Softw Syst Model (2012) 11:209–226
DOI 10.1007/s10270-010-0175-7

SPECIAL SECTION PAPER

Search-based model transformation by example

Marouane Kessentini · Houari Sahraoui · Mounir Boukadoum · Omar Ben Omar

Received: 1 June 2009 / Revised: 19 August 2010 / Accepted: 20 August 2010 / Published online: 22 September 2010

Abstract Model transformation (MT) has become an important concern in software engineering. In addition to its role in model-driven development, it is useful in many other situations such as measurement, refactoring, and test-case generation. Roughly speaking, MT aims to derive a target model from a source model by following some rules or principles. So far, the contributions in MT have mostly relied on defining languages to express transformation rules. However, the task of defining, expressing, and maintaining these rules can be difficult, especially for proprietary and non-widely used formalisms. In some situations, companies have accumulated examples from past experiences. Our work starts from these observations to view the transformation problem as one to solve with fragmentary knowledge, i.e. with only examples of source-to-target MTs. Our approach has two main advantages: (1) it always proposes a transformation for a source model, even when rule induction is impossible or difficult to achieve; (2) it is independent from the source and target formalisms; aside from the examples, no extra information is needed. In this context, we propose an optimization-based approach that consists of finding in the examples combinations of transformation fragments that best cover the source model. To that end, we use two strategies based on two search-based algorithms: particle swarm optimization and simulated annealing. The results of validating our approach on industrial projects show that the obtained models are accurate.

Keywords Search-based software engineering · Automated model transformation · Transformation by example

Communicated by Prof. Krzysztof Czarnecki.

M. Kessentini (✉) · H. Sahraoui · O. B. Omar
Département d'Informatique et Recherche Opérationnelle,
Université de Montréal, CP 6128, succ Centre-Ville,
Montréal, QC H3C 3J7, Canada
e-mail: kessentm@iro.umontreal.ca

H. Sahraoui
e-mail: sahraouh@iro.umontreal.ca

O. B. Omar
e-mail: benomaro@iro.umontreal.ca

M. Boukadoum
Département d'Informatique, Université du Québec à Montréal,
CP 8888, succ Centre-ville, Montréal, QC H3C 3P, Canada
e-mail: mounir.boukadoum@uqam.ca

1 Introduction

In the context of model-driven development (MDD) [1,65], the creation of models and model transformations (MTs) is a central task that requires a mature development environment, based on the best practices of software engineering principles. For a comprehensive approach to MDD, models and MTs must be designed, analyzed, synthesized, tested, maintained and subjected to configuration management to ensure their quality. This makes MT a central concern in the MDD paradigm: not used only in forward engineering, it allows concentrating the maintenance effort on models and using transformation mechanisms to generate code. As a result, many transformation languages are emerging.

Practical model-to-model transformation languages are of prime importance. Despite the many approaches [2–4] that addressed the request for proposals of OMG QVT RFP [5,6], the MT problem has no universal solution since most existing approaches to solve it depend on the source and target metamodels. A popular view attributes the situation to the difficulty of defining or expressing transformation rules, especially for proprietary or non-widely used formalisms.

Most contributions in MT are concerned with defining languages to express transformation rules. Transformation rules can be implemented using: (1) general programming languages such as Java or C#; (2) graph transformation languages like AGG [7] and VIATRA [8]; (3) specific languages such as ATL [9,10] and QVT [6,11]. Sometimes, transformations are based on invariants (pre-conditions and post-conditions specified in languages such as OCL [12]). These approaches have been successfully applied to transformation problems where knowledge exists about the mapping between the source and target models. Still, there exist situations where defining the set of rules is a complex task, and many difficulties accompany the results [13] (incompleteness, redundancy, inconsistency, etc.). In particular, the experts may find it difficult to master both the source and target meta-models [14].

On the other hand, it is recognized that experts are more easily keen on giving transformation examples than defining complete and consistent transformations rules [15]. This particularly applies to industrial organizations where a memory of past transformation examples can be found, thus motivating transformation-by-examples approaches such as the one proposed in [16] whose principle is to semi-automatically derive transformation rules from an initial set of examples (interrelated source and target models), using inductive logic programming (ILP). However, this kind of approach is not adaptable to situations where only a few examples are available.

We can alternatively view MT as an optimization problem where a (partial) target model is to be automatically derived from available examples. In this context, we recently introduced an optimization-based approach to automate MT called model transformation as optimization by examples (MOTOE) [17]. MOTOE views MT as a combinatorial optimization problem where the transformation of a source model is obtained by finding, for each of its constructs, a similar transformation from an example base. Due to the large number of possible combinations, a heuristic-search strategy is used to build the transformation solution as a set of individual construct transformations. Taking inspiration from our previous paper [17], we extend MOTOE with a more sophisticated transformation building process and use a larger scale validation with industrial data. In particular, we compare two strategies: (1) parallel exploration of different transformation possibilities (call it population-based MT) by means of a global search heuristic implemented with particle swarm optimization (PSO) [18], and (2) initial transformation possibility improvements (call it adaptation-based MT) implemented with a hybrid heuristic search that combines PSO with the local search heuristic simulated annealing (SA) [19].

The approach we propose has the advantage over rule-based algorithms that, for any source model, it always proposes a transformation, even when rule induction is impossible or difficult to achieve. Although, it can be seen as a form of case-based reasoning (CBR) [20], it actually differs from CBR in that all the existing models are used to derive a solution, not only the most similar one. Another interesting advantage is that our approach is independent from the source and target formalisms; aside from the examples, no extra information is needed. Still, our approach is not meant primarily to replace rule-based approaches; instead, it targets situations where rules are not available, difficult to define, or non-consensual.

In this paper, we illustrate and evaluate our approach on the well known case of transforming UML class diagrams (CLD) to relational schemas (RS). As will be shown in Sect. 4, the models obtained by using our transformation approach are comparable to those derived by transformation rules. Our choice of CLD-to-RS transformation is motivated by the fact that it is well known and reasonably complex; this allows us to focus on describing the technical aspects of our approach and comparing its results with those of a well-known alternative. Our approach can also be applied to more complex transformations such as sequence diagrams-to-colored Petri-nets [21].

The remainder of this paper is structured as follows: Sect. 2 is dedicated to the MT-problem statement. In Sect. 3, we describe the principles of our approach. The details are discussed in Sect. 3; they include the adaptation of two search algorithms for the MT problem. Section 4 contains the validation results of our approach with industrial projects and a comparison between the global- and adaptation-based strategies. In Sect. 5, the related work in MT is discussed. We conclude and suggest research directions in Sect. 6.

2 Approach overview

This section shows how, under some circumstances, MT can be seen as an optimization problem. We also show why the size of the corresponding search space makes heuristic search necessary to explore it. Finally, we give the principles of our approach.

2.1 Problem statement

Defining transformations for domain-specific or complex languages requires proficiency in high-level programming languages, knowledge of the underlying metamodels, and knowledge of the semantic equivalency between the metamodels' concepts [1]. Therefore, creating MT rules may become a complex task [22]. On the other hand, it is often easier for experts to show transformation examples than to express complete and consistent transformation rules [14]. This observation has led to a new research direction: model transformation by example (MTBE), where, like in [16], rules are semi-automatically derived from examples.

In the absence of rules or an exhaustive set of examples that allows rule extraction, an alternative solution is to derive a partial target model from the available examples. The generation of such models consists of finding, in the examples, model fragments that best match the model to transform. To characterize the problem, we start with some definitions.

Definition 3.1 (*Model to transform*) A model to transform, M, is a model composed of n constructs expressed in a predefined abstract syntax.

Definition 3.2 (*Model construct*) A construct is a source or target model element, for example, a class in a CLD. It may contain properties that describe it, e.g., its name.

Complex constructs may contain sub-constructs; for example, a class could have attributes. For graph-based formalisms, constructs are typically nodes and links between nodes. For instance, classes, associations, and generalizations are model constructs in UML CLDs.

Definition 3.3 (*Block*) A block defines a previously performed transformation trace between a set of constructs in the source model and a set of constructs in the target model.

Constructs that should be transformed together are grouped within the same block. For example, a generalization link g between two classes $\underline{A}$ and $\underline{B}$ cannot be mapped independently from the mapping of $\underline{A}$ and $\underline{B}$. In our case, we assume that blocks are manually defined by domain experts when transforming models. Finally, blocks are not general rules since they involve concept instances (e.g., class *Student*) instead of just concepts (e.g., *class* concept). In other words, where transformation rules are expressed in terms of metamodels, blocks are expressed in terms of concrete models.

Definition 3.4 (*Transformation example*) A transformation example, TE, is a mapping of constructs from a source model to the corresponding target model. Formally, we define a TE as a triple $\langle SMD, TMD, MB \rangle$, where SMD denotes the source model, TMD denotes the target model, and MB is a set of mapping blocks that relate sets of constructs in SMD to their equivalents in TMD. For example, the creation of a database schema from a UML CLD describing student records is a transformation example. The base of examples is a set of transformation examples.

Our goal is to combine and adapt transformation blocks—which are fragments coming from one or more MTs in the base of examples—to generate a new transformed model by similarity. A fragment from an example model is considered to be similar to one from the source model if it shares the same construct types with similar properties. For instance, in a CLD, a fragment with an association *pays* between two classes *Client* and *Bill* is similar to a fragment from another diagram containing an association *evaluates* relating classes *ControlExam* and *Module*. The degree of similarity depends on the properties and associations of the classes (attributes types, cardinalities, etc.). In the absence of transformation rules, any combination of blocks is a transformation possibility. For example, when transforming a CLD to a database schema, any class can be translated into a table, a foreign key in an existing table, two tables, or any other possible combination of target constructs. However, with transformation examples, the possibilities are reduced to the transformations of similar constructs in the examples.

