e-Informatica Software Engineering Journal, Volume 18, Issue 1, 2024, pages: 240109, DOI: 10.37190/e-Inf240109

An N-Way Model Merging Approach Based on Artificial Bee Colony Algorithm

Tong Ye*, Gongzhe Qiao**

*College of Computer and Software, Nanjing Vocational University of Industry Technology
**College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics
yetong@nuaa.edu.cn, qgz@nuaa.edu.cn

Abstract

Background: In N-way model merging, model matching plays an important role. However, the N-way model matching problem has been recognized as NP-hard.
Aim: To search the optimal or near-optimal matching solution efficiently, this paper proposes an N-way model matching algorithm based on the Artificial Bee Colony (ABC) algorithm.
Method: This algorithm combines global heuristic search and local search to deal with the complexity of N-way model matching. We evaluated the proposed N-way model merging approach through case studies and we evaluated the proposed ABCMatch algorithm by comparing it with Genetic Algorithm (GA) and Elephant Herding Optimization (EHO).
Results: The experimental results show that ABCMatch can obtain more accurate model matching solutions in a shorter time, and the average model matching accuracy of ABCMatch is 2.7725% higher than GA and 1.8804% higher than EHO.
Conclusion: Results demonstrate that our method provides an effective way for software engineers to merge UML models in collaborative modeling scenarios.

Keywords: Model driven development, Tools for software researchers or practitioners, Project management

1. Introduction

Model-Driven Software Engineering (MDSE) is an important direction of Software Engineering. It refers to the systematic use of models as first-class entities throughout the software engineering life cycle [1]. Models are less bound to the underlying implementation technology and are much closer to the problem domain. They accelerate the development process of complex systems by improving the abstraction level of software development. With the increasing complexity of software systems, it is almost impossible to model complex systems by a single user. The efficiency of software development can be greatly improved by adopting the Collaborative MDSE approach [2] where multiple stakeholders manage, collaborate, and are aware of each other's work on a set of shared models. Since UML (Unified Modeling Language) class diagram [3] is one of the most commonly used models for software modeling, this paper focuses on collaborative modeling using UML class diagrams.

Generally, collaborative modeling are divided into online (real-time) collaboration and offline collaboration. Our approach is proposed to support offline collaboration where users modify their models locally and push the changes later. In the process of offline

Submitted: 7 Nov. 2023; Revised: 21 Jun. 2024; Accepted: 22 Jun. 2024; Available online: 31 Jul. 2024

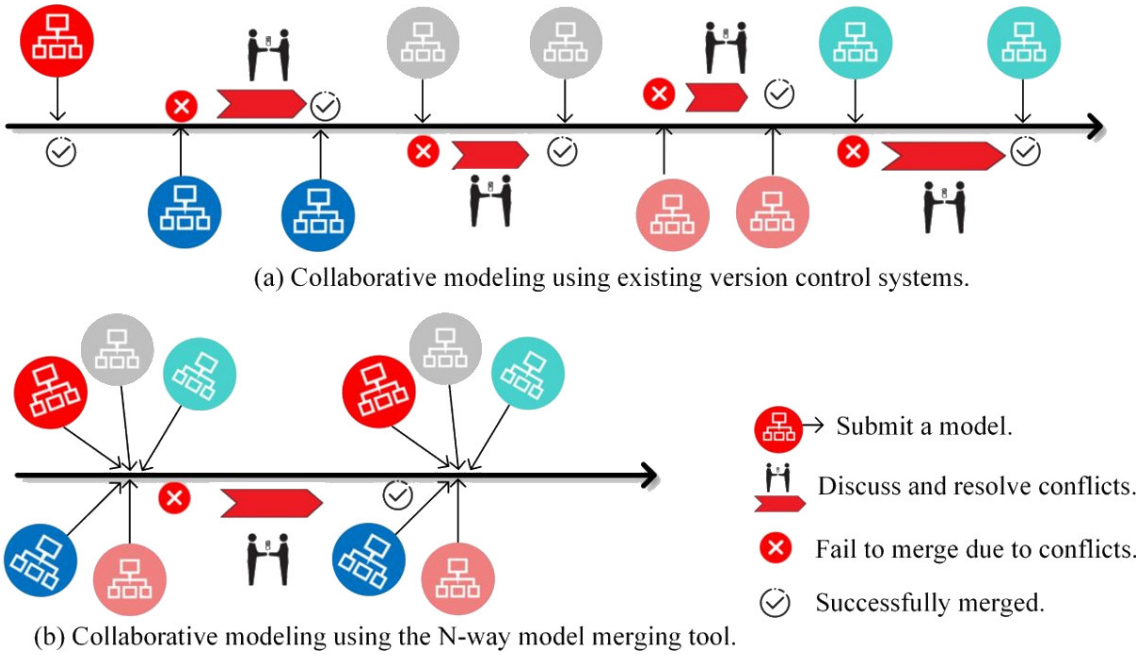


Figure 1. Comparison of collaborative modeling using existing version control systems and the N-way model merging tool

collaborative modeling, any modification will lead to different branches of the model. So it is important to merge different versions and branches periodically to obtain an integrated single model. Collaborative teams often use version control systems such as EMFStore [4] and CDO model repositor [5]. However, these tools [4, 5] only support two-way or three-way model merging. To make the motivation clear, Figure 1 gives an example of the model merging process of a five-member team, circles of each color represent the models submitted by each member. As shown in Figure 1(a), using the existing version control system, each member needs to wait for others to deal with the conflicts immediately, and only after resolving the conflicts can the next merge be carried out. To save the extra waiting time, we propose a practical N-way model merging approach to merge N models at a time. As shown in Figure 1(b), this approach not only reduces the number of negotiations but also saves time for submitting one by one.

In N-way model merging, the first challenge is that it is hard to well examine the overall search space effectively because N-way model matching is known as NP-hard as it requires to cope with a huge search space of possible element combinations [6]. How to efficiently check the entire search space to obtain more accurate N-way model matching solutions is a complex optimization problem. Optimization is one of the most important hot topics in scientific and technical areas [7]. Solving complex optimization problems in real life is considered to be a huge challenge. In recent years, numerous researchers have made efforts to solve optimization problems [8–11]. Some researchers use classical methods such as gradients, Lagrange, and linear mathematical to solve optimization problems [7]. However, due to the complex mathematical processes and nonlinear objective functions, classical optimization algorithms are unable to solve complex optimization problems efficiently [7]. In contrast, meta-heuristic algorithms based on group and cooperation are very effective in solving NP-hard problems [7].

Meta-heuristic methods search the optimal and near-optimal solutions by simulating natural behaviors or events [8, 12, 13]. At present, the meta-heuristic method has been listed as one of the most promising methods to solve optimization problems. Widely used meta-heuristic algorithms include Artificial Bee Colony (ABC) [14], Genetic Algorithm (GA) [15], Grey Wolf Optimizer (GWO) [16], Symbiotic Organisms Search (SOS) [17], Whale Optimization Algorithm (WOA) [18], Farmland Fertility Algorithm (FFA) [19], Elephant Herding Optimization (EHO) [20], Sparrow Search Algorithm (SSA) [21], Tunicate Swarm Algorithm (TSA) [22], Honey Badger Algorithm (HBA) [23], Northern Goshawk Optimization (NGO) [24]. Compared with traditional optimization methods, meta-heuristic algorithms have the advantages of simplicity, fewer parameters, avoiding local optimization and strong flexibility [25]. Because of these advantages, meta-heuristic methods have been widely used to solve various complex and difficult optimization problems.

Although most of the existing meta-heuristic algorithms have the above advantages, different meta-heuristic algorithms also have different weaknesses when facing different optimization issues [7]. With the change of the problem set, different optimization algorithms may show different performances [7]. Therefore, in order to solve complex optimization problems, an effective optimization method should consider all aspects of the problem. And it is necessary to select the most appropriate optimization algorithm according to the type of problem and search space [7].

The Artificial Bee Colony (ABC) algorithm [14] searches for the optimal solution through the random and objective evolution of candidate solution sets. In this algorithm, each food source represents a feasible solution to the problem to be solved, and the nectar quantity of the food source represents the fitness of the feasible solution. Bees are divided into three roles: employed bees, onlookers, and scouts. Through the cooperation of these three types of bees, the optimal solution or near-optimal solution can be obtained efficiently. ABC has good performance in searching possible solutions quickly and globally. At present, the ABC algorithm has been successfully applied in many areas including software engineering, medical image processing, economics, financial analysis, and network communication. Existing research has proved that the ABC algorithm is very suitable for solving complex and difficult combination problems [26]. Since the N-way model matching problem needs to search the model matching solution with the highest matching degree from the combinations of a large number of model elements, which is a complex combination problem, this paper selects ABC and improved it to solve the N-way model matching problem.

The second challenge is that models are complex structures connected with model relationships, so it is necessary to merge not only model elements but also their related nodes. Existing N-way model merging approaches [6, 27, 28] focus mainly on matching model elements. These approaches [6, 27, 28] ignore relationships in the model and break the chain into pieces rather than reshuffling chained elements. Unlike these methods [6, 27, 28], the proposed approach supports merging chained elements.

Conflict resolution is also an important challenge in model merging [29, 30]. Existing approaches [29, 30] transfer the responsibility of resolving conflicts to users and it is not applicable in complex merging situations where numerous conflicts lead to too many decisions to make. To prevent conflicts automatically, we present the matching model which is an intermediate form between the model matching results (generated by the proposed ABCMatch algorithm) and the merged model. Each model element in the matching model is assigned a priority number which is the same as the priority number of the model. When conflicts occur, the model element with the highest priority is picked automatically.

This paper makes the following contributions:

- We propose a new model matching algorithm ABCMatch which combines global heuristic search and local search together to deal with the complexity of N-way model matching.
- We propose a new N-way model merging approach based on the ABCMatch algorithm to merge UML class diagrams of different versions.
- We evaluated ABCMatch by comparing it with GA and EHO. The results show that ABCMatch performs better than GA and EHO in N-way model matching. The average model matching accuracy of ABCMatch is 2.7725% higher than GA and 1.8804% higher than EHO.
- We implemented the N-way model merging approach in Java and evaluated it by comparing it with EMFStore through case studies. The results show that the proposed approach performs better than EMFStore when a large number of models are required to be merged at one time in collaborative modeling.

The rest of the paper is structured as follows. Section 2 discusses related works. Section 3 presents the overview of the proposed approach. Section 4 introduces the model comparison method. Section 5 describes the ABCMatch algorithm. Section 6 presents the model merging method. Section 7 evaluates the proposed approach. Finally, Section 8 concludes this paper.

2. Related work

2.1. Two-way and three-way model merging

Model merging is a problem that has been studied for a long time in the area of collaborative modeling. In the aspect of two-way model merging, Buneman et al. propose a model merging algorithm named BDK [31], which creates the duplicate free union of two models based on the name equality of model elements. However, BDK can only identify one-type conflict as the proposed meta-meta-model contains only two relationships, *Is-a* and *Has-a*, where *Has-a* must obey one-type restriction. Pottinger and Bernstein improve BDK by defining the operator *Merge* and *take* mapping as its input [32]. However, this approach does not scale well since the manual definition of each mapping is a labor-intensive and time-consuming process.

Other studies [33–36] apply the three-way model merging technique that performs model merging on two models derived from the same ancestor model. Sharbaf et al. [33] present a novel three-way model merging approach which uses pattern-based method to detect and resolve conflicts in the merging process. Thao and Munson propose a three-way merging algorithm [34] based on LCS (Longest Common Subsequence) algorithm. Debreceni et al. propose a three-way operation-based merging algorithm [35]. However, they define only two change annotations *"must"* and *"may"*. When two changes are both annotated by *"must"* or *"may"*, the algorithm cannot determine which one to choose automatically. Our approach defines different priorities for each model thus avoiding this problem. Schwagerl et al. implement a three-way merging tool [36] for models in the Eclipse Modeling Framework (EMF). It is more general than the EMF Compare match meta-model but still fails to handle relationships as they omit the graph-like structure in the model. In this paper, we consider not only model elements but also their relationships.

2.2. *N*-way model merging

Due to the problem of selection order in two-way and three-way model merging, some approaches [6, 27, 28, 37–41] have been proposed to generate merged models from N existing variants. Schultheiÿ et al. [37] propose a heuristic N-way model matching algorithm named RaQuN, which uses multi-dimensional search trees to find suitable match candidates. Kasaei et al. [38] present a formalism for specifying N-way model merging rules. They implemented a syntax-aware editor and a parser to promote N-way merging rules for EMF-based models. Boubakir et al. [39] propose a pairwise approach for model merging, which improves the quality of the results by considering the order of combining the set of input models. Rubin et al. present the NwM algorithm [6] for the N-way merging of model variants. Assuncao et al. propose a search-based merge method [27] for UML model variants. However, these methods [6, 27] ignore relationships in the model. As models are complex structures connected with model relationships, it is necessary to merge not only model elements but also their related nodes. In this paper, we consider all input relationship chains and reshuffle elements from distinct chains by extracting the prior element link and storing it in matching models.

Jiang et al. propose an entropy-based merging tool [40] to help merge models generated by different modelers. However, it requires users to model in the way specified by the tool and cannot support merging models built with existing widely used UML modeling tools. In this paper, the proposed approach supports merging models built in the famous Papyrus modeling environment [42].

Martinez et al. present a generic framework [41] for constructing merged models from a set of model variants. But they assume that the variants are not independently generated out of the same family of models and do not target to address the problem of model similarity analysis. In addition, Reuling et al. [28] claim that Martinez et al. fail to support imprecise matching. They propose a precise N-way model merging method [28] by encoding the variability information using language-specific variability-encoding operators. However, in this method, the new class contains all duplicated class properties which require further manual handling. Our method merges duplicated model elements automatically rather than simply enumerating them.

