A

Multi-step learning and adaptive search for learning complex model transformations from examples

ISLEM BAKI, Université de Montréal
HOUARI SAHRAOUI, Université de Montréal

Model-driven engineering promotes models as main development artifacts. As several models may be manipulated during the software-development life cycle, model transformations ensure their consistency by automating model generation and update tasks. However, writing model transformations requires much knowledge and effort that detract from their benefits. To address this issue, Model Transformation by Example (MTBE) aims to learn transformation programs from source and target model pairs supplied as examples. In this paper, we tackle the fundamental issues that prevent the existing MTBE approaches from efficiently solving the problem of learning model transformations. We show that, when considering complex transformations, the search space is too large to be explored by naive search techniques. We propose an MTBE process to learn complex model transformations by considering three common requirements: element context and state dependencies, and complex value derivation. Our process relies on two strategies to reduce the size of the search space and to better explore it, namely, multi-step learning and adaptive search. We experimentally evaluate our approach on seven model transformation problems. The learned transformation programs are able to produce perfect target models in three transformation cases, whereas precision and recall values larger than 90% are recorded for the four remaining cases.

Categories and Subject Descriptors: D.2.1 [**Software Engineering**]: Requirements/Specifications; D.2.2 [**Software Engineering**]: Design Tools and Techniques; I.2.6 [**Artificial Intelligence**]: Learning—*Knowledge acquisition*; G.1.6 [**Numerical Analysis**]: Optimization

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Genetic programming, model-driven engineering, model transformation, model transformation by example, simulated annealing

ACM Reference Format:
Islem Baki and Houari Sahraoui, 2015. Multi-step learning and adaptive search for learning complex model transformations from examples. *ACM Trans. Softw. Eng. Methodol.* V, N, Article A (January YYYY), 36 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Over the last decade, model-driven engineering (MDE) has proven to be an efficient approach to develop software, as it alleviates overall development complexity, promotes communication, and increases productivity through reuse [Mohagheghi et al. 2013]. MDE advocates the use of models as first-class artifacts. It combines domain-specific modeling languages to capture specific aspects of the solution, and transformation engines and generators in order to move back and forth between models while ensuring their coherence, or to produce from these models low level artifacts such as source code, documentation, and test suites [Schmidt 2006]. Model transformation (MT) is

Authors' address: I. Baki and H. Sahraoui, Département d'informatique et de recherche opérationnelle, Université de Montréal
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© YYYY Copyright held by the owner/author(s). Publication rights licensed to ACM. 1049-331X/YYYY/01-ARTA $15.00
DOI: http://dx.doi.org/10.1145/0000000.0000000

at the very heart of the MDE paradigm. As such, several transformation languages and tools have been made available to MDE practitioners. Unfortunately, developing MTs remains a challenging task [Siikarla and Systa 2008] as it requires many types of knowledge, e.g., semantic equivalence between source and target domains, source and target metamodels, model transformation language, etc., that domain experts do not generally have. Moreover, developing large sets of transformations requires a lot of effort as candidate transformation rules need to be written and debugged iteratively in order to produce the correct output.

The automation of MT activities is one of the major obstacles that threaten the success of the MDE vision. Over the last decade, many researchers tackled this subject. A particularly interesting idea that has motivated many contributions is learning model transformations by example (MTBE) [Varró 2006]. MTBE approaches aim to derive transformation rules starting from a set of interrelated source and target model pairs. Over the past few years, different approaches have been proposed to achieve MT automation using examples with ad-hoc heuristics [Varró 2006; Wimmer et al. 2007], inductive logic programming [Balogh and Varró 2009], particle swarm optimization [Kessentini et al. 2008], relational concept analysis [Dolques et al. 2010], genetic programming [Faunes et al. 2012], etc.

Independently from their nature, these approaches search for a model transformation in a space whose boundaries are defined by a transformation language (the infinity of programs that could be written in this language) and the source and target metamodels (the infinity of programs that take, as input, instances of the source metamodel and produce, as output, instances of the target metamodel). Consequently, and although these approaches make important advances towards the resolution of the MTBE problem, they all face the search-space explosion issue when targeting complex transformations.

In this paper, we discuss the considerations that contribute to the search-space explosion when attempting to derive complex MTs from examples. We also propose a learning process that copes with huge search spaces, and that can thus target such transformations. Our process derives an MT from a set of interrelated source-target example pairs. First, transformation traces between the supplied source and target models (mappings) are refined and analyzed to build example pools. Then, genetic programming is used to derive for each pool, a set of rules that best transforms its examples. Finally, rule sets are merged into a single transformation program that is refined using a simulated annealing algorithm.

In our approach, we contend with huge search spaces using two strategies. First, the multi-phase aspect of our learning process allows us to reduce the size of the search space to be explored at each step. We thus learn the transformation program incrementally by addressing different requirements separately. Second, we take advantage of adaptive search techniques to adjust our search to the dynamics of the solution spaces and escape local optima. We validated our approach on seven transformation problems that exhibit diverse characteristics and different complexity levels. Our results show that common transformations are fully learned, and high precision and recall values can be achieved for the most complex ones.

The rest of this paper is organized as follows. Sections 2 and 3 introduce basic MTBE notions and motivate our work. In Section 4, we discuss the existing contributions in the area of MTBE. We briefly introduce our approach in Section 5 and then detail each phase of the learning process in Section 6. Section 7 describes the settings and the results of our validation. In this section, we also discuss the threats to the validity of our results. In Section 8, we discuss our findings. Finally, Section 9 concludes our paper and outlines future work.

2. BACKGROUND

2.1. Model Transformation

A model transformation can be defined as the automatic generation of a target model (TM) from a source model (SM) according to a transformation definition [Kleppe et al. 2003]. This definition is usually expressed as a set of transformation rules where each rule analyzes some aspects of the SM given as input, and synthesizes the corresponding TM as output. Both source and target models conform to their respective metamodels [Revault et al. 1995].

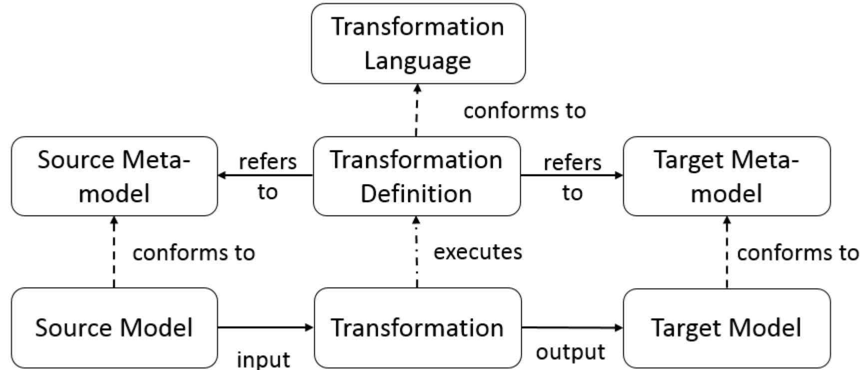

Fig. 1. Model transformation concepts.

The generation of the target model is said *automatic* because an MT is defined at the metamodel level (see Figure 1). If SM and TM have the same metamodel, the transformation of SM into TM is said endogenous [Czarnecki and Helsen 2006], whereas if SM and TM conform to different metamodels SMM and TMM, the transformation is said exogenous. In this paper, we are interested in exogenous model transformations.

2.2. Transformation Language

The traditional approach toward implementing an MT is to specify the transformation rules using an executable, preferably declarative, transformation language. In this contribution, we use Jess [Hill 2003] (Java Expert Shell System) as a transformation language and engine. Jess is a Java rule engine that holds knowledge organized as a collection of facts. It is then possible to reason on this knowledge base using declarative rules. Rules are continuously applied on the data through pattern matching techniques (Jess uses an enhanced version of the Rete algorithm [Forgy 1982]). We decided to use this generic, simple, and declarative language to separate, in a first phase of this research project, the intrinsic complexity of example-based transformation learning from the accidental complexity of using specific tools (e.g., ATL with Ecore) with interoperability and dependency concerns.

Jess facts are very similar to Java objects. Each fact has a name and a list of attributes called *slots*. Facts can be defined using templates, which are in turn similar to the concept of classes in Java. Jess rules react to the changes that occur in the collection of facts. Each rule is composed of two parts: conditions expressed in the left-hand side (LHS) of the rule, and actions defined in its right-hand side (RHS). When the LHS of a rule is satisfied, its RHS is executed.

In the context of MT, we represent source and target models as facts. Each model conforms to its metamodel, which is supplied as a set of Jess fact templates. The transformation program consists in a set of Jess rules. The LHS patterns of the rules are matched with elements from the SM (or the already created elements of the TM). The RHS part of the rules allows us to generate fragments of the TM by asserting new facts. Listing 1 illustrates a single rule program that transforms a single-element source model into the corresponding target model. When this trivial transformation is executed, the engine will look for facts that satisfy the condition specified in the rule *Source2Target*, i.e., the presence of an element *sourceElement*. For each match in the source model, the rule asserts a new target model element *targetElement*, whose name attribute is set with the name of the matched source element (the assignment is done by means of the same variable $?c00$ in the rule's LHS and RHS).

Listing 1. A simple example of source and target metamodels, a source model to transform, a transformation rule, and a produced target model, expressed in the Jess syntax.

```
;source metamodel
(deftemplate sourceElement (slot name))

;target metamodel
(deftemplate targetElement (slot name))

;source model
(assert (sourceElement (name SC1)))

;rule
(defrule Source2Target
(sourceElement(name ?c00))
=>
(assert (targetElement(name ?c00))))

;output
(targetElement (name SC1))
```

2.3. Model Transformation by Example

As mentioned in Section 1, writing model transformations may be a challenging and time-consuming task. Unlike the example given above, real-world MT may consist of dozens of complex interdependent rules. MTBE is an elegant solution to this problem. The goal of MTBE is to learn a transformation program from examples. Each example is a pair consisting in a source model and the corresponding target model. Instead of writing the transformation, the user provides such pairs to illustrate its behavior. The system then learns the sought transformation automatically. The idea behind MTBE is that experts are much more comfortable in defining real source-target examples, expressed in a concrete syntax (modeling notation), rather than specifying transformation rules in abstract syntax (computer representation) [Egyed 2002].

In many MTBE papers, the source and target models supplied within each example pair are interrelated. Thus, a transformation example consists usually in a triple (SM,TM,TT), where SM denotes the source model, TM the target model and TT are the transformation traces, also called mappings. Transformation traces are many-to-many links that associate a group of n source elements to a group of m target elements. Figure 2 shows an example pair with three identified traces (not all the traces are displayed). For each trace, the target fragment (bottom) has been determined as corresponding to the source fragment (top). Transformation mappings can be identified by an expert during the design process or recovered (semi-)automatically using,

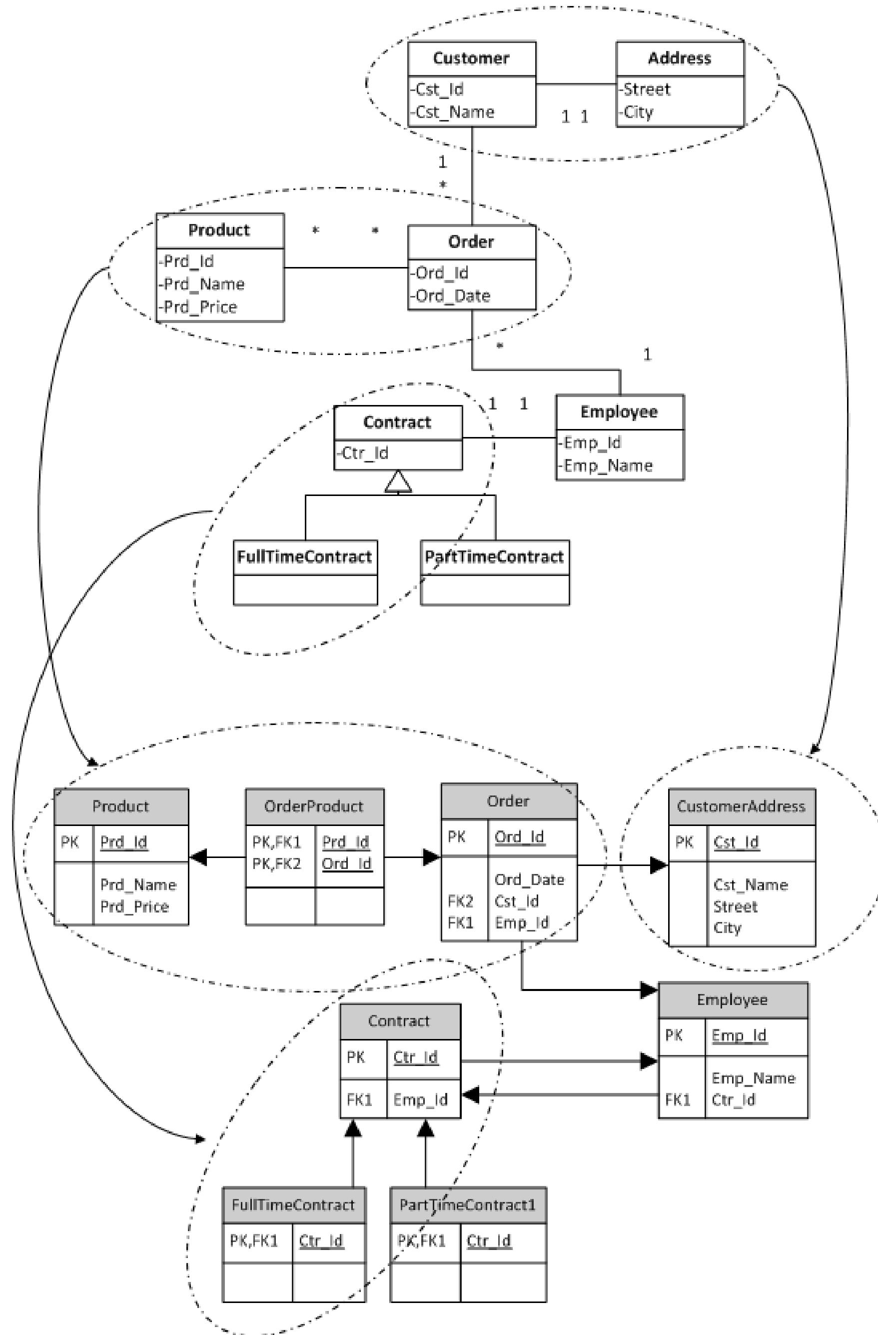

Fig. 2. A source-target example pair with three identified traces.

for instance, approaches such as the ones proposed by Saada et al. [2013] or Grammel et al. [2012].