The transformation of a model M with n constructs, using a set of examples that globally define m possibilities (blocks), consists of finding the subset from the m possibilities that best transforms each of the n constructs of M. "Best transforms" means that each construct can be transformed by one of the selected possibilities and that construct transformations are mutually consistent. In this context, m^n possible combinations have to be explored. This number can quickly become huge. For example, an average UML CLD with 40 classes and 60 links (generalization, associations, and aggregations) defines 100 constructs (40 + 60). At the same time, an example base with a reasonable number of examples may contain hundreds of blocks, say 300. In this case, 300^{100} possible combinations should be explored. If we limit the possibilities for each construct to only blocks that contain similar constructs, the number of possibilities becomes $m_1 \times m_2 \times m_3 \times \cdots \times m_n$ where each $m_i \leq m$ represents the number of transformation possibilities for construct i. Although the number of possibilities is reduced, it could still be very large for large CLDs. In the same example, assuming that each of the 100 constructs has 8 or more mapping possibilities leads to exploring at least 8^{100} combinations. Considering these magnitudes, exploring all the possibilities cannot be done within a reasonable time frame. This calls for alternative approaches such as heuristic search.

2.2 Approach overview

We propose an approach that uses knowledge from previously solved transformation cases (examples) to solve new MT problems by using combinations of the past problem solutions. More specifically, the (partial) target models are automatically derived by an optimization process that exploits the available examples.

Figure 1 shows the general structure of MOTOE. The approach takes as inputs a base of examples (i.e., a set of transformation examples) and a source model to transform, and generates as output a target model. The generation process can be viewed as the selection of the subset of transformation fragments (blocks) in the example base that best matches the constructs of the source model (using a similarity function). In other words, the transformation is done as

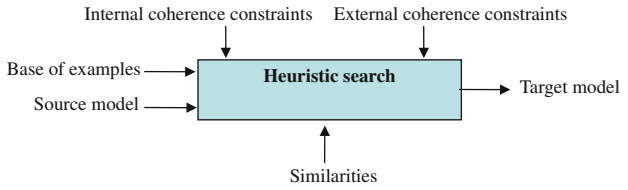

Fig. 1 MOTOE overview

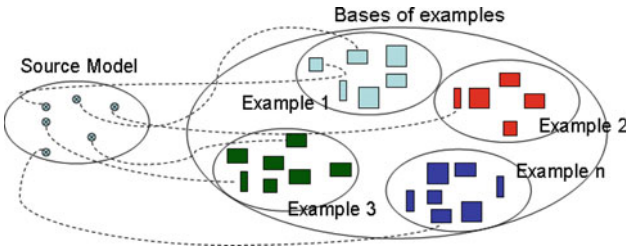

Fig. 2 Illustration of the proposed transformation process

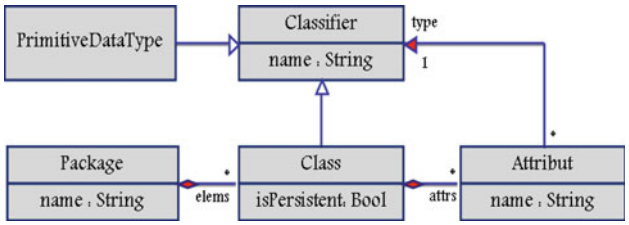

Fig. 3 Class diagram metamodel

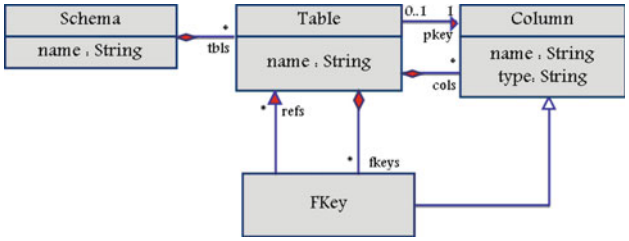

Fig. 4 Relational schema metamodel

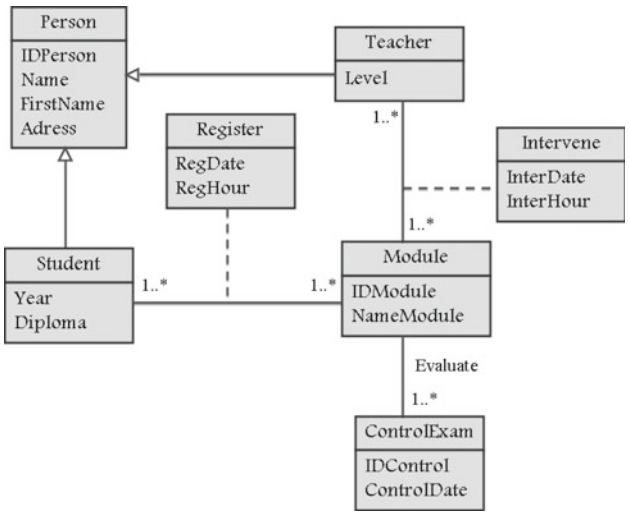

Fig. 5 Example of a CLD source model

an assembly of building blocks. The quality of the produced target model is measured by the conformance of the selected fragments to structural constraints, i.e., by answering the following two questions: (1) did we choose the right blocks? and (2) did they fit together?

Figure 2 illustrates the case of a source model with six constructs to transform represented by dots. A transformation solution consists of assigning to each construct c_i a mapping block, i.e., a transformation possibility from the example base (blocks are represented by rectangles in Fig. 2). A possibility is considered to be adequate if the assigned block contains a construct similar to c_i (similarity evaluation is discussed in Sect. 3.3).

As many block assembly schemes are possible, the transformation is a combinatorial optimization problem. The number of possible solutions quickly becomes huge as the number of blocks increases. A deterministic search is not practical in such cases, hence the use of heuristic search. The dimensions of the solution space are the constructs of the source model to transform. A solution is determined by the assignment of a transformation fragment (block) to each source model construct. The search is guided by the quality of the solution according to internal coherence (inside a block), and external coherence (between blocks).

To explore the solution space, we study two different search strategies in this work. The first one uses a global heuristic search by means of the PSO algorithm [18]. The second one first uses a global heuristic search to shrink the search space and find a first transformation solution; then it uses a local heuristic search, using SA algorithm [19], to refine the found solution.

To illustrate our example-based transformation mechanism, consider the case of MT between UML CLD and RS. Figure 3 shows a simplified metamodel of the UML CLD, containing concepts like class, attribute, relationship between classes, etc. Figure 4 shows a partial view of the RS metamodel, composed of table, column, attribute, etc. Normally, the transformation mechanism, based on rules, will specify how the persistent classes, their attributes and their associations should be transformed into tables, columns and keys.

The choice of this particular example is only motivated by considerations of clarity. MOTOE is independent from the nature of the transformation problem since it does not depend on the source and target metamodels. Instead, it applies to any formalism where prior examples of successful transformation are available.

A transformation example of a CLD to a RS is presented in Figs. 5 and 6. The CLD is the source model (a) and the RS is the target one (b). The CLD contains 12 constructs that represent 7 classes (including 2 association classes), 3 asso-

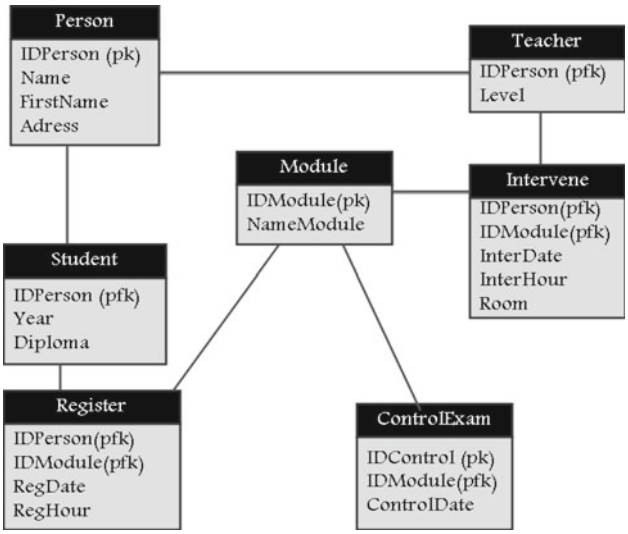

Fig. 6 Equivalent RS target model to the CLD source model of Fig. 5

ciations, and 2 generalization links. The five non-associative classes are mapped to tables with the class attributes mapped to columns of the tables. The associations between *Student* and *Module*, and between *Teacher* and *Module*, are respectively, translated into tables *Register* and *Intervene* with, as columns, the attributes of the associative classes. Each of these tables also contains two foreign keys to their related tables. Association *evaluate* becomes a foreign key in table *ControlExam*. Finally, the generalization links are mapped as foreign keys in the tables corresponding to the subclasses.

The decisions made in this transformation example are not unique alternatives. For instance, we can find many rules (point of views) to transform a generalization link. One of them maps abstract class *Person* as a duplication of its attributes in the tables that correspond to classes *Student* and *Teacher*.

Following Definition 3.4 of Sect. 2.1, SMD corresponds to the CLD, TMD represents the corresponding RS and MB is the set of mapping blocks between the two models. For example, a block describes the mapping of the association *evaluate* and classes *Module* and *ControlExam* in Fig. 5. This block respectively assigns tables *Module* and *ControlExam* to the two classes, and foreign key *IDModule* to the association (Fig. 5). As mentioned earlier, the transformations of the three constructs are grouped within the same block since they are interdependent.

To ease the manipulation of the source and target models and their transformation, the models are described using a set of predicates that correspond to the included constructs. Each construct is represented by one or more predicates. For example, Class Teacher in Fig. 5 is described as follows:

```
Class(Teacher).
Attribute(Level, Teacher,_).
```

The first predicate indicates that Teacher is a class. The second one states that Level is an attribute of that class and that its value is not unique ("_" instead of "unique").