N-way model matching plays an important role in N-way model merging. The N-way model matching problem is a complex optimization problem that requires to use optimization methods to efficiently search the optimal or near-optimal model matching solution in the huge search space. Approximate methods for solving optimization problems are divided into heuristic methods and meta-heuristic methods [43]. Heuristic algorithms usually search the optimal solution in a reasonable computing time. However, heuristic algorithms cannot guarantee the optimal solutions and are easy to fall into local optimums [43]. Due to the weaknesses of heuristic algorithms, many existing research studies have proposed meta-heuristic algorithms to solve complex optimization problems.

Meta-heuristic algorithms are inspired by natural behavior or events. The existing meta-heuristic algorithms can be divided into three categories: evolution-based algorithms, physics-based algorithms, and population-based algorithms[43]. Evolution-based algorithms mainly simulate the evolution process in nature to realize the overall progress of the population. Physics-based algorithms usually imitate physical rules to achieve optimization. Because of the strong flexibility and high performance, population-based algorithms have attracted more attention in recent years. In this type of algorithm, each population is

a biological population. Population-based algorithms search the global optimal solution through the cooperative behavior among individuals in the population.

Common population-based optimization algorithms are Artificial Bee Colony (ABC) [14], Grey Wolf Optimizer (GWO) [16], Symbiotic Organisms Search (SOS) [17], Whale Optimization Algorithm (WOA) [18], Elephant Herding Optimization (EHO) [20], Sparrow Search Algorithm (SSA) [21], Tunicate Swarm Algorithm (TSA) [22], etc. These algorithms and their variants have been widely used to find the optimal values of functions, solve multi-machine scheduling problems, multi-objective optimization problems, and so on. For example, Mohammadzadeh et al. [8] proposed the BMAMH algorithm combined with multiple swarm intelligence optimization algorithms to detect spam e-mail. Gharehchopogh [44] improved the tunicate swarm algorithm and proposed the QLGCTSA algorithm with higher performance to solve complex optimization problems. Abdollahzadeh1 et al. [45] proposed three effective binary methods based on symbiotic biological search (SOS) algorithm to solve the feature selection problem in information preprocessing. Bonab et al. 10 proposed a new hybrid method based on fruit fly algorithm (FFA) and ant optimization algorithm (ALO) to improve the performance of intrusion detection system.

Among existing population-based optimization algorithms, the ABC algorithm and its variants avoid local optimal solution by using global and local search, which has the advantages of high performance and strong flexibility. In recent years, more and more researchers choose to use the ABC algorithm and its variants to solve complex optimization problems in various fields. Öztürk et al. studied the role of the ABC algorithm and its variants in the field of medical image processing [46]. The ABC algorithm maintains a good balance between global search and local search. It can not only be used for medical image enhancement, including improving contrast, edge, artifact elimination, intelligent noise reduction, but also has played an important role in medical image segmentation, such as tumor detection, classifying image pixels into anatomical regions [46].Existing research has confirmed that the ABC algorithm has better performance than other meta-heuristic algorithms in solving various complex combination problems [26]. Because the N-way model matching problem needs to search the model combination with the best matching degree from the combinations of a large number of model elements, which is a complex combination problem, we chose the ABC algorithm and improved it to solve the N-way model matching problem.

2.3. Operation-based merging approach

Operation-based merging tries to solve the merge problem by merging operation sequences. Mansoor et al. propose an operation-based model merging method [47]. They consider merging different model versions as a multi-objective optimization problem. But the importance score of each composite option is determined by different developers. It is hard for them to compare importance scores of operations with each other while developing as they are not sure what scores others might set and this might cause their important operations disabled. In this paper, we define different priorities for input models from a global perspective thus avoiding this problem.

Some existing approaches [48–50] apply rule-based methods in operation-based model merging. Anwar et al. propose a formal approach [48] for model composition. RuCORD [49] is a rule-based composite operation detection and recovery framework for merging models in Eclipse. Chong et al. propose an operation-based approach [50] to merge different versions of UML models. However, these methods [48–50] are not applicable in large-scale projects

as they require identifying corresponding model elements and defining formal composition rules for each model element manually in the matching step. In addition, the detection and recovery processes are supposed to be guided by users which are not applicable when there are too many operations. To solve these problems, in this paper, we identify groups of corresponding input model elements automatically by the ABCMatch algorithm and handle conflicts by identifying the prior model. Furthermore, we consider N models simultaneously which is more suitable for large-scale model merging.

2.4. Conflicts resolving

Koegel et al. present an approach [29] to make conflicts part of the model and represent them as first-level entities based on issue modeling. However, this approach transfers the responsibility of resolving conflicts to users and it is not applicable in complex merging situations where numerous conflicts lead to too many decisions to make. Dam et al. propose an approach [30] to automatically resolve all inconsistencies that arise during the merging of model versions. They create a validation tree to evaluate constraint instances and build a repair tree based on the validation tree which gives repair suggestions and checks if a repair causes other inconsistencies. However, they fail to consider the situation where repair suggestions form a cycle. This method is not applicable when merging large-scale models as it may cause many cycle errors which cannot be solved automatically.

To summarize, in two-way and three-way model merging [31, 32, 34–36], results are influenced by the order to pick input model elements. Among existing N-way model merging methods [6, 27, 28, 40, 41], Rubin et al. [6], Assuncao et al. [27] and Reuling et al. [28] ignore relationships in models, Jiang et al. [40] merge UML models which are modeled in a specified way using their tool rather than common UML models, Martinez et al. [41] fail to address the problem of model similarity analysis, and Reuling et al. [28] fail to handle duplicated class properties. Among existing operation-based model merging methods [47–50], Mansoor et al. [47], fail to define the priority from a global view which might lead to important operations being disabled, and rule-based methods [48–50] require much user interaction and fail to resolve conflicts automatically. In addition, existing conflicts resolving methods [29, 30] are not applicable in N-way model merging where numerous conflicts lead to too many decisions for users to make.

To address these problems and fill the research gap, we propose an N-way model merging approach based on the ABC (Artificial Bee Colony) algorithm [14]. We identify the prior model element and prior element link to generate the merged model from the matching model. In this way, inconsistencies are avoided automatically. To solve the matching problem of N-way model merging, we propose the ABCMatch algorithm which can explore the search space and obtain the optimal matching solution efficiently.

3. Overview of the proposed approach

Existing studies [6, 48, 51] suggest that model merging can be divided into three steps: model comparison, model matching, and model combination. In the comparison step, similarity degrees between elements are calculated by comparing their corresponding sub-elements and weighing the results using empirically determined weights. These weights represent the contribution of model sub-elements to the overall similarity of their owning elements. In the matching step, pairs of elements from the input models as well as their similarity degrees

are taken as inputs, and the outputs are groups of model elements that are considered similar. In the model combination step, a new duplicate-free model that combines groups of matched elements is generated.

In this paper, we propose a novel N-way model merging approach following the above-mentioned three steps. The overview of the proposed approach is given in Figure 2.

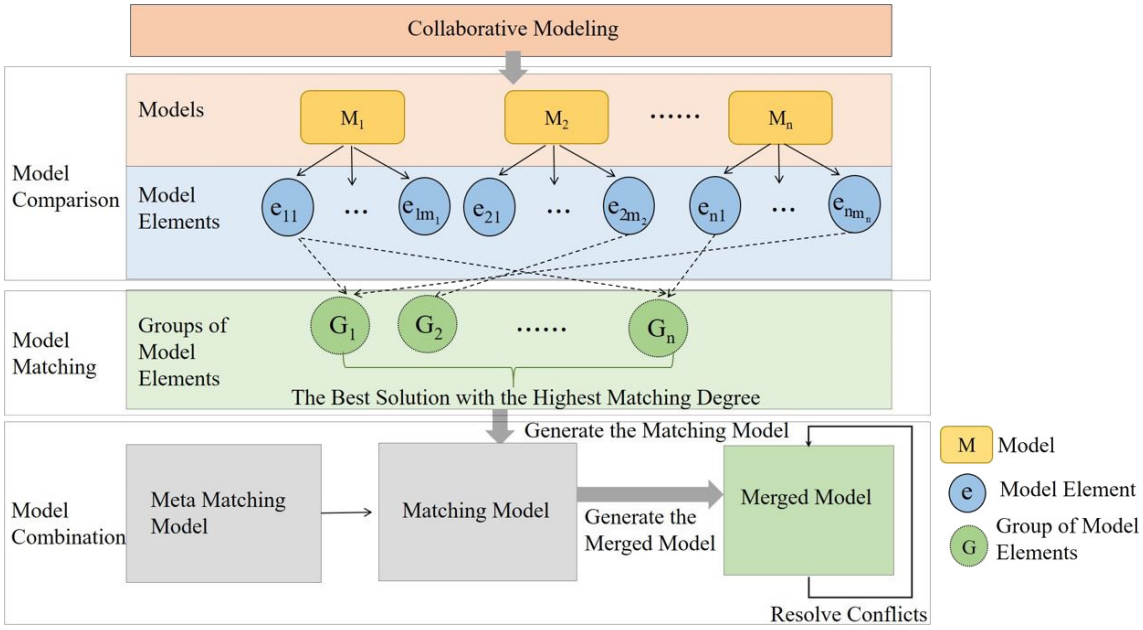


Figure 2. Overview of the proposed approach

Model comparison. The similarity degree between two classes can be calculated as a weighted sum of the similarity degrees of their names, properties, and methods. Although numerous auto or semi-auto methods have been proposed to calculate similarity degrees, gaps still exist when applying to N-way model merging. This is because, in N-way model merging, multiple input models are considered at the same time. So model comparison needs to calculate the similarity of a group of model elements rather than only two model elements in two-way or three-way model merging. To solve this problem, we propose a model comparison approach that can calculate the similarity of a group of model elements (see details in Section 4). As shown in Figure 2, models to be merged are denoted as M_1, M_2,...,M_n. Each model M_i contains m_i model elements denoted as $e_{i1}, e_{i2}, ..., e_{im_n}$. We define a group of models with different versions of common elements as a matching path, which is denoted as G_i in Figure 2. Multiple matching paths without common elements constitute a model matching solution. To distinguish similarity degrees of a group of model elements and a pair of elements, the sum of similarity degrees of model elements in a matching path/solution is called the matching degree. In our approach, similarity degrees between two classes are calculated based on the Jaccard similarity coefficient [52] which is an index to measure the similarity between two sets. We calculate the matching degrees of the matching paths from two dimensions: the class name and properties/methods. Both metrics are complementary and assess two different aspects of the matching path: the first one compares the string of a group of model elements while the second one focuses on the number of properties/methods in common. A matching solution is a set of disjoint

matching paths, so the matching degree can be obtained by simply adding the matching degrees of all matching paths.

Model matching. The goal of this step is to find the optimal matching solution with the highest matching degree. The challenge is that it requires coping with a huge search space of possible element combinations [6]. To solve this problem, we propose an N-way model matching algorithm ABCMatch (see details in Section 5). First, the model matching problem is transformed into the weighted maximum matching problem of graph theory. Second, a two-dimensional integer array coding scheme of the food source is proposed by improving the food source encoding in the original ABC algorithm. Third, the strategy for generating candidate solutions is given. Then, all the feasible solutions (food sources) are exploited by employed bees, onlookers, and scouts. Finally, the best matching solution $\{G_1, G_2, ..., G_n\}$ with the highest matching degree is obtained which is used in the next step to generate the matching model.

Model combination. In this step, a single global merged model is generated by combining matched model elements. There exist two challenges in model combination. The first one is that existing approaches [6, 28, 40, 41] ignore relationships in the model and break the chain into pieces rather than reshuffling chained elements, while for model elements connected with relationships, it is necessary to merge not only model elements but also their related nodes. The second one is that N-way model merging is too complex for users to handle conflicts manually, conflicts should be resolved automatically. To solve these problems, we propose a novel approach to reshuffle elements from distinct chains (see details in Section 6). We present the matching model which is an intermediate form between the model matching results (generated by the ABCMatch algorithm) and the merged model. Relevant information needed for conflict resolving and structural merging is represented in the matching model. We present the meta matching model which consists of the type definitions for the objects of the matching model. As shown in Figure 2, we build a temporary matching model based on the proposed meta matching model and the best solution $\{G_1, G_2, ..., G_n\}$ obtained in the model matching step. Based on groups of matched model elements obtained by ABCMatch, matching model elements are generated. And for the relationships in input models, we extract the prior element links and store them in the matching model to memorize the related nodes as well as relationships between them in original models. Finally, we transform the matching model to the merged model.

4. Model comparison

The proposed matching algorithm is for UML class diagrams. We compare model elements from two dimensions: (1) name and (2) properties/methods. Assuming that the vocabulary used for naming model elements, properties, and methods are from corresponding domain terminology, then we can determine whether two elements are similar by checking if they use a similar vocabulary. This section gives the calculation method of similarity degree between any two model elements, based on which, we present equations for calculating matching degrees of matching paths and matching solutions. To make the idea more concrete, an example is given to describe the process of model comparison.

4.1. Calculation of model matching degree

Jaccard similarity coefficient [52] is an index to measure the similarity between two sets. It is widely used to compare the similarity between sample sets of limited sizes. It calculates

the similarity of sets from multiple dimensions. In each dimension, the value is usually between [0, 1]. For example, given two sets A and B, the Jaccard coefficient is defined as the ratio of the size of the intersection of A and B to the size of the union of A and B. In this paper, we use the Jaccard similarity coefficient to calculate the similarity between two model elements. For each pair of elements e_1 and e_2, the similarity degree $Si(e_1, e_2)$ is defined as the average value of the similarity degrees of their names and properties/methods. String comparison is used in the calculation of the name dimension, where the number of characters in the overlapping sub-string is divided by the total number of characters. For the property/method dimension, the similarity degree is calculated by dividing the number of common properties/methods of input elements by the number of properties/methods in the union set of these two elements.

In the following, we extend the above-mentioned calculation method for two elements to support the comparison of multiple elements in a matching path.

