MOMM: Multi-Objective Model Merging

Marouane Kessentini[1], Mazen Alnofal[1], Philip Langer[2], Manuel Wimmer[2], Slim Bechikh[1] and Kalyanmoy Deb[3]

[1] University of Michigan, USA
firstname@umich.edu
[2] Vienna University of Technology, Austria
lastname@big.tuwien.ac.at
[3]Michigan State University, USA
kdeb@egr.msu.edu

Computational Optimization and Innovation Laboratory, MSU, USA
http://www.egr.msu.edu/~kdeb/report.shtml

COIN Report Number 2014008

Abstract. Nowadays, software systems are complex and large. To cope with this situation, teams of developers have to cooperate and work in parallel on software models. Thus, techniques to support the collaborative development of models are a must. To this end, several approaches exist to identify the change operations applied in parallel, to detect conflicts among them, as well as to construct a merged model by incorporating all non-conflicting operations. Conflicts often denote situations where the application of one operation disables the applicability of another one. Consequently, one operation has to be omitted to construct a valid merged model in such scenarios. When having to decide which operation to omit, the importance of their application, depending on the operation type and the application context, has to be taken into account. However, existing work treat the operations to merge with equal importance. We introduce in this paper, for the first time, a multi-objective formulation of the problem of model merging, based on NSGA-II, that aims to find the best trade-off between minimizing the number of omitted operations and maximizing the number of successfully applied important operations. We evaluated our approach using seven open source systems and compared it with different existing model merging approaches. The merging solutions obtained with our approach were found in 100% of the scenarios of our experiments to be comparable in terms of minimizing the number of conflicts to those suggested by existing approaches and to carry a high importance score of merged operations. Our results also revealed an interesting feature about the trade-off between the two conflicting objectives that demonstrates the practical value of taking the importance of operations into account in model merging tasks since the shape of the Pareto front represents an interesting guidance for developers to select best solutions based on their preferences.

Keywords: Search-based software engineering, software maintenance, merging, multi-objective optimization.

1 Introduction

Nowadays, the majority of industrial software companies are dealing with projects involving high number of requirements, hard deadlines and high expectations in terms of efficiency and quality of the resulting software [24]. Thus, Model-Driven Engineering (MDE) is applied increasingly to cope with the complexity of software systems by raising the level of abstraction. To address the size of software systems, teams of developers have to cooperate and work in parallel on software models. Consequently, techniques to support building models collaboratively are highly required.

When models are changed in parallel, they have to be merged eventually to obtain a consolidated model. Therefore, several approaches have been proposed for detecting the operations that have been applied in parallel by developers. Once the applied operations are available, conflict detection algorithms are used to identify pairs of operations that interfere with each other (cf. [16] for a survey on model versioning approaches). In this regard, a conflict denotes a pair of operations, whereas one operation masks the effect of the other (i.e., they do not commute) or one operation disables the applicability of the other. An example for the former is a pair of parallel operations that update the same feature in the model with different values. The latter case is at hand if one operation's preconditions are not valid anymore after applying the operations of the other developer. Such a scenario frequently occurs if composite operations, e.g., model refactorings [4][28], are applied, because they have potentially complex preconditions that may easily be invalidated by parallel operations.

For resolving conflicts, empirical studies [27] showed that users prefer to work with a tentative merged model acting as a basis for reasoning about possible conflict resolutions, instead of working with the list of operations in terms of choosing to reject one or the other conflicting operation for creating a merged model. A few approaches respect this preference and produce a merged model by applying all non-conflicting operations; conflicting operations are omitted. However, especially in case of a large number of conflicts, many operations are not merged with this strategy, leading to a tentative merged model that lacks in reflecting the maximal combined effect of the parallel operations. Furthermore, the majority of existing work [1][2][3][12][13][14] treat the refactoring operations to merge with equal importance. However, in a real world scenario these operations have different importance and priority scores that can depend on the type of the refactoring and the context of the refactoring

application. Thus, in existing work the developer cannot integrate her/his priority preferences related to the importance of some refactorings that should be included in the merging process.

In this paper, we extended our previous work published at GECCO'13 [38] where a mono-objective genetic algorithm is proposed for model merging to minimize the number of conflicts. We propose in this paper, for the first time, a novel multi-objective formulation of the problem of model merging based on NSGA-II proposed by Deb et al. [39]. Our multi-objective approach aims to find the best trade-off between minimizing the number of conflicts and maximizing the number of successfully applied *important* operations. As mentioned already, an important kind of conflict denotes pairs of operations where one operation disables the applicability of the other. Whether one operation disables the other, however, often depends on the order in which they are applied. Thus, we aim at computing an operation sequence that minimizes the number of disabled operations among all parallel operations, including the conflicting ones. At the same time, our multi-objective formulation minimizes the number of disabled refactorings that are important. Our approach takes as input the initial model and the revised ones (i.e., the different parallel versions), a list of the applied operations, which is computed as described in previous work [25][26] and an importance score for each composite operation, and generates as output a set of merging solutions that reflect the best trade-off between the two conflicting objectives mentioned before. The importance score for each composite operation can be calculated manually or automatically. In our experiments, we calculated this importance score using quality metrics that formalize the complexity of a class that will be modified and a classification of the types of composite operations based on their complexity. In general, larger composite operations (combination of atomic and smaller composite operations) are more important than smaller ones.

The primary contributions of this paper are as follows:

- This paper introduces a novel formulation of the model merging problem as a multi-objective problem that takes into account the importance and priority of the proposed composite operation during the merging process. To the best of our knowledge, and based on recent search-based software engineering (SBSE) surveys [29][30], this is the first work to use multi-objective optimization for model merging.
- The paper reports on the results of an empirical study of our multi-objective proposal as applied to seven open source systems. We compared our approach to random

search, multi-objective particle swarm optimization (MOPSO) [40], mono-objective model merging (GECCO2013) [38] and a practical model merging technique [1]. The results provide evidence to support the claim that our proposal enables the generation of efficient model merging solutions to be comparable in terms of minimizing the number of conflicts to those suggested by existing approaches and to carry a high importance score of merged operations. Our results also revealed an interesting feature about the trade-off between the two conflicting objectives that demonstrates the practical value of taking the importance of operations into account in model merging tasks.

The remainder of this paper is structured as follows. Section 2 provides the background of model merging and demonstrates the challenges addressed in this paper based on a motivating example. In Section 3, we give an overview of our proposal and explain how we adapted the NSGA-II algorithm to find optimal operation sequences. Section 4 discusses the design and results of the empirical evaluation of our approach. After surveying related work in Section 5, we conclude with some pointers to future work in Section 6.

2 Model Merging Challenges

In this section, we briefly introduce the context, concepts, and challenges of model merging based on a motivating example.

2.1 Background on Model Merging

In general, two kinds of merge approaches can be distinguished [15]. First, *state-based* merge approaches aim at merging two model versions by combining their model elements into one merged model. Second, *operation-based* merge approaches in contrast do not reason about the models' states, but consider recorded operation histories and apply the combination of the parallel histories to the common initial version to compute the merged version.

For both approaches, the notion of conflict is essential, because when having two parallel evolutions of one model, not all operations may be combined to compute one unique merged model. Basically, we can distinguish between two kinds of conflicts. First, two operations are conflicting if one operation masks the effect of the other operation in the merged version: e.g., for update/update conflicts, the latter update in the operation sequence applied on the model is effective, while the former update is lost. Thus, such conflicting operations are not confluent: different operation sequences result in different models. Second, a conflict also occurs if one operation disables the applicability of the other. Every operation has specific preconditions,

e.g., an update of an element can only be performed when the element still exists; otherwise a delete/update conflict is raised.

Operation-based merge approaches [17] usually consider besides *atomic operations* (i.e, additions, deletions, and updates), also composite operations, such as model refactorings, which consist of a set of atomic operations and additional potentially more complex preconditions. For instance, in the case of model refactorings, certain conditions have to be fulfilled before a refactoring is applicable in order to preserve the semantics of a model after the refactoring has been applied. While in the operation history produced by one single developer, the preconditions of the operations are clearly fulfilled, this is not guaranteed when two operation histories from two different developers are combined. But this is exactly what is required to perform operation-based merging with the goal of maximizing the combined effect of the parallel operations.

A pure phasing-based approach is in general not solving this problem. For instance, applying first the operation history of developer A and afterwards the operation history of developer B, only the applicability of the operations of developer A is maximized. But there may be better solutions by intermingling the operation histories of developer A and B. Considering all possible permutations of two operation histories, we end up with a complexity of n! (where n is the amount of all applied operations of both developers). Considering the length of operation histories in practice, using an exhaustive approach is not feasible.

2.2 Model Merging Challenges: A Motivating Example

For making the model merging challenges more concrete, we make use of a motivating example. The starting point is the UML class diagram shown in Figure 1a. This version of a person management system has been subject to parallel evolution by two developers who concurrently applied a set of atomic and composite operations leading to the version shown in Figure 1b and 1c, respectively.

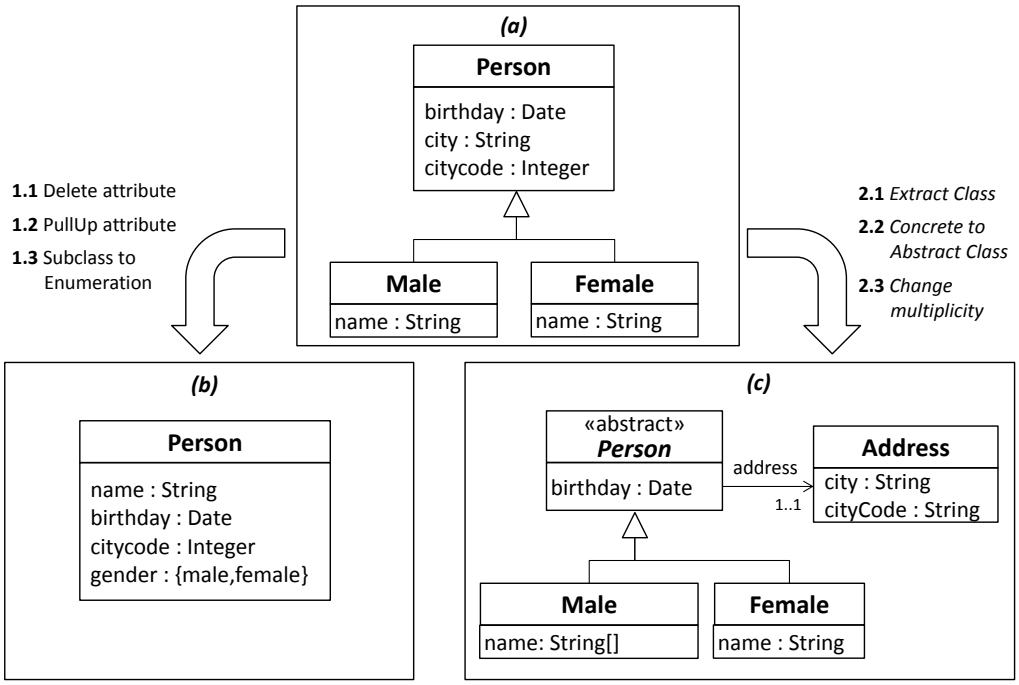

Figure 1: Parallel Evolution of a Class Diagram: (a) Initial Model v_0, (b) Revised Model v_{1a}, (c) Revised Model v_{1b}.