The mapping blocks relate the predicates in the source model to their equivalent constructs in the target model. For instance, in Fig. 7, block B37[1] which contains the generalization link and the two classes Teacher and Person is described as follows:

```
Begin b37
Class(Person) : Table(Person).
Attribute(IDPerson, Person, unique) : Column(IDPerson, Person,
                                                          pk).
Attribute(Name, Person,_) : Column(Name, Person,_).
Attribute(FirstName, Person,_) : Column(FirstName, Person,_).
Attribute(Address, Person,_) : Column(Address, Person,_).
Class(Teacher) : Table(Teacher).
Attribute(Level, Teacher,_) : Column(Level, Teacher,_).
Generalization(Person, Teacher) : Column(IDPerson, Teacher, pfk).
End b37
```

Mappings are expressed with the ':' character. So, the mapping between predicates Attribute (IDPerson, Person, unique) and Column (IDPerson, Person, pk) indicates that the "unique" attribute IDPerson in class Person is transformed into the column IDPerson in table Person with the status of primary key. Similarly, the mapping between Generalization(Person, Teacher) and Column(IDPerson, Teacher, pfk) indicates that the generalization link is represented by the primary-foreign key (pfk) IDPerson in table Teacher.

A model M_i to transform is characterized by its description SMD_i, i.e. a set of constructs expressed by predicates. A construct can be transformed in many ways, each having a degree of relevance. This depends on three factors: (1) the adequacy of the individual construct transformations; (2) the internal consistency of the individual transformations inside the blocks; (3) the transformation (external) coherence between the related blocks, i.e., blocks sharing the same constructs. For example, consider a model to transform that has two classes, *Dog* and *Animal*, related by a generalization link *g*. *g* could become a table, many tables, a column, a foreign key, or any other possibility. A possibility is considered adequate if there exists a block in the example base that maps a generalization link. For instance, the mapping of block B37 (Fig. 7) is adequate because it also involves a generalization link. It is also internally consistent since it maps a similar pair of classes. Finally, it is externally coherent if *Dog* and *Animal* are only mapped to tables in the other blocks that contain them.

The transformation quality of a source model is the sum of the transformation qualities of its constructs. Consequently, finding a good transformation is equivalent to finding the combination of construct transformations that maximizes the

[1] For ease of traceability, the blocks are sequentially numbered, starting from the first transformation example in the example base. For instance, the nine blocks of example TE1 are labeled B_1 to B_9. The 13 blocks of TE2 B_{10} to B_{22}, and so on. When a solution is produced, it is relatively easy to determine which examples contributed to it.

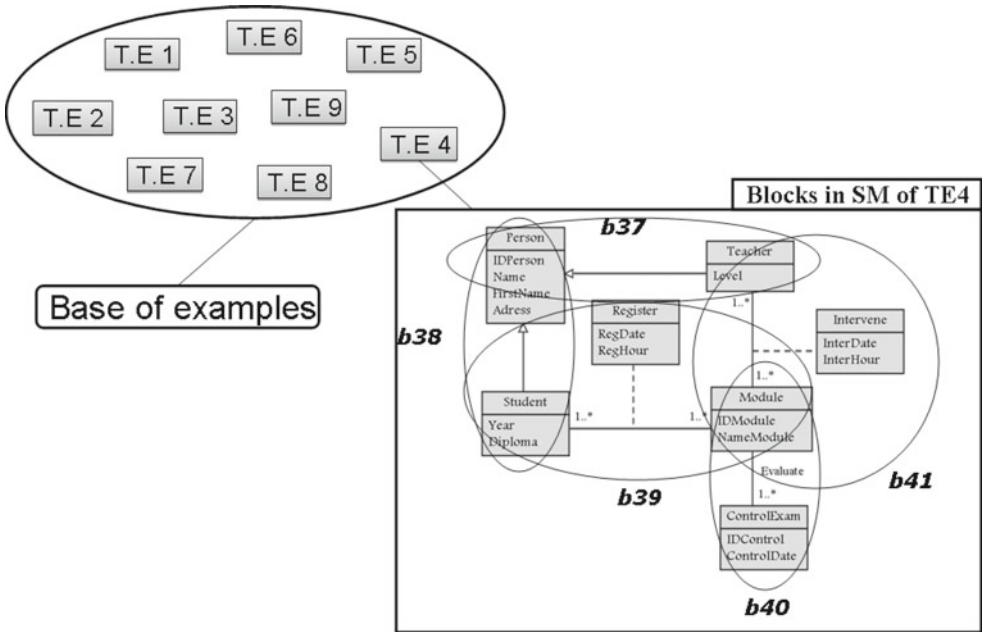

Fig. 7 Base of transformation examples and block generation in the source model of TE4

global quality. But since 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 previously, heuristic search offers a good alternative in this case. The search space dimensions are the constructs and the possible coordinates in these dimensions are the block numbers. A solution then consists of choosing a block number for each construct. The exploration of the search space using heuristic algorithms is presented next.

3 Transformation using search-based methods

We describe in this section the adaptation of PSO and SA to automate MT. To apply them to a specific problem, one must specify the encoding of solutions, the operators that allow movement in the search space so that new solutions are obtained, and the fitness function to evaluate a solution's quality. These three elements are detailed in Sects. 3.1, 3.2, and 3.3, respectively. Their use by PSO and SA to solve the MT problem is presented in Sects. 3.4 and 3.5.

3.1 Representing transformation solutions

One key issue when applying a search-based technique is finding a suitable mapping between the problem to solve and the techniques to use, i.e., in our case, encoding transformations between the source and target models. As stated in Sect. 2, we view the set of potential solutions as points in a n-dimensional space where each dimension corresponds to

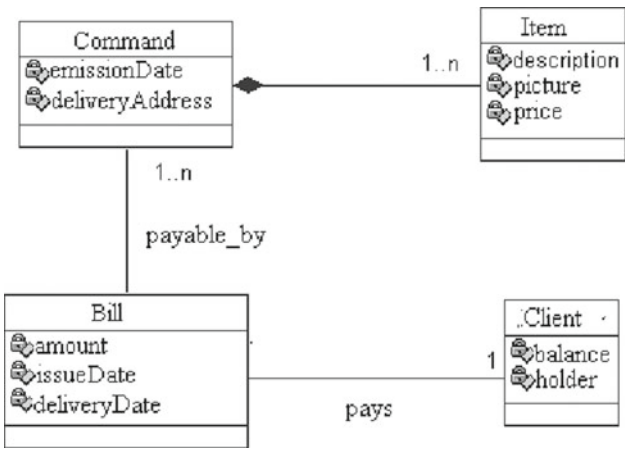

Fig. 8 Example of source model (UML-class diagram)

one of the n constructs of the model to transform. For instance, the transformation of the model shown in Fig. 8 will generate a seven-dimensional space that accounts for the four classes and three relationships.

Each construct is mapped from a finite set of m blocks extracted from the transformation examples. As a result, the dimensions of the search space take discrete values $b = \{i | 1 \leq i \leq m\}$. Thus, to define a particular solution, we associate with each dimension (construct) a block number that contains a transformation possibility and defines a coordinate in the corresponding dimension. The resulting n-tuple of block numbers then defines a vector position in the n-dimensional space. For instance, the solution shown in Table 1 suggests that *construct1* (class *Command*) be

Table 1 Solution representation

Dimension	Construct	Block number
1	Class(Command)	28
2	Class(Bill)	3
3	Class(Article)	21
4	Class(Seller)	13
5	Aggregation	9
6	Association(payable_by)	42
7	Association(pays)	5

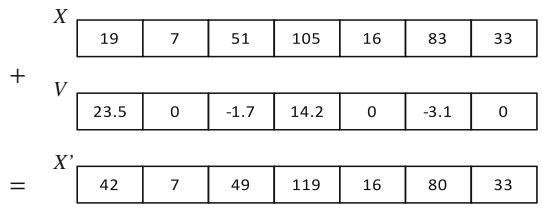

Fig. 9 Change Operator in PSO

transformed according to block28, construct2 (class *Bill*) be transformed according to block3, etc. To summarize, a solution is implemented as a vector where the constructs of the model to transform are the elements and the block numbers that refer to transformation possibilities from the example base are the element values. The proposed coding is valid for both heuristics. In the case of PSO, as an initial population, we create k solution vectors with a random assignment of blocks. Alternatively, SA starts from a solution vector produced by PSO.

3.2 Deriving a transformation solution

A change operator is needed to modify a candidate solution in order to produce a new one. In our case, it modifies a transformation of the source model in order to produce a new one. This is accomplished by changing the blocks for some constructs, which is equivalent to changing the coordinates of the solution in the search space. Unlike solution encoding, change operators are implemented differently by the PSO and SA heuristics. PSO changes blocks as a result of movement in the search space driven by a velocity function; SA performs the change randomly.

In the case of PSO, a translation (velocity) vector is regularly updated and added to a position vector to define new solutions (see Sect. 3.4, (3) and (4) for details). For example, the solution shown in Table 1 may lead to the new solution shown at the bottom of Fig. 9. Velocity vector V assigns a real-valued translation for each element of the position vector. After adding the two vectors, the elements of the result are each rounded to the nearest integer to represent block numbers. (The allowable values are bound by 1 and the maximum

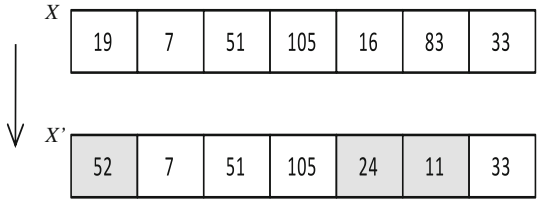

Fig. 10 Change Operator in SA

number of available blocks). As shown in Fig. 9, the new solution updates the block numbers of all construct. Thus, block 42 replaces block 19, block 7 remains the same, block 49 replaces block 51, etc.

For SA, the change operator involves randomly choosing l dimensions ($l < n$) and replacing their assigned blocks by randomly selected ones from the example base. For instance, Fig. 10 shows a new solution derived from the one of Table 1. Constructs 1, 5 and 6 are selected for change. They are assigned blocks 52, 24, and 11 in place of 19, 16, and 83, respectively. The other constructs keep their transformation blocks. The number of blocks to change is a parameter of the SA algorithm (three in this example).

In summary, regardless of the search heuristic, a change consists of assigning new block numbers to one or more dimensions. Thus, it creates a new transformation solution vector X_{i+1} from the previous one X_i.

3.3 Evaluating transformation solutions

The *fitness function* quantifies the quality of a transformation solution, which is basically a 1-to-1 assignment of blocks from the example base to the constructs of the source model. As discussed in Sect. 2, the fitness function must consider the three following aspects for a construct j to transform:

- Adequacy of the assigned block to construct j;
- Internal coherence of the individual construct transformation;
- External coherence with the other construct transformations.