The matching path $d = \{e\}_d$ is composed of a set of model elements, and the matching degree $Pi(d)$ of d is the similarity among all model elements in set $\{e\}_d$. Similar to the above-mentioned method, the calculation of $Pi(d)$ also contains the same two dimensions. The first dimension of name is calculated by Equation (1), where $|\bigcap c_d|$ represents the number of characters in the overlapping sub-string of names of all model elements in $\{e\}_d$ and $\sum |c_d| - |\bigcap c_d|$ is the number of the characters in the union set of all names.

$$Similarity\_c\_mul(\{e\}_d) = \frac{|\bigcap c_d|}{\sum |c_d| - |\bigcap c_d|} \tag{1}$$

Suppose that there are N model elements in the matching path $d = e_d$. The similarity degree of properties/methods in d is calculated by Equation (2).

$$Similarity\_pm\_mul\left(\{e\}_d\right) = \frac{\sum\limits_{e_{i,j} \in \{e\}_d, i \neq j} Similarity\_pm(e_i, e_j)}{N(N-1)/2} \tag{2}$$

In Equation (2), the numerator represents the sum of the similarity degrees between any two different model elements in set $d = \{e\}_d$. Normalization of the result is implemented by dividing the total cases of taking any two model elements from N model elements in matching path d.

$$SiD(\{d\}_D) = \sum_{d_i \in \{d\}_D} Pi(d_i) \tag{3}$$

The matching solution $D = \{d_i\}(1 \leq i \leq n)$ contains multiple disjoint matching paths. As shown in Equation (3), SiD is the matching degree of D which can be obtained by simply calculating the sum of the matching degrees of all matching paths. With the help of the proposed equations, we can calculate the matching degree of any given matching path/solution.

4.2. An example of model comparison

In collaborative modeling, suppose that there are k models M_i $(1 \leq i \leq k)$, and each model M_i has m_i $(m_i \geq 1)$ model elements, which are denoted as e_{ij} $(1 \leq j \leq m_i)$. First, we calculate similarity degrees by pairs. Then, model elements from different models whose similarity degrees are larger than the predefined threshold are put into a matching

path d. Multiple matching paths without intersection constitute a matching solution D. The predefined threshold is supposed to be set by users according to actual needs. In this paper, we use $e_{ij} - e_{mn}(se)$ to indicate that the model element e_{ij} is similar to e_{mn} and the similarity degree is se. A case is given to illustrate the calculation process of model comparison.

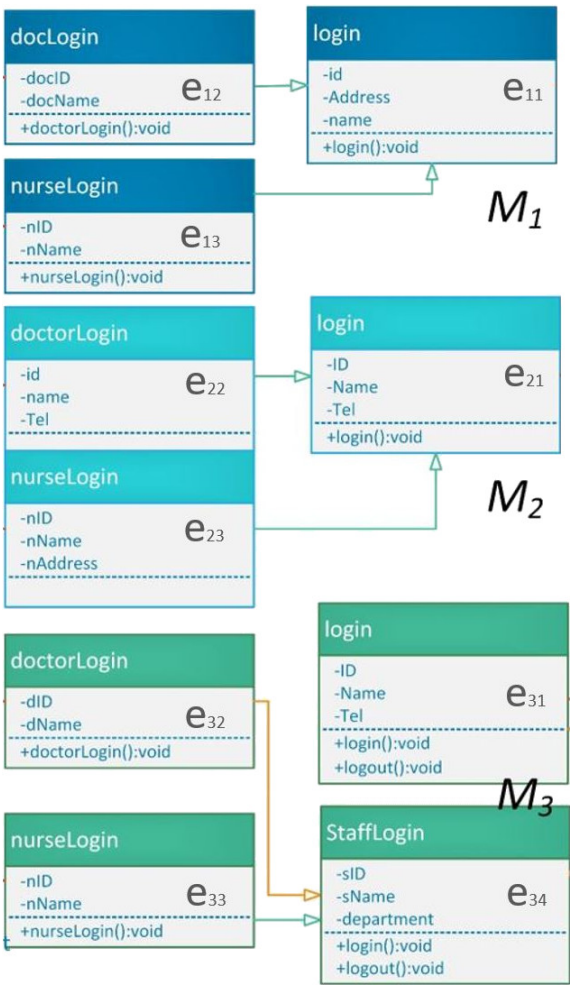


Figure 3. Three models of a "login" function in a medical system software

Figure 3 shows three models of a "login" function in a medical system software. The input models to be merged are in three colors: deep blue (M_1), light blue (M_2) and green (M_3). In the three models, the classes e_{11}, e_{21} and e_{31} describe the login function, the classes e_{12}, e_{22} and e_{32} describe the doctor login function and the classes e_{13}, e_{23} and e_{33} describe the nurse login function, which are supposed to be merged. Calculated by the proposed model comparison method, the similarity degrees of each pair of model elements are: $e_{11} - e_{21}$ (0.8750), $e_{11} - e_{22}$ (0.5983), $e_{11} - e_{31}$ (0.8333), $e_{11} - e_{32}$ (0.3125), $e_{12} - e_{21}$ (0.3846), $e_{12} - e_{22}$ (0.4211), $e_{12} - e_{31}$ (0.3846), $e_{12} - e_{32}$ (0.7179), $e_{21} - e_{31}$ (0.94), $e_{21} - e_{32}$ (0.3125), $e_{22} - e_{31}$ (0.4875), and $e_{22} - e_{32}$ (0.5).

In this example, the predefined threshold is set to 0.5. A matching solution consists of multiple matching paths without intersection. For example, $d_{a1} = \{e_{11}, e_{21}, e_{31}\}$, $d_{a2} = \{e_{12}, e_{22}, e_{32}\}$ and $d_{a3} = \{e_{13}, e_{23}, e_{33}\}$ are three valid matching paths. The three matching

paths form a matching solutions $D = \{d_{a1}, d_{a2}, d_{a3}\}$. According to Equation (1) and Equation (2), the the model matching degrees of three follows: $Pi(d_{a1}) = 0.8462$, $Pi(d_{a2}) = 0.6875$, $Pi(d_{a3}) = 0.9444$. Substituting $Pi(d_{a1})$, $Pi(d_{a2})$, and $Pi(d_{a3})$ into Equation (3), we can obtain the model matching degree of D is $SiD(D) = 2.4781$. The details of the generation of matching paths and matching solutions are described in Section 5.

5. The ABCMatch algorithm

In N-way model matching, it requires coping with a huge search space of possible element combinations to find the optimal matching solution. In this section, first, the model matching problem is transformed into the weighted maximum matching problem of graph theory. Second, a search-based matching approach based on the ABC algorithm [14] is proposed. Then, a two-dimensional integer array coding scheme of the food source is proposed by improving the food source encoding in the original ABC algorithm [14] and the strategy for generating candidate solutions is given. Finally, all the feasible solutions are exploited by employed bees, onlookers, and scouts. By searching for the best model matching solution through the bee colony's exploration of food sources, this approach can find the optimal matching solution with high efficiency.

5.1. The problem of model matching

In an undirected graph G, the points covered by an edge are defined as the endpoints of the edge. The maximum matching problem is to find the largest edge set S that contains the most edges where any endpoint in this graph is covered by only one edge. For weighted graphs, the maximum matching problem is to find an edge set S with the maximum sum of weights.

In this paper, the model elements to be matched are regarded as endpoints in the graph, the matching path containing multiple elements is regarded as the edge of the graph. And the matching degree of the matching path is regarded as the weight of the edge. The goal is to find the optimal matching solution with the highest matching degree. Suppose that $G = (V, E)$ is an undirected graph and endpoint set $V = V_1 \cup V_2 \cup ... \cup V_n$ is composed of n disjoint subsets, where each subset V_i $(1 \leq i \leq n)$ denotes the set of model elements in the i-th model M_i. The edge $E(i, j)$ represents the matching correspondence between element e_i of M_i and model element e_j of M_j. Assuming that there is an edge E_{ik} between e_k and e_i, and e_k does not belong to model M_i or M_j, then the edges $E(i, j)$ and $E(i, k)$ can form a matching path $E(i, j, k)$. The goal to search for the best matching solution is to find a group of matching paths with the maximum weight sum, where endpoints of each path do not intersect with each other.

Suppose that there are k models and the i-th model M_i contains m_i model elements. If no more than one model element is selected from each model to participate in model matching, there will be a total of $\prod_{i=1}^{k}(m_i + 1)$ matching paths. After excluding the situation of empty set or the situation of only one model element, the number of paths p reduces to $\prod_{i=1}^{k}(m_i + 1) - 1 - \sum_{i=1}^{k} m_i$. The set of matching solutions is the power set of all paths

minus the empty set, so there are $2^p - 1$ matching solutions. The optimal solution cannot be obtained in linear time using enumeration methods and it is easy to miss the global optimal solution using greedy algorithms. This paper improved the original ABC (Artificial Bee Colony) algorithm [14] and proposes a search-based N-way model matching algorithm ABCMatch.

5.2. Encoding

In the original ABC algorithm [14], each food source represents a feasible solution of the problem to be solved, and the nectar quantity of the food source represents the fitness of the feasible solution. Bees are divided into three roles: employed bees, onlookers, and scouts. Through the cooperation of these three types of bees, the optimal solution or approximate optimal solution is obtained with high efficiency. In this section, we propose the ABCMatch algorithm improved by the ABC algorithm to solve the N-way model matching problem. In the ABCMatch algorithm, the corresponding relationship between bee colony foraging behavior and model matching problem is given in Table 1.

Table 1. Corresponding relationship between foraging behavior of bee colony and model matching problem

Bees foraging	Model matching
Food source position	Model matching solution
Nectar quality	Model matching degrees
Speed of searching and foraging	Speed of algorithm optimization
The best food source	The optimal model matching solution
Dimension of food source	The first dimension represents matching path The second dimension represents matching solution

In the original ABC algorithm [14], the food source position represents the feasible solution to the optimization problem, which is denoted by a multidimensional vector. In the model matching problem, each candidate solution represents a feasible model matching solution. Therefore, it is necessary to improve the food source encoding and the strategy for generating candidate solutions. Here, we use a two-dimensional integer array coding scheme to code the model matching solution. A matching path is represented by a two-dimensional array d_i, where $d[i][j] = 1$ indicates that the model element e_{ij} is in the matching path, and $d[i][j] = 0$ indicates that it is not in the matching path. Multiple disjoint matching paths compose a matching solution.

5.2.1. Initialization

The number of food sources is denoted as SN. In the initialization phase of the ABC-Match algorithm, SN feasible solutions are generated randomly. Each matching path is a two-dimensional matrix with m rows and m columns, where m is the maximum number of model elements. According to the proposed coding scheme, to select the j-th element from model M_i, we need to set the value of the element in the j-th row and i-th column to "1". If all values in a column are set to "0", which means no model element of the corresponding model is taken.

Next, we check whether all the elements in the matrix are "0" or if there is only one "1" in the matrix. If so, it means that the matching solution is empty or there is only

one model element. In that case, a new matching path is generated. In our approach, a threshold *GSiD* of matching degrees is defined, and the matching degrees of S matching paths are calculated. If the matching degree is less than *GSiD*, a new matching path will be generated until matching degrees of S matching paths are all over *GSiD*. After generating S valid paths, we compose them to generate SN matching solutions. This process needs to satisfy the following rules:

1. Each matching solution has at least one matching path.
2. Matching paths must not intersect.

In the process of path combination, if the rules are not satisfied, a new combination will be generated until SN feasible solutions are generated.

According to Equation (3), the model matching degree $SiD(D_i)$ of each matching solution D_i is calculated as the fitness value of the feasible solution. Among the fitness values of SN feasible solutions, the maximum fitness value max_v and the optimal matching solution $best\_s$ are recorded.

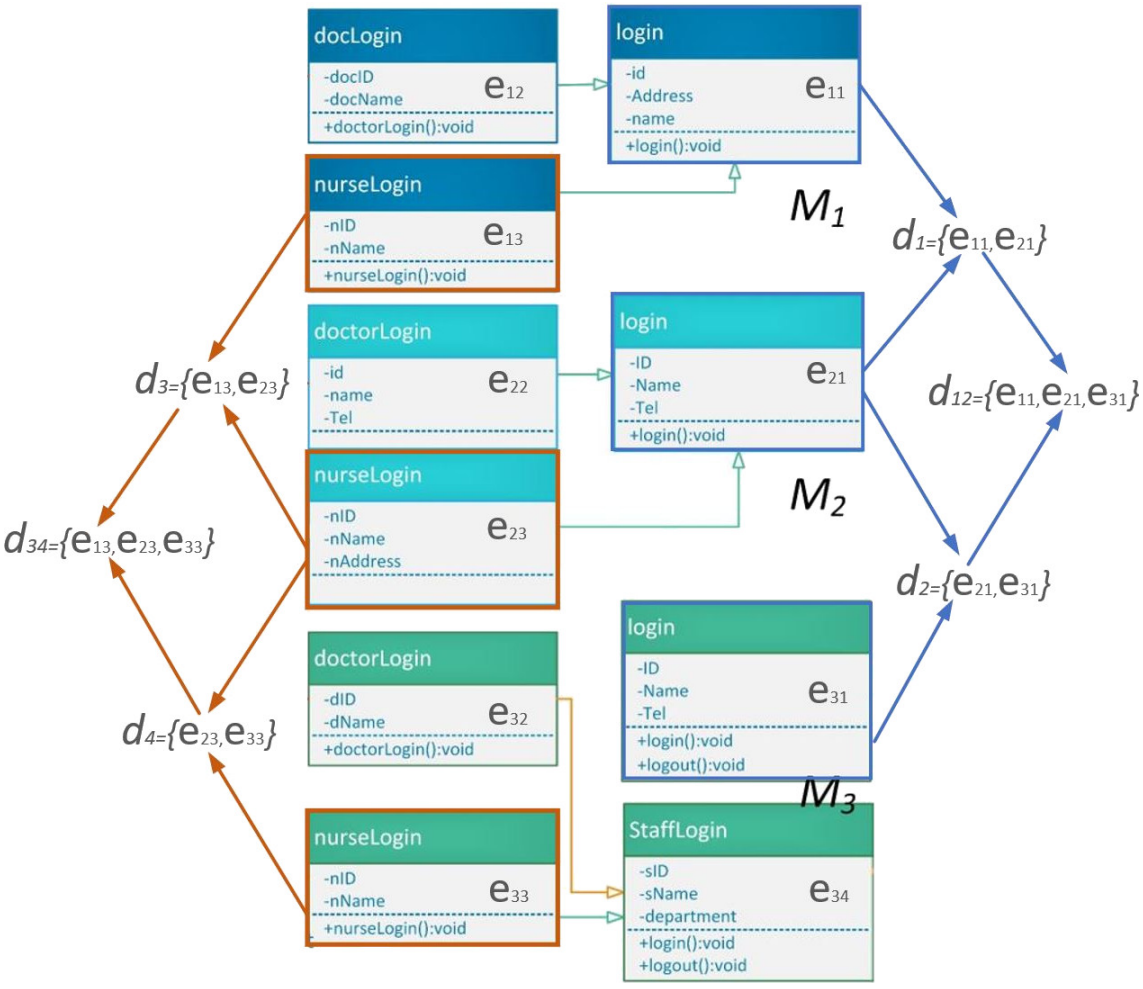


Figure 4. Matching paths in the example

An example of the matching paths is shown in Figure 4. It shows three models of a "login" function in a medical system software. The input models to be merged are in three colors: deep blue (M_1), light blue (M_2) and green (M_3). Matching paths are

a set of elements (class diagrams) from different models. For example, the matching path $d_1 = \{e_{11}, e_{21}\}$ means "login: M_1" and "login:M_2" are matched. Matching paths need to be merged if they have intersections. For example, in Figure 4, the matching paths d_1 and d_2 have intersection, which is consistent with (5), so they are merged to a new path d_{12}. Similarly, d_3 and d_4 are merged to a new path d_{34}.