First, developer A deletes the *city* attribute of class Person, because she identifies that this attribute is redundant, as the information is already covered by *cityCode* attribute. Furthermore, developer *A* is applying the *Pull Up Attribute* refactoring in a second step: the *name* attribute of classes *Male* and *Female* are substituted by introducing this attribute to the class *Person*. This refactoring is represented by one composite operation which contains all atomic operations. The precondition of this refactoring is clearly fulfilled: both subclasses of class *Person* have the *name* attribute with the same property values, i.e., same data type and multiplicities. Finally, developer A applies another refactoring: substituting the now empty subclasses by an enumeration-typed attribute. The precondition for applying this refactoring is that the superclass receiving the enumeration-typed attribute is a concrete class and not an abstract one. The resulting model incorporating all mentioned changes is shown in Figure 1b.
Developer *B* applies the *Extract Class* refactoring to create an explicit class for the address information, and subsequently, the *Concrete to Abstract Class* refactoring. Again, these two refactorings are represented by composite operation consisting of several atomic operations. Please note that the *Concrete To Abstract Class* refactoring comprises a dedicated precondition that has to be fulfilled, namely the subject for this refactoring has to provide at

least one concrete subclass. Furthermore, developer B changes the upper bound cardinality of the name attribute of class Male from one to many by marking this attribute as an array. The resulting model is depicted in Figure 1c.

Now a naive operation-based merge approach may apply the operations of developer *A* to the initial model, and subsequently, on this intermediate version, the changes of developer *B*. The result of this merge strategy is illustrated in Figure 2a. However, in this sequence, the *Extract Class* refactoring is not applicable anymore, because the *city* attribute is already missing. Furthermore, the *Concrete to Abstract Class* refactoring is disabled as well due to the unsatisfied precondition for the intermediate version that is produced by applying the operations of developer A. Starting with the operations of developer *B* and continuing with the operations of developer *A*, also leads to two disabled operations: the *Pull Up Attribute* refactoring cannot be executed, because the *name* attributes in both subclasses have different cardinalities after applying the operations by developer B. Thus, the precondition of the refactoring is not fulfilled, and consequently, the operation cannot be applied. The same is true for the *Subclass to Enumeration* refactoring.

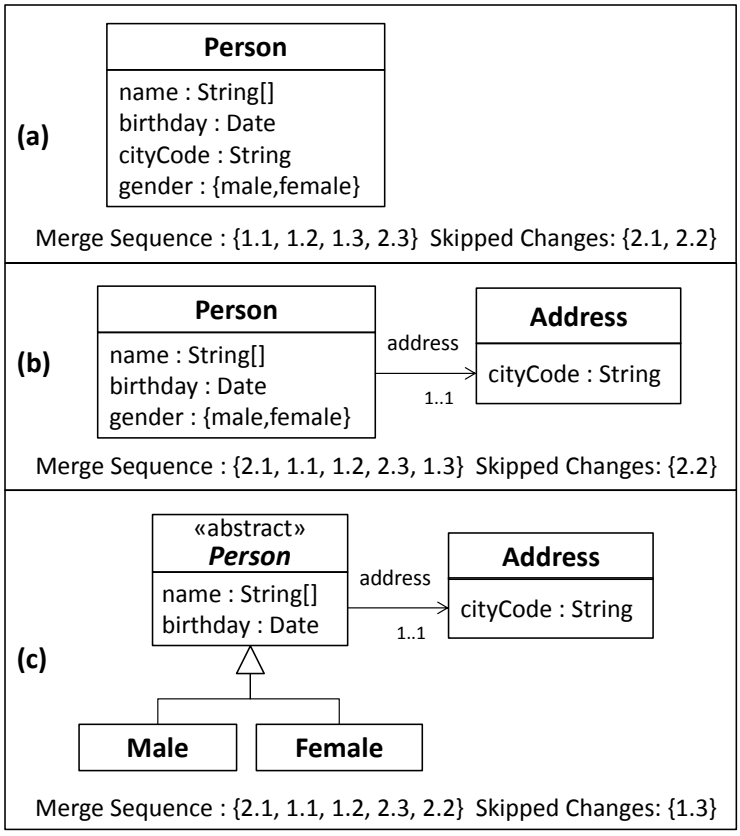

Figure 2: Partially Merged Solutions: (a) Pure phasing approach, (b) Operation interleaving and prioritizing *Subclasses to Enumeration* and (c) Operation interleaving and prioritizing *Concrete to Abstract Class.*

Now the question arises whether there are more appropriate solutions in terms of sequences of operations that enable constructing a merged model that maximizes the combined effect of both developers. In our example, 720 different operation sequences have to be considered in addition to the previous two. In fact, for our example, solutions exist that allow for applying *more operations successfully* by intermingling the operations of developer A and B than using a pure phasing approach. For instance, executing the *Extract Class* refactoring, deleting the *cityName* attribute, executing the *Pull Up Attribute* and *Subclass to Enumeration* refactorings, and finally, setting the upper bound cardinality represents one operation sequence (cf. Figure 2b) that contains more enabled operations as the pure phasing approach (cf. Figure 2a). However, one operation is disabled, namely Concrete to Abstract Class is in conflict with Subclass to Enumeration irrespective of the application sequence. Thus, the merge solution shown in Figure 2c contains also the successful application of five operations and one

disabled operation. Due to the fact that both operations are mutual exclusive, it is only possible to enable one of them. Selecting the operation that should be considered requires having a statement on the importance of the operation. One statement may be derived from the magnitude of an operation, i.e., the amount of atomic changes applied. In this case, the merge solution shown in Figure 2b may be more preferable.

It is important to stress that for six operations, an exhaustive based approach is applicable, but when doubling the number of operations, we already have to explore over 470 million combinations because $n!$ solutions exist where n is the number of operations. Thus, in the next section a scalable approach to solve this problem is presented that is able to consider the importance of operations, as well as to explore the sequences that maximize the application of operations.

3 Model Merging as a Multi-Objective Problem

We now describe our proposal and how we consider merging different model versions as a multi-objective optimization problem. We start by giving an overview of our approach and provide subsequently a more detailed description on how we adapted and used the NSGA-II algorithm for the problem of model merging.

3.1 Overview

The goal of our approach is to construct a tentative merged model that minimizes the number of disabled operations and maximizes the number of important enabled ones. Therefore, we use a multi-objective optimization algorithm to compute an optimal sequence of merging operations in terms of finding trade-offs between minimizing the number of operations that are disabled by preceding operations and maximizing the number of important operations that are enabled.

The general structure of our approach is sketched in Figure 3 The search-based process takes as inputs the sequences of operations that have been applied concurrently to a model by an arbitrary number of developers with an importance score for each composite operation. These sequences can be detected using operation detection algorithms presented in previous work [26]. Note that these sequences may also be obtained alternatively by tools that record operations directly in the modeling editor. The sequences are composed of operation applications, thereby each entry in a sequence states the operation type as well as the elements on which it has been applied. Having these sequences at hand, we may now combine them

into one common sequence of operations and compute the number of disabled operations. Therefore, we use composite operation specifications that contain explicitly specified preconditions in combination with a condition evaluation engine [12] to verify whether the preconditions of each operation in a sequence are fulfilled in a certain state of a model after the preceding operations in the sequence have been applied. If we determine an operation with invalid preconditions in a certain state of the model, we consider this operation to be disabled in the respective operation sequence.

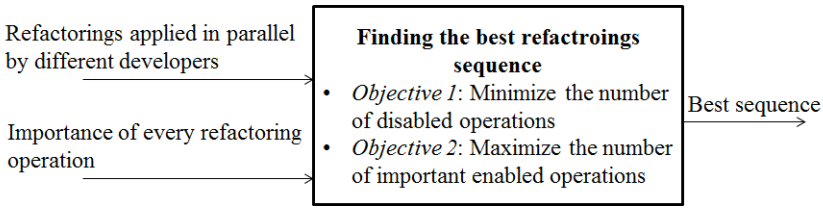

Figure 3. Multi-objective model merging: overview

The process of generating a solution can be viewed as the mechanism that finds the best order among all possible operation sequences that minimizes the number of disabled operations and maximizes the number of enabled important ones. The size of the search space is determined not only by the number of operations applied by the different developers on the same model, but also by the order in which they are applied. Due to the large number of possible refactoring sequences and the two conflicting objectives to optimize, we considered refactoring merging as a multi-objective optimization problem. In the next subsection, we describe the adaptation of NSGA-II [39] to our problem domain.

3.2 NSGA-II Adaptation for the Model Merging Problem

3.2.1 *Background definitions*

Before describing the adaptation of NSGA-II to the multi-objective model merging problem, some background definition related to multi-objective optimization should be given [41]:

Definition 1 (MOP). A multi-objective optimization problem (MOP) consists in minimizing or maximizing an objective function vector under some constraints. The general form of a

MOP is as follows:

$$\begin{cases} Min\, f(x) = [f_1(x), f_2(x), \ldots, f_M(x)]^T \\ g_j(x) \geq 0 & j = 1, \ldots, P; \\ h_k(x) = 0 & k = 1, \ldots, Q; \\ {x_i}^L \leq x_i \leq {x_i}^U & i = 1, \ldots, n. \end{cases} \quad (1)$$

where M is the number of objective functions, P is the number of inequality constraints, Q is the number of equality constraints, x_i^L and x_i^U correspond to the lower and upper bounds of the variable x_i. A solution x_i satisfying the (P+Q) constraints is said feasible and the set of all feasible solutions defines the feasible search space denoted by Ω. In this formulation, we consider a minimization MOP since maximization can be easily turned to minimization based on the duality principle by multiplying each objective function by -1. The resolution of a MOP consists in approximating the whole Pareto front.

Definition 2 (Pareto optimality). A solution $x^* \in \Omega$ is Pareto optimal if $\forall x \in \Omega$ and $I = \{1, \ldots, M\}$ either $\forall\, m \in I$ we have $f_m(x) = f_m(x^*)$ or there is at least one $m \in I$ such that $f_m(x) > f_m(x^*)$.

The definition of Pareto optimality states that x^* is Pareto optimal if no feasible vector x exists which would improve some objective without causing a simultaneous worsening in at least another one. Other important definitions associated with Pareto optimality are essentially the following:

Definition 3 (Pareto dominance). A solution $u = (u_1, u_2, \ldots, u_n)$ is said to dominate another solution $v = (v_1, v_2, \ldots, v_n)$ (denoted by $f(u) \preceq f(v)$) if and only if $f(u)$ is partially less than $f(v)$. In other words, $\forall\, m \in \{1, \ldots, M\}$ we have $f_m(u) \leq f_m(v)$ and $\exists\, m \in \{1, \ldots, M\}$ where $f_m(u) < f_m(v)$.

Definition 4 (Pareto optimal set). For a MOP $f(x)$, the Pareto optimal set is $P^* = \{x \in \Omega \mid \neg \exists\, x' \in \Omega, f(x') \preceq f(x)\}$.

Definition 5 (Pareto optimal front). For a given MOP $f(x)$ and its Pareto optimal set P^*, the Pareto front is $PF^* = \{f(x), x \in P^*\}$.