3. PROBLEM STATEMENT

Despite important advances in MTBE, state-of-the-art contributions are unable to handle many complex, nonetheless common, model transformation scenarios. To illustrate the most common characteristics of what we consider in this paper as complex transformation cases, let us consider a motivating example of the well-known problem of transforming UML class diagrams into relational schemas (CL2RE).

An ideal simple situation when transforming class diagrams to relational schemas is when the transformation rules map a single source element of a given type to a single target element of another type. For example, a basic rule consists of creating a table for each class, regardless of the other elements related to the class, and simply assigning the class name to the table name. The search space in such a situation is relatively small and is defined by all the possible sets of pairs <source-element-type, target-element-type>, where in each set, all the source element types are mapped to target element types. Unfortunately, in a more complete specification, many considerations will increase this search space dramatically. Although many requirements can contribute to the search-space explosion, the three most recurrent ones that we identified are (1) dependency on the element state, (2) dependency on the element context, and (3) complex value derivation of target element attributes.

3.1. Dependency on the Element State

In many transformation problems, the same element type can be transformed differently depending on its state, i.e., the values of the element attributes. This generally holds for enumeration types, where the set of values that the attribute can take is finite. For example, in CL2RE, consider the case of two classes linked by an association. The association is transformed differently according to its state, which is defined by its cardinalities (one-to-one, one-to-many, many-to-many).

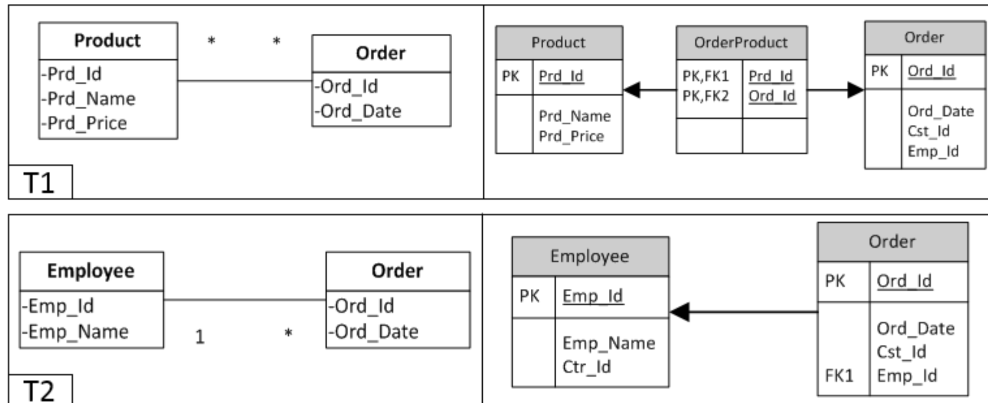

Fig. 3. Similar source fragments, extracted from the example of Figure 2, having elements with different states.

As illustrated in trace $T1$ of Figure 3, when two classes ($Product$ and $Order$) are linked by a many-to-many association, a table is created ($OrderProduct$), and keys link this table to the ones corresponding to the association source and target classes. However, if the cardinalities are one-to-many, as in the association between $Employee$ and $Order$ in $T2$, no specific table is created, and the class-corresponding tables are

linked by a key. Thus, when transforming an element, the search space is augmented by all the possible combinations of all the possible values of its attributes.

3.2. Dependency on the Element Context

In addition to the dependency on the element state, many transformation scenarios require to consider the context in which a source element appears, i.e., the relationships to the other elements, in order to transform it. To illustrate this characteristic on the CL2RE transformation problem, let us consider this time the case of a one-to-one association. It is a common practice to merge classes related through a "1-1" association into one table when one of these classes is not involved in any other association or inheritance relationship. If not, a table is created for each class. Both situations are illustrated by the two traces presented in Figure 4. In $T3$, the class $Address$ does not have links other than the one to $Customer$, and then a single table is created. Conversely, in $T4$, both classes $Employee$ and $Contract$ have at least one other link (see the initial source model in Figure 2). Therefore, two tables must be created and linked with foreign keys.

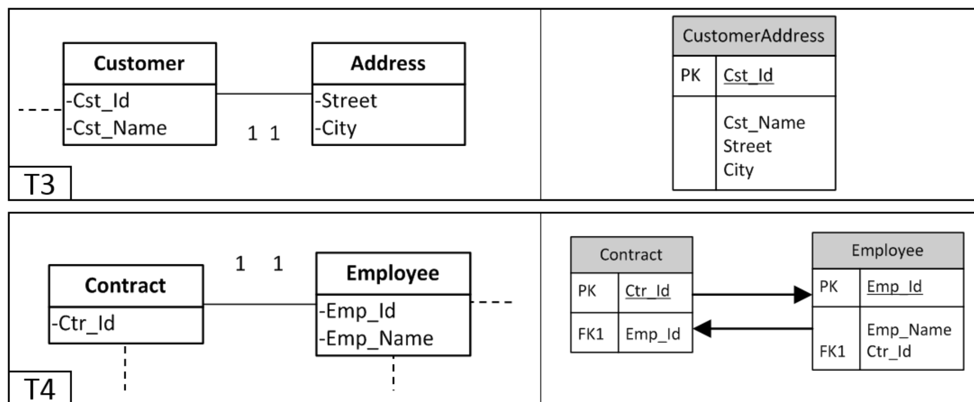

Fig. 4. Similar source fragments, extracted from the example of Figure 2, having elements with different contexts.

This situation illustrates the fact that, for some transformation problems, the search space increases by all the possible contexts of the element type to transform, i.e., existence of different direct or indirect links to the other elements of the transformed model. In such a case, a wider context must be explored to transform the elements, which requires to learn complex rules with sophisticated conditions.

3.3. Complex Value Derivation

While both previous requirements tackle sophisticated conditions that could be expressed in a transformation rule, considerations have also to be directed toward the action part of the rules. Indeed, although in many cases, the attributes of the target elements are set with the attributes' values of the source elements, many transformations require the generation of values using combinations of arithmetic, string and conversion operators as well as constant generation and aggregations (e.g., count, average, minimum). For example, in $T3$ of Figure 4, when a unique table is generated for two classes, as it is the case for the association that links the $Customer$ and $Address$

classes, the name of the corresponding table in the target model can be the concatenation of the two-class names. Similarly, the name of a table key could be the concatenation of a generated constant with a class attribute name or with the class name.

Other transformation scenarios that require complex value derivations are those whose target models are graphical representations [Pfister et al. 2012; Buckl et al. 2007]. In these transformations, numeric operators and functions are often required to perform the transformation, particularly for coordinate and size attributes derivation. Thus, in many transformation scenarios, there is a large number of operator-attribute combinations, which multiplies the possibilities in the search space.

These three requirements, namely, dependency on the element state, dependency on the element context, and complex value derivation are quite common (see the transformation problems described in Section 7.2). Therefore, they must be considered when trying to automate real-world model transformations. However, two considerations arise when using search based techniques to learn such transformation problems. The first one is the search-space size explosion that ensues. The expression of complex transformation rules with more sophisticated LHS conditions, and especially, complex RHS attribute derivations, increases exponentially the number of transformation possibilities. Second, many search-based techniques such as evolutionary algorithms may be subject to local optimum convergence, known as the premature convergence problem [Andre et al. 2001]. A fully automated MT learning process must, therefore, implement mechanisms that would permit to circumvent such a phenomenon.

4. RELATED WORK

Two main research axes investigate learning model transformations from examples: example-based (MTBE) and demonstration-based (MTBD) approaches. MTBE approaches aim to derive transformation programs by exploiting example pairs of source and target models, supplemented in most contributions with fine grained transformation traces. Demonstration-based approaches (MTBD) rely on recording and analyzing user editing actions when performing a transformation to learn its rules [Sun et al. 2009]. As pointed out by Kappel et al. [2012], existing example-based approaches deal exclusively with exogenous model transformations whereas demonstration-based approaches focus primarily on endogenous MT. A summary of the characteristics of MTBE and MTBD contributions is provided in Table I.

An alternative approach to automated transformation synthesis (MTBE and MTBD) is proposed by Avazpour et al. [2015]. The approach, CONVErT, uses visualization to allow a domain expert to interactively specify mappings between source and target elements in example model pairs. It also relies on recommendations to derive a transformation specification and to convert it into transformation rules.

4.1. Example-based Model Transformations

Varró [2006] propose a semi-automatic graph-based approach to derive transformation rules using interrelated source and target examples. The derivation is interactive and iterative and allows to derive 1-1 transformation rules. The extension by Balogh and Varró [2009] derives n-m rules using Inductive Logic Programming (ILP). Wimmer et al. [2007] propose a similar approach to derive 1-1 transformation rules in ATL. Their contribution differs from Varró and Balogh with respect to transformation mappings, which are defined in a concrete rather than an abstract syntax. Wimmer et al.'s contribution is improved in [Strommer et al. 2007; Strommer and Wimmer 2008] to also handle n-m rules with the mention of a basic string manipulation operator (lowercase). Similarly, García-Magariño et al. [2009] propose an ad-hoc algorithm to derive ATL n-m rules using example pairs with their mappings.

Table I. Features of current MTBE approches.

Approach	Algorithm	Input	Output	n-m rules?	Con-trol?	Con-text?	State?	Complex derivations?
MTBE approaches								
Varró [2006]	ad-hoc heuristic	Examples & traces	Rules	1-1	No	No	No	No
Wimmer et al. [2007]	ad-hoc heuristic	Examples & traces	Rules	1-1	No	No	No	No
Strommer et al. [2007; 2008]	Pattern matching	Examples & traces	Rules	n-m	No	No	No	String operator
Kassentini et al. [2008; 2012]	PSO/PSO-SA	Examples & traces	Target model	–	No	No	No	No
Balogh and Varró [2009]	ILP	Examples & traces	Rules	n-m	No	No	No	No
García-Magariño et al. [2009]	ad-hoc algorithm	Examples & traces	Rules	n-m	No	No	No	No
Kassentini et al. [2010]	PSO-SA	Examples only	Rules	1-m	No	No	No	No
Deloques et al. [2010]	RCA	Examples & traces	Pat-terns	–	No	No	No	No
Saada et al. [2012a; 2012b]	RCA	Examples & traces	Rules	1-m	No	No	No	No
Faunes et al. [2012; 2013]	GP	Examples only	Rules	n-m	No	No	No	No
Baki et al. [2014]	GP	Examples & traces	Rules	n-m	Yes	Yes	No	No
MTBD approaches								
Brosch et al. [2009a; 2009b]	Pattern matching	User actions	Rules	–	–	No	No	Added manually
Sun et al. [2009; 2011]	Pattern matching	User actions	Rules	–	–	No	No	Demonstrated by the user
Langer et al. [2010]	Pattern matching	User actions	Rules	n-m	Yes	No	No	Added manually

Kessentini et al. [2008; 2012] use analogy to perform transformations. Unlike the above-mentioned contributions, they do not produce transformation rules but derive the corresponding target model by considering MT as an optimization problem. The problem is addressed using particle swarm optimization (PSO), followed by a combination of PSO and simulated annealing (SA). The approach is taken a step further by producing transformation rules while overcoming the need for transformation mappings [Kessentini et al. 2010]. Another contribution, not initially intended for transformation rules, is proposed by Dolques et al. [2010]. The approach is based on relational concept analysis to learn transformation patterns. This approach is extended by Saada et al. [2012a; 2012b], in which the patterns are analyzed to select subsets, which are mapped to executable Jess rules. The most-recent contribution to MTBE is the one of Faunes et al. [2012; 2013] in which genetic programming (GP) is used to learn n-m transformation rules starting from source and target examples without the transformation traces. The approach is enhanced by Baki et al. [2014] to learn the rule execution control.

