lab, Missouri, USA. His research interests include multi-criteria decision making, evolutionary computation, multi-agent systems, portfolio optimization and search-based software engineering. Since 2008, he published several papers in well-ranked journals and conferences. Moreover, he obtained the best paper award of the ACM Symposium on Applied Computing 2010 in Switzerland among more than three hundreds participants. Since 2010, he serves as reviewer for several conferences such as ACM SAC and GECCO and various journals such as Soft Computing and IJITDM.