3.2.2 *NSGA-II*

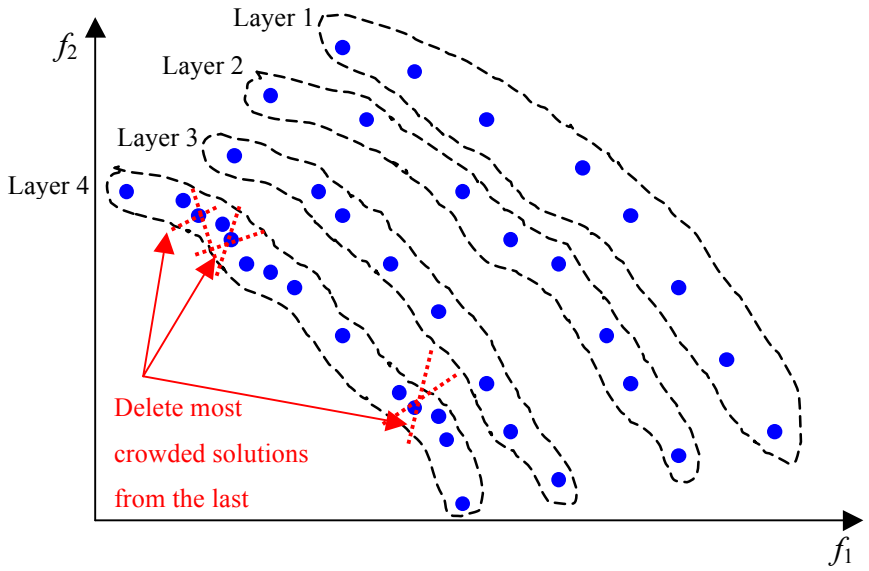

Figure 4. NSGA-II replacement scheme.

NSGA-II [39] is one of the most used multi-objective evolutionary algorithms (EAs) in tackling real world problems including SE ones [29][30]to find trade-offs between different objectives simultaneously. It begins by generating an offspring population from a parent one by means of variation operators (crossover and mutation) such that both populations have the same size. After that, it ranks the merged population (parents and children) into several non-dominated layers, called fronts, as depicted by figure 4. Non-dominated solutions are assigned a rank of 1 and constitute the first layer. Non-dominated solutions according to the population truncated of the layer 1 are assigned rank of 2 and constitute the layer 2. This process is continued until the ranking of all parent and children individuals. After that, each solution is assigned a diversity score, called crowding distance, frontwise. This distance corresponds to the half of the perimeter of the cuboid having the two closest neighboring solutions to the considered individual as vertices. It is important to note that extreme solutions are assigned an infinite crowding score since they are of great importance for diversity. The fitness in NSGA-II is not a scalar value. In fact, it is a couple (rank, crowding distance). Solutions having better ranks are emphasized. Among solutions having the same rank (belonging to the same layer), solutions having larger crowding distances are emphasized since they are less crowded than the others. Once all individuals of the merged population are assigned a rank and a diversity score, we perform the environmental selection to form the parent population for the next generation. Indeed, solutions belonging to the best layers are selected. Figure 4 illustrates this

process where the last selected layer is the 4th one. Usually, the cardinality of the last layer (layer 4 in figure 4) is greater than the number of available slot in the parent population of the next generation. As denoted by figure 4, solutions of the 4th layer are selected based on their crowding distance values. In this way, most crowded solutions are discouraged to remain in the race; thereby emphasizing population diversification. To sum up, the Pareto ranking encourages convergence and the crowding factor procedure emphasizes diversity, therefore NSGA-II is an elitist multi-objective EA which is today widely used to address real-world problems including search-based software engineering ones (SBSE).

3.2.3 *Problem Formulation*

The model merging problem involves searching for the best sequence of applying composite operations among the set of candidate ones, which constitutes a huge search space. A solution is a sequence of operations where the goal of applying the sequence to a software system *S* is typically to minimize the number of disabled refactorings and maximize the number of important enabled ones. In a real-world setting, operations that are applied by different developers cannot be treated with an equal importance. The operations introduced in parallel can be related to new requirements/features, fixing bugs, or improve the quality [23]. In addition, these operations are applied to different classes that are not equally important in terms of size, number of features implemented, number of methods, etc. In general, the size of the class is important to determine its importance [28]. Furthermore, a team manager can have different preferences and priorities for the new release such as including some important features or fixing bugs. Thus, it is important to prioritize some operations that can stratify the preferences of managers during the merging process. However, most of existing work did not take into consideration this aspect/objective when merging parallel software changes.

We propose, in this paper, a multi-objective formulation of the model merging problem that takes the importance of the refactoring operations to merge into account. Consequently, we have two objective functions in our problem formulation: (1) minimize the number of disabled refactorings, and (2) maximize the number of important enabled refactorings. Analytically speaking, the formulation of the model merging problem can be stated as follows:

$$\begin{cases} Minimize f_1(S) = ndo \\ Maximize f_2(S) = \sum_{i=1}^{nb\_enabled\_ref} Importance(x_i) \end{cases}$$

where *ndo* is the number of disabled operations, x_i is the *i*-th enabled operations in the sequence *S*, and *Importance*(x_i) is the importance of the operation x_i in *S*.

The goal is to define an efficient and simple—in the sense that it is not computationally expensive—fitness function in order to reduce the computational complexity. This function should evaluate the number of disabled operations. Therefore, we use previously developed tools to specify composite operations including their preconditions in combination with an engine for evaluating the conditions in a certain model state (cf. [26][38] for details). As evaluating conditions can be rather expensive, we only compute whether one operation in a sequence disables another for each possible pair-wise combination of operations in advance, instead of checking the preconditions of each operation with the combined effect of every operation that precedes the operation in the sequence. Please note that this might miss to detect some disabled operations in certain scenarios: the preconditions of an operation might be valid with each *single* preceding operation, but the preceding operations *in combination* might still invalidate the preconditions of the subsequent operations. As this is a rather rare case, we left this issue as a topic of future work.

The information on which operation in a sequence disables the other is represented in terms of a matrix $n \times n$ where *n* is the number of operations applied originally by the different developers in total (after eliminating duplicates). Each item in this matrix represents a combination of two operations and holds a value of either 0 or 1: if an operation *i* disables the operation *j* then, the item (*i,j*) in the matrix takes the value 1, otherwise it takes 0. Based on this matrix, we may determine easily the number of disabled operations for a specific operation sequence by summing up all values in the matrix.

We define the refactoring importance as follows:

$$Importance(x_i) = \text{type\_importance}(\mathrm{x_i}) + \sum_{j=1}^{numberMetrics} M_j(c_i)$$

such that M_j represents a software metric and c_i is the class where the refactoring x_i will be applied. Thus, we defined the class importance using a set of software metrics. In our

experiments, we considered the following metrics: Weighted Methods per Class (WMC), Response for a Class (RFC), Lack of Cohesion of Methods (LCOM), Cyclomatic Complexity (CC), Number of Attributes (NA), Attribute Hiding Factor (AH), Method Hiding Factor (MH), Number of Lines of Code (NLC), Coupling Between Object Classes (CBO), Number of Association (NAS), Number of Classes (NC), Depth of Inheritance Tree (DIT), Polymorphism Factor (PF), Attribute Inheritance Factor (AIF) and Number of Children (NOC).

We use a model that considers two categories of composite operations: an operation can be a Low-Level Operation (LLO) or a High-Level Operations (HLO). HLO is composed by the combination of two or more operations and LLO is an elementary operation. We classify the following basic operations in the LLO category: Create_class, delete_method, add_field, move_method, rename_method, create_relationship, etc. The HLO or complex refactoring are Extract_class, Extract_subclass, Pull up method, Push down Field, etc. In fact, the application of these complex refactorings include the execution of some LLO operations. For example, *extract_class* includes *create_new_class*, *move_methods*, etc. In our experiments, we defined the *type_importance* measure as follows:

$$\text{type\_importance}(x_i) = \begin{cases} 1, \text{if } x_i \in \text{HLO} \\ 0.5, \text{if } x_i \in \text{LLO} \end{cases}$$

There are of course many ways in which the importance of classes and code changes could be measured, and one of the advantages of the search-based approach is that this definition could be easily replaced with a different one. In summary, the basic idea behind this work is to take into account the importance of the operations to merge while maximizing simultaneously the number of enabled refactorings. These two objectives are in conflict. Thus, the goal is to find a good compromise. In fact, once the bi-objective trade-off front is obtained, the user can navigate through this front in order to select his/her preferred merging solution. This is achieved through sacrificing some degree of number of enabled refactorings while gaining in terms of including important ones in the merging process.

3.2.4 *Solution Approach*

The solution approach proposed in this paper lies within the SBSE field. We use the well-known multi-objective evolutionary algorithm NSGA-II [39] to try to solve the model merging problem. As noted by Harman et al. [29][30], a generic algorithm like NSGA-II

cannot be used ‘out of the box’ – it is necessary to define problem-specific genetic operators to obtain the best performance. To adapt NSGA-II to our problem, the required steps are to create: (1) solution representation, (2) solution variation and (3) solution evaluation. We examine each of these in the coming paragraphs.

Solution representation. To represent a candidate solution (individual), we use a vector containing all operations that have been applied by the developers in parallel, whereas each item in the vector represents a single operation (with links to the elements to which it is applied) and the order of operations in this vector represents the sequence in which the operations are applied. Please note that some operations can be eliminated in case they are equivalent; that is, two developers applied the same operation to the same model elements. Thus, we exclude duplicates. Consequently, all vectors, each representing one candidate solution, have the same number of dimensions that corresponds to the number of all parallel operations applied by all developers.

Figure 5 depicts a possible population of operation sequences for the running example introduced above, whereas *R*** refers to the label of the respective refactoring. For instance, the solutions represented in Figure 5 are composed of five dimensions corresponding to five operations proposed by two different developers. All the solutions have the same length, but they constitute a different order.

The proposed algorithm first generates a population randomly from the list of all operations. Second, crossover and mutation operators are used to generate new populations in the next iterations as explained in the following.

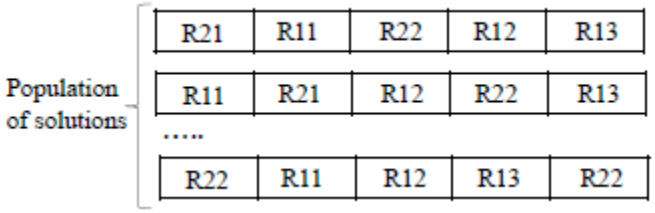

Figure 5. Population of operation sequences

Solution variation. In a search algorithm, the variation operators play the key role of moving within the search space with the aim of driving the search towards better solutions. We used the principle of the Roulette wheel [39] to select individuals for mutation and crossover. The probability to select an individual for crossover and mutation is directly proportional to its relative fitness in the population. In each iteration, we select *population_size / 2* individuals

from the population pop_i to form population pop_i+1. These (*population_size / 2*) selected individuals will "give birth" to another (*population_size / 2*) new individuals using a crossover operator. Therefore, two parent individuals are selected, and a sub-vector is picked on each one. Then, the crossover operator swaps the dimensions and their relative sub-vector from one parent to the other. When applying the crossover, we ensure that the length of the vector remains the same. The crossover operator allows creating two offspring *p'1* and *p'2* from the two selected parents *p1* and *p2*. It is defined as follows: a random position, *k*, is selected. The first *k* operations of *p1* become the first *k* elements of *p'1*. Similarly, the first *k* operations of *p2* become the first *k* operations of *p'2*.