Except for the work of Strommer and colleagues [Strommer et al. 2007; Strommer and Wimmer 2008], where a single string operator is considered, all current MTBE contributions do not support complex target value derivations that involve string manipulations or arithmetic operations. Moreover, to our best knowledge, none of the contributions above can derive rules that require testing string attribute values or exploring a global context except for Baki et al. [2014] where the result returned by a user-defined query can be used in rule conditions. We also noticed that most MTBE contributions illustrate their respective approaches using simplified transformation

problems or toy examples without showing the complexity and correctness of the rules that can be learned.

4.2. Demonstration-based Model Transformations

Brosch et al. [2009a; 2009b] propose an approach to alleviate the complexity of developing model refactoring operations. Their contribution derives semi-automatically endogenous MT specifications by analyzing user editing actions when refactoring models. The derivation process consists of two phases. During the first phase all atomic operations performed by the user are collected by performing a state-based comparison between the initial and final models. A unique ID is automatically assigned to all elements of the initial model to allow a precise detection of all atomic changes. In the second phase, the collected operations are saved in a *diff* model and a set of pre- and post-conditions of the refactoring operations is proposed to the user for manual refinement. The refinement step can also include the definition of complex value derivations that should be used during the transformation. Finally, the *diff* model and the revised pre- and post-conditions are used for the transformation sought.

Langer et al. [2010] extend Brosch et al.'s contributions to support exogenous model transformations. The learning process consists of three steps. As in the previous contributions, the user is first invited to create the source and corresponding target elements in his favorite modeling environment. The user can select a context prior to certain editing actions to express rule dependencies. During the second phase, the transformation scenario is generalized in a set of transformation templates than can be manually refined. A final phase produces the sought transformation rules from the refined templates using a higher-order transformation.

Sun et al. [Sun et al. 2009; Sun et al. 2011] propose a similar approach for deriving endogenous MTs. Unlike, the contributions from Brosch et al. and Langer et al., an extension is added to the modeling environment in order to monitor the user editing actions. The recorded operations are then analyzed to remove incoherent and unnecessary ones. An inference engine is subsequently used to express user intentions as reusable transformation patterns. Similarly to Brosch et al.'s approach, certain complex value derivation are supported. These derivations must be demonstrated by the user rather than added manually during a distinct refinement phase.

5. APPROACH OVERVIEW

In this paper, we present a process to learn complex exogenous model transformations. Our process takes as input source and target metamodels as well as a set of source and target model pairs supplemented with their traces. It produces as output a declarative and executable transformation program. For the sake of simplicity, we describe our approach using a single example pair, but we also explain how multiple example pairs are handled at each step of our process in Section 6.

As illustrated in Section 3, learning a complex model transformation can involve exploring a very large search space that cannot be achieved in a reasonable time when using standard meta-heuristics. We believe that many strategies have to be combined in order to reduce the size of the search space and to ease the convergence towards an acceptable transformation solution. Our approach is based on two such strategies: multi-phase learning rather than a single phase one, and adaptive-search techniques rather than classical search techniques.

5.1. Multi-Phase Learning

As mentioned earlier, dependencies on context and state are two major reasons of the search-space explosion that makes MT learning challenging. To deal with these considerations, the sought transformation program is derived incrementally in three phases,

as illustrated in Figure 5. We leverage transformation traces that are split into several pools in order to reduce the size of the search space to be explored in each phase.

Fig. 5. The process of learning a model transformation.

Analyzing transformation traces. The goal of the first phase is to analyze the mappings between source and target fragments of the input example pair. During the analysis, the traces are split into pools (Figure 5 - 1), so that each pool contains only source fragments of the same type (cf. Section 6.1) that are transformed identically. When two source fragments of the same type are transformed into two similar target fragments, this means that the source fragments have contexts and states that trigger the same transformation. Conversely, source fragments of the same type mapped to different target fragments implies that the contexts and/or states are important in choosing the transformation alternatives.

Learning transformation rules. Now that in each pool, traces of the same type are mapped to similar target fragments, it is possible to learn the transformation rules of each pool (in parallel) without exploring the contexts and states, which reduces considerably the size of the search space. The search of the best transformation program of each pool is performed by a genetic programming algorithm [Koza 1992], to produce a transformation program (a set of rules) per pool (Figure 5 - 2). Each transformation program is then automatically processed to eliminate incorrect and redundant rules before starting the final phase.

Refining the transformation program. In the third phase, the transformation programs of all the pools are merged into a single program that is used as the initial solution to be refined by a simulated annealing algorithm [Aarts and Korst 1988] (Figure 5 - 3). Some of the merged rules, which were learned in different pools, will search for the same source fragment types but produce different target fragments because the contexts and states are not tested. To solve this issue, the refinement algorithm

must explore the possibilities of enriching rules with more sophisticated conditions that may consider a wider context, test source element states, check the absence of constructs (negative conditions), or add implicit control through target-pattern testing in the rules.

5.2. Adaptive Search

The proposed 3-steps process reduces the search space by learning basic rules in the second phase and by sophisticating/specializing these rules in the final phase. Still, the third space-explosion consideration, i.e., complex value derivation is not yet addressed. Whereas the LHS of the rules (the conditions) are incrementally built by the second and third phases of the learning process, their RHS (actions) are exclusively derived during the second phase carried out by the genetic programming algorithm.

Genetic programming, as for many population-based heuristic algorithms, is subject to the premature convergence problem [Andre et al. 2001], when parents near local optima breed many similar offsprings. As such, it is necessary to implement mechanisms to ensure that GP dynamically adapts to the search progression in order to achieve a good trade-off between performance (optimal solution found) and exploration power. For our problem, we propose to use two techniques to increase convergence towards a good transformation solution: adaptive mutations and memory-based mutations.

6. APPROACH

6.1. Analyzing Transformation Traces

Given a source-target model example pair (SM, TM), we define a transformation trace, as a set of mappings $TR = \{T_1, T_2, ..., T_m\}$ where each mapping is a pair of source and target fragments $T_i = (SF_i, TF_i)$, $i \in [1..m]$. A source (resp. target) fragment is a set of interconnected source (resp. target) elements. Each source fragment contains one main element that is the element being transformed. More formally, $T_i = (\{se_{i0}, ..., se_{ip}\}, \{te_{i0}, ..., te_{iq}\})\ p > 0, q > 0$ where se_{i0} is the main element of SF_i. We then say that a trace is of type t, denoted as T_i^t, if its main source element is of type $t \in SMM$.

Since we use a search-based algorithm, we do not require traces to be precise. For instance, the target fragment mapped to a 1-n association can include the referenced table in addition to the referencing table with its foreign key. We hold that, in practice, experts may identify fragments rather than specific target elements. However, we do assume that mappings are consistent.

After gathering the set of traces associated with a source-target example pair, each trace is automatically completed by supplying its source fragment with elements referenced by its main element, but not present in the trace (according to the minimal cardinalities expressed in the source metamodel). Elements referenced by the main element are likely to be used to transform it. In our example, if the expert defines a mapping between a 1-1 association and a table with a source fragment that contains only the association itself, we will complete this fragment with the two classes that are referenced by the association.

Transformation traces are then analyzed and conflicting traces are separated into distinct pools. We say that two traces are conflicting if their source fragments are of the same type but are transformed differently. Such cases are identified by comparing, on the one hand, the structural similarity of their respective target fragments, and on the other, the lexical similarity between string source and target attribute values in each trace. This latter operation is performed to identify cases where the source fragments of two traces are mapped to two structurally-similar target fragments but where source attributes are mapped to different target attributes in each trace.

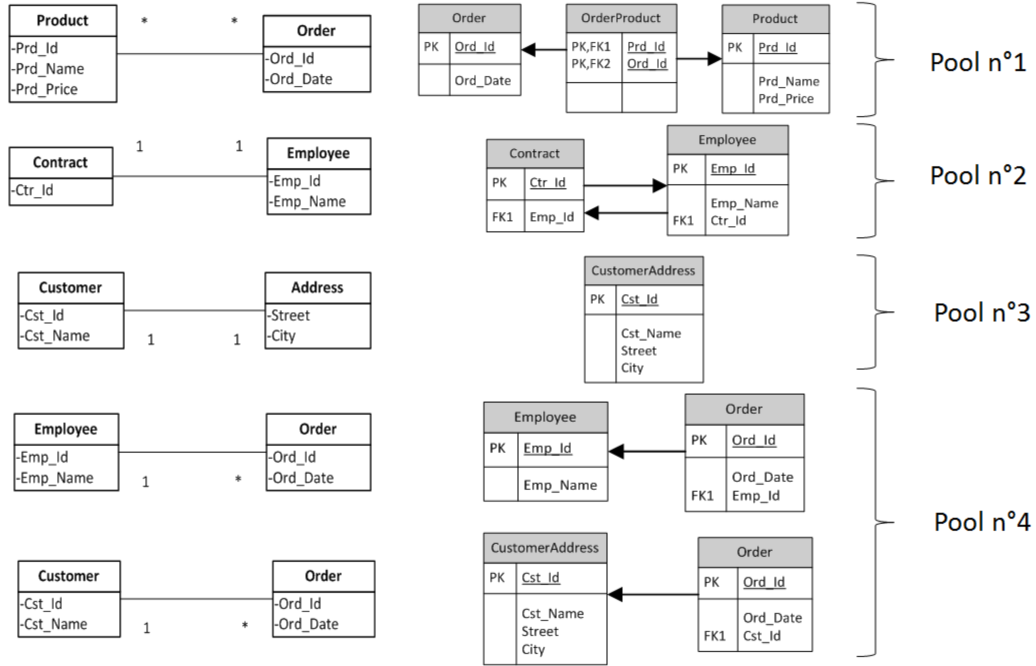

Fig. 6. Identification of conflicting traces.

For instance, consider Figure 6 that depict five traces of type association. These traces are separated into four different pools because they are all transformed differently except for the two 1-$*$ associations that will be grouped into pool n°4. The reason we separate such traces across multiple pools is that since we neither explore the context nor test the state of an element, learning a transformation in one pool using traces with similar source but different target fragments will lead to contradictory rules that have the same conditions and different actions.

Depending on the number of conflicts in each type of trace, n pools may be built (see Figure 5 - 2). Each pool P_k, $k \in [1..n]$, contains a set of traces $TR_k = \{T_{k_1}, T_{k_2}, ..., T_{k_{mk}}\}$. Traces of a given type that are all transformed in the same manner, are just duplicated across all the pools. In Figure 6 all the traces of types class and inheritance are simply copied into each pool. Thus, we have $TR = \bigcup_{k=1}^{n} TR_k$ and $\bigcap_{k=1}^{n} TR_k \neq \emptyset$.

Finally, for each pool (SM_k,TM_k), the initial model example pair is modified to remove elements that are not in the pool traces. For instance, given that association traces are split as illustrated in Figure 6, SM_1 the source model of pool n°1 is presented in Figure 7.

Multiple example pairs are handled similarly to a single example pair. The set of traces analyzed is the union of the traces of each example pair. Traces are then analyzed without considering their origin, and similar traces of different example pairs are grouped in the same pool.

6.2. Learning Transformation Rules using GP

The goal of this phase is to derive a transformation program TP_k for each pool P_k such that $TP_k(SM_k) = (TM_k)$. To this end, we use genetic programming. In the following

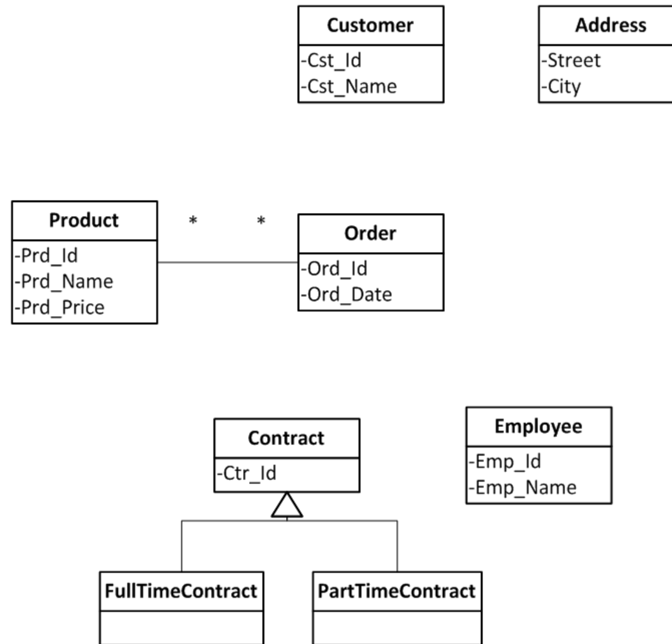

Fig. 7. Source model of pool n°1 of Figure 6.

subsections, we first introduce GP. Then, we detail how we adapt GP to our learning phase.

6.2.1. Genetic Programming. GP is a variant of genetic algorithms whose goal is to automatically creates computer programs to solve problems [Poli et al. 2008]. It evolves a population of programs into new, hopefully better ones, through an iterative process inspired by Darwinian evolution.

Figure 8 shows the typical steps of genetic programming. The algorithm starts with an initial population of programs randomly generated. Each program is represented with a tree-like structure having functions as internal nodes and terminals (variables and constants) as leafs. The set of functions and terminals is problem dependent, and it defines the search space to be explored. In the next step, each program is evaluated using a fitness function in order to assess its ability to solve the given problem. The fitness function is the main mechanism that communicates to the computer the description of the task to be accomplished, i.e., direct the search.