In this context, we define the fitness function of a solution as the sum of qualities of the n individual construct transformations. Formally,

$$f = \sum_{j=1}^{n} a_j \times (\mathrm{ic}_j + \mathrm{ec}_j) \quad (1)$$

where j stands for a construct, a_j is the adequacy factor, ic_j the internal-coherence factor and ec_j the external-coherence factor. a_j has value 1 if the block associated to construct j contains at least one construct of the same type, and value 0 otherwise. It basically penalizes the assignment

of irrelevant blocks by assigning a zero adequacy value to them; this serves to reduce the search space.

The internal-coherence factor ic_j measures the similarity, in terms of properties, between the construct to transform and the construct of the same type in the assigned block. As shown in Sect. 3.1, the properties of the constructs are represented by the parameters of the predicates. Formally:

$$\text{ic}_j = \frac{\text{number of matched parameters in the predicates of the } j\text{th construct}}{\text{total number of parameters in the predicates of the } j\text{th construct}}$$

In general, a block assigned to a construct j contains more constructs than the one that is adequate with j. The external-coherence factor evaluates to which extent these constructs match the constructs that are linked to j in the source model. ec_j is defined as

$$\text{ec}_j = \frac{\text{number of matched constructs related to the } j\text{th construct}}{\text{total number of constructs related to } j\text{th construct}}$$

To illustrate the fitness calculation, consider again the example of Fig. 8. The association *payable_by* (sixth dimension) is defined by the predicate

Association (1,n,1,1,_,Command, Bill)

where the first four parameters indicates the multiplicities (1..n and 1..1), the fifth the name of the associative class if it exists, and the last two the source and target classes (*Command* and*Bill*). Consider a solution s_1 that assigns block 42 to this association:

```
Begin b42
Class(Client) : Table(Client).
Attribute(NClient, Client, unique) : Column(NClient, Client, pk).
Attribute(ClientName, Client, ) : Column(ClientName, Client, ).
Attribute(Address, Client,_) : Column(Address, Client,_).
Attribute(Tel, Client,_) : Column(Tel, Client,_).
Class(Reservation) : Table(Reservation).
Attribute(NReservation, Reservation, unique) :
Column(NReservation, Reservation, pk).
Attribute(StartDate, Reservation,_) : Column(StartDate,
Reservation,_).
Attribute(EndDate, Reservation,_) : Column(EndDate,
Reservation,_).
Attribute(Region, Reservation,_) : Column(Region, Reservation,_).
Association (1,n,0,n,_, Client, Reservation) : Column(N_Client,
reservation, fk).
End b42
```

In this case, a_6 (adequacy for the 6th construct) is equal to 1 because block 42 contains a predicate *Association* that relates classes *Client* and *Reservation*. This association predicate has five parameters out of seven that match the ones of *pays* (1, n, x, x, _, origin and destination class names). As a result, we have $ic_6 = 5/7 = 0.71$. Moreover, according to block 42, to be consistent with the transformation of *payable_by,* classes *Bill* and *Command* have to be mapped to tables. On the other hand, s_1 also assigns blocks 28 and 3 to classes *Bill* (dimension 2) and *Client* (dimension 4), respectively. These two blocks are defined as follows.

```
Begin b28
Class(Position) : Table(Position).
….
Class(Employee) : Table(Employee).
…
Association(0,1, ,n,_, Position, Employee) : Column(IDPosition,
Employee, fk).
End b28

Begin b3
Class(Manager) : Table(Manager).
…
Class(Employee) : Table(Employee).
…
Generalization(Employee, Manager) : Column(IDEmployee, Manager,
fk).
End b3
```

In both blocks, classes are transformed into tables. Since this does not conflict with block 42, we have for the two related constructs $c_6 = 2/2 = 1$.

The fitness function also evaluates the completeness of a transformation indirectly. A solution that does not transform a subset of constructs will be penalized. Those constructs will have null values (a_j being always equal to 0). Finally, to make the values comparable across models with different numbers of constructs, a normalized version of the fitness function is used. For a particular construct, the fitness varies between 0 and 2 (ic_j and ec_j can be both equal to 1). Considering the n constructs, we normalized the fitness function as follows:

$$f_{\text{nor}} = \frac{f}{2 \times n} \tag{2}$$

We used this normalized fitness function for both PSO and SA.

3.4 Global search (PSO)

3.4.1 PSO principles

Particle swarm optimization is a parallel population-based computation technique [18]. It was originally inspired from the flocking behavior of birds, which emerges from very simple individual conducts. Many variations of the algorithm have been proposed over the years, but they all share a common basis. First, an initial population (named swarm) of random solutions (named particles) is created. Then, each particle flies in the n-dimensional problem space with a velocity that is regularly adjusted according to the composite flying experience of the particle and some, or all, the other particles. All particles have fitness values that are evaluated by the objective function to be optimized. Every particle in the swarm is described by its position and velocity. A particle position represents a possible solution to the optimization problem, and velocity represents the search distances and directions that guide particle flying. In this paper, we use basic velocity and position update rules defined by Kennedy and Eberhart [18]:

$$V_{\text{id}} = W \times V_{\text{id}} + C_1 \times Rand() \times \left(P_{\text{id}} - X_{\text{id}}\right) + C_2 \times Rand() \left(P_{\text{gd}} - X_{\text{id}}\right) \quad (3)$$

$$X_{\text{id}} = X_{\text{id}} + V_{\text{id}} \quad (4)$$

At each time (iteration), V_{id} represents the particle velocity and X_{id} its position in the search space. P_{id} (also called *pbest* for local best solution), represents the ith particle's best previous position, and P_{gd} (also called *gbest* for global best solution), represents the best position among all particles in the population. w is an inertia term; it sets a balance between the global and local exploration abilities in the swarm. Constants c_1 and c_2 represent cognitive and social weights associated to the individual and global behavior, respectively. There are also two random functions *rand()* and *Rand()* (normally uniform in the interval [0, 1]) that represent stochastic acceleration during the attempt to pull each particle toward the *pbest* and *gbest* positions. For a n-dimensional search space, the i^{th} particle in the swarm is represented by a n-dimensional vector, $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id})$. The velocity of the particle, *pbest* and *gbest* are also represented by n-dimensional vectors. Algorithm 1 summarizes the generic PSO procedure.

PSO algorithm

```
1: Initial population (particles) creating (initialization)
2: while Termination criterion not met do
3:         for each particle do
4:                 Evaluate fitness
5:                 Update local/global best (if necessary)
6:                 Update velocity and position
7:         end for
8: end while
9: Return solution corresponding to the global best
```

3.4.2 PSO for MT

The PSO swarm is represented as a set of K particles, each defined by a position vector corresponding to the n constructs of the model to transform. For a particle position, the values of the position vector's elements are the mapping blocks selected for each construct. Our version of PSO starts by randomly generating the particle positions and velocities in the swarm. This is done by randomly affecting a block number to each of the n constructs (dimensions). Thus, the initial particle population represents K different possibilities (solutions) to transform the source model by combining blocks from the transformation examples. The fitness of each particle is measured by the fitness function defined by (1) and (2).

The particle with the highest fitness is memorized as the global best solution during the search process. At each iteration, the algorithm compares the fitness of each particle with those of the other particles in the population to determine the *gbest* position for use to update the swarm. Then, for each particle, it compares its current positions with *pbest*, and update the latter if an improvement is found. The new positions affect the velocity of each particle according to (3). The algorithm iterates until the particles converge towards a good transformation solution of the source model. In our case, we define a maximum number of iterations after which we select the *gbest* as the transformation solution. The algorithm stops before if all the particles converge to the same solution.

The parameters in Eq.(3) have an important effect on the search efficiency of the PSO algorithm. Acceleration constants c_1 and c_2 adjust the amount of "tension" in the system. Low values allow particles to roam far from target regions before being tugged back, while high values result in abrupt movement toward, or past, target regions [23]. Based on past research experience, we set both constants to 1. Equations (3) and (4) may lead to large absolute values for V_{id} and X_{id}, so that a particle may overshoot the problem space. Therefore, V_{id} and X_{id} should be confined to a maximum velocity $V_{\max}$, and a maximum position $X_{\max}$, such that

$$X_{\max} = N; \quad X_{\text{id}} = \min\left(\max\left(0, X_{\text{id}} + V_{\text{id}}\right), X_{\max}\right) \quad (5)$$

$V_{\max}$ serves as a constraint to control the global exploration ability of a particle swarm. It should take values in the interval $[-m, m]$, m being the number of blocks in the existing transformation examples. X_{id} represents the block number affected to a construct; it must be a positive integer. Hence, a real value for X_{id} is rounded to the closest block number by dropping the sign and the fractional part.

The inertia weight (w) is another important parameter of the PSO search. A proper value for w provides a balance between global and local exploration, and results in less iterations to find a solution on average. In practice, it is often linearly decreased through the course of the PSO, for the PSO to have more global search ability at the beginning of the run and more local search ability near the end. For the validation experience in this paper, the parameter was set as follows [23]:

$$W = W_{\max} - \left(\frac{W_{\max} - W_{\min}}{iter_{\max}}\right) \times iter \quad (6)$$

where $W_{\max}$ is the initial value of weighting coefficient, $W_{\min}$ is a minimal value of weighting coefficient, $iter_{\max}$ is the maximum number of iterations, and *iter* is the current iteration.

3.5 Local search (SA)

3.5.1 SA principles

In the case of a quick run of PSO (only a few iterations), the best transformation solution can be improved by using another search heuristic. We propose in this work to use SA in combination with PSO. SA [19] is a search algorithm that gradually transforms a solution following the annealing principle used in metallurgy.