In the the initialization phase, first, valid paths $d_1 = \{e_{11}, e_{21}\}$, $d_2 = \{e_{11}, e_{31}\}$, $d_3 = \{e_{12}, e_{22}, e_{32}\}$, $d_4 = \{e_{11}, e_{22}\}$, $d_5 = \{e_{13}, e_{23}, e_{33}\}$ are generated randomly. Then, five feasible matching solutions are obtained by composing the five paths $d_1, d_2, ..., d_5$. After deleting solutions with intersecting paths, the matching degrees of all matching solutions are calculated, and the results are shown in Table 2.

Table 2. Matching solutions and model matching degrees

No.	Matching solution	Matching degree
1	$D_1 = \{d_1, d_3\}$	1.5625
2	$D_2 = \{d_2, d_5\}$	1.7777
3	$D_3 = \{d_3, d_4\}$	1.2857
4	$D_4 = \{d_1, d_4\}$	1.4732
5	$D_5 = \{d_2, d_3\}$	1.5208

5.3. Iteration process

After initialization, all the feasible solutions (food sources) will be exploited. Each cycle includes the behaviors of employed bees, onlookers, and scouts.

5.3.1. Employed bees phase

We assign an employed bee to each food source, thus the number of the employed bees is SN too. In the initial matching solution, there are only S matching paths. The employed bee exploits the neighborhood of the food source and adds matching paths. Because adding a new matching path may cause conflicts with existing matching paths, it is necessary to determine whether to delete the conflict elements in existing paths and add the new path or just stay unchanged. In our approach, the decision is made according to the matching degree. At the beginning of the cycle, each matching path d_y in the food source is matched with each path d_x in solution D_i and the conflict rate between them is calculated. Suppose that there are k models where each model M_i has m_i model elements, the conflict rate is denoted as P, and the boolean variable m is used to indicate whether the composition of paths is necessary. There are three situations as follows:

1. If the collision rate of matching paths is 0 which means that all paths are disjoint, then there is no need to merge the paths, which is formally expressed as (4).

$$\forall i \in [1,k], \forall j \in [1,m_i], d_x[i][j] + d_y[i][j] \leq 1 \rightarrow P = 0, M = 0 \tag{4}$$

2. If two paths have only one common element and there is no other model elements from the same model in these paths, then the conflict rate is 0 and the composition of paths is necessary. It is formally expressed as (5).

$$\begin{aligned} &\forall i, p, q \in [1,k], \forall j, t \in [1,m_i], \exists d_x[i][j] + d_y[i][j] > 1 \\ &\wedge\, j \neq t \wedge d_x[p][t] + d_y[q][t] \leq 1 \rightarrow P = 0, M = 1 \end{aligned} \tag{5}$$

3. If there exist elements in the intersecting path from the same model, a conflict occurs. It is formally expressed as (6).

$$\begin{aligned} &\forall i,p,q \in [1,k], \forall j,t \in [1,m_i], \exists d_x[i][j] + d_y[i][j] > 1 \\ &\wedge j \neq t \wedge d_x[p][t] + d_y[q][t] > 1 \rightarrow P > 0 \end{aligned} \tag{6}$$

The computation method of the conflict rate P is as (7).

$$P = \frac{n}{(m_i + m_k) \times (m_i + m_k - 1)/2} \tag{7}$$

In (7), n is the number of conflict element pairs. The conflict rate is the ratio of the number of conflict pairs to the number of possible cases of taking any pair of elements in the two matching paths.

In this approach, we add the new path according to the above three situations. If $P = 0$, $M = 0$, the path will be directly added to the matching solution; If $P = 0$, $M = 1$, the path will be merged into the path that intersects with it in the matching solution. If $P > 0$, we select the matching path d_y whose conflict rate P is less than the predefined threshold P', and add the path d_y to the matching solution D_i to generate matching solution *Neighbour_*D_i. Then, we delete the model elements in the original solution that conflict with the new path. In the above-mentioned example, the paths of D_2 are re-matched. After generating new matching paths according to the rules, a new matching solution D'_2 is obtained. A new matching solution *Neighbour_*D_i is a new food source. If the model matching degree of *Neighbour_*D_i is greater than that of D_i, the food source will be updated.

In the above example, for the first employed bee, it selects food source D_1 and adds it to D_2. Then, the path in D_1 with the least matching conflict to D_2 is added to solution D_2, and the new food source is $[d_1, d_2, d_3, d_5]$. As d_1 and d_2 have intersection, which is consistent with (5), they are merged to a new path d'_{12} as shown in Figure 4. After merging, we can get the new matching solution $D'_2 = \{d'_{12}, d_3, d_5\}$. Then we calculate the matching degree $SiD(D'_2) = 2.4781$, which is better than the original $D_2 = 1.7777$, so we accept the new food source. When all employed bees have completed the search, the new population is shown in Table 3. The number updated in this iteration is indicated by the underline.

Table 3. Updated matching solutions and matching degrees

No.	Matching solution	Matching degree
1	$D_1 = \{d_1, d_3\}$	1.5625
2	$D'_2 = \{d'_{12}, d_3, d_5\}$	<u>2.4781</u>
3	$D'_3 = \{d'_3, d_4\}$	**1.5382**
4	$D_4 = \{d_1, d_4\}$	1.4732
5	$D_5 = \{d_2, d_3\}$	1.5208

5.3.2. Onlookers phase

Each onlooker selects a food source according to the probability which is proportional to the nectar quality. The selecting probability P_i is calculated by (8) [53].

$$P_i = \frac{0.9 \times \text{fit}(D_i)}{\max_{i=1}^{SN} \text{fit}(D_i)} + 0.1 \tag{8}$$

In (8), P_i is the fitness value of solution D_i, and in this paper, fit calculates the matching degree *SiD*. Onlookers select the food source by the roulette mechanism. First, it generates a random number. If the number is greater than the random number, then the onlooker will not move. Otherwise, the onlooker will attach itself to the food source and exploit its neighborhood. The greedy selection strategy is used to update the food source. Obviously, according to the selection method, the food source with higher fitness will attract more onlookers.

In the above example, the selecting probabilities are 1, 0.95, 0.87, 0,43, 0.1. For the third onlooker, first, a random number is generated, which is smaller than P_3, and then this onlooker will exploit the solution. And the new solution is $D'_3 = \{d_2, d'_5\}$. The new matching degree $SiD = 1.79 > 1.52$, thus the matching solution will be updated. For the fifth onlooker, the random number is greater than 0.1, so it does not move. After all the onlookers finish the search, the new population is shown in Table 3, and the updated number is indicated in bold.

5.3.3. Scouts phase

If a solution is not updated after limited iterations (set to 5 in the proposed algorithm), then this food source will be abandoned. The associated employed bee will become a scout and randomly generate a new food source. For the above example, after the employed bees and onlookers finish the search, the trial is [1, 3, 3, 4, 1], which did not reach the maximum value limit, thus the scout will not appear. After all the food sources are explored, the best matching solution *best_s* and the optimal matching degree *max_v* will be updated, and the next iteration will begin. Until now, *max_v* is 2.4781, and the corresponding *best_s* is $\{d'_{12}, d_3, d_5\}$. In this matching solution, the following three sets of classes are matched: $\{e_{11}, e_{21}, e_{31}\}$, $\{e_{12}, e_{22}, e_{32}\}$, and $\{e_{13}, e_{23}, e_{33}\}$.

6. Model combination

This section defines the matching model and introduces the approach of transforming the matching model to the final merged model. First, the meta-model is presented. Second, an example is given on how to build a matching model from the matching elements obtained in the model matching step. Then, we introduce how to generate the final merged model from the matching model. Finally, we prove that our approach satisfies the Generic Merge Requirements [32].

6.1. Meta-model

The proposed meta-model is given in Figure 5. It consists of the following type definitions for the objects in the matching model.

1. ***Matching element.*** *Matching elements* are the first-class objects in a matching model. Each *matching element* consists of four properties: the *Input_Mo*

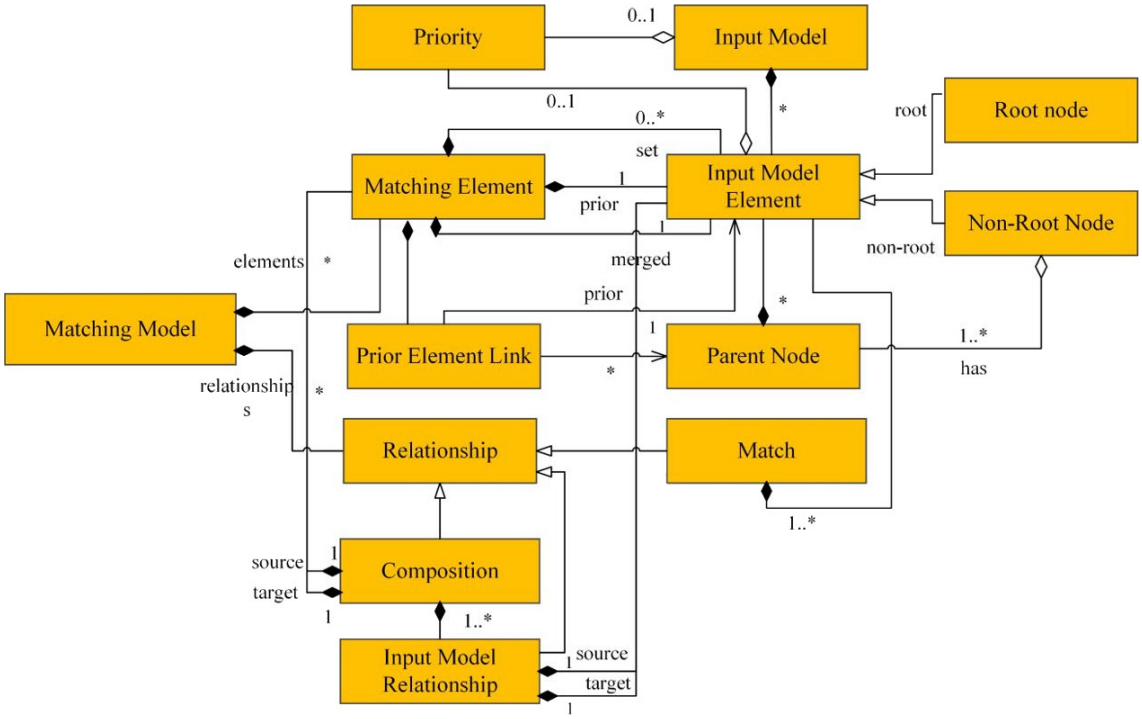


Figure 5. The proposed meta-model of the matching model

del_Element_Set, the *Prior_Input_Model_Element*, the *Merged_Input_Model_Element* and the *Prior_Element_Link*.

The *Input_Model_Element_Set* is a set of matched model elements obtained by the proposed ABCMatch algorithm. By merging the model elements in an *Input_Model_Element_Set*, the *Merged_Input_Model_Element* is generated. Each input model to be merged is given a different priority number. When conflicts occur during model merging, the model element with the highest priority is picked, and this element is called the *Prior_Input_Model_Element*. By searching the direct and indirect parent nodes until finding a root node of the *Prior_Input_Model_Element*, the *Prior_Element_Link* is obtained, which is used to build relationships of the *Merged_Input_Model_Elements* in the matching model.

2. ***Relationship.*** There are three types of relationships in the matching model: *Match*, *Composition* and *Input_Model_Relationship*. The *Match* is the relationship among multiple model elements, it represents correspondences among similar/common model elements from different models. We define *Composition* as the binary relationship between matching elements which indicates that a matching element consists of another matching element. For example, suppose that MA and MB are two matching elements where A and B are *Merged_Input_Model_Elements* in MA and MB, respectively. If B is the subclass of A, then a *Composition* relationship should be built from MB to MA. As matching is not only between the model elements themselves but also their related model elements. In addition to source and target matching elements, the relationship *Composition* also has a property that records relationship types between input model elements. The *Input_Model_Relationships* are binary relationships between input model elements, they are defined by the meta-model of input models. In this paper, input

models to be merged are UML class diagrams, so *Input_Model_Relationships* are UML relationships such as generalization, association, etc.