Then, new generations are iteratively derived from existing ones by applying genetic operators, namely, reproduction, crossovers, and mutations. Indeed, at each iteration, programs are probabilistically selected to reproduce and survive based on their fitness (ability to solve the given problem). Reproduction consists in simply copying the selected individuals into the new population. The crossover operator mimics the sexual recombination of organisms. It recombines two individuals by exchanging two randomly selected sub-trees from each parent. Finally, mutations allow to introduce new genetic material into the population by randomly altering a terminal or a sub-tree of the selected individual. The execution of the genetic program is carried on until the stop criterion is met, and the best program is then returned.

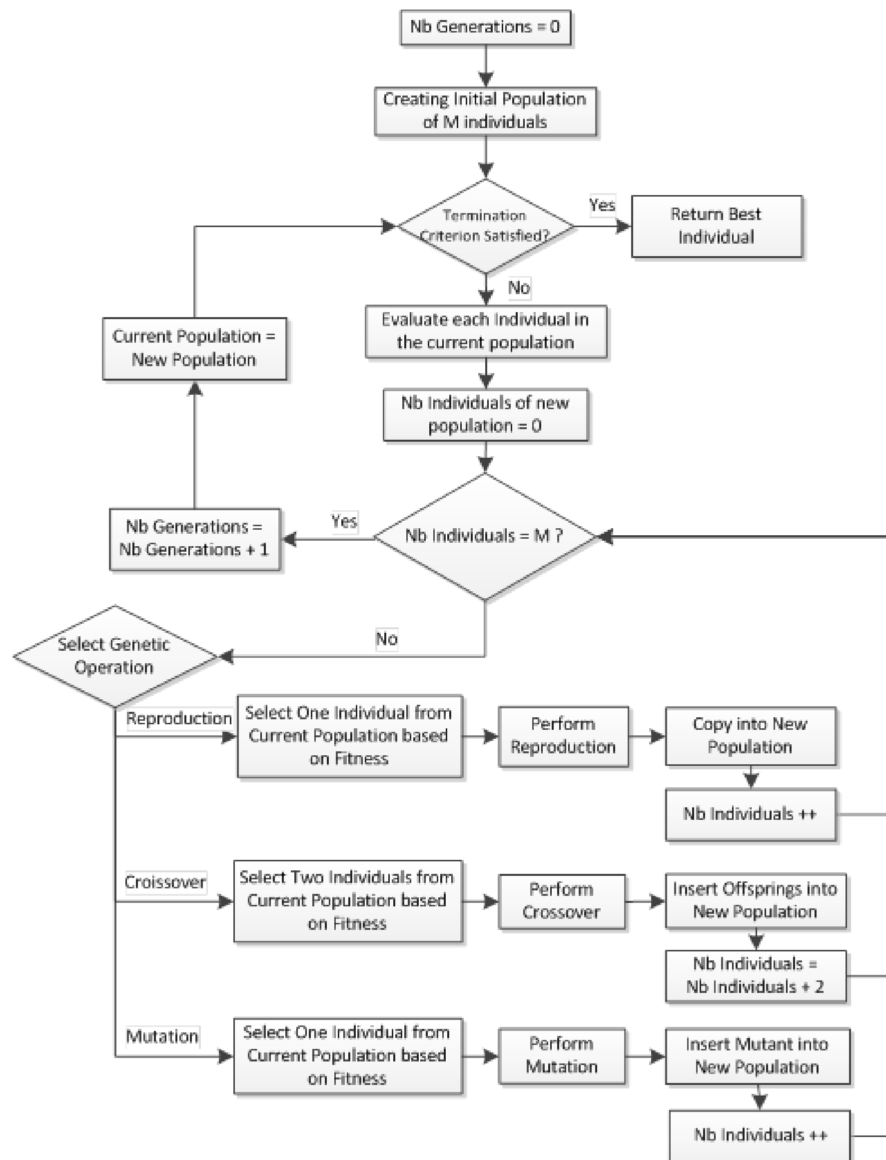

Fig. 8. Flowchart of genetic programming.

The execution of a GP is controlled by a set of parameters such as the size of the population, the probability of selecting each genetic operator, and the stop criterion, e.g., a maximum number of generations or a given result that must be achieved by the best program.

In the subsequent paragraphs, we explain how GP is adapted to derive a transformation program in each pool. We discuss, in particular, how rule sets are encoded, evaluated and derived.

6.2.2. Rule Set Encoding. For a given pool P_k, we define a candidate transformation program TP_k as a set of rule groups $TP_k = \{RG_k^t, t \in SMM\}$ where each rule group

$RG_k^t = \{R_{k_1}^t, R_{k_2}^t, ..., R_{k_x}^t\}$ transforms traces of type $t \in SMM$. Each transformation rule is encoded as a couple $R_{k_i}^t =< LHS_{k_i}^t, RHS_{k_i}^t >$, $i \in [1..x]$. $LHS_{k_i}^t$ is a pattern to search for in the source model SM_k and $RHS_{k_i}^t$ is the pattern to instantiate in the target model. For example, Listing 2 shows a rule that transforms an inheritance relationship into a foreign key. $LHS_{k_i}^t$ determines the conditions to be satisfied by SM_k for the rule to be fired. It is composed of one or more interconnected bricks (see Listing2). A brick is a set of interconnected source model elements with a main element. A brick is self-contained, i.e., respects the minimum cardinality of its main element as defined in the metamodel. Thus, a brick of type class would contain a single element whereas an inheritance brick contains three elements, i.e., the inheritance, the subclass, and the superclass as illustrated in Listing 2 (three first lines after the rule name). We say that a brick is of type t if its main source element is of type $t \in SMM$. For instance, the LHS of the rule depicted in Listing 2 contains two bricks, one of type inheritance and the other of type attribute (two lines before the symbol "=>").

$LHS_{k_i}^t$ can contain an arbitrary number of bricks. However, it must contain at least one brick of type t, the type of traces transformed by the rule group to which it belongs. Moreover, $LHS_{k_i}^t$ cannot contain an element of type $t' \in SMM$ if t' does not appear in source fragments of the set of traces of type t in the pool k. Finally, the bricks must be interconnected to be matched by concrete model fragments during the transformation execution. The interconnection is made through a common element. For example, in Listing 2, the superclass $?c10$ of the inheritance brick is the class $?c10$ to which belong the attribute $?a20$ in the second brick. Note that in this phase, LHS does not include negative conditions, target model bricks, navigation primitives or attribute values testing.

Listing 2. Example of a rule.

```
(defrule Inheritence_to_ForeignKey
(inheritance(class ?c00)(superclass ?c10))
(class(name ?c00))
(class(name ?c10))

(attribute(name ?a20)(class ?c10)(unique ?a22))
(class(name ?c10))
=>
(assert (fk(column ?a20)(table ?c00)(fktable ?c10))))
```

The RHS of the rule contains the target elements to be created if the rule is fired. As for $LHS_{k_i}^t$, $RHS_{k_i}^t$ cannot contain a target element of type $t' \in TMM$ if t' does not appear in the target fragments of the set of traces of type t in the pool k. The attributes of target elements can be initialized with three types of terminals: attributes from the LHS, constants defined by enumeration types of TMM, or functions that can take as parameters the two previous types of terminals. When performing the binding of a target attribute only terminals of a compatible type are considered. Furthermore, for string target attributes, a vector of lexical similarities built from the analysis of the pool traces is used to reduce the number of candidate terminals and introduce more effectively functions as they become the only binding candidates of a target attribute when its lexical similarity vector reveals no full match amongst constants and LHS variables.

The derivation of a transformation program using GP requires the creation of an initial population of programs. The size of the initial (and subsequent) population is determined by the $generation_size$ parameter. Each program of the initial population has as many rule groups as there are types $t \in SMM$. The number of rules to be created for each group is chosen randomly from a given interval. Each rule is then created

by generating a random number of bricks for its LHS and a random number of target elements for its RHS. Random source and target elements of each rule are selected amongst elements present in the set of traces transformed by the rule owning group. Next, each target element attribute is bound to a random compatible terminal (LHS variable, constant or a computed function). The generation algorithm ensures that the obtained rules are syntactically (w.r.t Jess) and semantically (w.r.t the metamodels) correct.

6.2.3. Rule Set Evaluation. Each candidate transformation program TP_{ki}, where $i \in [1..generation_size]$, is run with SM_k as input. The fitness of TP_{ki} is then evaluated by comparing its output model TM_{ki} to the expected one TM_k. Programs that produce target models more similar to the expected ones will have a higher fitness and are thus favored during the breeding of the next generation.

The similarity between the produced models and the expected ones is computed by comparing the elements of each model. The comparison considers the type of elements produced in order to avoid a bias towards frequent elements of a certain type. For T_{TM_k} the set of target types defined by the target metamodel, the fitness function is defined as the average of the result obtained by the comparison of elements of type $t \in TMM$ (Equation 1).

$$f_i(TM_k, TM_{ki}) = \sum_{t \in T_{TM_k}} \frac{f_i^t(TM_k, TM_{ki})}{|T_{TM_k}|} \quad (1)$$

$$f_i^t(TM_k, TM_{ki}) = \alpha fm^t + \beta pm^t, \textit{ with } \alpha + \beta = 1 \quad (2)$$

Comparing TM_k and TM_{ki} for a type t is performed by first identifying produced elements that fully match the expected ones. Two elements that fully match have the exact same attribute values and references. Elements are removed when they match so that each produced element is not matched to more than one expected element and vice versa. For the remaining elements, partial matches are identified for same-type elements that were not selected in the first phase. Partial matches are considered when evaluating programs in order to favor partially correct rules over completely incorrect ones. $f^t(TM_k, TM_{ki})$ is then computed according to Equation 2 where fm^t (resp. pm^t) are the percentages of full (resp. partial) matches. The weights α and β were empirically set to respectively 0.8 and 0.2.

When the learning process involves multiple example pairs, each candidate transformation program TP_{ki} is run once for each example pair. The fitness of the candidate is then computed as the average of the fitness score obtained at each run.

Each rule set is composed of groups where a group RG_k^t contains rules that transform traces of type $t \in SMM$. An alternative evaluation to the one described previously is to evaluate each group RG_k^t on traces of its type rather than evaluated all the groups on the source model SM_k. However, as we do not require the traces to be precise, this will result in several rules creating the same element. For instance, the target fragment of an association may contain two tables and their respective foreign keys. Evaluating rules of such groups on the corresponding traces would drive those rules to produce not only the expected foreign-key elements but the tables as well, whereas these table elements were likely also derived by the rule group responsible for transforming the class type.

6.2.4. Rule Set Derivation. After evaluating each candidate program of the current generation, a new generation of programs is derived. First, an elitist operator is used to insert the x fittest individuals into the new generation. Then and until the new population is complete, two individuals are selected from the previous generation using a

binary selection tournament, and are subject to a crossover with a given probability in order to produce two child candidates. These are, in turn, possibly mutated before joining the new population.

Crossover: the crossover that we defined operates on the group level with a one-cut-point strategy. It consists in producing two child rules set from two existing ones by selecting a cut-point and then exchanging the groups of each side. For instance, consider the two transformation program candidates TP_{k1}=$\{RG_{k1}^{t_1}, RG_{k1}^{t_2}, RG_{k1}^{t_3}, RG_{k1}^{t_4}\}$ and TP_{k2}=$\{RG_{k2}^{t_1}, RG_{k2}^{t_2}, RG_{k2}^{t_3}, RG_{k2}^{t_4}\}$. As groups must be maintained in the children (i.e. each candidate rule set must transform all source fragment types), the same crossover point is used for both parents, let us say 3. The offsprings obtained are the candidates O_{k1}=$\{RG_{k1}^{t_1}, RG_{k1}^{t_2}, RG_{k1}^{t_3}, RG_{k2}^{t_4}\}$ and O_{k2}=$\{RG_{k2}^{t_1}, RG_{k2}^{t_2}, RG_{k_2}^{t_3}, RG_{k1}^{t_4}\}$.

Mutation: each offspring can be the subject of a random mutation with a given probability. Mutations can occur at two levels: at the rule groups level by adding or deleting a rule from a randomly selected group, or at the rules level where seven mutations can be used. The first three mutations are applied on the LHS of the rule and consist in: adding a brick, deleting one, or recreating the LHS. For example, the rule in Listing 3 was mutated by enforcing a new condition that requires the n-m association not to have a class-as reference. The four other rule level mutations apply to the RHS of a rule by adding or deleting an element, recreating the whole RHS, or finally, rebinding the attributes and references of RHS. An example of the latter case is presented in Listing 4

Listing 3. Example of a LHS Rule mutation.

```
Rule before the mutation
(defrule R347261
(associationnm (name ?s00)(classfr ?c00)(classto ?c10)(classas ?s03))
(class(name ?c00))
(class(name ?c10))
=>
(assert (table (name ?s00)(altername nil))))

Rule after the mutation
(defrule R347261
(associationnm (name ?s00)(classfr ?c00)(classto ?c10)(classas ?s03))
(class(name ?c00))
(class(name ?c10))

(test (eq ?s03 nil))
=>
(assert (table (name ?s00)(altername nil))))
```

Mutations at the group level must prevent empty groups and respect the group size limitation. At the rule level changes that occur on the LHS must be propagated to the RHS to avoid, for instance, having rules with a RHS referencing variables that do not exist anymore in their LHS. Finally, mutations at the rule set level (i.e adding or deleting groups) cannot be possible since we expect a rule set to transform all source model types that are involved in the transformation.