The generic behavior of this heuristic is described by Algorithm 2. After defining an initial solution, the algorithm iterates the following three steps:

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.

```
                        SA algorithm
current_solution ← initial_solution
current_cost ← evaluate (current_solution)
T ← T_initial
while (T > T_final ) do
        for i=1 to iterations (T) do
                new_solution ← move (current_solution)
                new_cost ← evaluate(new_solution)
                Δcost ← new_cost – current_cost
                if (Δcost≤0 OR e^(-Δcost/<T) < random() )
                        current_solution ← new_solution
                        current_cost ← new_cost
                end if
        end for
        T← next_temp(T)
end while
```

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 this case, the new solution is accepted with probability $\mathbf{e}^{-\Delta \mathbf{cost}/\mathbf{T}}$. The introduction of a stochastic element in the decision process avoids being trapped in a local minimum solution. Parameter T, called temperature, controls the acceptance probability of a lesser good solution. T begins with a high value, for a high probability of accepting a solution during the early iterations. Then, it decreases gradually (cooling phase) to lower the acceptation probability as we advance in the iteration sequence. For each temperature value, the three steps of the algorithm are repeated for a fixed number of iterations.

One attractive feature of the SA algorithm is that it is problem-independent and can be applied to most combinatorial optimization problems [24,25]. However, SA is usually slow to converge to a solution.

3.5.2 SA for MT

To obtain a more robust optimization technique, it is common to combine different search strategies in an attempt to compensate the deficiencies of individual algorithms [24]. In our case, the search for a solution is done in two steps. First, a global search is quickly performed to locate the portion of search space where good solutions are likely to be found. This is performed by PSO and results in a near-optimal solution. In the second step, the obtained solution is refined by the SA algorithm.

As described in Sect. 3.1, solutions are coded by assigning a block number to each construct in order to form a vector. The SA algorithm starts with an initial solution generated by a quick run of PSO. As for PSO, the fitness function presented in Sect. 3.3 measures the quality of the solution at the end of each iteration. The generation of a neighboring solution is obtained by randomly changing a number of dimensions with new randomly selected blocks. The way in which we decrement the temperature coefficient is critical to the success of the algorithm. SA theory states that we should allow enough iterations for each temperature value to allow the system to stabilize at that temperature. Unfortunately, the theory also states that the number of iterations at each temperature to achieve this might vary exponentially with problem size. As this is impractical, a compromise measure is to either do a large number of iterations for a few temperature values, a small number of iterations for many temperatures values, or strike a balance between the two. One way to decrement the temperature is use a geometric cooling schedule [19]. The temperature is reduced using:

$$T_{i+1} = \alpha \times T_i \tag{7}$$

where α is a constant less than 1. Experience has shown that α should be between 0.8 and 0.99, with better results being found at the higher end of the range. On the other hand, the higher the value of α, the longer it will take to decrement the temperature to the stopping criterion.

4 Evaluation and comparison

To evaluate the feasibility of our approach, we conducted an experiment with industrial data. We start by presenting our experimental setting. Then, we describe and discuss the obtained results. We compare in particular the results of PSO with the PSO-SA combination. Finally, we evaluate the impact of the example base size on transformation quality.

4.1 Setting

We used 12 examples of class-diagrams to RS transformations to build an example base EB $= \{\langle \text{CLD}_i, \text{SR}_i \rangle | 1 \leq i \leq 12\}$. The examples were provided by an industrial partner. As showed in Table 2, the size of the CLDs varied from 28 to 92 constructs, with an average of 58. Altogether, the 12 examples defined 257 mapping blocks. Because our industrial partner uses Rational Rose to derive RS from UML class models, we did not have transformation blocks defined by experts during the transformation. For the need of the experience, we automatically extracted the transformation traces from XMI files produced by Rational Rose. Then, we manually partitioned the traces into blocks.

Table 2 12-fold cross validation with PSO

Source model	Number of constructs	Fitness	AC	MC
SM 1	72	0.696	0.618	0.882
SM 2	83	0.714	0.682	0.928
SM 3	49	0.762	0.721	0.943
SM 4	53	0.796	0.719	0.931
SM 5	38	0.773	0.789	0.952
SM 6	47	0.746	0.652	0.918
SM 7	78	0.715	0.772	0.957
SM 8	34	0.896	0.822	0.981
SM 9	92	0.61	0.634	0.87
SM 10	28	0.892	0.908	0.969
SM 11	59	0.773	0.717	0.915
SM 12	63	0.805	0.762	0.938
Average	58	0.764	0.733	0.932

To evaluate the quality of transformations produced by MOTOE, we used a 12-fold cross validation procedure. For each fold, one class diagram CLD_j is transformed by using the remaining 11 transformation examples ($\mathrm{EB}_j = \{\langle \mathrm{CLD}_i, \mathrm{RS}_i \rangle | i \neq j\}$). Then the transformation result of each fold is checked for correctness. The correctness of a transformation $t\mathrm{CLD}_j$ was measured by two methods: automatic correctness (AC) and manual correctness (MC). Automatic correctness consists of comparing the derived transformation $t\mathrm{CLD}_j$ to the known RS_j, construct by construct. This method has the advantage of being automatic and objective. However, since a given CLD_j may have different transformation possibilities, AC could reject a valid construct transformation because it yields a different RS from the one provided. To account for those situations, we also use MC which manually evaluates $t\mathrm{CLD}_j$, here again construct by construct. In both cases, the correctness of a transformation is the proportion of constructs that are correctly transformed.

To set the parameters of PSO for the global search strategy, we started with commonly found values in the literature [26–28] and adapted some of them to the particularities of the transformation problem. The final parameters values were set as follows:

- The swarm is composed of 40 particles. We found this number to provide a good balance between population diversity and the quantity of available examples.
- The inertia weight W is initially set to 1.3 and gradually decreased after each iteration according to (6), until it reaches 0.3,
- C_1 and C_2 are both equal to 1 to give equal importance to local and global search.
- The maximum number of iterations is set to twice the size of the population, i.e. 80. This is a generally accepted heuristic [23].
- Since two different executions of a search heuristic may produce different results for the same model, we decided, for each of 12 folds, to take the best result from 5 executions.

As mentioned previously, the initial particle positions are randomly generated. The range of values for each particle coordinate (construct) is defined as [0, *MaxBlocks*] where *MaxBlocks* is the total number of blocks extracted from the 11 examples of the fold. In our case, *MaxBlocks* is 257 minus the number of blocks of the fold example.

For the hybrid search strategy, the SA algorithm was applied using the following parameters:

- The initial temperature of the process is randomly selected in the range [0, 1];
- The geometric cooling coefficient α is 0.98;
- The iteration interval for temperature update is 10,000 (to account for SA's slowness);
- The number of dimensions to change for generating a neighboring solution is set to 2. This value offers a good balance with the large number of iterations;
- The stopping criterion (temperature threshold) is 0.1.

To quickly generate an initial solution for SA, we limited the number of particles to 10 and the number of iterations to 20 for PSO.

With these parameter values, the transformation of the largest of the used diagrams only took a few seconds of runtime.

4.2 Results and discussion

4.2.1 Results

Tables 2 and 3, respectively, show the obtained correctness for each of the 12 folds, when using global and hybrid search. Both automatic and manual correctness values were high and, as expected, manual evaluation yielded better correctness since it considered all the correct transformations and not only the specific alternatives chosen by our industrial partner. We consider correctness values (74 and 94% for, respectively, the automatic and the manual validation) as relatively high relatively given the context of no transformation rules and the limited number of used examples.

Table 2 shows the correctness of the generated transformations using the PSO heuristic. The automatic correctness measure had an average value of 73.3%, with most of the models transformed with at least 70% precision. The manual correctness measure was much greater, with an average value of 93.2%; this indicates that the proposed

Table 3 12-fold cross validation with PSO-SA

Source model	Number of constructs	Fitness	AC	MC
SM 1	72	0.735	0.696	0.947
SM 2	83	0.784	0.723	0.962
SM 3	49	0.632	0.69	0.912
SM 4	53	0.619	0.672	0.956
SM 5	38	0.742	0.733	0.93
SM 6	47	0.737	0.704	0.953
SM 7	78	0.743	0.79	0.942
SM 8	34	0.845	0.813	0.975
SM 9	92	0.648	0.667	0.931
SM 10	28	0.846	0.873	0.983
SM 11	59	0.796	0.73	0.92
SM 12	63	0.772	0.72	0.964
Average	58	0.742	0.734	0.948

transformations were almost as correct as the ones given by experts. The worst model (SM9) had an acceptable MC of 87% and four models obtained an MC greater than 95%, with a value of 98.1% for SM8.

The hybrid search gave slightly better correctness results as shown in Table 3. Both automatic and manual correctness were slightly better on average (93.4% for AC and 94.8 for MC). With regards to MC, the quality of eight MTs was improved while that of four was slightly degraded. For instance, MC for the worst transformed model (SM9) improved from 87 to 93.1%, while that for the best transformed model (SM5) decreased from 95.2 to 93%.

4.2.2 Discussion

One observation to be made from the results in Tables 2 and 3 is that, with the exception of model SM7, hybrid search yielded better results than global search for the models with the highest numbers of constructs. This may be due to the fact that, when the number of dimensions is high, the search space is very large and the use of PSO leads to particle movement steps that can only approximate the location of the target solution. A more focused search consisting of global search followed by local exploration produces better results in this case. In contrast, for a smaller search space (less dimensions), area coverage by the particles is easier, and a global search appears to be more efficient to zero in on the solution.