6.2. The model matching process

We define two operators for model matching. The operator *match* is to find a set of matched model elements from multiple models built by different collaborators and generate matching elements. The operator *connect* is to build relationships between matching elements and model elements. After operating *match* and *connect* in sequential order on a group of input models, a matching model is generated. In our approach, the models are merged based on the matching models. The inputs of *match* are as follows:

1. Input models: $M_1, M_2, \ldots, M_n$.
2. Priority of input models: $Pr_1, Pr_2, \ldots, Pr_n$.

Each model element has a priority number the same as the model it belongs to. When conflicts occur, the model element with the highest priority is picked.

The semantics of *match* is defined as follows:
The function $match((M_1, Pr_1), (M_2, Pr_2), \ldots, (M_n, Pr_n)) \longrightarrow ME$ matches n models $M_1, M_2, \ldots, M_n$ based on similarity, the detailed description of similarity calculation is given in Section 4. The function consists of three steps as follows. First, it searches for the optimal matching solution which contains multiple matching paths of similar model elements. Second, it generates a merged element for each matching path. When facing representation conflicts, the representation of the element with the highest priority is picked. Third, it records the related elements and relationships of the prior element. Finally, a matching model element ME is generated.

The semantics of *connect* is defined as follows:
The function $connect(ME_1, ME_2, \ldots, ME_n) \longrightarrow GM$ connects n matching elements ME_1, $ME_2, \ldots, ME_n$ based on original model relationships and generates a matching model GM. Relationships are added according to three different situations. (1) For each matching element ME_i $(1 \leq i \leq n)$, we search its *Prior_Element_Link*. If the model element in the link does not exist in GM, then we add it to GM as well as its relationships. (2) If the model element already exists in a matching element ME_j $(1 \leq j \leq n, i \neq j)$ in GM and there is no relationship between ME_i and ME_j, then the relationship *Composition* is added from ME_j to ME_i. And its property *Input_Model_Relationship* is set the same as the relationship in the *Prior_Element_Link*. (3) If the added model element or the matching element has related nodes, then we search its element link and repeat the steps in the first two situations.

6.3. An example of the matching model

An example of the matching model is shown in Figure 6. It shows three models of a "login" function in a medical system software. The input models to be merged are in three colors: deep blue (M_1), light blue (M_2) and green (M_3). The matching elements generated are in red and the *Merged_Input_Model_Elements* are in blue. We define that the priority order of these models is $M_3 > M_2 > M_1$.

As shown in Figure 6, matching elements are generated based on these matching paths. Take the the matching path $d_1 = \{e_{11}, e_{21}, e_{31}\}$ as an example, "Matching element *login*" is generated which consists of three properties, the set {login: M_1, login: M_2, login: M_3}, the prior element "login: M_3", and the merged element "Merged_Input_Model_Element *login*". As "login" is the root node which does not have a parent node in this example, there is no

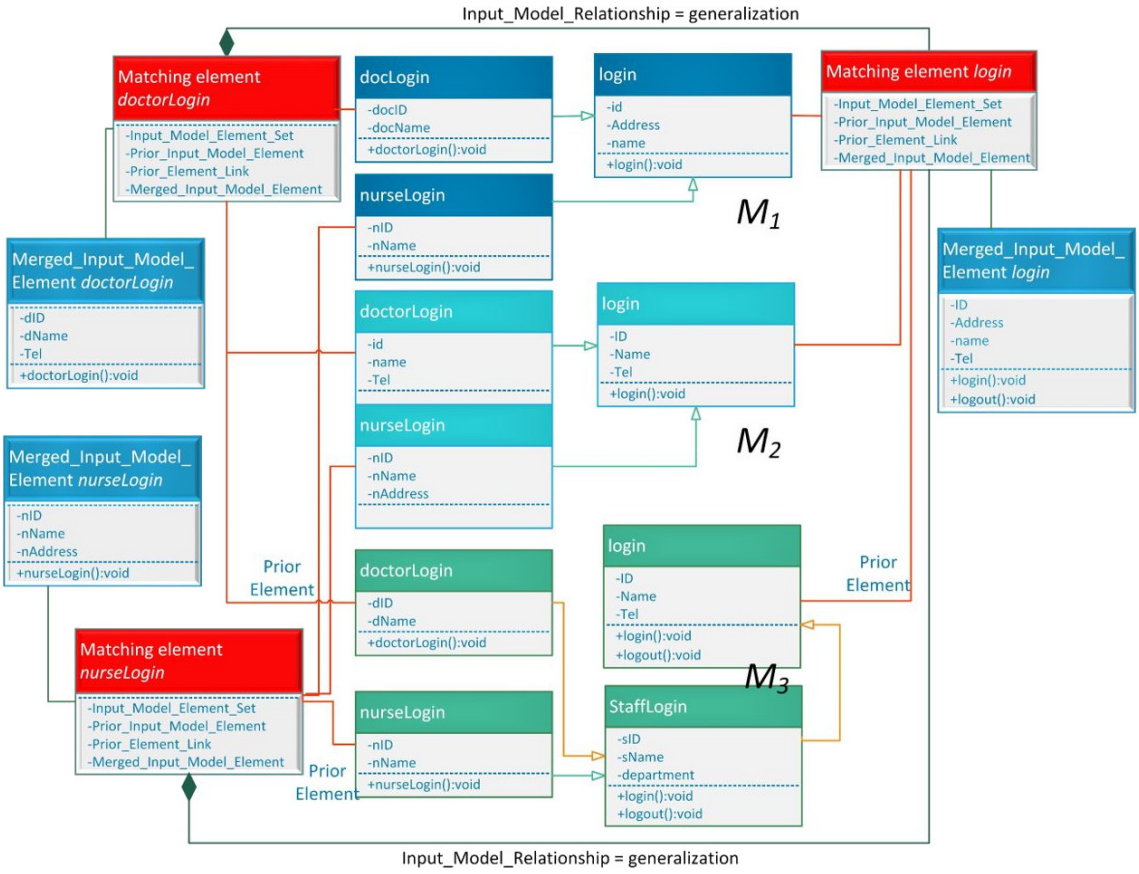


Figure 6. An example of the matching model

Prior_Element_Link in this matching element. "Merged_Input_Model_Element *login*" contains all properties and methods of the original models. It is a new "duplicate-free" model element obtained by merging "login: M_1", "login: M_2", and "login: M_3".

Models have relationships, so it is necessary to compare matched elements' related nodes which are connected by model relationships. Searching all related nodes of model elements is time-consuming. To improve the efficiency of this process, we search the prior element link of each merged element from itself to the root node and compare these nodes to other matching elements' merged elements. First, we find out all groups of matched elements and generate a merged element for each group. Second, the prior element and prior element link is memorized in each group where the merged element shares the same link with the prior element. Then, matching elements are generated and relationships between merged elements are built with the help of the *Prior_Element_Links*. For example, in Figure 6, "doctorLogin: M_3" is the prior element of "Merged_Input_Model_Element *login*". And its *Prior_Element_Link* is highlighted in orange in Figure 6. As "login: M_3" in the *Prior_Element_Link* exists in the *Input_Model_Element_Set* of "Matching element *login*", so it is replaced by "Merged_Input_Model_Element *login*".

6.4. Model merging

We merge model elements and generate a new global model M' by operating on the matching model. First, model elements that are not in *Prior_Element_Links* are deleted.

For example, "docLogin: M_1" and "login:M_1" are deleted while "StaffLogin: M_3" is reserved as it is in the *Prior_Element_Link* of "Matching element *doctorLogin*".

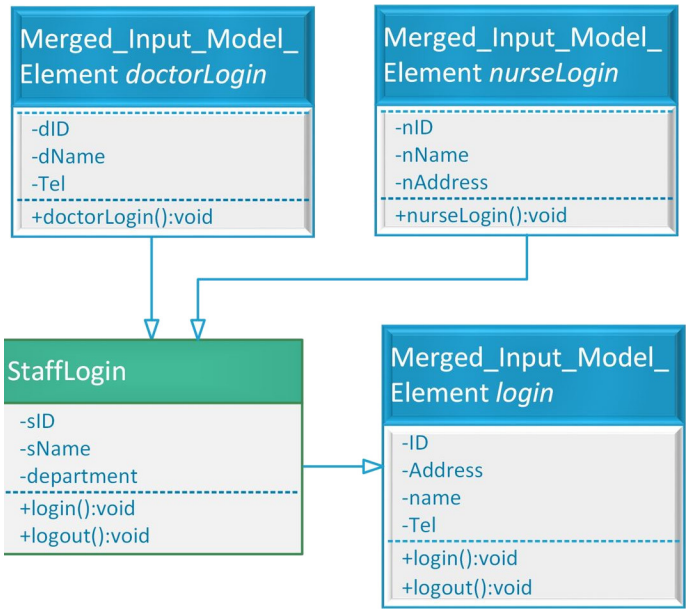


Figure 7. The merged model M' in the example

Second, merged elements are connected to model elements in *Prior_Element_Links*. For example, in the *Prior_Element_Link* of "Matching element *doctorLogin*", there is a generalization relationship which connects "doctorLogin: M_3" with "StaffLogin: M_3", so we build the same relationship between "Merged_Input_Model_Element *doctorLogin*" and "StaffLogin: M_3". Note that in this step, if the model element in the *Prior_Element_Link* exists in any matching element's model element set, it means there is a merged element of this model element. In that case, we replace this model element with the corresponding merged element. For example, "Login: M_3" is replaced by "Merged_Input_Model_Element *login*". Finally, input model elements that do not match with other elements are added to the merged model directly. In this example, the final merged model M' is given in Figure 7.

The proposed model merging method satisfies the following Generic Merge Requirements [32]. Signals used are given in Table 4.

1. ***Element preservation.*** Each element of source models has a corresponding element in the target model. Formally, for each element $e_{ij} \in M_i (1 \leq i \leq n, j > 0)$, if it has one or more matching elements in other input models, they will be matched and generate a merged model element MME_k $(k > 0)$ in the matching model ME_i, which will be added into the merged model M'. So e_{ij} must have a corresponding element $e' = MME_j \in M'$. If e_{ij} does not have any matching element but it exists in the prior element link PEL_q $(q > 0)$, then it is added to M' and connected with the merged element MME_q or model elements in $PELq$. If e_{ij} does not have any matching element and does not exist in any prior element link, it will also be added to M' with its related model elements and relations in the source model. So we can conclude that for any element $e_{ij} \in M_i$ $(1 \leq i \leq n, j > 0)$, there is always a corresponding element $e'_{ij} \in M'$.
2. ***Equality preservation.*** Input elements of source models are mapped to the same element in merged model M' if and only if they are equal in the mapping, where equality

Table 4. Signals illustration

Signal	Meaning
n	The number of input models.
M_i	The i-th input model ($1 \leq i \leq n$).
Pr_i	The priority number of model M_i.
e_{ij}	The j-th model elements in $M_i (1 \leq i \leq n)$.
M'	The global model after model merging.
GM	The matching model.
ME_i	The i-th matching element in $GM (1 \leq i \leq n)$.
MES_i	The set of model elements in matching element ME_i.
PE_i	The prior model element in matching element ME_i.
MME_i	The merged model element in matching element ME_i.
PEL_i	The prior element link in matching element ME_i.
$M(e, e')$	The input model element e has a unique corresponding element e' in M'.
$Eq(e_i, e_j)$	Input model elements e_i and e_j are equal in model merging.
$R(e_i, e_j)$	The relationship between model elements e_i and e_j.
$V(e, p)$	The value of the property p in the model element e.

in the mapping is transitive. Formally, suppose that $e_i \in M_i$, $e_j \in M_j$ and $Eq(e_i, e_j)$, which means that e_i and e_j are equal in the process of generating a matching element ME_i. Then $e_i, e_j \in MES_i$. And $M(e_i, e')$, $M(e_j, e')$ where $e' = MME_i \in M'$. Suppose that $Eq(e_i, e_j)$, $Eq(e_i, e_k)$, then $e_i, e_j \in MES_1$, $M(e_i, e'_1)$, $M(e_j, e'_1)$ and $e_i, e_k \in MES_2$, $M(e_i, e'_2)$, $M(e_k, e'_2)$. As e_i's corresponding element e' is unique in M', $e'_1 = e'_2 = e'$. So $M(e_j, e')$, $M(e_k, e')$ which means that equality in the mapping is transitive. If e_i and e_j are not equal in the mapping, then there is no such e', so that e_i and e_j correspond to different element in M'.

3. ***Relationship preservation.*** Each input relationship is explicitly in or implied by target model M'. Formally, for each relationship $R(e_{ik}, e_{ij})$ between e_{ik} and e_{ij} in M_i, there are two cases. If one or both of e_{ik} and e_{ij} have matching elements such that $e_{ik} \in MES_i$ or $e_{jk} \in MES_j$, then $R(e_{ik}, e_{ij})$ is recorded in PEL_i or PEL_j. And $R(e_{ik}, e_{ij})$ will be added to M'. If neither of e_{ik} and e_{ij} has matching element, they will both be added to M' as well as the relationship $R(e_{ik}, e_{ij})$ between them in the last step of model merging.
4. ***Similarity preservation.*** Elements that are declared to be similar (but not equal) to one another in mapping from one model to another retain their separate identity in target model and are related to each other by some relationship. Formally, for each pair of similar but not equal elements $e_i \in M_i$ and $e_j \in M_j$. There exist elements e'_i and $e'_j \in M'$ and $M(e_i, e'_i)$, $M(e_j, e'_j)$. As $Eq(e_i, e_j)$ is not true, e_i and e_j correspond to different elements in M', so $e'_i \neq e'_j$. Suppose $R(e_i, e_j)$ is the relationship between e_i and e_j. As relationship preservation is proved, there must be a relationship $R(e'_i, e'_j)$ between e'_i and e'_j in M'.
5. ***Meta-meta-model constraint satisfaction.*** There are meta-meta-model conflicts caused by one-type constraint and no-cycle constraint in model merging, we solve these problems by appointing a prior model and picking the elements in the prior model when conflicts occur. As the prior model is a correct UML model that satisfies all meta-meta-model constraints, the merged model M' satisfies them too.
6. ***Extraneous item prohibition.*** Other than the elements and relationships specified in source models, no additional elements or relationships exist in merged model. Formally, for each model element $e' \in M'$, $e' = e_i \in M_i$ if e_i has no matching element or $e' = MME_i$ which is a merged element whose properties and methods all come from

elements in $MES_i \in \{M_1, M_2...M_n\}$. For each relationship $R(e'_i, e'_j) \in M'$, there exist $e_i, e_j \in M_i$, $M(e_i, e'_i)$, $M(e_j, e'_j)$ such that $R(e'_i, e'_j) = R(e_i, e_j) \in M_i$.