The mutation operator can be applied to pairs of dimensions selected randomly. Given a selected solution, the mutation operator first randomly selects one or many pairs of dimensions of the vector. Then, for each selected pair, the dimensions are swapped.

Solution evaluation. The solution is evaluated based on the two objective functions defined in the previous section. Since we are considering a bi-objective formulation, we use the concept of Pareto optimality to find a set of compromise (Pareto-optimal) merging solutions. By definition, a solution *x* Pareto-dominates a solution *y* if and only if *x* is at least as good as *y* in all objectives and strictly better than *y* in at least one objective. The fitness of a particular solution in NSGA-II corresponds to a couple (*Pareto Rank*, *Crowding distance*). In fact, NSGA-II classifies the population individuals (of parents and children) into different layers, called non-dominated fronts. Non-dominated solutions are assigned a rank of 1 and then are discarded temporary from the population. Non-dominated solutions from the truncated population are assigned a rank of 2 and then discarded temporarily. This process is repeated until the entire population is classified with the domination metric. After that, a diversity measure, called *crowding distance*, is assigned front-wise to each individual. The crowding distance is the average side length of the cuboid formed by the nearest neighbors of the considered solution. Once each solution is assigned its Pareto rank, based on the values of the fitness functions, in addition to its crowding distance, mating selection and environmental selection are performed. This is based on the crowded comparison operator that favors solutions having better Pareto ranks and, in case of equal ranks, it favors the solution having larger crowding distance. In this way, convergence towards the Pareto optimal bi-objective

front (importance, number of enabled refactoring) and diversity along this front are emphasized simultaneously.

The basic iteration of NSGA-II consists in generating an offspring population (of size N) from the parent one (of size *N* too) based on variation operators (crossover and mutation) where the parent individuals are selected based on the crowded comparison operator. After that, parents and children are merged into a single population *R* of size 2*N*. The parent population for the next generation is composed of the best non-dominated fronts. When including the last front, there are usually not enough available slots for all its members. Hence, based on the crowded comparison operator, solutions having largest crowding distances from the last front are included in the new parent population and the remaining ones are discarded. This process continues until the satisfaction of a stopping criterion. The output of NSGA-II is the last obtained parent population containing the best of the non-dominated solutions found. When plotted in the objective space, they form the Pareto front from which the user will select his/her preferred merging solution.

4 Empirical Evaluation

This section describes our empirical study to evaluate our multi-objective model merging proposal including the research questions to address, the seven open source systems examined, evaluation metrics considered in our experiments and the statistical tests results.

4.1 Research Questions

We defined six research questions that address the applicability, performance in comparison to existing merging approaches, and the usefulness of our robust multi-objective merging. The six research questions are as follows:

RQ1: Search Validation. To validate the problem formulation of our approach, we compared our NSGA-II formulation with Random Search. If Random Search (RS) outperforms an intelligent search method thus we can conclude that our problem formulation is not adequate.

RQ2. To what extent can the proposed approach reduce the number of disabled operations and maximize the number of enabled important operations? It is important to evaluate the performance of our model merging approach, based on NSGA-II, when applied to real-world scenarios.

The next four questions are related to the comparison between our proposal and the state-of-the-art model merging approaches.

RQ3.1: How does NSGA-II perform compared to another multi-objective algorithm? It is important to justify the use of NSGA-II for the problem of model merging. We compare NSGA-II with another widely used multi-objective algorithm, MOPSO (Multi-Objective Particle Swarm Optimization), [40] using the same fitness functions.

RQ3.2: How does our multi-objective model merging formulation performs compared to a mono-objective one? A multi-objective algorithm provides a trade-off between the two objectives where developers can select their desired merging solution from the Pareto-optimal front. A mono-objective approach uses a single fitness function that is formed as an aggregation of both objectives and generates as output only one merging solution. This comparison is required to ensure that the solutions provided by NSGA-II and MOPSO provide a better trade-off between the two objectives than a mono-objective approach. Otherwise, there is no benefit to our multi-objective adaptation.

RQ3.3: How does NSGA-II perform compared to existing search-based model merging approaches? Our proposal is the first work that treats the problem of model merging using a multi-objective approach. However, in our previous work [38], we used a mono-objective genetic algorithm for model merging using only one objective which is minimizing the number of conflicts.

RQ3.4: How does NSGA-II perform compared to existing model merging approaches not based on the use of metaheuristic search? While it is very interesting to show that our proposal outperforms existing search-based merging approaches, developers will consider our approach useful, if it can outperform other existing model merging tools [1] that are not based on optimization techniques where the operations are applied as they arrive without trying the find the best sequence/order of applying them.

RQ4: Can our multi-objective approach for model merging be useful for software engineers in real-world setting? In a real-world problem, it is important to show that it is useful to consider the importance/priority score related to each refactoring when merging models. Some scenarios are required to illustrate the importance of the use of a multi-objective approach for model merging in a real-world setting.

4.2 Experimental Setup

4.2.1 Systems Studied

We chose to analyze the extensive evolution of three Ecore metamodels coming from the Graphical Modeling Framework (GMF) [8], an open source project for generating graphical

modeling editors. We considered the evolution from GMF's release 1.0 over 2.0 to release 2.1 covering a period of two years. For achieving a broader data basis, we analyzed the revisions of three models, namely the Graphical Definition Metamodel (GMF Graph), the Generator Metamodel (GMF Gen), and the Mappings Metamodel (GMF Map). Therefore, the respective metamodel versions had to be extracted from GMF's version control system and, subsequently, manually analyzed to determine the actually applied operations between successive metamodel versions. In addition to GMF, we used model fragments extracted from three open source projects: GanttProject (Gantt for short) [9], JHotDraw [10], ApacheAnt [42] and Xerces-J [11]. We considered the evolution across three versions of Gantt (v1.7, v1.8, and v1.9.10), three versions of JHotDraw (v5.1, v5.2, and v5.3), four versions of ApacheAnt (v1.6.1, v1.6.2, v1.6.3 and v1.6.4) and four versions of Xerxes-J (v1.4.4, v2.5.0, v2.6.0, and v2.6.1). Xerces-J is a family of software packages for parsing XML. GanttProject is a cross-platform tool for project scheduling. ApacheAnt is a build tool and library specifically conceived for Java applications. Finally, JHotDraw is a GUI framework for drawing editors. Table 1 summarizes for each model evolution scenario the number of applied refactorings, as well as the number of model elements for the smallest and largest model version.

Additionally, we had to specify all operation types (i.e., their comprised atomic operations and preconditions) that have been applied across all versions leading to 38 different types of operations. The evolution of the analyzed models provides a relatively large set of revisions containing overall 659 different applications of the operation types as shown in Table 1.

Due to the lack of existing parallel revision histories that we could have used for evaluating our approach, we emulate parallel evolution by dividing the applied operations from the single revisions into parallel sequences of operations manually and asked five graduate students to additionally modify different model fragments of these open source systems in order to cause disabled operations in the considered evolutions.

Table 1. Systems studied in our experiments.

Model	ɔer of refactorings	f elements (min, max)
GMF Map	14	367, 428
GMF Graph	36	277,310
GMF Gen	112	883,1295
GanttProject	72	451, 572
Xerces-J	86	1698,1732
JHotDraw	81	985, 1457

ApacheAnt	258	2489, 2166

4.2.2 Performance Indicators

When comparing two mono-objective algorithms, it is usual to compare their best solutions found so far during the optimization process. However, this is not applicable when comparing two multi-objective evolutionary algorithms since each of them gives as output a set of non-dominated (Pareto equivalent) solutions. For this reason, we use the three following performance indicators [44] when comparing NSGA-II and MOPSO:

−*Hypervolume* (*IHV*) corresponds to the proportion of the objective space that is dominated by the Pareto front approximation returned by the algorithm and delimited by a reference point. Larger values for this metric mean better performance. The most interesting features of this indicator are its Pareto dominance compliance and its ability to capture both convergence and diversity. The reference point used in this study corresponds to the nadir point [43] of the Reference Front (*RF*), where the Reference Front is the set of all non-dominated solutions found so far by all algorithms under comparison.

−*Inverse Generational Distance* (*IGD*) is a convergence measure that corresponds to the average Euclidean distance between the Pareto front Approximation *PA* provided by the algorithm and the reference front *RF*. The distance between *PA* and *RF* in an *M*-objective space is calculated as the average *M*-dimensional Euclidean distance between each solution in *PA* and its nearest neighbour in *RF*. Lower values for this indicator mean better performance (convergence).

−*Contribution* (*IC*) corresponds to the ratio of the number of non-dominated solutions the algorithm provides to the cardinality of *RF*. Larger values for this metric mean better performance.

We note that the mono-objective EA provides only one refactorings merging solution, while NSGA-II and MOPSO generate a set of non-dominated solutions. In order to make meaningful comparisons, we select the best solution for NSGA-II and MOPSO using a knee point strategy [43]. The knee point corresponds to the solution with the maximal trade-off between minimizing the number of conflicts and maximizing the number of enabled important operations. Hence moving from the knee point in either direction is usually not interesting for the user. Thus, for NSGA-II and MOPSO, we select the knee point from the Pareto approximation having the median IHV value. We aim by this strategy to ensure

fairness when making comparisons against the mono-objective EA. For the latter, we use the best solution corresponding to the median observation on 51 runs. We use the trade-off "worth" metric proposed by Rachmawati and Srinivasan [44] to find the knee point. This metric estimates the worthiness of each non-dominated merging solution in terms of trade-off between our two conflicting objectives. After that, the knee point corresponds to the solution having the maximal trade-off "worthiness" value. We used these metrics mainly to compare between mono-objective and multi-objective approaches defined as follows:

−*Number of disabled refactoring operations (NDR)*.

−*Importance of enabled refactoring operations* (*IER*) is defined as follows:

$$IER(S) = \sum_{i=1}^{k} \text{importance}(R_i)$$

where *importance* is as defined in Section 3.2.

−*Automatic correctness (AC)* consists of comparing the suggested operation order to the expected one, operation by operation. AC method has the advantage of being automatic and objective. However, since different possibilities exist to avoid disabled operations, AC could reject a good solution because it yields different operation sequences from the original one. To account for those situations, we also use *manual correctness (MC)* for evaluating the correctness of the proposed sequence, again operation by operation. When calculating MC, we verify whether the proposed operation sequence preserves the semantics of the design (and not only avoids disabled operations).

−*Computational time* (*ICT*) is a measure of efficiency employed here since more than one objective may cause the search to use more time in order to find a set of Pareto-optimal trade-offs.

4.2.3 Statistical Tests

Since metaheuristic algorithms are stochastic optimizers, they can provide different results for the same problem instance from one run to another. For this reason, our experimental study is performed based on 51 independent simulation runs for each problem instance and the obtained results are statistically analyzed by using the Wilcoxon rank sum test [45] with a 95% confidence level (α = 5%). The latter verifies the null hypothesis H_0 that the obtained results of two algorithms are samples from continuous distributions with equal medians, as against the alternative that they are not, H_1. The p-value of the Wilcoxon test corresponds to the probability of rejecting the null hypothesis H_0 while it is true (type I error). A p-value that

is less than or equal to α (≤ 0.05) means that we accept H_1 and we reject H_0. However, a p-value that is strictly greater than α (> 0.05) means the opposite. In fact, for each problem instance, we compute the p-value of random search, MOPSO and mono-objective search results with NSGA-II ones. In this way, we could decide whether the outperformance of NSGA-II over one of each of the others (or the opposite) is statistically significant or just a random result.