To better analyze the performance of the hybrid strategy, Fig. 11 shows, for all models, the average final values of the fitness function after the quick search with PSO and their corresponding values after the refinement made by SA. As one can see, substantial fitness improvement occurred (>50% in many cases) in each case of the 12-fold cross validation. It appears then that the hybrid strategy brings a

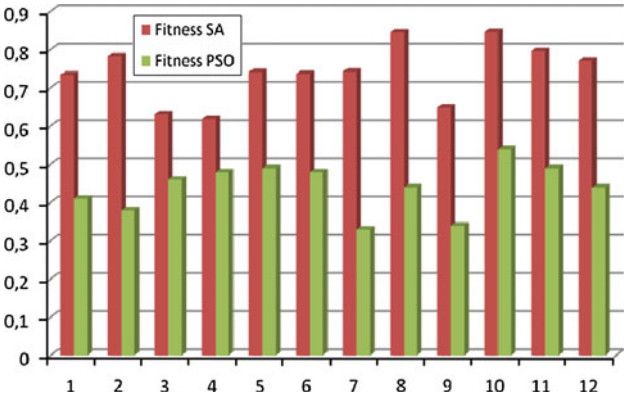

Fig. 11 Fitness improvement with SA after PSO initial pass

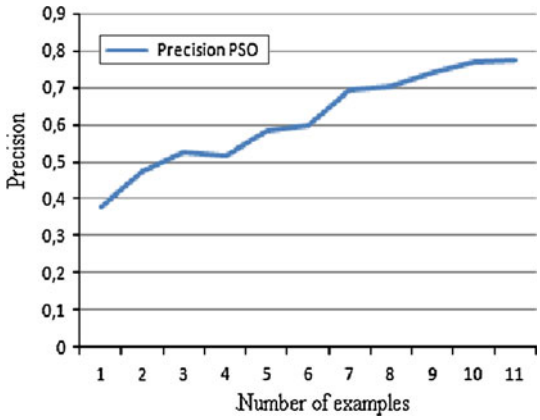

Fig. 12 Example-size variation with PSO

good compromise between correctness and execution time: it allows improving the transformation correctness with a slight increase of the execution time. The obtained results also show that our fitness function is a good estimator of transformation correctness.

An important consideration is the impact of the example base size on transformation quality. Drawn for SM7, the results of Figs. 12 and 13 show that our approach also proposes transformation solutions in situations where only few examples are available. When using the global search strategy, AC seems to grow steadily and linearly with the number of examples. For the hybrid strategy, the correctness seems to follow an exponential curve; it rapidly grows to acceptable values and then slows down. Indeed, AC improved from roughly 30 to 65% as the example base size went from 1 to 4 examples. Then, it grew only by an additional 15% as the size varied from 6 to 11 examples.

When manually analyzing the results, we noticed that some of the 12 models had constructs not present in the other models. Those constructs were generally not transformed as no adequate block could be found for them. However, others were transformed by adapting the transformation of constructs of the same nature. This was the case, for instance,

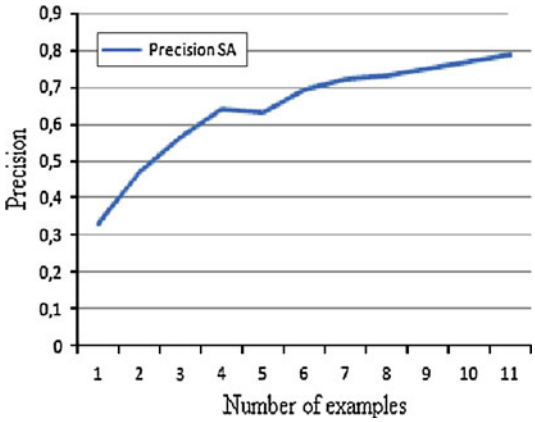

Fig. 13 Example-size variation with PSO-SA

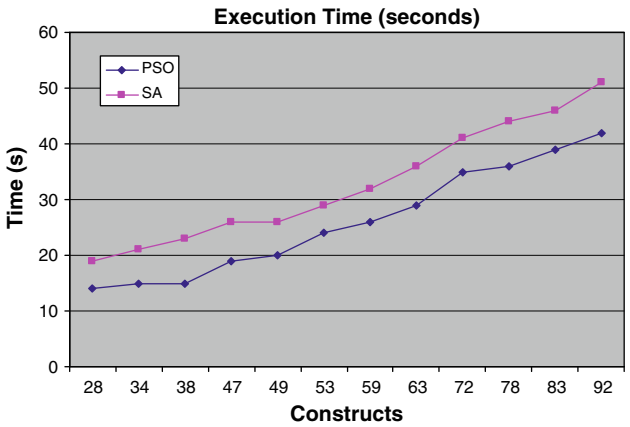

Fig. 14 Execution time

for an association with multiplicity $(1..N, 1..N)$. Since the multiplicity elements are considered as parameters of the construct, the transformation of an association $(0..N, 0..N)$ was applied with a penalty for the fitness function. Although these few cases of adaptation improved the global correctness scores, we did not specifically address the issue at the current stage of our research.

Finally, since we viewed the transformation 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 2 GHz with 1GB of RAM). The execution time is shown in Fig. 14 for a number of constructs to transform corresponding to small to medium-sized problems. The Figure reveals that greater execution times may be obtained in comparison with using rule-based tools for small-dimensional problems. In any case, our approach is meant to apply to situations where rule-based solutions are normally not readily available.

5 Related work

The work proposed in this paper can be related to three research areas in software engineering, of which the most relevant one is MT in the context of MDD. Some links can also be found with by-example and search-based software engineering (SBSE), but our concerns are different as will be discussed below. As a result, only a comparison to alternatives in the first area is warranted.

5.1 Model transformation

Several MT approaches can be found in the literature (see, for example, the classifications given in [29,30]). Czarnecki and Helsen [29] distinguish between two main types: model-to-model and model-to-code. They describe five categories of the former: Graph-transformation-based [31], relational [32], structure-driven [33], direct-manipulation and hybrid. They use various criteria to analyze them, like the consideration of model-driven architecture (MDA) as a basis for transformation, the complexity and scalability of the transformation mechanism, the use or not of annotations, the level of automation, and the used languages and implementation techniques. In general, the reported approaches are based on empirically obtained rules [15,34] in contradistinction to block transformation in MOTOE. In rules-based approaches, the rules are defined in metamodels while our blocks relate to specific problems, with a varying structure, for different problems.

In existing transformation approaches such as graph transformation [31,35–40], a transformation rule consists of two parts: a left-hand side (LHS) and a right-hand side (RHS). The LHS accesses the source model, whereas the RHS expands in the target model. By comparison, each block in MOTOE contains a transformation of source elements (LHS) to their equivalent target elements (RHS). However, in a graph-transformation approach, potentials conflicts between transformation elements are verified with pre and post condition. In our case, pre and post conditions are replaced by the fitness function that ensures transformations coherency.

In rule based approaches, a rule is applied to a specific location within its source scope. Since there may be more than one match for a rule within a given source scope, we need an application strategy. The strategy could be deterministic, non-deterministic or even interactive [41]. For example, a deterministic strategy could exploit some standard traversal strategy (such as depth-first) over the containment hierarchy in the source. In our work, the transformation possibilities (blocks) are randomly chosen with no strategy for rules application (e.g., rules scheduling).

Transformation rules are usually designed to have a functional character: given some input in the source model, they produce a concrete result in the target model [10].

A declarative rule (i.e., one that only uses declarative logic and/or patterns) can often be applied in the inverse direction. However, since different inputs may lead to the same output, the inverse of a rule may not be a function. We have the same problem in our approach since blocks are only defined in one direction (from CLD to RS for example). To ensure a bidirectional transformation property, we need to apply our methodology to examples from the other direction.

If we define cognitive complexity as the level of difficulty to design a MT, we believe that collecting/recording transformation examples may be less difficult than producing and maintaining consistent transformation rule sets. This is consistent with recent trend in industry where we find several tools to record transformations and automatically generate transformation traceability records [42].

The traditional approach for implementing MTs is to specify transformation rules and automate the transformation process by using a MT language [43]. Most of these languages are powerful enough to implement large-scale and complex MT tasks. However, the transformation rules are usually defined at the metamodel level, which requires a clear and deep understanding about the abstract syntax and semantic interrelationships between the source and target models. In some cases, domain concepts may be hidden in the metamodel and difficult to unveil [15,34] (e.g., some concepts are hidden in attributes or association ends, rather than being represented as first-class entities). These implicit concepts may make writing transformation rules difficult.

To help address the previous challenges, an alternative approach called MTBE was proposed in [14,16]. In it, users are asked to build a prototypical set of interrelated mappings between the source and target model instances, and then the metamodel-level transformation rules will be semi-automatically generated. Because users configure the mappings at the instance level, without knowing any details about the metamodel definition or the hidden concepts, combined with the generated rules, the simplicity of specifying MTs can be improved. Varrò and Balogh [14,16] propose a semi-automated process for MTBE using ILP. The principle of their approach is to derive transformation rules semi-automatically from an initial prototypical set of interrelated source and target models. Another similar work is that of Wimmer et al. [44] who derive ATL transformation rules from examples of business process models. Both works use semantic correspondences between models to derive rules. Their differences include the fact that [44] presents an object-based approach that finally derives ATL rules for MT, while [16] derives graph transformation rules. Another difference is that they respectively use abstract versus concrete syntax: Varro uses IPL while Wimmer relies on an ad hoc technique. Both approaches provide a semi-automatic generation of MT rules that needs further refinement by the user. Also, since both approaches are based on semantic mappings, they are more appropriate in the context of exogenous MTs between different metamodel. Unfortunately, the generation of rules to transform attributes is not well supported in most MTBE implementations. Our model is different from both previous approaches to MTBE. We do not create transformation rules to transform a source model, directly using examples instead. As a result, our approach does not rely upon formalisms.

Recently, a similar approach to MTBE, called Model Transformation By Demonstration (MTBD), was proposed [45]. Instead of the MTBE idea of inferring the rules from a prototypical set of mappings, users are asked to demonstrate how the MT should be done, through direct editing (e.g., add, delete, connect, update) of the source model, so as to simulate the transformation process. A recording and inference engine was been developed, as part of a prototype called MT-Scribe, to capture all user operations and infer a user's intention during a MT task. A transformation pattern is generated from the inference, specifying the preconditions of the transformation and the sequence of operations needed to realize the transformation. This pattern can be reused by automatically matching the preconditions in a new model instance and replaying the necessary operations to simulate the MT process. However, this approach needs a large number of simulated patterns to be efficient and it requires a high level of user intervention. In fact, the user must choose the suitable transformation pattern. Finally, the authors do not show how MTBD can be useful to transform an entire source model and only provide examples of transforming model fragments.