7. ***Property preservation.*** For each element property would be presented only if property of element is mapped exactly to element in target model. The goal is to prove for each element $e' \in M'$, e' has property p if and only if $\exists e_i \in M_i$, $M(e_i, e')$ and e_i has property p. Suppose that $\exists e_i \in M_i$, $M(e_i, e')$ and e_i has property p. If e' is not a merged element, then $e' = e_i$, so if e has property p, e' has p too. If e' is a merged element in matching element ME_i, and e_i is the prior element PE_i, then e' has all properties of e_i. If e_i is not PE_i, $\exists PE_i \in M_j$, and PE_i has the property p' which is the same property in different representation with p, such that e' has p' instead of p. If there is no such p' , then e' has p. If for each $e_i \in M_i (1 \leq i \leq n)$, $M(e_i, e')$, e_i does not have property p, then e' does not have property p.
8. ***Value preference.*** For each element in merged model M', its property value is chosen from mapping elements. Suppose that $\exists e_i \in M_i$, $M(e_i, e')$ and e_i has property p. For each element $e' \in M'$, if e' is not a merged element, then $e' = e_i$, $V(p, e_i) = V(p, e')$. If e' is a merged element in matching element ME_i, and e_i is the prior element PE_i, then $V(p, e_i) = V(p, e')$. If e_i is not PE_i, $\exists PE_i \in M_j$ has the property p' which is the same property in different representation with p, then $V(p', PE_i) = V(p', e')$. So for each $e' \in M'$, $p(e')$ is chosen from the mapping element corresponding to e' or the prior element PE_i corresponding to e'.

7. Case study

We implement a prototype tool of the proposed approach in Java. The source code of the tool is available online [54]. The code dictionary and the view of the proposed N-way model merging tool are given in Figure 8. This tool can be used to merge different versions of UML class diagrams modeled in the famous Papyrus modeling environment [42]. Users are supposed to run this tool in the Eclipse IDE. First, users need to move the different versions of UML class diagrams into the folder named $Models$ in the directory of the tool. Then, users need to set the relations of the UML versions as the example given in the file $version\_r$ in the $Models$ folder. After refreshing the tool view in Eclipse, users can see the diagram of the versions and the list of the versions as shown in Figure 8. Users need to select the versions to be merged by selecting corresponding checkbox. By editing the priority file following the example in the $Models$ folder, priority can be set to facilitate conflict prevention based on priority. After that, users need to click the button $Merge\ Selected\ Versions$ to accomplish model merging.

The proposed ABCMatch algorithm is a search-based meta-heuristic algorithm to search for the optimal matching solution. So we evaluate the algorithm by comparing it with existing novel meta-heuristic algorithms. The implemented tool merge the models according to the optimal matching solution. We evaluate the scalability of the tool by inviting participants to use the tool.

The purpose of the case study is to evaluate the effectiveness of the proposed method in real-world collaborative modeling. However, existing N-way model merging methods [6, 27, 28, 40] cannot directly process and generate complete UML diagrams as our tool does, they require a lot of manual processing. In addition, these methods do not provide publically available tools for us to compare with. EMFStore [4] is one of the most widely-used

version control systems in real-world collaborative modeling and it supports two-way model merging, so we choose it to compare our tool with.

This section evaluates the proposed approach by answering the following research questions.

- **RQ1.** Compared with the existing version control system, is the proposed N-way model merging approach more effective?
- **RQ2.** What are users' views on the usefulness of the proposed N-way model merging approach compared with the existing version control system?
- **RQ3.** How is the performance of the proposed ABCMatch algorithm compared with existing novel meta-heuristic algorithms?

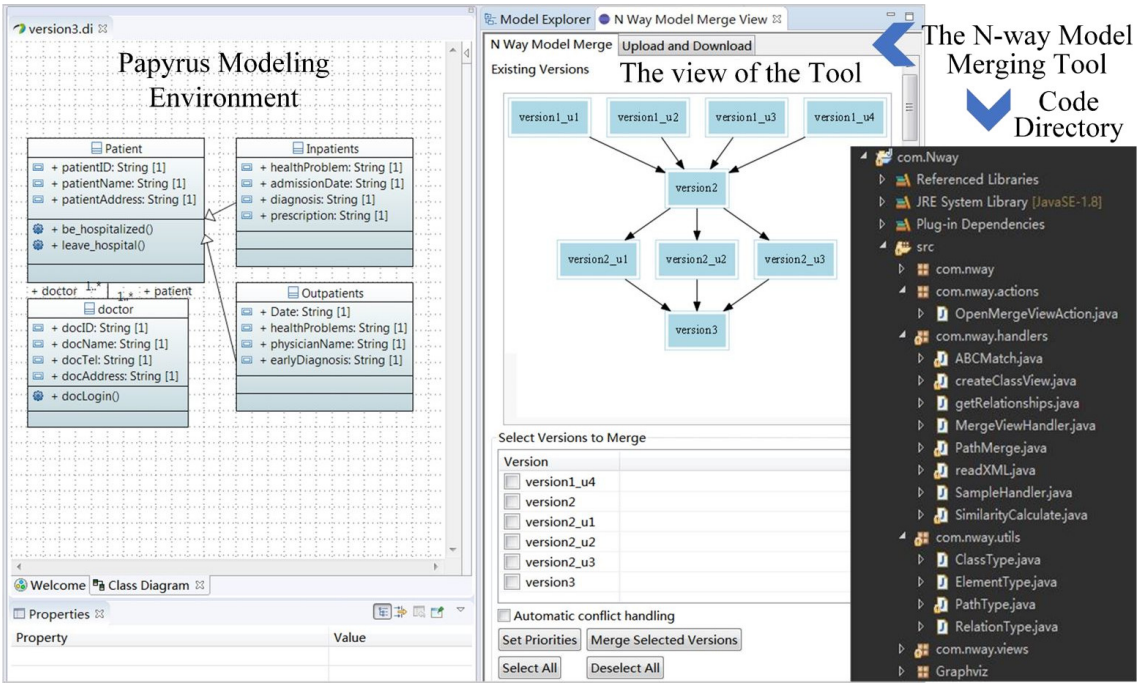


Figure 8. The code dictionary and the view of the proposed N-way model merging tool

7.1. Experiment

7.1.1. Participants

We selected 30 participants from the College of Computer Science and Technology of Nanjing University of Aeronautics and Astronautics. The selected participants are graduate students majoring in software engineering, The participants meet the following requirements:

1. At least 2 years of software modeling experience.
2. Have experience in using the model management tool EMFStore [4].

7.1.2. Modeling tasks

We designed the following two tasks with different workloads:

Task 1. Use the UML class diagram to model the following information in a library information system: (1) system users such as administrators, teachers, and students. (2) system functions available to each type of user. Require a minimum of 40 classes in the submitted model.

Task 2. Use the UML class diagram to model the following information in a common hospital information system: (1) departments, (2) medical staff, (3) system functions available to medical staff, (4) wards, (5) patients, (6) system functions available to patients. Require a minimum of 80 classes in the submitted model.

We divided the participants into two groups, Group 1 and Group 2. Based on the information provided by the mentors of the participating students, we tried to make the modeling levels of the two groups as equal as possible. Group 1 and Group 2 were the experiment group and the control group, respectively, and were supposed to work on the same tasks with the N-way model merging tool developed in this paper and EMFStore [4], respectively. We asked participants to record the following information during the experiment: (1) the time taken to complete the task, (2) the number of model merges performed, (3) the number of conflicts, and (4) the total time taken for resolving conflicts, including the time taken for group discussion, the time taken for modifying conflicting models and the time taken for remerging models using the tool. In addition, we asked participants in the same group to work together in the same period of time each day for modeling, model merging, and group discussion to facilitate time consumption statistics.

7.1.3. Questionnaire

To compare participants' views of the tools, we designed the following questionnaire:

1. How useful is the model merging function supported by the tool in collaborative modeling?
   A. Very useful; **B.** Useful; **C.** A little useful; **D.** Not useful.
2. How useful is the conflict handling function supported by the tool in collaborative modeling?
   A. Very useful; **B.** Useful; **C.** A little useful; **D.** Not useful.
3. Which tool do you prefer as the model merging tool for collaborative modeling?
   A. The N-way model merging tool; **B.** EMFStore.

7.1.4. Evaluation of ABCMatch

In recent years, numerous novel meta-heuristic algorithms have been proposed to solve optimization problems in various fields. However, most of these algorithms are not suitable for solving the N-way model matching problem. This is because in the N-way model matching problem, each matching solution consists of multiple matching paths, and each matching path contains a set of UML classes. When using meta-heuristic algorithms to solve the N-way model matching problem, the locations of different agents are supposed to represent different matching solutions. However, most of the existing novel meta-heuristic algorithms update the locations of agents through complex numerical calculations, which cannot represent the update of the matching paths and the UML classes contained in matching paths in each model matching solution. Therefore, this paper cannot compare ABCMatch with the latest meta-heuristic algorithms which use complex numerical calculations to update locations of agents, such as Sparrow Search Algorithm (SSA) [21], Honey Badger Aptimization (HBA) [23], and Northern Goshawk Optimization (NGO) [24].

As a meta-heuristic algorithm, the reason why the ABC algorithm can be improved to solve the N-way model matching problem is that ABC updates the locations of agents by searching adjacent locations with higher fitness without complex numerical calculations. Therefore, this paper can improve ABC and propose ABCMatch to solve the N-way model matching problem. ABCMatch regards two matching solutions with only one matching path to be different as neighbors when updating the locations of agents. We implemented the ABCMatch algorithm in Eclipse using Java. The parameter configuration of the algorithm is shown in Table 5.

Table 5. Parameter configuration

Algorithms	Description	Parameters	Value
GA[15]	Population size	PG	90
	Crossover rate	PC	0.9
	Mutation rate	PM	0.01
EHO[20]	Population size	PE	90
	Number of clans in elephant population	NClan	9
	Number of elephants in each clan	Nc	10
	Random number in the separating process	R	[0, 1]
ABCMatch	Population size	PA	90
	Number of employed bees	Ne	30
	Number of onlookers	No	30
	Number of scouts	Ns	30
	Initial number of matching paths in each matching solution	S	10

In order to evaluate the performance of the proposed ABCMatch algorithm by comparing it with existing novel meta-heuristic algorithms in the literature, we analyzed 23 existing state-of-the-art meta-heuristic algorithms to find the algorithms that can be used to solve the N-way model matching problem. These 23 algorithms are Genetic Algorithm (GA) [15], Gray Wolf Optimization (GWO) [16], Symbiotic Organisms Search (SOS) [17], Whale Optimization Algorithm (WOA) [18], Farmland Fertility Algorithm (FFA) [19],Elephant Herding Optimization (EHO) [20], Sparrow Search Algorithm (SSA) [21], Tunicate Swarm Algorithm (TSA) [22], Honey Badger Optimization (HBA) [23], and Northern Goshawk Optimization (NGO) [24] Ant Colony Optimization (ACO) [55], Particle Swarm Optimization (PSO) [56], Invasive Weed Optimization (IWO) [57], Firefly Algorithm (FA) [58], Fruit Fly Optimization (FFO) [59], Flower Pollination Algorithm (FPA) [60], Moth-Flame Optimization (MFO) [61], Crow Search Algorithm (CSA) [62], Dragonfly algorithm (DA) [63], Grasshopper Optimization Algorithm (GOA) [64], Spotted Hyena Optimization (SHO) [65], Emperor Penguin Optimization (EPO) [66], Butterfly Optimization Algorithm (BOA) [67]. Among the above algorithms, only GA[15] and EHO[20] do not need to update the positions of agents through complex numerical calculations, so they can be used to solve the N-way model matching problem.

Genetic algorithm (GA) [15]: GA is a classic meta-heuristic algorithm inspired by natural evolution theory. GA simulates the phenomena of replication, crossover and mutation in natural selection and genetics. Starting from any initial population, through random selection, crossover and mutation operations, it generates a group of individuals more suitable for the environment, so that the population evolves to a region with higher adaptability in the search space, and continues to reproduce and evolve from generation to generation, and finally converges to a group of individuals most suitable for the environment,

so as to obtain the optimal solution of the problem. To compare ABCMatch with GA, we implemented GA using Java in Eclipse, and searched for the optimal solution of the N-way model matching problem by selecting, crossing and mutating the matching paths in the model matching solutions. The parameter configuration of GA implemented in this paper is shown in Table 5.

Elephant Herding Optimization (EHO) [20]: EHO is an advanced swarm intelligence optimization algorithm that has been applied to optimization problems in many fields. This algorithm mainly simulates the herding behavior of elephant groups. In nature, an elephant group can be divided into multiple clans, and each clan has a matriarch as the leader. Elephants belonging to different clans live under the leadership of the matriarch (the best position in the clan). EHO has two operations: clan updating and separating. In the clan updating operation, the position of each elephant is updated according to its position and the position of the matriarch. In the separating operation, elephants with the worst finesses will be moved to new locations to increase the global search ability of the population. To compare ABCMatch with EHO, we implemented EHO in Eclipse using Java. In the implemented EHO algorithm, we regarded multiple sets of model matching solutions as multiple clans in an elephant group and took the best matching solution in each set as the matriarch in each group. For separating, new matching paths were randomly generated to replace the matching solutions with poor matching degrees. The parameter configuration of the EHO algorithm implemented in this paper is shown in Table 5.