Listing 4. Example of a RHS Rule mutation.

```
Rule before the mutation
(defrule R438783
(association1n (classfr ?c00) (classto ?c10))
(class(name ?c00))
(class(name ?c10))
```

```
(column (name ?o20)(table ?c00)(pk 1))
(table (name ?c00)(altername ?t21))
=>
(assert (fk (column ?o20) (table ?c00) (fktable ?c10))))


Rule after the mutation
(defrule R438783
(association1n (classfr ?c00) (classto ?c10))
(class(name ?c00))
(class(name ?c10))

(column (name ?o20)(table ?c00)(pk 1))
(table (name ?c00)(altername ?t21))
=>
(assert (fk (column ?o20) (table ?c10) (fktable ?c00))))
```

6.2.5. Adaptive Search. Many population-based optimization algorithms are subject to the premature convergence problem [Andre et al. 2001; Syrjakow et al. 1998] when parents near the local optima area breed many similar offsprings. In genetic algorithms, this can be circumvented by increasing the exploration power of the algorithm. However, this results in a population that reaches the global optimum very slowly. A lot of contributions have therefore been proposed to tackle this problem. In our approach, we consider mutations as the main mechanism that allows the exploration of the search space. Crossover has always been the most popular operator in GA and GP (this is mostly due to the building-block hypothesis [Holland 1975]). However, much research has cast doubt on the superiority of crossover over mutation in GP [Luke and Spector 1998]. Moreover, since the comparison of models is costly, our approach builds on rather a small population size with which mutation seems to be more successful [Luke and Spector 1998; Srinivas and Patnaik 1994]. We thus regard mutation as the most critical factor in the success of our GP algorithm.

Adaptive Mutations In [Yang and Uyar 2006], an adaptive mutation mechanism is proposed. The mutation probability of each locus is correlated to statistics about the gene-based fitness and the gene-based allele distributions in each gene locus. In our approach, we use a similar mechanism while taking advantage of our ability to evaluate subsets of each individual transformation program. When the GP produces the same fitness value during several generations, the program switches from performing regular mutations to selective mutations. We chose such starting criterion to avoid tuning an adaptive mutation parameter according to the beginning of each transformation problem on which the approach is applied.

During selective mutations, in addition to evaluate a candidate transformation program TP_{ki} on the example pair (SM_k, TM_k), each group of rules RG_{ki}^t is evaluated on the set of traces of type $t \in SMM$ that the group aims to transform. During each mutation, a binary tournament is conducted to elect the group that will be mutated. Two groups that participate to the tournament are randomly chosen, then the group with the lowest fitness win the tournament. The goal of this strategy is to focus on source elements that are not transformed correctly in order to allow a faster convergence toward the global optimum.

Using Memory Many contributions in the field of GP were directed towards protecting desirable partial solution in the genotype: Automatically Defined Functions (ADF) [Koza et al. 1996], Module Acquisition (MA) [Angeline and Pollack 1993], Adaptive Representation through Learning (ARL) [Rosca 1995], and Collective Memory(CM) [Bearpark and Keane 2005]. In this contribution, we implement a similar idea

by storing and reusing rules learned during earlier generations. We circumvent the difficulty of identifying good components of a transformation program by conducting a second evaluation on the fittest individuals in which rules are evaluated individually. This mechanism allows a faster convergence by protecting important material from disruptions caused by crossover and mutation operators.

In our implementation, we gradually enrich a memory of rules during generations in which the highest fitness score improves. We first evaluate each group $RG_k^t t \in SMM$ of the best individual on the set of traces of its type TR_k^t. Groups that achieve a higher score than earlier generations are selected. Each rule of the group is then evaluated individually on the same set of traces. If the rule produces a full match, it is inserted into the collective memory. As in the transformation programs, rules are grouped inside the memory according to the type of fragment they transform. When adaptive mutations are launched, a new memory mutation operator is added to the two mutations available at the rule group level. The mutation selects a rule R_i^t, $t \in SMM$, from the memory storage and adds it to the rule group RG_k^t that transform the type t in the candidate transformation program.

Post-Processing Transformation Programs At this stage, a transformation program for each pool is derived by a distinct GP run. Before merging all the transformations into a single transformation program, we conduct a cleaning step. First, rules that produce more than one target element are split into several rules with one target element each. This is done to make sure that each element is transformed when specific conditions are met, conditions that will be learned in the third phase. Rules are then evaluated independently in their pools on their respective source-target example pairs to remove those that do not produce matches. The remaining rules are analyzed to remove duplicate conditions. Next, duplicate rules are removed. Duplicate rules are identified by analyzing their respective LHS and RHS. Finally, rules that produce the same elements are analyzed to check if the LHS of one rule is subsumed by the other. After each processing action, the transformation program is evaluated on the pool's example pair to make sure that its fitness did not decrease. After correcting and cleaning each transformation program, the rules of all the produced transformations are merged into a single program. Rules that belong to conflicting groups (i.e. groups that transform conflicting traces) are all copied into the final program. For non-conflicting groups, each group is evaluated on the initial source-target example pair, and the group with the highest score is inserted into the final program. After building the final program, this latter is subject to the same processing conducted on each pool's transformation program.

6.3. Refining the Transformation Program

The refinement phase is carried out when a model transformation has been learned across several pools. After merging the pools' programs, we expect the derived rules to have incomplete LHS, as conditions that distinguish conflicting traces are not learned yet. The merged transformation program has a low precision (i.e. generates elements that are not expected) when applied on the original example pair. For example, consider the association rules derived by each pool of Figure 6. Each of these rules will have a rudimentary initial set of conditions that will match all the association elements of the initial source model and produce different target fragments for each match 5.

Listing 5. Two contradictory rules learned in different pool.

```
A Rule learned in Pool 1
(defrule R_174272
(MAIN::association(name ?s00)(classfr ?c00)(classfrcard ?s02)
```

```
(classto ?c10)(classtocard ?s04)(classas ?s05))
(MAIN::class(name ?c00))
(MAIN::class(name ?c10))
=>
(assert (MAIN::table(name ?s05)(altername nil))))

A Rule learned in Pool 3
(defrule R_374137
(MAIN::association(name ?s00)(classfr ?c00)(classfrcard ?s02)
(classto ?c10)(classtocard ?s04)(classas ?s05))
(MAIN::class(name ?c00))
(MAIN::class(name ?c10))
=>
(assert (MAIN::table(name ?c10)(altername (sym-cat ?c10 ?c00 )))))
```

For instance, association rules of the first pool have to be triggered only by association elements with $*-*$ cardinalities (state condition), whereas association rules of the third pool have to learn that only 1-1 association elements with a participating class that have no other link are merged (state and context conditions) (see Listing 6).

Listing 6. A refined class merge rule

```
(defrule R_2252064
(MAIN::association(name ?s00)(classfr ?c00)(classfrcard ?s02)
(classto ?c10)(classtocard ?s04)(classas ?s05))

(MAIN::class(name ?c00))
(MAIN::class(name ?c10))

; Conditions to learn during the third phase
(and (and
(test (eq ?s02 1))
(test (eq ?s04 1)))
(test (eq (count-query-results MAIN::allAssociation ?c00 ) 1)))
=>
(assert (MAIN::table(name ?c10)(altername (sym-cat ?c10 ?c00 )))))
```

Hence, the goal of this phase is to refine the LHS of the rules. Given a transformation program $\widetilde{TP}$, we define the refinement phase as a combinatorial optimization problem (S, f) where S is a finite space of all possible solutions and f is the fitness function defined in paragraph 6.2.3, $f : S \rightarrow [0, 1]$, for $s \in S$. Our objective is to find $TP = s*$ such as $f(s*) > f(s), \forall s \in S$. We use to this end a Simulated Annealing (SA) algorithm.

6.3.1. Simulated Annealing. Simulated Annealing is a generic heuristic approach for global optimization problem that drives inspiration from the field of statistical thermodynamics [Aarts and Korst 1988]. As described in Algorithm 1, it starts with an initial solution from which it moves to a neighbor trial solution. If the neighbor solution is better than the current one, it is selected as the current solution. If not, it still can be selected with a certain probability.

To ensure an overall improvement of the solutions found, the acceptance of less-good solutions is done in a controlled manner inspired by the way thermodynamic energy is reduced when a substance is subject to a controlled cooling. Thus, the probability of accepting non-improving solutions is proportional to the system temperature and inversely proportional to the degradation sustained when accepting the less-good solution. The temperature of the system is gradually reduced as the simulation proceeds according to a schedule, thus increasingly favoring better moves over worse ones. The feature of accepting non-improving moves distinguishes SA from tradition local-search

methods and gives it the ability to escape local optima. In fact, the SA algorithm provides a statistical guarantee that an optimal solution is reached when using a logarithmic annealing schedule [Ingber et al. 2012].

6.3.2. Solution Representation and Generation. At each iteration a new trial neighbor solution s_c is obtained by mutating the current solution TP_c. Unlike the initial conditions that compose each rule LHS, the additional conditions that we learn during this phase are combined through AND and OR operators to form a binary tree (with a fixed maximum size). The OR operator is introduced to handle rules that should fire in multiple situations since we do not add rules nor duplicate existing ones. A mutation operator is used to derive a new neighbor at each iteration by selecting a rule randomly from the current solution, then it either adds, modifies or deletes a condition from the binary tree part of the rule LHS. The conditions that can be added may involve source and target patterns, negation patterns, attribute values and reference testing, as well as navigation primitives, which explore the source model. Target patterns are used for the implicit control of transformations. When a learned condition in a rule tests the presence of a target element, it implicitly states that this rule cannot be fired before the ones that generate this target element. The initial conditions learned during the GP phase remain unchanged.

6.3.3. Solution Evaluation. The trial solution s_c is evaluated on the initial source-target example pair (SM, TM) using the fitness function defined in paragraph 6.2.3. TP_c is replaced with s_c if this latter has a higher fitness or if the condition expressed in Algorithm 1-15 is satisfied. The best solution that was found since the beginning of the algorithm is stored in TP_{best}.

As explained in Section 6.2.3, for a learning process involving multiple example pairs, the fitness of a candidate solution s_c is computed as the average of the fitness score obtained when evaluating the solution on each example pair.

6.3.4. Annealing Schedule. Four parameters must be determined to set up the annealing schedule of the algorithm: the initial temperature, the temperature decrement function, the number of iterations at each temperature, and the stop criterion. In our implementation, we set the initial temperature to 1. This temperature is hot enough to allow the algorithm to move to any neighborhood solution while preventing it from performing a random search in its early stages. Regarding the temperature decrement, we employ the widely used geometric cooling schedule $Temp_t = Temp_0 * \alpha^t$ with α set between 0.9 and 0.99. At each temperature, we experimented with different iteration numbers Nb_{reps} ranging from tens to thousands to find out which configuration (temperature decrement - iteration number) was the more appropriate. Finally, the SA algorithm terminates when the system is frozen, i.e. no solution is accepted, for several successive temperature values.

7. VALIDATION

As explained in Section 4 (see Table I), none of the existing automated approaches considers transformation problems with, at same time, context and state dependencies as well as the complex derivation of attribute values. Thus the addressed problems have a limited search space (transformation of simplified models). Our validation aims to show that despite the consideration of context and state dependencies, and complex derivation of attribute values, which together increase dramatically the search space, our approach can learn the transformation rules. To this end, we evaluated our approach on seven model transformation problems in order to investigate the following research questions:

RQ1. Are the target models produced by the learned transformations correct?

Algorithm 1 Simulated annealing

```
1:  TP_c ← ~TP
2:  TP_best ← ~TP
3:  Temp ← Temp_0
4:  while stop = false do
5:      for i = 0, Nb_reps do
6:          s_c = N(TP_c)
7:          σ ← f(s_c) − f(TP_c)
8:          if σ > 0 then
9:              TP_c ← s_c
10:             if f(TP_c) > f(TP_best) then
11:                 TP_best ← TP_c
12:             end if
13:         else
14:             u ← U([0,1])
15:             if u < exp(σ/Temp) then
16:                 TP_c ← s_c
17:             end if
18:         end if
19:     end for
20:     Temp ← Temp ∗ α
21: end while
```

RQ2. Are the transformation programs correct themselves?
RQ3. Are the results achieved by the approach steady over the different executions?
RQ4. Are the obtained results attributable to the approach itself or to the volume of explored solutions?

We first describe our experimental setting. Then, we answer each research question, and finally discuss the threats that could affect the validity of our study.

7.1. Approach Implementation

We implemented the three steps of our approach in Java. The candidate transformation programs were encoded as Jess rule sets. The metamodels are encoded as Jess fact templates and the models as Jess facts. For the calculation of the fitness scores of candidate transformations, we use the Jess java library version 7.0, including the rule engine[1]. Additionally, we used logging, testing and chart libraries (Log4j, JUnit, JFreeChart). The latter allows us to report visually on the progress of the learning process.

7.2. Experiment Setting

7.2.1. Transformation Problems. We selected seven model transformations that exhibit diverse requirements. Except for SPEM2BPMN transformation case [Debnath et al. 2007], all the transformations were selected from the ATL database[2]. The model transformations are briefly described below.