Some others metamodel matching works can also be considered as variants of by-example approaches. Garcia-Magarino et al. [46] propose an approach to generate transformation rules between two meta-models that satisfy some constraints introduced manually by the developer. In [47], the authors propose to automatically capture some transformation patterns in order to generate matching rules in the metamodel level. This approach is similar to MTBD, but it is used at the meta-model level. The fundamental difference of these approaches with ours is that we do not generate transformation rules and MOTOE does not need to specify the source and target metamodels as input.

To conclude, the previous problems limit the applicability of MTBE/MTBD for some transformation problems. In such situations, MOTOE may leads to more relevant solutions. In our approach, the definition of transformation examples is based on the use of traceability information [42,48–52]. Traceability usually allows tracing artifacts within a set of chained operations, where the operations may be performed either manually (e.g., crafting a software design for a set of software requirements) or with automated assistance (e.g., generating code from a set of abstract descriptions). For example, Triple Graph Grammars (TGG) [53] explicitly maintain the correspondence of two graphs

by means of correspondence links. These correspondence links play the role of traceability links that map elements of one graph to elements of the other graph, and vice versa. With TGG, one has to explicitly describe the correspondence between the source and target models, which is difficult if the transformation is complex and the intermediate models are required during the transformation. In [54], a traceability framework for Kermeta is discussed. This framework supports the creation of traces throughout a transformation chain. Marvie describes a transformation composition framework [55] that allows manual creation of linkings (traces). This framework does not support the automatic generation of traces.

Finally, a large part of the work on traceability in MDE uses it for detecting model inconsistency and fault localization in transformations. In MOTOE, this is not the goal as the purpose is to use trace information as input to automate the transformation process. The trace information (model correspondence) between a source and target model define a transformation example that is decomposed in some independent blocks as explained before.

Our approach is also different from CBR methods where the level of granularity must be the example as a whole, i.e., a transformation example [20]; in our case, we do not select the most similar example and adapt its transformation; rather, we aggregate the best transformation possibilities coming from different examples.

5.2 By-example software engineering

The approach proposed in this paper is based on using past transformation examples. Various such "by-example" approaches have been proposed in the software engineering literature [56,57]. However, the problems addressed by them differ from ours in both nature and objectives. The closest work to ours is program transformations by demonstration [58,59], in which a user manually changes program examples while a monitoring plug-in to the development environment records the changes. Then, the recorded data are analyzed to create general transformations that can be reused in subsequent programs. However, the overall process is not automated and requires frequent interaction with the user, and the generated transformation patterns are found via a different algorithms than the one used by MOTOE.

5.3 Search-based software engineering

Our approach is inspired by contributions in SBSE [60–64,66]. As the name indicates, SBSE uses a search-based approach 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. To the best of our knowledge, inspired among others by the roadmap paper of Harman [62], the idea of treating MT as a combinatorial optimization problem to be solved by a search-based approach was not studied before our proposal in [17]. For this reason, we cannot compare our approach to existing works in SBSE since their application domain is different.

6 Summary and conclusion

In summary, we described MOTOE, a novel approach to automate MT 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, for each possibility, a quality is associated depending on its conformance with the examples at hand. The search space is explored with two methods. In the first one, we use PSO with transformation solutions generated from the examples at hand as particles. Particles progressively converge toward a good solution by exchanging and adapting individual construct transformation possibilities. In the second method, a partial run of PSO is performed to derive an initial solution. This solution is then refined using a local search with SA. The refinement explores neighboring solutions obtained by trying individual construct transformation possibilities derived from the example base. In both methods, the quality of a solution considers the adequacy of construct transformations as well as their mutual consistency.

We illustrated MOTOE with the transformation of UML CLDs to RS. In this context, we conducted a validation on real industrial models. The experiment results clearly indicate that the derived models are comparable to those proposed by experts (correctness of more than 90% with manual evaluation). They also reveal also that some constructs were correctly transformed although no transformation examples were available for them. This was possible because the approach uses syntactic similarity between construct types to adapt their transformations. We also showed that the two methods used for the space search produced comparable results when properly applied, and that PSO alone is enough with small-to-medium models while the combination PSO-SA is more suitable when the size of the models to transform is larger. For both methods, our transformation process derives a good quality transformation in an acceptable execution time. Finally, the validation study showed that the quality of MT improves with the number of examples. However, it reaches a stable score after as few as nine examples.

Our proposed method also has limitations. First, MOTOE's performance depends on the availability of transformation examples, which could be difficult to collect.

Second, the generation of blocks from the examples is done manually in our present work; we could partially automate this task using decomposition heuristics. Third, due to the nature of our solution, i.e., an optimization technique, the transformation process can be time consuming for large models. Finally, as we use heuristic algorithms, different execution for the same source models may lead to different target models. Nevertheless, we showed in our validation that solutions that have high fitness values also have good correctness. Moreover, this is close to what happens in the real world where different experts could propose different target models.

From the applicability point of view, our approach can theoretically be applied to the transformation of models for any pair of source/target formalisms. To practically assess this claim, we are currently experimenting with other formalisms such as sequence diagrams to Petri nets. We also plan to work on adapting our approach to other transformation problems such as code generation (model-to-code), refactoring (code-to-code), and reverse engineering (code-to-model). The refactoring problem also has the advantage of exploring endogenous transformations where source and target models conform to the same metamodel. Regarding the quality evaluation of transformations, the fitness function we used could be improved. In this work, we gave equal importance to all constructs. In the real world, some construct types may be more important than others.

References

1. France, R., Rumpe, B.: Model-driven development of complex software: a research roadmap. In: Briand, L., Wolf, A. (eds.) International Conference on Software Engineering (ICSE 2007): Future of Software Engineering IEEE Computer Soceity Press, Los Alamitos (2007)
2. Interactive Objects and Project Technology, MOF Query/Views/Transformations, Revised Submission. OMG Document: ad/03-08-11, ad/03-08-12, ad/03-08-13 (2003)
3. Object Management Group (OMG): Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification Final Adopted Specification, ptc/05-11-01. http://www.omg.org/docs/ptc/05-11-01.pdf (2005)
4. QVT-Merge Group. MOF 2.0 Query/Views/Transformations RFP, Revised submission, version 1.0. OMG Document ad/2004-04-01. http://www.omg.org/cgi-bin/doc?ad/2004-04-01 (2004)
5. OMG: Meta Object Facility (MOF). Version 1.4. www.omg.org/technology/documents/formal/mof.htm
6. OMG: Request For Proposal: MOF 2.0/QVT. OMG Document ad/2002-04-10. http://www.omg.org/cgi-bin/doc?ad/2002-04-10 (2002)
7. Taentzer, G.: AGG: a graph transformation environment for system modeling and validation. In: Proceedings of Tool Exihibition at Formal Methods 2003, Pisa, Italy (2003)
8. Varro, D., Pataricza, A.: Generic and meta-transformations for model transformation engineering. In: Baar, T., Strohmeier, A., Moreira, A., Mellor, S.J. (eds.) UML 2004. LNCS, vol. 3273, Springer, Heidelberg (2004)
9. ATLAS Group. The ATLAS Transformation Language. http://www.eclipse.org/gmt (2000)
10. Jouault, F., Kurter, I.: Transforming models with ATL. In: Proceedings Of the Model Transformations in Practice Workshop at MoDELS 2005, Jamaica (2005)
11. Compuware, SUN. MOF 2.0 Query/Views/Transformations RFP, Revised Submission. OMG Document ad/2003-08-07. http://www.omg.org/cgi-bin/doc?ad/2003-08-07 (2003)
12. Clark, T., Warmer, J.: Object Modeling with the OCL, The Rationale behind the Object Constraint Language. Springer, London (2002)
13. Behrens, U., Flasinski, M., Hagge, L., Jurek, J., Ohrenberg, K.: Recent developments of the ZEUS expert system ZEX. IEEE Trans. Nucl. Sci. **43**, 65–68 (1996)
14. Varro, D.: Model transformation by example. In: Proc. MODELS 2006, vol. 4199 of LNCS, pp. 410–424. Springer, Heidelberg (2006)
15. Egyed, A.: Automated abstraction of class diagrams. ACM Trans. Softw. Eng. Methodol. **11**(4), 449–491 (2002)
16. Varro, D., Balogh, Z.: Automating model transformation by example using inductive logic programming. ACM Symposium on Applied Computing | Model Transformation Track (2007)
17. Kessentini, M., Sahraoui, H., Boukadoum, M.: Model transformation as an optimization problem. In: Proceedings of MODELS 2008, vol. 5301 of LNCS, pp. 159–173. Springer, Heidelberg (2008)
18. Kennedy, J., Eberhart, R.C.: Particle swarm optimization. In: Proceedings of IEEE International Conference on Neural Networks, pp. 1942–1948 (1995)
19. Kirkpatrick, S., Gelatt, C.D. Jr.., Vecchi, M.P.: Optimization by simulated annealing. Sciences **220**(4598), 671–680 (1983)
20. Aamodt, A., Plaza, E.: Case-based reasoning: foundational issues, methodological variations, and system approaches. Artif. Intell. Commun. **7**(1), 39–52 (1994)
21. Kessentini, M., Bouchoucha, A., Sahraoui, H., Boukadoum, M.: Example-based sequence diagrams to colored petri nets transformation using heuristic Search. In: Proceedings of 6th European Conference on Modeling Foundations and Applications (ECMFA 2010), Paris (2010)
22. Siikarla, M., Syst, T.: Decision reuse in an interactive model transformation. In: 12th European Conference on Software Maintenance and Reengineering, CSMR 2008, April 1–4, 2008, Athens, Greece, pp. 123–132 (2008)
23. Eberhart, R.C. Shi, Y.: Particle swarm optimization: developments, applications and resources. In: Proceedings of IEEE Congress on Evolutionary Computation (CEC 2001), pp. 81–86 (2001)
24. Mitra, D., Romeo, F., Sangiovanni-Vincentelli, A.: Convergence and finite-time behaviour of simulated annealing. In: Proceedings of 1985 Decision and Control Control, vol. 5 (1985)
25. Geman, S., Geman, D.: Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. IEEE Trans. Pattern Anal. Mach. Intell **6**(6), 721–741 (1984)
26. Salman, A., Imtiaz, A., Al-Madani, S.: Particle swarm optimization for task assignment problem. In: IASTED International Conference on Artificial Intelligence and Applications (2001)
27. Windisch, A., Wappler, S., Wegener, J.: Applying particle swarm optimization to software testing. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (London, England, 07–11 July 2007). GECCO '07, pp. 1121–1128. ACM, New York (2007)
28. Coming, D.S., Staadt, O.G.: Velocity-aligned discrete oriented polytopes for dynamic collision detection. IEEE Trans. Visual. Comput. Graph. **14**(1), 1–12 (2008)
29. Czarnecki, K., Helsen, S.: Classification of model transformation approaches. In: OOSPLA'03 Workshop on Generative