To answer the first research question RQ1 an algorithm was implemented where merging solutions were randomly generated (random sequence) at each iteration. The obtained Pareto fronts were compared for statistically significant differences with NSGA-II using *IHV*, *IGD* and *IC*.

To answer RQ2, we evaluated the average of *NDR, IER, AC* and *MC* scores for non-dominated merging solutions proposed by NSGA-II. To answer RQ3.1 we implemented a widely used multi-objective algorithm, namely multi-objective particle swarm optimization (MOPSO) and we compared NSGA-II and MOPSO using the same quality indicators used in RQ1. In addition, we used boxplots to analyze the distribution of the results and discover the knee point (best trade-off between the objectives).

To answer RQ3.2 we implemented a mono-objective Genetic Algorithm (Agg-GA) [5] where one fitness function is defined as an average of the two objectives. The multi-objective evaluation measures (*IHV*, *IGD* and *IC*) cannot be used in this comparison thus we considered the *NDR, IER, AC* and *MC* metrics. To answer RQ3.3 we compared NSGA-II with our previous mono-objective work for model merging [38] where the importance/priority of operations is not taken into account. We considered different metrics for the comparison such as *NDR, IER, AC* and *MC*. To answer RQ3.4 we used an existing model merging tool [1] not based on optimization techniques where the operations are applied as they arrive without trying the find the best sequence/order of applying them. We compared the results of this tool with NSGA-II using *NDR, IER, AC* and *MC* metrics since only one solution can be proposed by [1] and not a set of "non-dominated" solutions.

To answer the last question (RQ4) we discussed on how the shape of the Pareto front can help developers to select the best merging solution based on their preferences.

4.2.4 Parameter Setting and Tuning

Parameter setting has a significant influence on the performance of a search algorithm on a particular problem instance. For this reason, for each algorithm and for each system (cf. Table

2), we perform a set of experiments using several population sizes: 50, 100, 200, 300 and 500. The stopping criterion was set to 100000 evaluations for all algorithms in order to ensure fairness of comparison. The MOPSO used in this paper is the Non-dominated Sorting PSO (NSPSO) proposed by Li [40]. The other parameters' values were fixed by trial and error and are as follows: (1) crossover probability = 0.8; mutation probability = 0.5 where the probability of gene modification is 0.3; stopping criterion = 100000 evaluations. For MOPSO, the cognitive and social scaling parameters c_1 and c_2 were both set to 2.0 and the inertia weighting coefficient w decreased gradually from 1.0 to 0.4.

Table 2. Best population size configurations.

System	**A-II**	**PSO**	**-GA**	**A**	**S**
MF Map	)0	)0	)0	0	0
MF Graph	)0	)0	)0	0	0
MF Gen	)0	)0	)0	0	0
nttProject	)0	)0	)0	0	0
erces-J	)0	)0	)0	0	0
lotDraw	)0	)0	)0	0	0
acheAnt	)0	)0	)0	0	0

4.3 Results Analysis

This section describes and discusses the results obtained for the different research questions of Section 4.1.

4.3.1 Results for RQ1

We do not dwell long in answering the first research question (RQ1) that involves comparing our approach based on NSGA-II with random search. The remaining research questions will reveal more about the performance, insight, and usefulness of our approach.

Table 3 confirms that both multi-objective algorithms NSGA-II and MOPSO are better than random search based on the three quality indicators IHV, IGD and IC on all the seven open source systems. The Wilcoxon rank sum test showed that in 51 runs both NSGA-II and MOPSO results were significantly better than random search. In addition, Tables 4 and 5, and Figures 6 and 7 confirm the outperformance of NSGA-II and MOPSO comparing to random search in terms of number of disabled refactoring, importance of enabled refactoring and automatic and manual correctness. In fact, random search is not efficient to generate good merging solutions using all the above metrics in 100% of the experiments. Thus, an intelligent algorithm is required to find good trade-offs to propose efficient merging solutions. We conclude that there is empirical evidence that our multi-objective formulation surpasses the

performance of random search thus our formulation is adequate and the use of metaheursitic search is justified (this answers RQ1).

The significantly best algorithm among random search, NSGA-II and MOPSO diff. means that NSGA-II and MOPSO are significantly better than random not statistically different).

System	*IHV*	*IGD*	*IC*
GMF Map	NSGA-II	NSGA-II	SGA-II
GMF Graph	NSGA-II	*ot sign. diff.*	*sign. diff.*
GMF Gen	*Not sign. diff.*	*MOPSO*	*MOPSO*
GanttProject	NSGA-II	NSGA-II	SGA-II
Xerces-J	NSGA-II	NSGA-II	SGA-II
JHotDraw	NSGA-II	NSGA-II	SGA-II
ApacheAnt	NSGA-II	NSGA-II	SGA-II

4.3.2 Results for RQ2

In this section, we evaluate the performance of NSGA-II to find good trade-offs between the two objectives of minimizing the number of disabled refactorings and maximizing the number of important enabled ones. Table 4 confirms that our NSGA-II adaptation was successful in generating merging sequences that minimize the number of disabled refactorings. In fact, the highest number of disabled refactorings is 26 (ApacheAnt) and the lowest one is 3 (GMF Map). The number of disabled refactorings seems very reasonable if we consider the high number of refactorings, more than 250, to merge for ApacheAnt.

Table 4. Median number of disabled refactorings on 51 independent runs. The results were statistically significant on 31 independent runs using the Wilcoxon rank sum test with a 99% confidence level ($\alpha < 1\%$).

stems	*SGA-II*	*OPSO*	*RS*	*ggGA*	*GA[38]*	*ractical [1]*
F Map	3				3	8
F Graph	6				5	16
IF Gen	6				6	49
tProject	21				9	33
rces-J	2				2	29
tDraw	1				1	37
cheAnt	26				23	71

Table 5 shows that NSGA-II provides merging solutions that not only minimize the number of disabled operations, as described in Table 4, but also maximize the number of important enabled ones. This confirms that most of the disabled refactorings are not very important and it is interesting that the importance of the operations can help the algorithm which operation

to disable when conflicts are detected. The highest importance score of enabled refactorings is 471.6 for ApacheAnt and the lowest importance score is 28.6 for GMF Map. An interesting observation that the highest and lowest number of disabled refactorings are also found in the same systems Ant-Apache and GMF Map thus it is evident that NSGA-II finds the best trade-off between our two conflicting objectives. This is can be explained by the fact that the algorithm tried to disabled operation with the lowest importance when a conflict is detected to optimize simultaneously both objectives.

Table 5. Median importance scores of enabled refactoring on 51 independent runs. The results were statistically significant on 31 independent runs using the Wilcoxon rank sum test with a 99% confidence level ($\alpha < 1\%$).

stems	*GA-II*	*PSO*	*S*	*gGA*	*GA[38]*	*actical [1]*
F Map	3.6	4		4	3.8	17.4
F Graph	9.4	1	5	8	9.6	18.1
F Gen	4.5	4	8	.1	8.4	01.8
tProject	8.3	2	2	3	1.4	78.6
rces-J	9.1	4		1	6.3	81.3
tDraw	2.8	7	4	.4	4.5	80.6
cheAnt	1.6	2	6	.4	8.6	83.4

Since it is not sufficient to evaluate the performance of our NSGA-II adaptation using metrics similar to the fitness functions, we considered two other metrics related to automatic and manual correctness. Figure 6 shows that the suggested merging solutions using NSGA-II are similar to the "reference" solution provided manually by developers with more 78% for all the 7 systems. However, it is a fastidious process for developers to manually merge parallel operations thus sometimes better solutions can be provided by our automated merging algorithm. A deviation with the set of reference solutions did not means sometimes that there is an error/conflict with our multi-objective solutions but it could be another possible good merging solution different from the reference ones. To this end, we evaluated the suggested solutions by NSGA-II manually and we calculated a manual correctness score. Figure 7 confirms that most of the suggested merging solutions are good with an average of more than 85% in all the seven systems. This manual validation provides strong evidence that our merging solutions make sense semantically and could be applied to provide coherent model versions/releases.

To better evaluate the performance of our NSGA-II adaptation for the problem of model

merging, we compare in the next section its performance with existing model merging techniques.

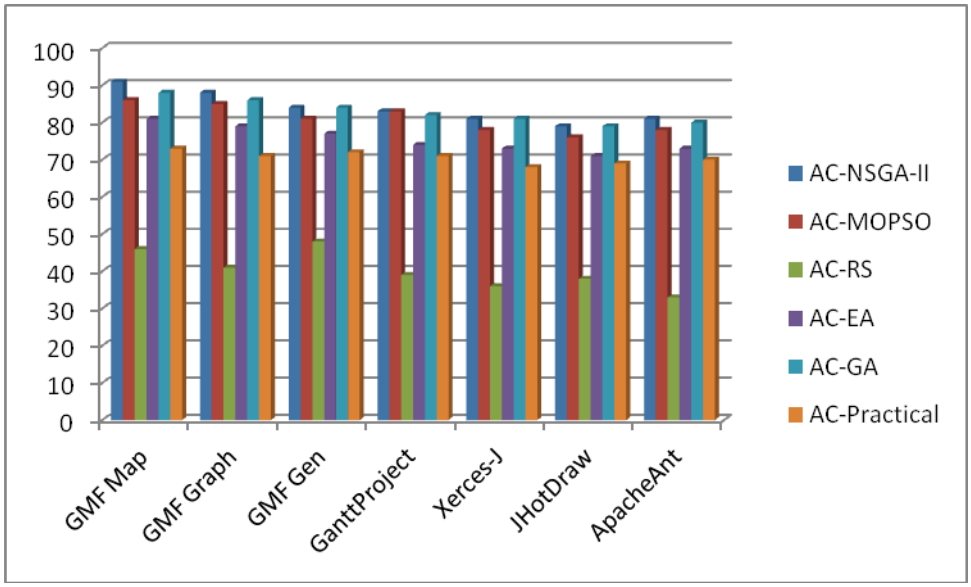

Figure 6. Automatic correctness median values on 51 independent runs for the case studies

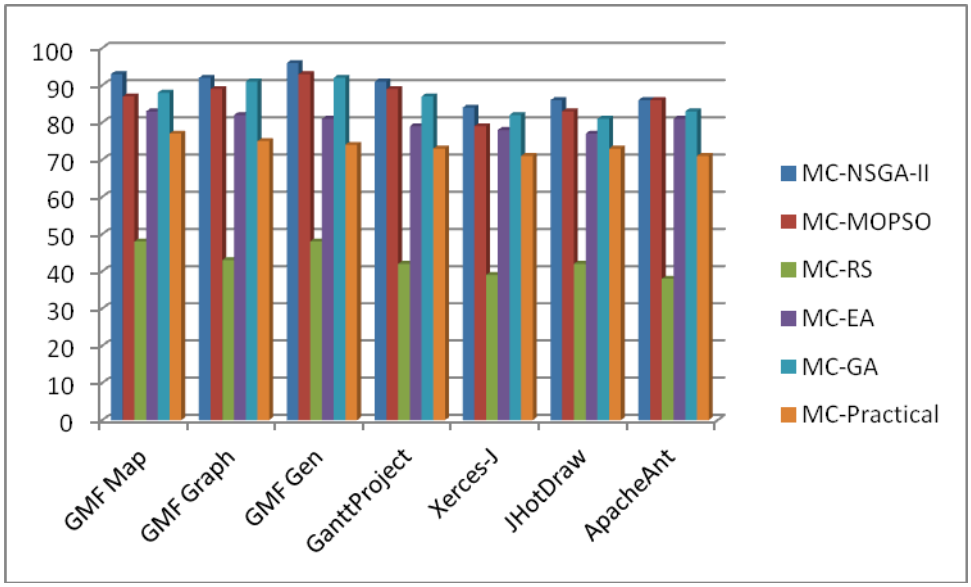

Figure 7. Manual correctness median values on 51 independent runs for the case studies

4.3.3 Results for RQ3

In this section, we compare our NSGA-II adaptation to the current, state-of-the-art model merging approaches. To answer RQ3.1, we compared NSGA-II to another widely used multi-objective algorithm, MOPSO, using the same adapted fitness function. Table 3 shows the overview of the results of the significance tests comparison between NSGA-II and MOPSO.