UML Class Diagram to Relational Schema (CL2RE): We use this well-known model transformation to evaluate the ability of our approach to learn complex structural transformations. The specification we considered includes transforming associations

[1] http://www.jessrules.com/
[2] http://www.eclipse.org/atl/atlTransformations/

with and without an association class as well as merging 1-1 associations when possible, which requires context exploration. Our specification also involves value computations, as merging two classes entails the concatenation of their respective names. A specification of this transformation can be found in [Kessentini 2010].

Domain Specific Language to Kernel Metamodel (DSL2KM3): This case was selected from a transformation chain proposed as a bridge between Microsoft Domain Specific Language (DSL) and Eclipse Modeling Framework (EMF). As for CL2RE, this problem exhibits interesting structural transformations. DSL and KM3 metamodels are fairly similar. However, KM3 supports multiple inheritance whereas DSL does not. Moreover, DSL's classes and relationships have the same properties (i.e. relationships can have attributes and supertypes) whereas this is not the case for KM3. Simple DSL relationships are thus transformed into reference pairs whereas complex ones are transformed into KM3 classes.

Ant to Maven (ANT2MAVEN): This transformation specifies how to generate a Maven file from an Ant file. Considering that Maven is an extension of Ant, this transformation is relatively simple with many one-to-one mappings. We selected this case, however, for two reasons. First, the source and target metamodels are large and the transformation requires 26 rules. Second, most of the model elements have many attributes, which allows us to assess how our approach responds to the proliferation of attributes, a characteristic that hampered our previous contributions [Baki et al. 2014].

Software Process Engineering Metamodel to Business Process Management Notation (SPEM2BPMN) : We consider a transformation specification described in [Debnath et al. 2007] which aims to automate the management of software development activities. SPEM activities are mapped to BPMN sub-processes, which can then be translated into a specification in a standard language such as BPEL (Business Process Execution Language). One interesting feature of this case is that a source element can be transformed into different target elements depending on its references and attribute values.

Microsoft Excel to XML (EXCEL2XML): This transformation aims to generate an Excel workbook (a well-formed Excel XML file) from an Excel model that conforms to the simplified Spreadsheet-ML metamodel. We consider only the first part of the specification as it encompasses all the transformation logic. The second part of the specification is an XML extraction which consists in serializing the XML model into a valid XML file. This transformation is interesting because many target element attributes are mapped to constants rather than to source attribute values.

Table to SVG Bar Charts (TB2BC): The goal of this transformation is to generate a bar chart expressed in a Scalable Vector Graphics model (SVG) according to the data present in a table supplied as input. The SVG model includes graph elements, rectangle elements having dimensions and coordinates as well as text elements placed according to their coordinate elements. This transformation specification entails the use of arithmetic operations and functions to compute dimension and coordinate attributes for the rectangle and text elements generated in the target SVG model.

UML Model to Amble (UML2AMBLE): This case describes a transformation from a set of interrelated UML models expressing different aspects of a distributed program into an implementation of it, in the Amble programming language (a distributed programming language based on Objective Caml). The UML models consist of multiple state diagrams, each diagram representing a type of process involved in the Amble program, and a single class diagram that describes the different processes (as classes) and their communication channels (as associations). The specification of this transformation requires context exploration, testing reference and attribute values, and the

derivation of some complex target attribute values. Additionally, this transformation involves many source models.

7.2.2. Data Set. As mentioned earlier, we used in our setting Jess, a declarative rule-based programming language. Metamodels were represented as fact templates and models as facts. After representing all the metamodels in Jess, we defined two source-target model pairs for each transformation problem. Table II shows the number of elements in each example pair. We chose to design each source model in order to meet all the specifications of the transformation problem in a single example pair. As ground truth, we also wrote the transformation program associated with each transformation case, tested them on the defined model pairs and inspected each target model to make sure that they were correct with respect to the transformation specifications.

Table II. The size of the example models in number of elements.

Transformation	First Pair		Second Pair		Written Pgm
	Source	Target	Source	Target	Nb. Rules
CL2RE	30	42	42	59	25
DSL2KM3	31	38	46	60	9
ANT2MAVEN	26	29	33	36	25
SPEM2BPMN	14	14	22	23	9
EXCEL2XML	18	22	31	47	10
TB2BC	13	42	17	54	12
UML2AMBLE	41	33	62	66	14

We evaluated our process on each transformation problem using a 2-fold cross-validation strategy. For each transformation problem, the process is run with the first example pair to learn the sought program. This latter is then evaluated on the second example pair. For the second fold, the pairs are exchanged and the process is launched a second time. For each case, we thus built the traces associated with both source-target example pairs, as mentioned in Section 6.1, we tried to mimic a real-life scenario by avoiding precise traces where a source fragment is mapped to the exact target element that would be produced by applying the corresponding transformation rule.

7.2.3. Algorithm Parameters. We ran our approach once for each transformation problem with the learning example pair and its traces as input. The runs of the genetic program were all made of 200 generations and 100 programs. Crossover and mutation probabilities were both set to 0.9 and elitism to five programs. Regarding the simulated annealing algorithm, we opted for a slowly decreasing temperature with a small number of iterations each time. We thus set the α parameter of the annealing schedule to 0.99 with only 10 iterations at each temperature value. The learning process, including the post-processing at the end of the second phase, was fully automated and did not involve any human intervention.

The genetic program was each time supplied with functions and constants specific to the source and target domains. To avoid injecting knowledge about each transformation, we also added functions that are not used by the transformation. For instance, the TB2BC case requires the derivation of the total number of rows and the maximum cell values. We then supplied the algorithm with many functions such as count, minimum, maximum, and sum that could be applied to all the numerical attributes. In addition to domain-specific constants and functions, we included in the seven cases, basic string and integer operations (string concatenation, addition, subtraction, etc.).

7.2.4. Validation Method. To answer **RQ1**, we assess each derived transformation program by comparing its produced target model with the expected one. The comparison is done automatically using the precision and recall measures. We define the recall as the ratio between the number of expected elements that are produced and the total number of expected elements, whereas the precision is the ratio between the number of expected elements produced and the total number of produced elements. We say that an expected element is produced if there is an exact match with an expected element, i.e., its name, attribute values, and references are the same. Unlike in the fitness function defined for the GP and SA programs, the reported results exclude partial matches, thus, an element is considered as incorrect even if only one of its attribute is incorrectly derived. For PE and EE, respectively the set of produced elements and the set of expected elements, the evaluation of the target models consists in the two following formulas:

$$Rec_{TM} = \frac{|EE \cap PE|}{|EE|} \quad (3)$$

$$Prec_{TM} = \frac{|EE \cap PE|}{|PE|}, \quad (4)$$

We answer **RQ2** in two ways. First, we run the derived transformations on the evaluation example pairs (unseen pairs) and assess the precision and recall. Second, we manually inspect each derived transformation program. Indeed, at the program level, we check if the target elements are produced only by the rule groups that are supposed to produce them, at the rule-group level, we assess on one hand, the completeness and the correctness of the conditions that should be satisfied to produce an element. On the other hand, we check to what extent were the target attributes correctly derived.

To assess the steadiness of the obtained results (**RQ3**) considering the probabilistic aspect of our approach, we run the learning process 30 times on the first pair of the CL2RE transformation problem, and we report and analyze the variations observed in the recorded results.

Finally, we answer **RQ4** in two ways. Firstly, we assess the benefits of our learning process by comparing the fitness scores obtained for the transformation problem used in RQ3 (CL2RE) with two other samples obtained respectively by a random exploration of the search space, and a single-phase standard GP search as the described in Baki et al. [2014] and Faunes et al. [2013]. Secondly, we run the second phase of our approach several times with and without its adaptive features to assess the benefits brought by the proposed mutation strategies.

To have a fair *Approach vs Random* comparison, we consider transformations programs obtained by a 30 random exploration of the search space (to compare with the 30 programs derived for RQ3). In each random exploration, we selected the best transformation program from a pool of randomly-generated transformation programs. The total number of random programs in the pool is equal to the number of solutions that are explored by the whole process of our approach (GP and SA) with the parameters given above. This resulted in pools of approximately 100 000 programs. Each random generation is performed in a similar way to the one used to build the initial population of GP, particularly by exploiting the transformation traces of the first CL2RE example pair. However, unlike GP programs, the random algorithm can explore the whole search space, i.e., it may include conditions that test the context and/or state of elements.

The *Approach vs GP* comparison is performed by contrasting the data gathered in RQ3 with an equal number of programs obtained using a single-step standard GP

algorithm. The algorithm is similar to the one described in Baki et al [Baki et al. 2014] except it uses transformation traces to reduce the size of the search space. Unlike for the proposed approach, the standard GP algorithm aims to learn all the transformation program in a single step. As for the random strategy, it, therefore, can explore the whole search space, including element states and context. The standard GP algorithm is run with the same parameters used in our approach on the first CL2RE example pair as well.

7.3. Results and Interpretation

Table III. Fitness, Recall and Precision for the pool learning with GP (TP_k) and after the rule integration ($\widetilde{TP}$).

Transformation	Pools	Pair	TP_k			$\widetilde{TP}$		
			Avg(Fitness)	Avg(Recall)	Avg(Precision)	Fitness	Recall	Precision
CL2RE	5	1st	96.2%	97%	95.5%	53.4%	100%	37%
		2nd	96.6%	96.9%	96.4%	53.5%	100%	37.6%
DSL2KM3	2	1st	98.3	100%	96.9%	73.5%	100%	64.9%
		2nd	98%	100%	96.2%	79.6%	100%	70.1%
ANT2MAVEN	2	1st	100%	100%	100%	99.3%	100%	98.8%
		2nd	97.9%	97.6%	98.6%	100%	100%	100%
SPEM2BPMN	2	1st	94.9%	97.3%	93.7%	92.4%	100%	88%
		2nd	95.1%	100%	92.8%	92.4%	100%	88%
EXCEL2XML	1	1st	100%	100%	100%	100%	100%	100%
		2nd	100%	100%	100%	100%	100%	100%
TB2BC	1	1st	93.3%	90.0%	100%	93.3%	90.0%	100%
		2nd	93.3%	90.0%	100%	93.3%	90.0%	100%
UML2AMBLE	4	1st	93.5%	94.4%	95.1%	78.5%	96.9%	71.9%
		2nd	93.9%	94.2%	95.8%	78.8%	95.8%	73%

Table III shows the results achieved on the training example pairs at each step of the learning process. Five of the seven studied problems contained conflicting traces and were thus separated into multiple pools. For these cases, a high fitness ranging from 93% to 100% is reached in each pool. When the transformation programs are merged and post-processed into a single program $\widetilde{TP}$, a perfect recall is achieved for four of the five cases. However, and as expected after the merge, the precision of each resulting program drops, which results in a relatively low fitness ranging from 50% to 80% for three cases CL2RE, DSL2KM3 and UML2AMBLE. After conducting the final step, the quality of each transformation plan is considerably enhanced and a fitness higher than 92% is observed on both pairs of all the transformation cases having conflicting traces.

Regarding transformation problems learned in a single pool, our approach reached a perfect score for the Excel2XML transformation case, whereas for TB2BC, although the precision was 100%, only 90% of the expected elements were produced on both learning pairs. For these transformation cases, the simulated annealing phase was not conducted as all the rules were learned from a single pool. The final program of each case thus corresponds to the post-processed version of the genetic program's output.

7.3.1. Target Model Correctness (RQ1). There are two transformation problems where the recall in $\widetilde{TP}$, although very high, was not perfect, TB2BC and UML2AMBLE. When inspecting each program rules we found out that in the TB2BC, there are two situations that our approach could not handle. First, in certain elements, a target attribute is computed through an addition of a source attribute and the value returned by a function (nested functions). Second, some elements contained not one but many attributes requiring complex value derivation. Rules responsible for producing such elements were not learned, see, for example, Listing 7 where R11 is successfully learned whereas R12 is not. This is because our fitness function does not evaluate attributes

Table IV. Fitness, Recall and Precision after the refinement step with SA.

Transformation	Pools	Pair	TP			
			Fitness	Recall	Precision	Nb. Rules
CL2RE	5	1st	93.8%	95.2%	92.8%	21
		2nd	93.5%	92.1%	95.3%	22
DSL2KM3	2	1st	100%	100%	100%	9
		2nd	99.4%	98.9%	100%	9
ANT2MAVEN	2	1st	100%	100%	100%	30
		2nd	100%	100%	100%	29
SPEM2BPMN	2	1st	100%	100%	100%	9
		2nd	100%	100%	100%	9
EXCEL2XML	1	1st	100%	100%	100%	10
		2nd	100%	100%	100%	10
TB2BC	1	1st	93.3%	90.0%	100%	12
		2nd	93.3%	90.0%	100%	12
UML2AMBLE	4	1st	95.6%	95.8%	97.2%	16
		2nd	94.9%	93.5%	99%	20

independently, and if a complex value is derived correctly but another is not, the element will not be favored against other partial matches. Regarding the UML2AMBLE case, our algorithm could not derive two attributes of a particular element because they require building an ordered collection of elements and navigating through this collection to compute values.