Table 6. Details of the data sets

Data sets		Number of class diagrams	Number of matching paths	Number of class diagrams that can be matched
Class diagrams of the library information system (Task 1)	ModelSet1	332	25	233
	ModelSet2	380	28	271
	ModelSet3	417	32	300
	ModelSet4	422	32	304
	ModelSet5	426	33	308
Class diagrams of the hospital information system (Task 2)	ModelSet6	701	60	589
	ModelSet7	785	69	651
	ModelSet8	803	73	702
	ModelSet9	812	74	710
	ModelSet10	820	76	718

Data sets. In order to evaluate the performance of ABCMatch in solving the N-way model matching problem, this paper uses the class diagrams of the library information system and the hospital information system established by the participants in Task 1 and Task 2 as the experimental data sets, and compared the ABCMatch algorithm proposed in this paper with Genetic algorithm (GA) [15] and Elephant Herding Optimization (EHO) [20]. The details of the data sets are shown in Table 6. In Task 1, Group 1 conducted a total number of 13 times of model merging, and we randomly selected five times among them to collect the models to be merged. The selected models are from the first, the third, the tenth, the eleventh, and the thirteenth times of model merging and we named them as ModelSet1 to ModelSet5 in sequence as the experimental data sets. In Task 2, Group 1 conducted a total number of 29 times of model merging. We randomly selected the models from the fourth, the tenth, the thirteenth, the twenty-second and the twenty-ninth

times of model merging, and named them as ModelSet6 to ModelSet10 in sequence as the experimental data sets.

Table 6 presents the details of each model set, including the total number of class diagrams in the model set, the number of matching paths in the model set, and the total number of class diagrams that can be matched in the model set. In order to avoid the statistics bias, the results of each algorithm are averaged over 30 runs.

Experimental environment. The experiment was performed on 64-bit Windows desktop computer with the following configuration: 3.19 GHz CPU and 16 GB RAM.

7.2. Results and discussion

RQ1.Compared with existing version control system, is the proposed N-way model merging approach more effective?

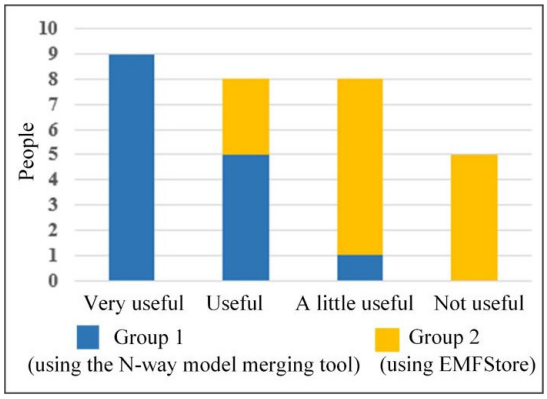
Table 7. Experimental data statistics of the two groups

		Group 1	Group 2
Task 1	Total Time Taken	12 h (6 days, 2 hours per day)	14 h (7 days, 2 hours per day)
	Total Number of Model Merging	13	131
	Total Number of Conflicts	92	96
	Total Time Taken for Resovling Conflicts	Total: 3 h 16 min Group Discussion: 1 h 50 min Modeling: 1 h 10 min Merging by Tool: 16 min	Total: 5 h 13 min Group Discussion: 2 h 10 min Modeling: 1 h 15 min Merging by Tool: 1 h 48 min
		Group 1	Group 2
Task 2	Total Time Taken	24 hours (12 days, 2 hours per day)	30 hours (15 days, 2 hours per day)
	Total Number of Model Merging	29	307
	Total Number of Conflicts	284	302
	Total Time Taken for Resovling Conflicts	Total: 9 h 22 min Group Discussion: 5 h 15 min Modeling: 3 h 10 min Merging by Tool: 57 min	Total: 15 h 51 min Group Discussion: 7 h 20 min Modeling: 4 h 18 min Merging by Tool: 4 h 13 min

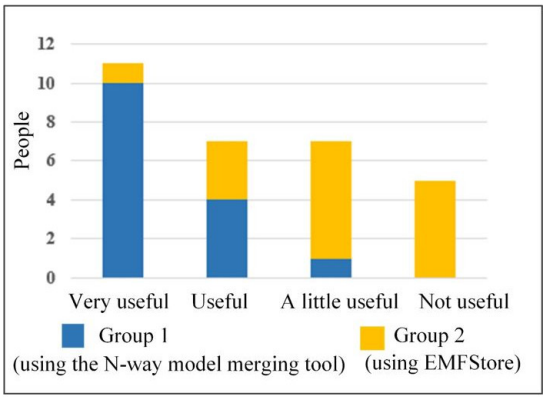
Two groups of participants completed their tasks and submitted the models and related experimental records. The models built by *Group 1* have 43 classes (Task 1) and 82 classes (Task 2), respectively, and the models built by *Group 2* have 44 classes (Task 1) and 83 classes (Task 2), respectively. As we expected, the complexity of models built by the two groups is relatively similar due to our design when grouping participants. The statistical results of the experiment are given in Table 7. The results show that *Group 1* (using the N-way model merging tool) spent less time on both tasks than *Group 2* (using EMFStore [4]). And the time consumption gap is greater in the second task. The reason is that *Group 2* spent more time on merging models using EMFStore when resolving conflicts, taking 1 hour and 48 minutes in Task 1 and 4 hours and 13 minutes in Task 2. This is much more than that of *Group 1* which spent only 16 minutes (Task 1) and 57 minutes (Task 2) using the proposed N-way model merging tool.

The essential reason why our tool is more effective than EMFStore [4] is that EMFStore cannot support N-way model merging. Each group member needs to submit the model one by one. When conflicts occur, each member needs to wait for others to deal with the

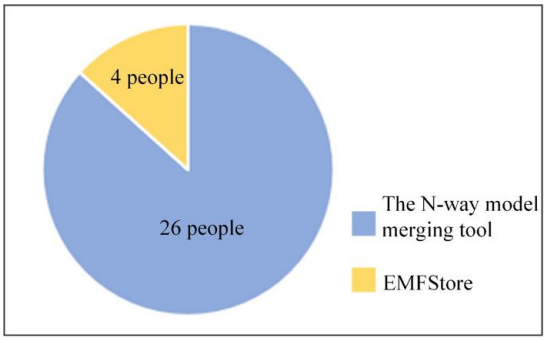
conflicts immediately, and only after resolving the conflicts can the next merge be carried out. Unlike EMFStore, our tool supports N-way model merging thus saving the extra waiting time of submitting the model one by one. The results show the applicability of our approach in merging real-world collaborative models.



(a) Comparison of the merging function between the N-way model merging tool and EMFStore



(b) Comparison of the conflict handling function between the N-way model merging tool and EMFStore



(c) Comparison of user favoritism between the N-way model merging tool and EMFStore

Figure 9. Statistics on the results of the answers to the questions in the questionnaire

RQ2. What are users' views on the usefulness of the proposed N-way model merging approach compared with the existing version control system?

According to the statistical results of questionnaires given in Figure 9, most participants think that the N-way model merging tool proposed in this paper is more useful than the existing version control system EMFStore [4] in model merging and conflict handling and are more willing to choose our tool as the model merging tool.

As shown in Figure 9(a), most of the participants (14 out of 15) in Group 1 (using the N-way model merging tool) stated that the N-way model merging tool was very useful or useful. And only one participant stated that the N-way model merging tool is a little useful. Compared with the N-way model merging tool, most participants in Group 2 (using EMFStore) found that the model merging function provided by EMFStore was not as useful as they had expected. And only a few participants (3 out of 15) stated that EMFStore's model merging function was useful. The results show that most participants agreed that the N-way model merging tool is useful and EMFStore is not useful enough. This is because a large number of models were required to be merged in the experiment. Unlike the N-way model merging tool which can merge a large number of models at one time, EMFStore requires more manual efforts to merge these models.

As shown in Figure 9(b), most of the participants (14 out of 15) using the N-way model merging tool stated that the conflict handling function provided by the N-way model merging tool was very useful or useful. In contrast, only a few participants (4 out of 15) felt that the conflict handling function of EMFStore was useful or very useful. This is because unlike EMFStore, the N-way model merging tool supports to show all the conflicts between the models to be merged at one time, which greatly improves the efficiency of conflict handling.

Figure 9(c) shows the comparison results of participants' favoritism for the N-way model merging tool and EMFStore. Most of the participants (26 of 30) prefer to use the N-way model merging tool in collaborative modeling. They stated that the N-way model merging tool can improve the efficiency of model merging when there are a large number of models to be merged. Other participants (4 out of 30) chose EMFStore because they were used to using EMFStore to manage their models.

RQ3. How is the performance of the proposed ABCMatch algorithm compared with existing novel meta-heuristic algorithms?

Table 8 shows the model matching accuracy and the time cost of ABCMatch, GA [15], and EHO [20] when the maximum model matching degree tends to be stable. The calculation method of model matching accuracy is shown in Equation (9). Where $N_{\text{correct\_path}}$ is the number of the correct matching paths in a matching solution, and $N_{\text{total\_path}}$ represents the total number of paths in a matching solution. A matching path is a correct matching path if and only if all the UML classes in this matching path are different versions of the same UML class.

$$Accuracy = \frac{N_{\text{correct\_path}}}{N_{\text{total\_path}}} \times 100\% \tag{9}$$

Table 8. Comparison of the model matching results between ABCMatch, GA [15], and EHO [20]

Data sets	ABCMatch		GA[15]		EHO[20]	
	Accuracy [%]	Time [min]	Accuracy [%]	Time [min]	Accuracy [%]	Time(min)
ModelSet1	98.0341	0.9687	95.6442	1.4283	96.5336	1.2557
ModelSet2	96.8774	0.9841	95.3084	1.4392	95.4013	1.2892
ModelSet3	97.8759	1.0259	94.2405	1.5870	96.7265	1.3066
ModelSet4	98.1253	1.0373	94.8720	1.5932	96.5449	1.3592
ModelSet5	96.9697	1.0429	93.3201	1.6079	95.1102	1.3623
ModelSet6	97.2340	1.8214	95.1853	3.1047	94.7228	2.9968
ModelSet7	98.0599	1.9028	94.9275	3.2824	96.0849	3.0294
ModelSet8	97.0538	2.0215	96.0284	3.2879	95.3972	3.0851
ModelSet9	98.1667	2.0250	95.0828	3.2901	95.9601	3.0883
ModelSet10	97.8613	2.0316	93.9238	3.2962	94.9730	3.0935

As shown in Table 8, for the five model sets ModelSet1–ModelSet5 from Task 1, the average value of the maximum model matching accuracy that GA and EHO can achieve when they tend to be stable are 94.6770% and 96.0633%, respectively. Compared with GA and EHO, the ABCMatch algorithm proposed in this paper can achieve a higher model matching accuracy of 97.5765% in a shorter time. For the five model sets ModelSet6-ModelSet10 from Task 2, the average value of the maximum accuracy that GA and EHO can achieve when they tend to be stable are 95.0296% and 95.4276%, respectively. Compared with GA and EHO, ABCMatch can achieve a higher model matching accuracy of 97.6751% in a shorter time. For all data sets in the experiment, ABCMatch has better

performance than GA and EHO. The average accuracy of model matching has increased by 2.7725% and 1.8804% compared with GA and EHO, respectively. And the average time cost has decreased by 0.9056 mins and 0.7005 mins compared with GA and EHO, respectively. The reason why ABCMatch performs better than EHO and GA is that ABCMatch can maintain a good balance between global search and local search through the cooperation of the employed bees, onlookers, and scouts. Unlike ABCMatch, GA has poor local search ability due to its randomness, so its performance is not as good as ABCMatch. Although EHO has a strong local search ability, its performance is not as good as the ABCMactch algorithm because of its poor global search ability. This problem makes it easy to fall into local optimum.

7.3. Threats to validity

In this section, we analyze the threats to the validity of the case study using the method [68] proposed by Wohlin et al.

1. ***Internal validity.*** The internal threat is that the modeling level of the participants could affect the results of the experiment. We set the modeling level that the participants need to achieve and tried to make the modeling levels of the two groups as equal as possible to relieve this threat.
2. ***External validity.*** The external validity means that the conclusion drawn from existing examples in the experimental data set is also valid for out-of-set examples [68]. Our experiments assume that the collaborative modeling is performed by a large group of people in the same period of time together. In this case, our approach is more effective than existing version control tools. We acknowledge that existing version control tools are more effective in the case that there is no need for a large group of users to resolve conflicts and merge models together during the same period of time.

The limited sample size poses a threat to the validity of our experimental results. Due to the limited time and manpower, in the current research work, only two group of participants were involved in the experiment. In the future, we will invite external practitioners and researchers to use the proposed method and continuously improve it.

To mitigate the threats to external validity brought by the particular tasks delivered in the experiment, we had selected two common and representative models of different scales in different fields. The selected models are similar to other common information systems in the real world.

3. ***Construct validity.*** The construct validity reflects the degree to which the case study truly represents what needs to be studied according to the research questions [68]. The two research questions of this paper are the effectiveness and users' views on the usefulness of the proposed approach compared with the existing version control tool. For the first question, we let two different groups (an experiment group and a control group) work on the same tasks with the two different approaches. For the second question, we obtained users' views on the usefulness of the proposed approach by questionnaire. So this case study totally represents what needs to be studied according to the two research questions.
4. ***Conclusion validity.*** As the conclusions of case studies are usually drawn from several cases, the results suffer from the threat to conclusion validity [68]. In this case study, we target to simulate the real-world collaborative modeling process to demonstrate the effectiveness of the proposed approach compared with the existing version control

system. However, this requires much manual modeling effort and time consumption. So it is difficult to provide a large number of examples.