NSGA-II outperforms MOPSO in most of the cases: 17 out of 21 experiments (81%). MOPSO outperforms the NSGA-II approach only in GMF Gen, which is one of the smallest open source system considered in our experiments, having a low number of operations to merge, so it appears that MOPSO's search operators make a better task of working with a smaller search space. In particular, NSGA-II outperforms MOPSO in terms of IHV values in 6 out 7 experiments with one 'no significant difference' result. Regarding IGD and IC, NSGA-II outperformed MOPSO in 5 out of 7 experiments, where only one case was not statistically significant, namely GMF Graph and one case where MOPSO provides better results namely GMF Gen.

A more qualitative evaluation is presented in Figure 8 illustrating the box plots obtained for the multi-objective metrics on the different projects. We see that for almost all problems the distributions of the metrics values for NSGA-II have smaller variability than for MOPSO. This fact confirms the effectiveness of NSGA-II over MOPSO in finding a well-converged and well-diversified set of Pareto-optimal merging solutions.

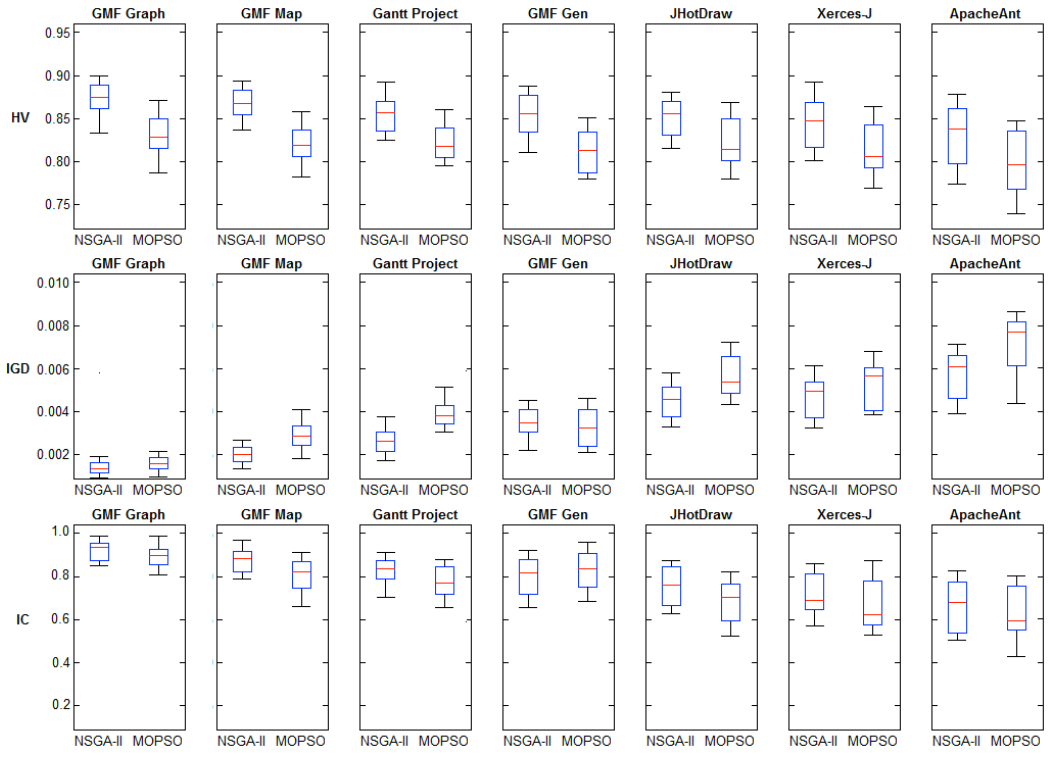

Figure 8. Boxplots using the quality measures (a) IC, (b) IHV, and (c) IGD applied to NSGA-II and MOPSO.

Next, we use all five metrics NDR, IER, AC, MC and ICT to compare three robust refactoring algorithms: our NSGA-II adaptation, MOPSO, a mono-objective genetic algorithm (Mono-EA) that has a single fitness function aggregating the two objectives, mono-objective model merging [38] and a manual merging tool [1]. In order to make meaningful comparisons, we select the best solution for NSGA-II and MOPSO using a knee point strategy [43]. Thus, for NSGA-II and MOPSO, we select the knee point from the Pareto approximation having the median IHV value. For the latter, we use the best solution corresponding to the median observation on 51 runs. The results of the comparison from 51 runs are depicted in Table 3, Table 4, Table 5, Figure 6, Figure 7 and Figure 9. It can be seen that both NSGA-II and MOPSO provide a better trade-off between quality and robustness than a mono-objective EA in all seven systems.

As described in Table 3, for NDR, the number of disabled refactorings using NSGA-II is lower than MOPSO in all systems however the lowest number of disabled refactoring is provided by the mono-objective genetic algorithm approach [38] where only one objective (maximizing the number of enabled refactorings) is considered. The number of disabled refactorings is the same between NSGA-II and [40] in 50% of the cases thus NSGA-II can provide similar results to the mono-objective approach while taking into account the importance of enabled refactorings (Table 4). The mono-objective GA approach that aggregates all the two objectives in one provides highest number of disabled refactorings than NSGA-II, MOPSO and [38] in 100% of the cases. This confirms that it is difficult to find trade-offs between conflicting objectives aggregated in only one fitness function. The random search and the practical merging tool residing on a manual merge process [1] provides the highest number of disabled conflicts due the huge search space to explore to find the best sequence.

Table 4 describes the results of the comparison of our NSGA-II adaptation with the state-of-the-art merging approaches in terms of enabling the most of important refactorings for model merging (IER). NSGA-II suggests the best merging sequences that increase the number of enabled important refactorings for all systems except for Gantt. MOPSO outperforms NSGA-II in terms of IER only for Gantt and GMF Graph. The deviation between the performance of NSGA-II comparing to the performance of other approaches (except MOPSO) is high since all the mono-objective and manual approaches provides low importance score of the enabled

refactoring on all the seven systems. This can be explained by the fact that multi-objective formulation selected the least important refactorings to disable when a conflict is detected. A more qualitative evaluation of NDR and IER is presented in Figure 9. It clearly shows the high deviation between the multi-objective and mono-objective formulation on 51 runs in terms of IRE and also it is important to note that the results of our multi-objective formulation is similar to the mono-objective ones in terms of NDR as described in Figure 9.

For AC and MC, Figure 6 and Figure 7 show that the solutions provided by NSGA-II have the highest manual and automatic correctness values. In fact, the average AC value for NSGA-II is 83% and it is lower than 80% for all the remaining algorithms for the seven systems. The same observation is valid for MC, NSGA-II has the highest MC average value with 86% while the remaining algorithms their MC average is lower than 82%. Figures 6 and 7 reveal also an interesting observation that there is no correlation between the number of operations to merge and the correctness values. More precisely, we sort AC and MC based on the number of refactorings for each open source system. From this data, we conclude that AC and MC are not necessarily affected negatively by a larger number of refactorings. For example, MC even increases from 82% to 88% when the number of refactorings increases from 86 to 112. Thus, we can conclude that our proposal shows a good scalability and is not affected negatively by the number of refactorings. However, when the number of operations increases, it does not necessarily mean that the number of disabled operations does.

Regarding IC, the execution time of NSGA-II is invariably lower than that of MOPSO with the same number of iterations (10.000), however the execution time required by Mono-EA (1h34 in average) is lower than both NSGA-II and MOPSO (1h58 in average for NSGA-II and 2h09 for MOPSO). It is well known that a mono-objective algorithm requires less execution time for convergence since only one objective is handled. However, the execution times of NSGA-II and MOPSO are not so far from those of mono-objective algorithms. For this reason, we can say that our proposal has an accepted efficiency since it generates high quality solutions with a CPU time that is not significantly larger than the time needed by mono-objective algorithms.

In conclusion, we answer RQ3.1-3.4, the results support the claim that our NSGA-II formulation provides a good trade-off between both objectives, and outperforms on average

the state-of-the-art of model merging approaches, both search-based and manual.