Listing 7. Derivation of multiple complex attributes.

```
(defrule R11
(row (name ?r) (value ?v) (number ?nb) (table ?t))
(table (name ?t) (posx ?x) (posy ?y))
=>
(assert (abscoord (ref ?r) (x ?x) (y  (+ ?y ?nb)))))

(defrule R12
(row (name ?r) (value ?txt) (number ?nb) (table ?t))
(table (name ?t) (posx ?x) (posy ?y))
(cell (name ?n) (value ?v) (ref ?r))
=>
(assert (abscoord (ref ?txt) (y (+ ?y ?nb)) (x (+ ?x ?v)))))
```

It is important to stress out that we do not expect pool transformations to achieve a perfect recall or precision. This is because when building each pool model pair, some elements may be added despite not being transformed in the pool itself because they are referenced by a main element in that pool. For example, in the first learning pair of SPEM2BPMN, the average pool recall and precision are respectively 97.3% and 93.7%, when merged, the transformation program has a recall of 100% and a similar precision after the refinement phase. Regarding the CL2RE and UML2AMBLE case, it is worth mentioning that we obtained a perfect score for the final programs TP in an alternative setting in which, we increased the size of the additional condition tree of each rule LHS in the refinement phase. However, when applying the derived programs to the validation pair, a large variability was recorded in the results. Allowing complex condition trees while using a single example drove the SA algorithm to learn conditions sometimes specific to the example pair that did not always hold for the validation pair.

7.3.2. Rule Quality (RQ2). For the second research question, Table V shows the results obtained when we applied the final transformation programs TP on the validation examples. For single-pool transformations, the results were identical to those achieved on the training pairs, whereas for the four other cases, the reported results were slightly lower except for SPEM2BPMN where a perfect score is obtained for the validation case

Table V. Fitness, Recall and Precision when evaluating on the validation example pairs.

Transformation	Pair	TP		
		Fitness	Recall	Precision
CL2RE	1st	93.8%	95.1%	92.8%
	2nd	93.5%	92.1%	95.3%
DSL2KM3	1st	98.4%	97.9%	98.9%
	2nd	98.5%	100%	97.2%
ANT2MAVEN	1st	100%	100%	100%
	2nd	100%	100%	100%
SPEM2BPMN	1st	100%	100%	100%
	2nd	100%	100%	100%
EXCEL2XML	1st	100%	100%	100%
	2nd	100%	100%	100%
TB2BC	1st	93.3%	90.0%	100%
	2nd	93.3%	90.0%	100%
UML2AMBLE	1st	90.3%	92.1%	92.7%
	2nd	90.8%	95.8%	90.9%

as well. These results confirm that the derived transformations are not specific to the learning examples and can be successfully applied to new models.

Regarding the quality of the learned rules, during the GP phase, we found out that elements are always produced by the rule groups that are supposed to produce them. However, given that we do not require precise traces, we also came across groups that produced elements they were not supposed to, which resulted in the same elements produced twice. These cases were handled by the post-processing phase. In the example of Listing 8, the second rule is deleted because its conditions subsume that of the first one.

Complex and correct conditions were also successfully learned in the SA phase. For instance, in the rule of Listing 9, a role end in DSL2KM3 is transformed into a reference to the corresponding class only if the DSL relationship is not complex. Otherwise, it is transformed into a KM3 class. The derived rule verifies if the relationship is simple by ensuring that it has not a value property, and that it is not involved in an inheritance.

Listing 8. A post-processing case.

```
(defrule R_649407
(MAIN::class(name ?c00))
=>
(assert (MAIN::table(name ?c00)(altername nil))))

(defrule R_649408
(MAIN::attribute(name ?a00)(class ?c00)(unico 0))
(MAIN::class(name ?c00))
=>
(assert (MAIN::table(name ?c00)(altername nil))))
```

7.3.3. Results steadiness(RQ4). The first line of Table VI shows the results obtained when executing the learning process 30 times on the first example pair of the CL2RE transformation problem. The stop criterion for the SA algorithm was modified to ensure the same number of iterations in each run. The score of derived transformation programs ranges between 92.2% and 100%.

Listing 9. A complex derived rule.

```
(defrule R_514281
(role(name ?ro00)(source ?c00)(rtype ?c10)
```

```
(relationship ?re00)(min ?ro04)(max ?ro05)(isordered ?ro06))
(class(name ?c00)(supertype ?c01))
(class(name ?c10)(supertype ?c11))
(relationship(name ?re00)(supertype ?re01)(isembedding ?re02))

; Conditions learned during second phase
(and  (and  (not (relationship(name ?re1800)(supertype ?re00)(isembedding ?re1802)) )
(not (valueproperty(name ?vp800)(vtype ?vp801)(owner ?re00))))
(not (relationship(name ?re01)(supertype ?re3121)(isembedding ?re3122))))
=>
(assert (reference(name ?c00)(owner ?c10)(rtype ?c00)(lower ?ro04)(upper ?ro05)
                                (iscontainer ?re02)(opposite ?c10))))
```

7.3.4. RQ4 - Approach vs GP & Random. In order to confirm that the results achieved by our approach are due to a better exploration of the search space, we compare the quality (fitness) of the transformation programs derived by our approach to one of the programs obtained by a random exploration of the search space or a single-phased standard GP search. The results obtained for the three experiments are summarized in Table VI. The scores of the best randomly-generated programs ranged from 54% to 61% with an average fitness of 61%. The standard GP approach achieved, on the other hand, better results with an average of 76% over the 30 runs and a maximum of a 80% fitness score.

Table VI. Descriptive statistics of each group.

Group	N	Min	Max	Mean	Std. Dev.	Std. Err.
Approach	30	0.92	1	0.95	0.018	0.003
GP	30	0.63	0.8	0.76	0.056	0.01
Random	30	0.54	0.61	0.56	0.012	0.002

To confirm that the observed disparity of results achieved by the different alternatives is statistically significant, we perform a mean difference t-test (the three samples being normally distributed). The results are shown in Table VII. The difference in quality of the programs derived by our approach and the ones produced by a random exploration approximates 40% in average. This difference is around 20% for the comparison with a standard GP approach. Both differences are statistically significant with a p-value <0.001.

7.3.5. RQ4 - Adaptive GP. Answering the second part of RQ4, we executed our approach with and without the adaptive features, i.e., adaptive and memory mutations. We observed, in general, that for complex transformations (those that require more exploration in the GP phase) such as CL2RE and UML2Amble, the better fitness values are reached quickly when adaptive features are used.

Table VII. Significance testing for mean differences of fitness.

	t-Test			
	t	df	Significance (2-tailed)	Mean difference
vs Random	96.122	49.9	<0.001	0.393
vs GP	96.122	35.4	<0.001	0.194

Figure 9 illustrates such a pattern for one pool of the CL2RE transformation. The adaptive GP run outperforms its opponent and converges more quickly. It achieves a perfect score around the 145th generation, whereas the standard GP run got stuck in a local optima with a fitness of 96.7%. This pattern, however, was not observed for all our transformation problems, especially for the simple ones.

7.4. Threats to validity

Although our approach produced good results on seven model transformations, a threat to the validity of our results resides in the generalization of our approach to all exogenous model transformations. While our sample – and then our approach– does not cover all transformation scenarios, especially complex non-deterministic transformations [Huszerl et al. 2002], we believe that it is representative enough of a wide range of real transformation problems.

Regarding the applicability of our approach to other transformation languages, we believe that our approach can be tailored to declarative MT languages as we are using a fully declarative language without sophisticated features. The only particularity of Jess is that, unlike other MT languages, creating the same model element (asserting a same fact) twice does not results in two model elements. We mitigated this threat by ensuring that all the transformation programs do not assert the same fact more than once through our post-processing step. The applicability of our approach is also concerned with the types of metamodels that can be considered. Indeed, in our approach, we make the assumption that metamodel elements have unique names and use these names to represent the relations between them (e.g., the use of class names in source and target slots of a fact representing an association). When such elements do not have unique names, our approach is not able to learn rules that transform them.

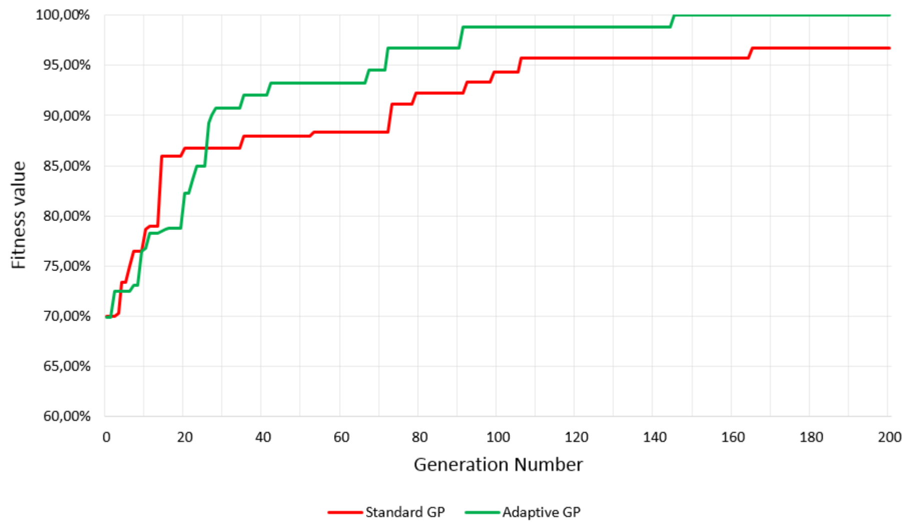

Fig. 9. Evolution of the fitness function for standard and adaptive GP run.

Another threat to the validity of our results relates to the use of a single example pair to learn each transformation program. We chose this alternative (instead of gathering a set of examples that cover the transformation specification) to simplify our experiment. We do believe that the approach would perform equally when using several example pairs, provided that they cover the transformation specifications. Moreover, our process could be run each time a new example pair is added to enhance to quality of the learned transformation since our process is not time consuming (few hours) and that the identification of traces can be tool-supported.

Finally, the last aspect to consider given these results is the quality of the transformation traces provided to the algorithm. When provided with imperfect traces in

another scenario, our algorithm may erroneously separate coherent traces into distinct pools. Such cases are handled by the post-processing step to eliminate duplicate rules. Considering that we use a search-based approach, we conjecture that our process is robust enough to such imperfections as long as the fraction of imperfect traces is small.

8. DISCUSSION

8.1. Using Transformation Traces

Unlike our previous contributions, the approach proposed in this paper requires the user to supply transformation mappings along with each example pair (as for most MTBE contributions). While this can be seen as a limitation, it is necessary when learning complex model transformations. We also mitigate the overload of defining transformation traces manually by accepting imprecise traces.

While the extra work needed for defining such traces is largely justified by the benefits of MTBE, we believe that it can be further reduced by assisting the expert with a tool, especially when dealing with large models. For instance, the expert could define the mapping of one source element, and the tool would then analyze the defined mapping and suggests mappings for similar elements.

8.2. Learning Complex Rules

During the validation of the approach, the algorithm did not produce all the expected elements ($recall < 100\%$ in $\widetilde{TP}$ in Table III) for two transformations. For TB2BC, nested functions and multiple-attribute computations were required. The latter can be circumvented by defining a more fine-grained fitness function that ranks partial matches depending on the ratio of well-derived attributes. This, however, comes at the cost of much more sophisticated evaluations (model comparison).

The second type of rule that could not be learned is the one of UML2Amble where it required iterating on an ordered collection of source elements while computing values. It is possible to deal with this situation by adding sets' operators (ordering and iterations) and by allowing functions/operators composition. Here again, this will increase dramatically the size of the search space.

8.3. Rules Generalization

When conducting our experiments, we were always able to achieve full precision after the refinement phase by increasing the size of the condition tree. The approach tends, however, to learn conditions specific to the example. This is because there are usually many transformation programs that can produce a target model from a source model. In order to improve the correctness of the programs, more examples are required. The learning can also be iterative, i.e., test-driven [Kappel et al. 2012], by running the process each time the example base is enriched with a new model pair, and if the transformation is not satisfactory, the expert can revise the defined mappings or the transformation program.

8.4. Rules Quality

During the refinement phase, we aim to learn the implicit transformation control by testing target elements in rule conditions. In some cases, this results in rules, harder to understand and to debug. A possible solution is to add a quality measurement to the fitness function to favor source pattern testing. Further investigation should determine how this could be done without affecting implicit control learning.

8.5. Scalability

Regarding the scalability, we measured the execution time and average heap usage at each phase of the learning process. All the executions were run on a desktop computer with Intel Core i7 3.40GHz and 32 gigabytes of memory. Table VIII details the results obtained for each transformation case when launching the learning process with the parameters given in Section 7. For transformation problems involving multiple GP pools, the average execution time is reported. Results for both time and memory usage were rounded up to the upper whole number.

The average execution time for the GP learning phase ranged from 32 to 92 minutes. The ANT2MAVEN registered the highest execution time and average heap usage during the second phase. This is most likely due to the size of the transformation programs derived in each pool that reached 30 rules for this transformation. The execution time of the SA phase varied greatly depending on the complexity of the refinements to perform. It took only 2 minutes the merge and refine the final transformation program for the ANT2MAVEN and SPEM2BPMN transformation cases, whereas it took between 11 and 19 minutes for DSL2KM3, CL2RE and UML2AMBLE transformation problems.