Techniques in the Context of Model-Driven Architecture. Anaheim, USA (2003)
30. Mens, T., Van Gorp, P.: A taxonomy of model transformation. in: Proceedings of International Workshop on Graph and Model Transformation (2005)
31. Andries, M., Engels, G., Habel, A., Hoffmann, B., Kreowski, H.-J., Kuske, S., Kuske, D., Plump, D., Schürr, A., Taentzer, G.: Graph Transformation for Specification and Programming. Technical Report 7/96, Universität Bremen. http://citeseer.nj.nec.com/article/andries96graph.html (1996)
32. Akehurst, D.H., Kent, S.: A Relational approach to defining transformations in a metamodel. In: Jézéquel, J.-M., Hussmann, H., Cook, S. (eds.): UML 2002—The Unified Modeling Language 5th International Conference, Dresden, Germany, September 30–October 4, 2002. Proceedings, LNCS vol. 2460, pp. 243–258 (2002)
33. Interactive Objects Software GmbH, Project Technology, Inc. MOF 2.0 Query/Views/Transformations RFP, Revised Submission. OMG Document ad/2003-08-11, http://www.omg.org/cgi-bin/doc?ad/2003-08-11 (2003)
34. Egyed, A.: Heterogeneous views integration and its automation. PhD Thesis, University of Southern California (2000)
35. Taentzer, G., Ehrig, K., Guerra, E., de Lara, J., Lengyel, L., Levendovsky, T., Prange, U., Varro, D., Varro-Gyapay, S.: Model transformation by graph transformation: a comparative study. In: Workshop on Model Transformations in Practice, September (2005)
36. Bunke, H.: Graph matching: theoretical foundations, algorithms, and applications. In: Proceedings of the Vision Interface 2000, Montreal/Canada, pp. 82–88 (2000)
37. Küster, J.M., Sendall, S., Wahler, M.: Comparing two model transformation approaches. In: Proceedings UML 2004 Workshop OCL and Model Driven Engineering, Lisbon, Portugal, October 12 (2004)
38. Heckel, R., Kuster, J.M., Taentzer, G.: Confluence of typed attributed graph transformation systems. In: Proceedings of ICGT'02, LNCS 2505, pp. 161–176, Springer, Heidelberg (2002)
39. Mens, T., Van Gorp, P., Varro, D., Karsai, G.: Applying a model transformation taxonomy to graph tansformation technology. In Karsai, G., Taentzer, G. (eds.) Proceedings of Graph and Model Transformation Workshop (to appear in ENTCS) (2005)
40. Mens, T., Van Gorp, P., Varro, D., Karsai, G.: Applying a model transformation taxonomy to graph transformation technology. In: Karsai, G., Taentzer, G. (eds.) Proceedings of Graph and Model Transformation Workshop, ENTCS (2005)
41. Visser, E. et al.: Program transformation with stratego/XT: rules, strategies, tools, and systems in StrategoXT-0.9. In: Lengauer, C. (ed.) Domain-Specific Program Generation, vol. 3016 of Lecture Notes in Computer Science, pp. 216–238. Spinger, Heidelberg (2004)
42. Vanhoof, B., Van Baelen, S., Joosen, W., Berbers, Y.: Traceability as input for model transformations. In: Proceedings of Traceability Workshop, European Conference in Model Driven Architecture (EC-MDA) (2007)
43. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Syst. J. (Special Issue on Model-Driven Softw. Dev. **45**(3), 621–645 (2006)
44. Wimmer, M., Strommer, M., Kargl, H., Kramler, G.: Towards model transformation generation by-example. In: Proceedings of HICSS-40 Hawaii International Conference on System Sciences. Hawaii, USA (2007)
45. Sun, Y., White, J., Gray, J.: Model Transformation by Demonstration. MoDELS09, Denver, CO, October 2009, pp. 712–726
46. Garcia-Magarino, I., Gomez-Sanz, J.J., Fuentes-Fernandez, R.: Model transformation by-example: an algorithm for generating many-to-many transformation rules in several model transformation languages. In: Proceedings of the 2nd International Conference on Theory and Practice of Model Transformations (ICMT'09), vol. 5563 of LNCS, pp. 52–66. Springer, Heidelberg (2009)
47. Fabro, M.D., Valduriez, P.: Towards the efficient development of model transformations using model weaving and matching transformations. Softw. Syst. Model. **8**(3), 305–324 (2009)
48. Jouault, F.: Loosely coupled traceability for atl. In: Proceedings of the European Conference on Model Driven Architecture (ECMDA) Workshop on Traceability (2005)
49. Galvão, I., Goknil, A.: Survey of Traceability Approaches in Model-Driven Engineering. In: EDOC'07, pp. 313–326 (2007)
50. Falleri, J.R., Huchard, M., Lafourcade, M.: Clémentine Nebut Meta-model Matching for Automatic Model Transformation Generation, ACM/IEEE 11th International Conference on Model Driven Engineering Languages and Systems (MODELS 2008) (2008)
51. Marvie, R.: A Transformation Composition Framework for Model Driven Engineering. Technical Report LIFL-2004-10, LIFL (2004)
52. DuanCheung, Y., Fu, X., Gu, Y.: A metamodel based model transformation approach. In: Proceedings of ACIS International Conference on Software Engineering Research, Management and Applications, pp. 184–191 (2005)
53. Giese, H., Wagner, R.: Incremental model synchronization with triple graph grammars. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) Models '06: Proceedings of the 9th International Conference on Model Driven Engineering Languages and Systems, vol. 4199 of LNCS, pp. 543–557. Springer, Heidelberg (2006)
54. Falleri, J.R., Huchard, M., Nebut, C.: Towards a traceability framework for model transformations in kermeta, HAL-CCSd-CNRS (2006)
55. Hearnden, D., Lawley, M., Raymond, K.: Incremental model transformation for the evolution of model-driven systems. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006, Genova, Italy, October 1–6, 2006, Proceedings, vol. 4199 of LNCS, pp. 321–335. Springer, Berlin (2006)
56. Krishnamurthy, R., Morgan, S.P., Zloof, M.M.: Query-by-example: operations on piecewise continuous data. In: Proceedings of 9th international conference on very large data bases, October 31–November 2, Florence, Italy, pp. 305–308 (1983)
57. Lechner, S., Schrefl, M.: Defining web schema transformers by example. In: Proceedings of DEXA'03. Springer, Heidelberg (2003)
58. Cypher, A. (ed.): Watch What I Do: Programming by Demonstration. The MIT Press, Cambridge (1993)
59. Repenning, A., Perrone, C.: Programming by example: programming by analogous examples. Comm. ACM **43**(3), 90–97 (2000)
60. Harman, M., Jones, B.F.: Search-based software engineering. Inf. Softw. Technol. **43**(14), 833–839 (2001)
61. Harman, M., Tratt, L.: Pareto optimal search based refactoring at the design level. In: GECCO'07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, pp. 1106–1113. ACM Press, New York (2007)
62. Harman, M.: The current state and future of search based software engineering. In: Proceedings of the 29th International Conference on Software Engineering (ICSE 2007), 20–26 May, Minneapolis, USA (2007)
63. Shousha, M., Briand, L., Labiche, Y.: A UML/SPT model analysis methodology for concurrent systems based on genetic algorithms. In: Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems MODELS08, pp. 475–489 (2008)
64. Bouktif, S., Sahraoui, H., Antoniol, G.: Simulated annealing for improving software quality prediction. In: GECCO2006: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, vol. 2, pp. 1893–1900 (2006)

65. Kleppe, A., Warmer, J., Bast, W.: MDA Explained. The Model Driven Architecture: Practice and Promise. Addison-Wesley, Reading (2003)
66. Seng, O., Stammel, J., Burkhart, D.: Search-based determination of refactorings for improving the class structure of object-oriented systems. In: GECCO '06: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, pp. 1909–1916, New York, NY, USA (2006)

Author Biographies

Marouane Kessentini received the MS degree in computer engineering from the University of Tunis (ENSI), Tunisia. He is currently a PhD student at the University of Montreal, Canada. His research interests include model transformation, software testing, design defects detection, and search-based software engineering. He is a student member of the IEEE and the IEEE Computer Society.

Houari Sahraoui is full professor at University of Montreal. Before joining the university, he held the position of lead researcher of the software engineering group at CRIM (Research center on computer science, Montreal). He holds an Engineering Diploma from the National Institute of computer science (1990), Algiers, and a PhD in Computer Science, Pierre & Marie Curie University LIP6, Paris, 1995. His research interests include the application of artificial intelligence techniques to software engineering, object-oriented metrics and quality, software visualization, and re-engineering. He has served as program committee member in several major conferences, as member of the editorial boards of many journals, and as organization member of different conferences and workshops.

Mounir Boukadoum received the MEE degree in electrical engineering from the Stevens Institute of Technology, Hoboken, NJ, in 1978, and the PhD degree in electrical engineering from the University of Houston, TX, in 1983. He was an Electronics Instructor at the Houston Community College, TX, for one year before joining the Université du Québec à Montréal (UQAM), in 1984, where he is now Full Professor at the Computer Science Department. He is also the current President of the Montreal Chapter of the IEEE Computational Intelligence Society. His research interests focus on the use of soft programming for data processing, pattern recognition, and instrument design.

Omar Ben Omar got his Bachelor degree in 2008 at the Université de Montréal in Computer Science. His topics of research range from model transformation by example to data and software visualization. He is pursuing his Master degree in software engineering at the Université de Montréal and will be graduating in 2010.