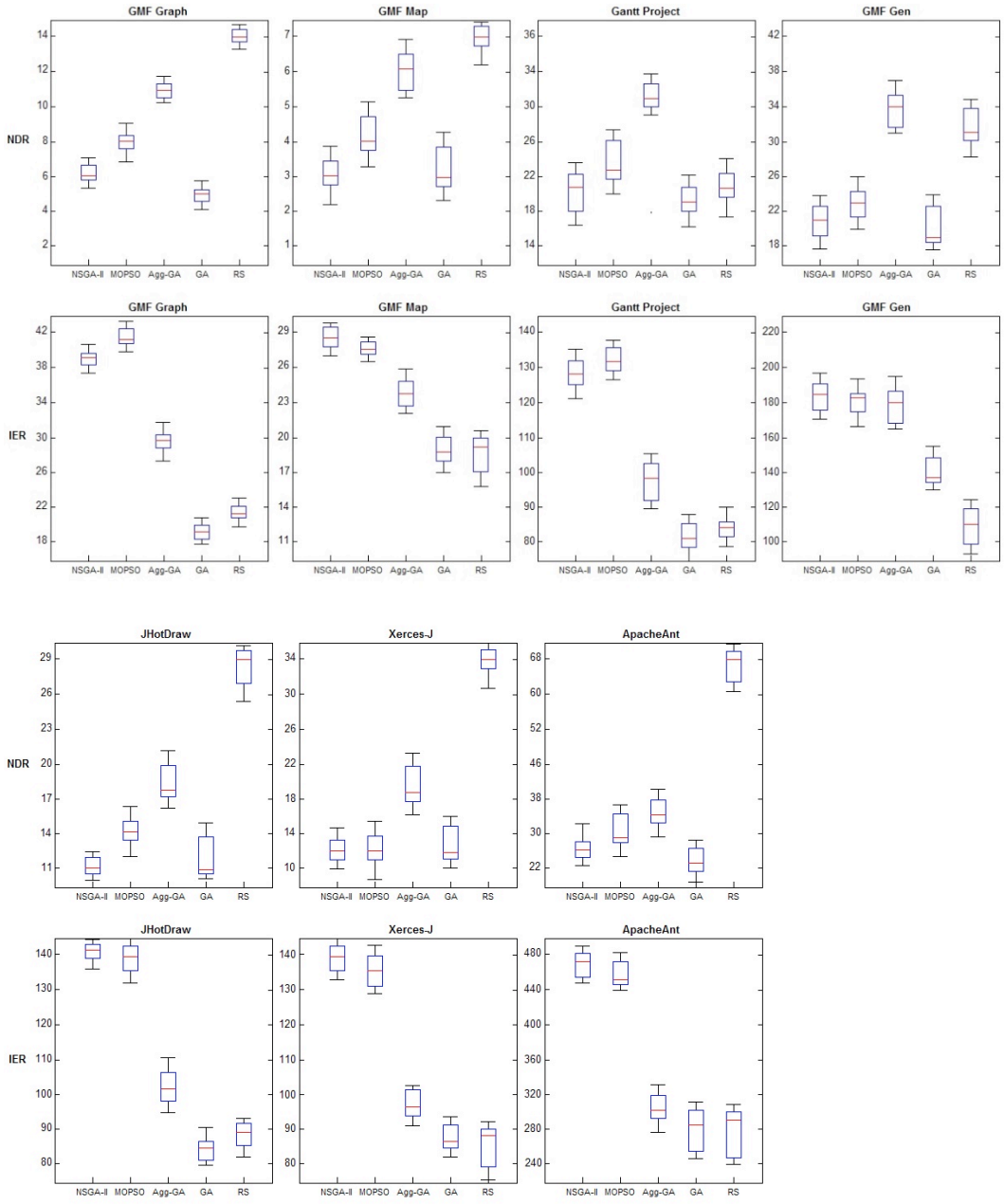

Figure 9. Boxplots using the measures *NDR* and *IER* applied to NSGA-II, MOPSO, Agg-GA, [38] and Random Search (RS) on the different systems for 51 independent runs: GMF Graph, GMF Map, GMF Gen, Gantt, JHotDraw, Xerces-J and ApacheAnt.

4.3.4 Results for RQ4

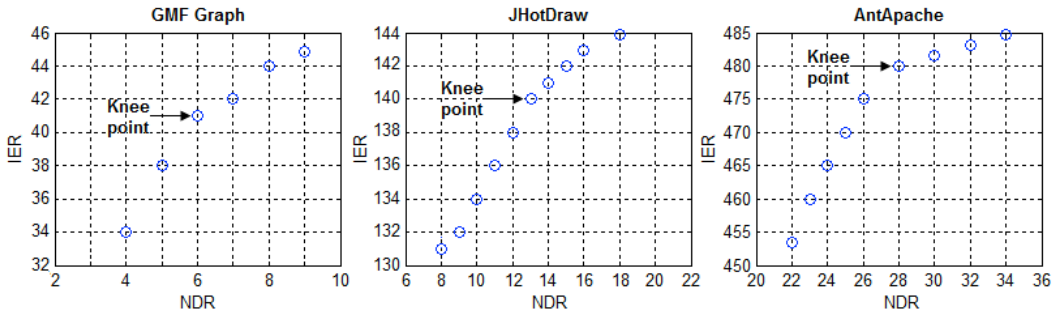

Figure 10. Pareto fronts for NSGA-II obtained on three open source systems:GMF Graph (small), JHotDraw (medium), and ApacheAnt (large).

Figure 10 depicts the different Pareto surfaces obtained on three open source systems (Apache Ant, JHotDraw and GMF Graph) using NSGA-II to optimize both objectives related to minimizing the number of disabled refactorings and maximizing the number of important enabled ones. Due to space limitations, we show only some examples of the Pareto-optimal front approximations obtained which differ significantly in terms of size. Similar observations were obtained on the remaining systems. The 2-D projection of the Pareto front helps software engineers to select the best trade-off solution between the two objectives based on their own preferences. Based on the plots of Figure 10, the engineer could increase the number of disabled operations in favor of some important operations to include while controlling visually the importance cost which corresponds to the number of conflicts created. In this way, the manager can select the preferred merging solution to realize.

One striking feature about all the three plots is that starting from the lowest number of disabled operations solution the trade-off between both objectives is not in favor of including important operations in the merging process, meaning that the number of enabled operations degrades slowly with a fast increase in the overall importance score of considered operations in the merging up to the knee point, marked in each figure. Thereafter, there is a sharp drop in the number of enabled operations with only an important increase in the importance score of considered operations in the merging solutions. It is very interesting to note that this property of the Pareto-optimal front is apparent in all the problems considered in this study. It is likely that a software engineer would be drawn to this knee point as the probable best trade-off between our both objectives. Without any consideration of the importance of operations in the search process, one would obtain the highest number of enabled operations in solution all the time, but Figure 10 shows how a better merging solution that include important operations

can be obtained by sacrificing just a little in the number of total enabled refactorings. Another interesting observation in Figure 10 is that the number of solutions in the Pareto fronts is not high, thus this can help the managers/developers to select the best solution based on their preferences.

We also compared the merging solution at the knee-point for JHotDraw with the best mering solution that maximizes only the number of enabled operations to understand why the former solution finds a good trade-off. We found that the knee-point solution disabled refactorings that were not important and not applied to important classes. Hence we conclude that RQ4 is affirmed and that the multi-objective approach has value for software engineers in a real-world setting that can help managers to prioritize some operations that should be included in a new model version.

4.4 Threats to Validity

Following the methodology proposed by Wohlin et al. [46], there are four types of threats that can affect the validity of our experiments. We consider each of these in the following paragraphs.

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

Internal validity is concerned with the causal relationship between the treatment and the outcome. When we observe an increase in the number of important operations that are enabled, was it caused by our multi-objective merging approach, or could it have occurred for another reason? We dealt with internal threats to validity by performing 51 independent simulation runs for each problem instance. This makes it highly unlikely that the observed increase in the number of important enabled operations was caused by anything other than the applied multi-objective merging approach.

Construct validity is concerned with the relationship between theory and what is observed. Most of what we measure in our experiments are standard metrics such as IHV, IGD, etc. that are widely accepted as good proxies for quality. The notion of refactoring operations importance we use in this paper is new and so constitutes a possible threat to construct

validity. However, the formula we use for refactoring operations importance is a routine size measure so, while many other definitions are possible, perhaps another formulation would yield very different results. We also assume that part of the importance formula is assigned on a per-type basis, so e.g., all the operations within the same type have almost the same importance score. In reality, a developer would probably want to assign different importance scores to different operations within the same type. While this is a weakness in our model, we do not anticipate that very different results would be obtained using a per-instance model. We considered the importance of each refactoring as an input of our approach; we will investigate in the future the impact of different definitions of the operation importance formula on the quality of the merging results.

External validity refers to the generalizability of our findings. In this study, we performed our experiments on seven different widely used open-source systems belonging to different domains and having different sizes, as described in Table 1. However, we cannot assert that our results can be generalized to industrial Java applications, other programming languages, and to other practitioners. Future replications of this study are necessary to confirm our findings.

5 Related Work

Research on versioning systems has a long tradition in software engineering dating back to the early 1970s [31]. Conradi & Westfechtel [32] proposed so-called version models to characterize and document the diversity of existing versioning approaches. In this paper, we consider extensional versioning in terms of having explicit model versions as well as the operation sequences between them available in versioning systems. Furthermore, we focus on the merge phase of the optimistic versioning process and base on existing approaches to support the predecessor phases, namely operation detection and conflict detection.

With respect to the contribution of this paper, namely to integrate two parallel operation histories into one operation sequence that maximizes the number of successfully applied important operations, related work dates back to the early 1990ies. Before that time, merging has been mostly achieved based on the states of the artifacts under version control [23]. The origin work on operation-based merging has been published by Lippe [14]. He pointed out several advantages of operation-based merging over purely state-based merging and contributed the important notion of frontier set. The frontier set, including frontier points, is

an indicator how far one can merge two sequences of operations. One goal is to shift the frontier points as far away from the original model version as possible to maximize the applicability of the performed operations. One way to shift the frontier points is to reorder the atomic operations, i.e., to apply all non-conflicting atomic operations before the conflicting ones. What we have contributed with our search-based approach is a mechanism to minimize the critical points in the merge process where users have to be involved, even when composite operations, such as refactorings, have been applied.

Operation-based merging has been heavily applied in asynchronous collaborative graphical editing. Edwards [13] has defined several strategies for combining two operation sequences into one sequence. The strategies range from fully automatic merging by computing each possible sequence of non-conflicting operations to interactive merging allowing the user to decide how each operation of a conflicting operation pair should be incorporated in the finally merged model. Ignat & Norrie [15] have compared an operation-based approach and a state-based approach for merging operation logs of collaborative graphical editors. They distinguish "real" conflicts from resolvable conflicts. The latter may be resolved by finding an appropriate order to incorporate the operations to the finally merged model. For finding the appropriate order, priority lists for operation types have to be defined. In our approach, we also use priorities for operations, but only for "real conflicts" and not for resolvable conflicts as proposed by Ignat & Norrie. The latter are resolved by searching for the sequences that allow applying the operations of resolvable conflicts. Munson & Dewan present a flexible framework for merging arbitrary objects, which may be configured in terms of merge policies [34]. Merge policies may be tailored by users to their specific needs and include rules for conflict detection and rules for automatic conflict resolution. Actions for automatic conflict resolution are defined in merge matrices and incorporate the kinds of atomic operations made to the object and the users who performed those operations. Thus, it may be configured, e.g., that operations of specific users always dominate operations of others, or that updates outpace deletions.

With the advent of MDE, the research topic collaborative modeling is gaining momentum. Several state-based approaches for model versioning have been proposed (cf. [16][23] for an overview), as well as a few operation-based approaches. Koegel et al. [18] record operations in modeling editors and provide conflict detection for two sequences of recorded operations.

They also support composite operations, but only consider how these operations are build up from atomic operations while explicit preconditions are disregarded. If they detect that a composite operation is in conflict with an atomic operation, they let the user decide which one to take. Similarly, Barret et al. [19] discuss pushing the frontier points as far as possible by incorporating all non-conflicting operations to produce a merged model and then let again the user decide which operation of a conflict pair to prioritize. Other operation-based approaches for models have been presented in [20][22][34], but no dedicated reordering strategies have been discussed. In [35], the authors present a fixed merge policy with a dedicated focus on raising the number of enabled parallel operations applied on ordered features by interleaving insert operations of both developers and relaxing the constraint that the inserted elements have to be exactly at the given index as long as the relative order of them is still given. In [21], the authors mention that finding an appropriate sequence for unifying the operations of two parallel operation sets may be considered as an optimization problem, but they based their approach on manual conflict resolution during the merge process. Cicchetti et al. [36] propose an adaptable merging algorithm by defining conflict patterns describing specific difference patterns which are supplemented with a reconciliation strategy. Such strategies state how the conflicts should be automatically resolved by specifying either which side should be preferred in the merge process or by introducing a transformation to resolve the conflict. While policy-based approaches require user intervention in certain conflict cases where no policy is at hand, [28] present a formal merge approach based on graph transformation theory, yielding a merged model by construction and defers the resolution of conflicts. A similar strategy is followed in [37] where conflicts are resolved after the merge collaboratively.