Table VIII. Execution Time (min) and Average Heap Usage (MB) at each phase of the Learning Process

Transformation	Phase 1		Phase 2 (GP)		Phase 3 (SA)	
	Time	AVG Heap	AVG Time	AVG Heap	Time	AVG Heap
CL2RE	<1	13	39	23	14	30
DSL2KM3	<1	25	32	13	11	13
ANT2MAVEN	<1	16	92	43	2	17
SPEM2BPMN	<1	15	35	16	2	13
EXCEL2XML	<1	25	36	15	-	-
TB2BC	<1	25	30	16	-	-
UML2AMBLE	<1	20	49	28	19	23

Although the approach would certainly take longer when using multiple example pairs, we believe that this can be easily dealt with by distributing the evaluation step of both the genetic programming and simulated annealing phases across multiple machines.

Note that in our validation, we use the tuning strategy to determine the algorithms' parameter values. We, consequently, applied these predefined parameters uniformly to all the problems in the experiments. Many transformations, such as EXCEL2XML and ANT2MAVEN, were fully learned faster and did not require to explore all the generations. Additionally, in some transformations, longer runs were required in the GP phase whereas, in others such as DSL2KM3, SA phase was responsible for learning the most complex patterns. A more sophisticated way of determining each algorithm parameters [Eiben et al. 1999; Eiben et al. 2007] would increase the efficiency of our approach.

9. CONCLUSION

In this paper, we propose an MTBE, search-based, approach to learn complex model transformations from a set of interrelated source-target model example pairs. The example traces are analyzed and split into non-conflicting pools, which are used to derive preliminary transformation programs by means of an adaptive genetic programming algorithm. The derived transformations are then combined into a single program which is refined using a simulated annealing algorithm to improve the rules with more complex conditions. We evaluated our approach on seven transformation problems. Our validation shows that the approach generally produces good transformations on

some complex problems that require mechanisms such as context exploration, source-attribute value testing and complex target-attribute derivation. Despite these encouraging results, there is still room for improvement. Concretely, we plan to (1) investigate the applicability of our approach on a more common environment such as the Eclipse Modeling Framework, (2) develop a graphical tool to assist experts in defining domain mappings between source and target models, and (3) assess with human subjects to what extent does our process facilitate developing model transformations. Regarding the approach itself, we will explore the idea of learning elements with multiple complex attribute values through a more fine-grained fitness function and/or using memory to store correct derivation functions. We also plan to handle collections in a more complete way. Another improvement axis worth investigating is the use of parameter control [Eiben et al. 1999; Eiben et al. 2007] to cope with the particularities of each transformation problem.

ACKNOWLEDGMENTS

This work was partially supported by NSERC, grant RGPIN-2014-06702.

REFERENCES

Emile Aarts and Jan Korst. 1988. *Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing*. Wiley.

Jerome Andre, Patrick Siarry, and Thomas Dognon. 2001. An improvement of the standard genetic algorithm fighting premature convergence in continuous optimization. *Advances in engineering software* 32, 1 (2001), 49–60.

Peter Angeline and Jordan Pollack. 1993. Evolutionary module acquisition. In *Proceedings of the second annual conference on evolutionary programming*. Citeseer, 154–163.

Iman Avazpour, John Grundy, and Lars Grunske. 2015. Specifying model transformations by direct manipulation using concrete visual notations and interactive recommendations. *Journal of Visual Languages and Computing* 28 (2015), 195 – 211.

Islem Baki, Houari Sahraoui, Quentin Cobbaert, Philippe Masson, and Martin Faunes. 2014. Learning Implicit and Explicit Control in Model Transformations by Example. In *Model-Driven Engineering Languages and Systems*. Springer, 636–652.

Zoltan Balogh and Dániel Varró. 2009. Model transformation by example using inductive logic programming. *Software and Systems Modeling* 8 (2009), 347–364. Issue 3.

Keith Bearpark and Andy J Keane. 2005. The use of collective memory in genetic programming. In *Knowledge Incorporation in Evolutionary Computation*. Springer, 15–36.

Petra Brosch, Philip Langer, Martina Seidl, Konrad Wieland, Manuel Wimmer, Gerti Kappel, Werner Retschitzegger, and Wieland Schwinger. 2009b. An example is worth a thousand words: Composite operation modeling by-example. In *Model Driven Engineering Languages and Systems*. Springer, 271–285.

Petra Brosch, Philip Langer, Martina Seidl, and Manuel Wimmer. 2009a. Towards end-user adaptable model versioning: The by-example operation recorder. In *Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models*. IEEE Computer Society, 55–60.

Sabine Buckl, Alexander M Ernst, Josef Lankes, Christian M Schweda, and André Wittenburg. 2007. Generating Visualizations of Enterprise Architectures using Model Transformations.. In *EMISA*, Vol. 2007. 33–46.

Krzysztof Czarnecki and Simon Helsen. 2006. Feature-based survey of model transformation approaches. *IBM Systems Journal* 45, 3 (2006), 621–645.

Narayan Debnath, Fabio Zorzan, German Montejano, and Daniel Riesco. 2007. Transformation of BPMN subprocesses based in SPEM using QVT. In *Electro/Information Technology, 2007 IEEE International Conference on*. IEEE, 146–151.

Xavier Dolques, Marianne Huchard, Clémentine Nebut, and Philippe Reitz. 2010. Learning Transformation Rules from Transformation Examples: An Approach Based on Relational Concept Analysis. In *International Conference on Enterprise Distributed Object Computing Workshops*. 27 –32.

Alexander Egyed. 2002. Automated abstraction of class diagrams. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 11, 4 (2002), 449–491.

Agoston Eiben, Zbigniew Michalewicz, Marc Schoenauer, and James Smith. 2007. Parameter control in evolutionary algorithms. In *Parameter setting in evolutionary algorithms*. Springer, 19–46.

Agoston Endre Eiben, Robert Hinterding, and Zbigniew Michalewicz. 1999. Parameter control in evolutionary algorithms. *Evolutionary Computation, IEEE Transactions on* 3, 2 (1999), 124–141.

Martin Faunes, Houari Sahraoui, and Mounir Boukadoum. 2012. Generating Model Transformation Rules from Examples using an Evolutionary Algorithm. In *Automated Software Engineering*. 1–4.

Martin Faunes, Houari Sahraoui, and Mounir Boukadoum. 2013. Genetic-Programming Approach to Learn Model Transformation Rules from Examples. In *Theory and Practice of Model Transformations*. Springer, 17–32.

Charles Forgy. 1982. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial intelligence* 19, 1 (1982), 17–37.

Iván García-Magariño, Jorge J. Gómez-Sanz, and Rubén Fuentes-Fernández. 2009. Model Transformation By-Example: An Algorithm for Generating Many-to-Many Transformation Rules in Several Model Transformation Languages. In *Proceedings of the International Conference on Theory and Practice of Model Transformations*. 52–66.

Birgit Grammel, Stefan Kastenholz, and Konrad Voigt. 2012. *Model matching for trace link generation in model-driven software development*. Springer.

Ernest Friedman Hill. 2003. *Jess in Action: Java Rule-Based Systems*. Manning Greenwich, CT.

John H Holland. 1975. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press.

Gábor Huszerl, István Majzik, András Pataricza, Konstantinos Kosmidis, and Mario Dal Cin. 2002. Quantitative analysis of UML statechart models of dependable systems. *The computer journal* 45, 3 (2002), 260–277.

Lester Ingber, Antonio Petraglia, Mariane Rembold Petraglia, and Maria Augusta Soares Machado. 2012. Adaptive simulated annealing. In *Stochastic global optimization and its applications with fuzzy adaptive simulated annealing*. Springer, 33–62.

Gerti Kappel, Philip Langer, Werner Retschitzegger, Wieland Schwinger, and Manuel Wimmer. 2012. Model transformation by-example: a survey of the first wave. In *Conceptual Modelling and Its Theoretical Foundations*. Springer, 197–215.

Marouane Kessentini. 2010. *Transformation by example*. Ph.D. Dissertation. University of Montreal, Canada.

Marouane Kessentini, Houari Sahraoui, and Mounir Boukadoum. 2008. Model transformation as an optimization problem. In *Model Driven Engineering Languages and Systems*. Springer, 159–173.

Marouane Kessentini, Houari Sahraoui, Mounir Boukadoum, and Omar Ben Omar. 2012. Search-based model transformation by example. *Software and System Modeling* 11, 2 (2012), 209–226.

Marouane Kessentini, Manuel Wimmer, Houari Sahraoui, and Mounir Boukadoum. 2010. Generating transformation rules from examples for behavioral models. In *Proc. of the 2nd Int. Workshop on Behaviour Modelling: Foundation and Applications*. Article 2, 2:1–2:7 pages.

Anneke G Kleppe, Jos Warmer, Wim Bast, and Explained. 2003. The model driven architecture: practice and promise. (2003).

John Koza, David Andre, Forrest Bennett III, and Martin Keane. 1996. Use of automatically defined functions and architecture-altering operations in automated circuit synthesis with genetic programming. In *Proceedings of the First Annual Conference on Genetic Programming*. MIT Press, 132–140.

John R Koza. 1992. *Genetic programming: on the programming of computers by means of natural selection*. Vol. 1. MIT press.

Philip Langer, Manuel Wimmer, and Gerti Kappel. 2010. Model-to-model transformations by demonstration. In *Theory and Practice of Model Transformations*. Springer, 153–167.

Sean Luke and Lee Spector. 1998. A revised comparison of crossover and mutation in genetic programming. *Genetic Programming* 98, 55 (1998), 208–213.

Parastoo Mohagheghi, Wasif Gilani, Alin Stefanescu, Miguel A Fernandez, Bjørn Nordmoen, and Mathias Fritzsche. 2013. Where does model-driven engineering help? Experiences from three industrial cases. *Software & Systems Modeling* 12, 3 (2013), 619–639.

Franois Pfister, Vincent Chapurlat, Marianne Huchard, and Clémentine Nebut. 2012. A proposed tool and process to design domain specific modeling languages. (2012).

Riccardo Poli, William B Langdon, Nicholas F McPhee, and John R Koza. 2008. *A field guide to genetic programming*. Lulu. com.

Nicolas Revault, Houari A Sahraoui, Gilles Blain, and Jean-François Perrot. 1995. A Metamodeling technique: The METAGEN system. In *Technology of Object-Oriented Languages and Systems, TOOLS EUROPE, Vol. 16*. Prentice Hall, 127–139.

Justinian P Rosca. 1995. Genetic Programming Exploratory Power and the Discovery of Functions.. In *Evolutionary Programming*. Citeseer, 719–736.

Hajer Saada, Xavier Dolques, Marianne Huchard, Clémentine Nebut, and Houari Sahraoui. 2012a. Generation of operational transformation rules from examples of model transformations. In *International Conference on Model Driven Engineering Languages and Systems*.

Hajer Saada, Xavier Dolques, Marianne Huchard, Clémentine Nebut, and Houari Sahraoui. 2012b. Learning Model Transformations from Examples using FCA: One for All or All for One?. In *CLA'2012: 9th International Conference on Concept Lattices and Applications*. 45–56.

Hajer Saada, Marianne Huchard, Clémentine Nebut, and Houari Sahraoui. 2013. Recovering model transformation traces using multi-objective optimization. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference*. IEEE, 688–693.

Douglas C Schmidt. 2006. Model-driven engineering. *IEEE Computer Society* 39, 2 (2006), 25.

Mika P Siikarla and Tarja J Systa. 2008. Decision reuse in an interactive model transformation. In *Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on*. IEEE, 123–132.

Mandavilli Srinivas and Lalit Patnaik. 1994. Adaptive probabilities of crossover and mutation in genetic algorithms. *Systems, Man and Cybernetics, IEEE Transactions on* 24, 4 (1994), 656–667.

Michael Strommer, Marion Murzek, and Manuel Wimmer. 2007. Applying model transformation by-example on business process modeling languages. In *Advances in Conceptual Modeling–Foundations and Applications*. Springer, 116–125.

Michael Strommer and Manuel Wimmer. 2008. A Framework for Model Transformation By-Example: Concepts and Tool Support. In *Objects, Components, Models and Patterns*. Springer, 372–391.

Yu Sun, Jeff Gray, and Jules White. 2011. MT-Scribe: an end-user approach to automate software model evolution. In *Proceedings of the 33rd International Conference on Software Engineering*. ACM, 980–982.

Yu Sun, Jules White, and Jeff Gray. 2009. Model Transformation by Demonstration. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 712–726.

Helena Syrjakow, Matthias Szczerbicka, and Michael Becker. 1998. Genetic algorithms: a tool for modelling, simulation, and optimization of complex systems. *Cybernetics & Systems* 29 (1998), 639–659.

Daniel Varró. 2006. Model Transformation by Example. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 410–424.

Manuel Wimmer, Michael Strommer, Horst Kargl, and Gerhard Kramler. 2007. Towards Model Transformation Generation By-Example. In *Annual Hawaii International Conference on System Sciences*. 285–295.

Shengxiang Yang and Şima Uyar. 2006. Adaptive mutation with fitness and allele distribution correlation for genetic algorithms. In *Proceedings of the 2006 ACM symposium on Applied computing*. ACM, 940–944.