In summary, reasoning on arbitrary application orders of the operations to unify (including composite operations) to find the order that is maximizing the successful application of the operations is not considered by existing operation-based merge approaches. State-of-the-art approaches mostly reside on a two-phase process: first, they apply the non-conflicting operations and then let the user select the operation to be prioritized out of two conflicting operations. In contrast, our approach explores arbitrary sequences and the result is the most applicable sequence of operations found by the genetic algorithm. Thus, we are able to minimize the critical and labor-intensive tasks involving user interaction in the merge process

going beyond existing state-of-the-art approaches.

Our proposal is part of the search-based software engineering (SBSE) contributions [29]. SBSE uses search-based approaches to solve optimization problems in software engineering. Based on surveys proposed by Harman et al. [29][30], our work represents the first attempt to treat the problem of model merging as a combinatorial optimization problem. SBSE techniques were applied to a close problem to software merging which is software refactoring. Search-based refactoring, i.e., fully automated refactoring driven by metaheuristic search and guided by software quality metrics and used subsequently to address the problem of automating design improvement [47]. Seng et al. [48] propose a search-based technique that uses a genetic algorithm over refactoring sequences. The employed metrics are mainly related to various class level properties such as coupling, cohesion, complexity and stability. Kessentini et al. [49] also propose a single-objective combinatorial optimization using a genetic algorithm to find the best sequence of refactoring operations that improve the quality of the code by minimizing as much as possible the number of code smells detected using a set of quality metrics. Harman and Tratt were the first to introduce the concept of Pareto optimality to search-based refactoring [50]. They use it to combine two metrics into a fitness function, namely CBO (coupling between objects) and SDMPC (standard deviation of methods per class), and demonstrate that it has several advantages over the weighted-sum approach. In all these search-based refactoring work, the problem of merging refactorings were not addressed.

6 Conclusion and Future Work

This paper proposes a novel multi-objective approach for merging parallel versions of models by finding the best operation sequence that take into account the importance of the operations to merge. Having such a sequence is very useful in model versioning to find a tentative merge, as a basis for subsequently resolving the remaining conflicts manually. Therefore, a merged model is necessary that maximizes the combined effect of all operations that have been applied by multiple developers in parallel to the same model. This is achieved by finding an optimal (potentially interleaved) order of operations that minimizes the number of disabled operations. Furthermore, it is useful for team managers to select merging solutions that include the most important operations while minimizing the number of disabled ones. As the search space in terms of all possible sequences of operations is potentially huge and we have

two conflicting objectives to optimize, we considered in this paper the merging process as a multi-objective optimization problem.

We evaluated our proposal with seven real-world model evolutions extracted from different open source systems. The experiment results indicate clearly that the number of disabled operations is reduced significantly in comparison to the number of disabled operations without taking into consideration the different possible operation orders. Furthermore, the results provide strong evidence to support the claim that our proposal enables the generation of efficient model merging solutions to be comparable in terms of minimizing the number of conflicts to those suggested by existing approaches and to carry a high importance score of merged operations.

Although our approach has been evaluated with real-world models with a reasonable number of applied operations, we are working now on larger models and with larger lists of operations applied in parallel. This is necessary to investigate more deeply the applicability of the approach in practice, but also to study the performance of our approach when dealing with very large models. We will be extending our evaluation on different industrial automotive projects in collaboration with several team managers. Moreover, we plan to include different other factors related to the importance of operations. More generally, we plan to extend this work by fixing detected conflicts and considering additional objectives to optimize during the merging process.

References

[1] D. Dig, K. Manzoor, R. E. Johnson, T. N. Nguyen. Effective Software Merging in the Presence of Object-Oriented Refactorings. *IEEE Transactions on Software Engineering,* 34(3):321-335, 2008.

[2] T. Ekman, U. Asklund. Refactoring-aware Versioning in Eclipse. *Electronic Notes in Theoretical Computer Science* 107:57-69, 2004.

[3] M. Koegel, M. Herrmannsdoerfer, Y. Li, J. Helming, D. Joern. Comparing State- and Operation-based Change Tracking on Models. In *Proceedings of EDOC,* 2010.

[4] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. Refactoring – Improving the Design of Existing Code. 1st ed. Addison-Wesley, 1999.

[5] J. R. Koza. Genetic Programming: On the Programming of Computers by Means of Natural Selection. *MIT Press*, Cambridge, 1992.

[6] S. Kirkpatrick, C. D. Jr. Gelatt, M. P. Vecchi. Optimization by simulated annealing. *Sciences,* 220(4598):671–680, 1983.

[7] http://web.mst.edu/~marouanek/gecco13/merging

[8] www.eclipse.org/gmf

[9] http://www.ganttproject.biz

[10] http://www.jhotdraw.org

[11] http://xerces.apache.org/xerces-j

[12] P. Langer. Adaptable Model Versioning based on Model Transformation By Demonstration. *PhD Thesis, Vienna University of Technology*, 2011.

[13] W. K. Edwards: Flexible Conflict Detection and Management in Collaborative Applications. In *Proceedings of Symposium on User Interface Software and Technology*, pages 139-148, 1997.

[14] E. Lippe, N. van Oosterom. Operation-based merging. In *Proceedings of SDE*, pages 78-87, 1992.

[15] C. Ignat and M. C. Norrie. Operation-based versus State-based Merging in Asynchronous Graphical Collaborative Editing. In *Workshop on Collaborative Editing,* 2004.
[16] P. Brosch, G. Kappel, P. Langer, M. Seidl, K. Wieland, M. Wimmer: An Introduction to Model Versioning. In *Proceedings of SFM,* 2012.
[17] P. Brosch, P. Langer, M. Seidl, K. Wieland, M. Wimmer, G. Kappel. The Past, Present, and Future of Model Versioning; *Emerging Technologies for the Evolution and Maintenance of Software Models*, IGI Global, 2011.
[18] M. Koegel, H. Naughton, J. Helming, M. Herrmannsdoerfer. Collaborative model merging. In *OOPSLA Companion,* 2010.
[19] S. Barrett, P. Chalin, G. Butler. Table-Driven Detection and Resolution of Operation-Based Merge Conflicts with Mirador. In *Proceedings of ECMFA,* 2011.
[20] C. Schneider, A. Zündorf, J. Niere. CoObRA - a small step for development tools to collaborative environments. In *Workshop on Directions in Software Engineering Environments,* 2004.
[21] M. Schmidt, S. Wenzel, T. Kehrer, U. Kelter. History-based Merging of Models. In *Workshop on Comparison and Versioning of Software Models*, 2009.
[22] A. Mougenot, X. Blanc, M. Gervais. D-Praxis: A Peer-to-Peer Collaborative Model Editing Framework. In Proceedings of *DAIS,* 2009.
[23] T. Mens. A State-of-the-Art Survey on Software Merging. *IEEE Transactions on Software Engineering*, 28(5):449-462, 2002.
[24] J. Bézivin. On the Unification Power of Models. *Software and Systems Modeling,* 4(2):171-188, (2005).
[25] A. ben Fadhel, M. Kessentini, P. Langer, M. Wimmer. Search-based Detection of High-level Model Changes. In *Proceedings of ICSM*, 2012.
[26] P. Langer, M. Wimmer, P. Brosch, M. Herrmannsdoerfer, M. Seidl, K. Wieland, G. Kappel. A Posteriori Operation Detection in Evolving Software Models. *Journal of Systems and Software*, 86(2):551-566, 2013.
[27] K. Wieland, P. Langer, M. Seidl, M. Wimmer, G. Kappel. Turning Conflicts into Collaboration - Concurrent Modeling in the Early Phases of Software Development. *Computer Supported Cooperative Work*, 22(2-3):181-240, 2013.
[28] G. Sunyé, D. Pollet, Y. Le Traon, J. M. Jézéquel. Refactoring UML Models. In *Proceedings of UML*, 2001.
[29] M. Harman. The current state and future of search based software engineering. In *Proceedings of ICSE*, 2007.
[30] M. Harman, S. A. Mansouri, Y. Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys* 45(1):Article11, 2012.
[31] Jacky Estublier, David B. Leblang, André van der Hoek, Reidar Conradi, Geoffrey Clemm, Walter F. Tichy, Darcy Wiborg Weber: Impact of software engineering research on the practice of software configuration management. ACM Trans. Softw. Eng. Methodol. 14(4): 383-430 (2005).
[32] Reidar Conradi, Bernhard Westfechtel: Version Models for Software Configuration Management. ACM Comput. Surv. 30(2): 232-282 (1998).
[33] Jonathan P. Munson, Prasun Dewan: A Flexible Object Merging Framework. CSCW 1994: 231-242
[34] Christian Gerth, Jochen Malte Küster, Markus Luckey, Gregor Engels: Detection and resolution of conflicting change operations in version management of process models. Software and System Modeling 12(3): 517-535 (2013)
[35] Marcus Alanen, Ivan Porres: Difference and Union of Models. UML 2003: 2-17
[36] Antonio Cicchetti, Davide Di Ruscio, Alfonso Pierantonio: Managing Model Conflicts in Distributed Development. MoDELS 2008: 311-325
[37] Konrad Wieland, Philip Langer, Martina Seidl, Manuel Wimmer, Gerti Kappel: Turning Conflicts into Collaboration. Computer Supported Cooperative Work 22(2-3): 181-240 (2013)
[38] Marouane Kessentini, Wafa Werda, Philip Langer, Manuel Wimmer: Search-based model merging. GECCO 2013: 1453-1460
[39] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II, *IEEE Trans. Evol. Comput.*, vol. 6, pp. 182–197, Apr. 2002.
[40] Li, X. 2003. A non-dominated sorting particle swarm optimizer for multiobjective optimization. In Proceedings of the 2003 international conference on Genetic and evolutionary computation (GECCO'03). 37-48.
[41] Saxena, D.K., Duro, J.A., Tiwari, A., Deb, K. and Zhang, Q. 2013. Objective Reduction in Many-objective Optimization: Linear and Nonlinear Algorithms. In Proceedings of IEEE Transactions on Evolutionary Computation. vol. 17, no. 1. 77–99.
[42] http://ant.apache.org/
[43] Bechikh, S., Ben Said, L. and Ghédira, K. 2010. Estimating Nadir Point in Multi-objective Optimization using Mobile Reference Points. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC'10). 2129–2137.
[44] Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M. and da Fonseca, V. G. 2003. Performance assessment of multiobjective optimizers: an analysis and review. IEEE Transaction on Evolutionary Computation. vol. 7, no. 2, 117-132.

[45] Arcuri, A. and Briand, L. C. 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In Proceedings of the 33rd International Conference on Software Engineering (ICSE '11). ACM, New York, NY, USA, 1-10. DOI=10.1145/1985793.1985795.

[46] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B. and Wesslén, A. 2012. Experimentation in Software Engineering. Springer.

[47] O'Keeffe, M. K. and Ó Cinnéide, M. 2008. Search-based refactoring for software maintenance. In Journal of Systems and Software. vol. 81, no.4. 502-516.

[48] Seng, O., Stammel, J. and Burkhart, D. 2006. Search-based determination of refactorings for improving the class structure of object-oriented systems. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'06). 1909–1916.

[49] Kessentini, M., Kessentini, W., Sahraoui, H., Boukadoum, M., and Ouni, A. 2011. Design Defects Detection and Correction by Example. In Proceedings of the 19th IEEE International Conference on Program Comprehension (ICPC'11). 81-90.

[50] Harman, M. and Tratt, L. 2007. Pareto optimal search based refactoring at the design level. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'07). 1106–1113.