8. Conclusion and future works

In this paper, we propose a novel N-way model merging approach and accompanying prototype tool for collaborative modeling. Our approach takes a set of model variants as inputs and generates a single global merged model by model comparison, model matching and model combination. For model comparison, this paper proposes a new calculation method for calculating the similarity of a group of model elements. For the most challenging step, matching, which is an NP-hard problem, we propose a novel N-way matching algorithm ABCMatch. Model combination is implemented by building a matching model which is then transformed into the merged model. Unlike other approaches, we reshuffle elements from distinct chains by extracting the prior element link and store it in matching models rather than breaking the chain into pieces. Theoretical analysis is given to prove that our approach satisfies the Generic Merge Requirements. A case study is conducted and the experiment results corroborate the effectiveness of the proposed approach. Compared with existing novel meta-heuristic algorithms GA and EHO which can be used to solve the N-way model matching problem, the proposed ABCMatch algorithm can obtain more accurate model matching solutions in a shorter time. The average model matching accuracy of ABCMatch is 2.7725% higher than GA and 1.8804% higher than EHO.

At present, the N-way model merging method proposed in this paper is only applicable to merging UML class diagrams and cannot be used to merge other types of models, such as UML sequence diagrams. This is because the model comparison method, the model matching method and the model combination method used in this paper are specially designed for UML class diagrams. In the future, we will extend the model merging method proposed in this paper to support the merging of other types of UML models.

References

[1] T. Stahl, M. Völter, J. Bettin, A. Haase, and S. Helsen, *Model-driven software development – Technology, engineering, management*. Hoboken, NJ, USA: John Wiley and Sons, Inc., 2006.

[2] M. Franzago, D.D. Ruscio, I. Malavolta, and H. Muccini, "Collaborative model-driven software engineering: A classification framework and a research map," *IEEE Transactions on Software Engineering*, 2017, pp. 1–1.

[3] J.E. Rumbaugh, I. Jacobson, and G. Booch, *The unified modeling language reference manual*. Reading, MA, USA: Addison Wesley Longman, Inc., 1999.

[4] M. Koegel and J. Helming, "EMFStore: a model repository for EMF models," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, Vol. 2, 2010, pp. 307–308.

[5] E. Stepper, "CDO model repository," 2010.

[6] J. Rubin and M. Chechik, "n-way model merging," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. Association for Computing Machinery, 2013, p. 301–311.

[7] F.S. Gharehchopogh, "Advances in tree seed algorithm: A comprehensive survey," *Archives of Computational Methods in Engineering*, Vol. 29, 2022, pp. 3281–3304.

[8] H. Mohammadzadeh and F.S. Gharehchopogh, "A multi-agent system based for solving high-dimensional optimization problems: A case study on email spam detection," *International Journal of Communication Systems*, Vol. 34, No. 3, 2021, p. e4670.

[9] F.S. Gharehchopogh, I. Maleki, and Z.A. Dizaji, "Chaotic vortex search algorithm: Metaheuristic algorithm for feature selection," *Evolutionary Intelligence*, Vol. 15, No. 3, 2022, pp. 1777–1808.
[10] T.S. Naseri and F.S. Gharehchopogh, "A feature selection based on the farmland fertility algorithm for improved intrusion detection systems," *Journal of Network and Systems Management*, Vol. 30, No. 3, 2022, p. 40.
[11] H. Mohammadzadeh and F.S. Gharehchopogh, "Feature selection with binary symbiotic organisms search algorithm for email spam detection," *International Journal of Information Technology and Decision Making*, Vol. 20, No. 01, 2021, pp. 469–515.
[12] F.S. Gharehchopogh, "Quantum-inspired metaheuristic algorithms: Comprehensive survey and classification," *Artificial Intelligence Review*, 2022, pp. 1–65.
[13] S. Ghafori and F.S. Gharehchopogh, "Advances in spotted hyena optimizer: A comprehensive survey," *Archives of Computational Methods in Engineering*, 2021, pp. 1–22.
[14] D. Karaboga, "Artificial bee colony algorithm," *Scholarpedia*, Vol. 5, No. 3, 2010, p. 6915.
[15] S. Katoch, S.S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," *Multimedia Tools and Applications*, Vol. 80, 2021, pp. 8091–8126.
[16] S. Mirjalili, S.M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Advances in Engineering Software*, Vol. 69, 2014, pp. 46–61.
[17] M.Y. Cheng and D. Prayogo, "Symbiotic organisms search: A new metaheuristic optimization algorithm," *Computers and Structures*, Vol. 139, 2014, pp. 98–112.
[18] S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Advances in Engineering Software*, Vol. 95, 2016, pp. 51–67.
[19] H. Shayanfar and F.S. Gharehchopogh, "Farmland fertility: A new metaheuristic algorithm for solving continuous optimization problems," *Applied Soft Computing*, Vol. 71, 2018, pp. 728–746.
[20] G.G. Wang, S. Deb, and L.d.S. Coelho, "Elephant herding optimization," in *3rd International Symposium on Computational and Business Intelligence (ISCBI)*. IEEE, 2015, pp. 1–5.
[21] J. Xue and B. Shen, "A novel swarm intelligence optimization approach: Sparrow search algorithm," *Systems Science and Control Engineering*, Vol. 8, No. 1, 2020, pp. 22–34.
[22] S. Kaur, L.K. Awasthi, A. Sangal, and G. Dhiman, "Tunicate swarm algorithm: A new bio-inspired based metaheuristic paradigm for global optimization," *Engineering Applications of Artificial Intelligence*, Vol. 90, 2020, p. 103541.
[23] F.A. Hashim, E.H. Houssein, K. Hussain, M.S. Mabrouk, and W. Al-Atabany, "Honey Badger Algorithm: New metaheuristic algorithm for solving optimization problems," *Mathematics and Computers in Simulation*, Vol. 192, 2022, pp. 84–110.
[24] M. Dehghani, Š. Hubálovskỳ, and P. Trojovskỳ, "Northern goshawk optimization: A new swarm-based algorithm for solving optimization problems," *IEEE Access*, Vol. 9, 2021, pp. 162 059–162 080.
[25] F.S. Gharehchopogh, M. Namazi, L. Ebrahimi, and B. Abdollahzadeh, "Advances in sparrow search algorithm: A comprehensive survey," *Archives of Computational Methods in Engineering*, Vol. 30, No. 1, 2023, pp. 427–455.
[26] Y. Huo, Y. Zhuang, J. Gu, S. Ni, and Y. Xue, "Discrete gbest-guided artificial bee colony algorithm for cloud service composition," *Applied Intelligence*, Vol. 42, 2015, pp. 661–678.
[27] W.K.G. Assunção, S.R. Vergilio, and R.E. Lopez-Herrejon, "Discovering software architectures with search-based merge of UML model variants," in *ICSR*, 2017.
[28] D. Reuling, M. Lochau, and U. Kelter, "From imprecise N-way model matching to precise N-way model merging." *The Journal of Object Technology*, Vol. 18, No. 2, 2019.
[29] M. Koegel, H. Naughton, J. Helming, and M. Herrmannsdoerfer, "Collaborative model merging," in *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*. Association for Computing Machinery, 2010, p. 27–34.
[30] H.K. Dam, A. Reder, and A. Egyed, "Inconsistency resolution in merging versions of architectural models," in *IEEE/IFIP Conference on Software Architecture*, 2014, pp. 153–162.

[31] P. Buneman, S. Davidson, and A. Kosky, "Theoretical aspects of schema merging," in *International Conference on Extending Database Technology: Advances in Database Technology*, 1992.
[32] R. Pottinger and P.A. Bernstein, "Merging models based on given correspondences," in *Proceedings VLDB Conference*, 2003, pp. 862–873.
[33] M. Sharbaf and B. Zamani, "Configurable three-way model merging," *Software: Practice and Experience*, Vol. 50, No. 8, 2020.
[34] C. Thao and E.V. Munson, "Using versioned trees, change detection and node identity for three-way XML merging," *Computer Science – Research and Development*, Vol. 34, No. 1, 2019, pp. 3–16.
[35] C. Debreceni, I. Rath, D. Varro, X. De Carlos, X. Mendialdua et al., "Automated model merge by design space exploration," in *Fundamental Approaches to Software Engineering*, 2016, pp. 104–121.
[36] F. Schwagerl, S. Uhrig, and B. Westfechtel, "Model-based tool support for consistent three-way merging of EMF models," *ACME*, 2013, p. 2.
[37] A. Schultheiß, P.M. Bittner, L. Grunske, T. Thüm, and T. Kehrer, "Scalable N-way model matching using multi-dimensional search trees," in *ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2021, pp. 1–12.
[38] M.S. Kasaei, M. Sharbaf, and B. Zamani, "Towards a formalism for specifying N-way model merging rules," in *27th International Computer Conference, Computer Society of Iran (CSICC)*, 2022, pp. 1–7.
[39] M. Boubakir and A. Chaoui, "A pairwise approach for model merging," in *Modelling and Implementation of Complex Systems*, S. Chikhi, A. Amine, A. Chaoui, M.K. Kholladi, and D.E. Saidouni, Eds. Cham: Springer International Publishing, 2016, pp. 327–340.
[40] Y. Jiang, S. Wang, K. Fu, W. Zhang, and H. Zhao, "A collaborative conceptual modeling tool based on stigmergy mechanism," in *Proceedings of the 8th Asia-Pacific Symposium on Internetware*, 2016, pp. 11–18.
[41] J. Martinez, T. Ziadi, T.F. Bissyande, J. Klein, and Y.L. Traon, "Automating the extraction of model-based software product lines from model variants," 2015, pp. 396–406.
[42] A. Lanusse, Y. Tanguy, H. Espinoza, C. Mraidha, S. Gerard et al., "Papyrus UML: An open source toolset for MDA," in *Proc. of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA 2009)*. Citeseer, 2009, pp. 1–4.
[43] F.S. Gharehchopogh, M.H. Nadimi-Shahraki, S. Barshandeh, B. Abdollahzadeh, and H. Zamani, "CQFFA: A Chaotic Quasi-Oppositional Farmland Fertility Algorithm for Solving Engineering Optimization Problems," *Journal of Bionic Engineering*, Vol. 20, No. 1, 2023, pp. 158–183.
[44] F.S. Gharehchopogh, "An improved tunicate swarm algorithm with best-random mutation strategy for global optimization problems," *Journal of Bionic Engineering*, Vol. 19, No. 4, 2022, pp. 1177–1202.
[45] B. Abdollahzadeh and F.S. Gharehchopogh, "A multi-objective optimization algorithm for feature selection problems," *Engineering with Computers*, Vol. 38, No. Suppl 3, 2022, pp. 1845–1863.
[46] Şaban Öztürk, R. Ahmad, and N. Akhtar, "Variants of Artificial Bee Colony algorithm and its applications in medical image processing," *Applied Soft Computing*, Vol. 97, 2020, p. 106799.
[47] U. Mansoor, M. Kessentini, P. Langer, M. Wimmer, S. Bechikh et al., "MOMM: Multi-objective model merging," *Journal of Systems and Software*, Vol. 103, 2015, pp. 423–439.
[48] A. Anwar, A. Benelallam, M. Nassar, and B. Coulette, "A graphical specification of model composition with triple graph grammars," in *Model-Based Methodologies for Pervasive and Embedded Software*, Vol. 7706, 2012, pp. 1–18.
[49] A. Koshima and V. Englebert, "RuCORD: Rule-based composite operation recovering and detection to support cooperative edition of (meta)models," in *2015 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, 2015, pp. 1–7.
[50] H. Chong, R. Zhang, and Z. Qin, "Composite-based conflict resolution in merging versions of UML models," in *17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, 2016, pp. 127–132.

[51] J. Rubin and M. Chechik, "Combining related products into product lines," in *Fundamental Approaches to Software Engineering*, 2012, pp. 285–300.
[52] P. Jaccard, "The distribution of the flora in the alpine zone. 1," *New Phytologist*, Vol. 11, No. 2, 1912, pp. 37–50.
[53] D. Karaboga and B. Gorkemli, "A combinatorial artificial bee colony algorithm for traveling salesman problem," in *International Symposium on Innovations in Intelligent Systems and Applications*, 2011, pp. 50–53.
[54] "*N* Way Model Merging Tool," https://github.com/YETONG1219/NWay, [accessed 7 Nov. 2023].
[55] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Computational Intelligence Magazine*, Vol. 1, No. 4, 2006, pp. 28–39.
[56] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 – International Conference on Neural Networks*, Vol. 4. IEEE, 1995, pp. 1942–1948.
[57] S. Karimkashi and A.A. Kishk, "Invasive weed optimization and its features in electromagnetics," *IEEE Transactions on Antennas and Propagation*, Vol. 58, No. 4, 2010, pp. 1269–1278.
[58] X.S. Yang and X. He, "Firefly algorithm: Recent advances and applications," *International Journal of Swarm Intelligence*, Vol. 1, No. 1, 2013, pp. 36–50.
[59] B. Xing, W.J. Gao, B. Xing, and W.J. Gao, "Fruit fly optimization algorithm," *Innovative Computational Intelligence: A Rough Guide to 134 Clever Algorithms*, 2014, pp. 167–170.
[60] X.S. Yang, M. Karamanoglu, and X. He, "Flower pollination algorithm: A novel approach for multiobjective optimization," *Engineering Optimization*, Vol. 46, No. 9, 2014, pp. 1222–1237.
[61] S. Mirjalili, "Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm," *Knowledge-based Systems*, Vol. 89, 2015, pp. 228–249.
[62] A. Askarzadeh, "A novel metaheuristic method for solving constrained engineering optimization problems: Crow search algorithm," *Computers and Structures*, Vol. 169, 2016, pp. 1–12.
[63] S. Mirjalili, "Dragonfly algorithm: A new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems," *Neural Computing and Applications*, Vol. 27, 2016, pp. 1053–1073.
[64] S.Z. Mirjalili, S. Mirjalili, S. Saremi, H. Faris, and I. Aljarah, "Grasshopper optimization algorithm for multi-objective optimization problems," *Applied Intelligence*, Vol. 48, 2018, pp. 805–820.
[65] G. Dhiman and V. Kumar, "Multi-objective spotted hyena optimizer: A multi-objective optimization algorithm for engineering problems," *Knowledge-Based Systems*, Vol. 150, 2018, pp. 175–197.
[66] G. Dhiman and V. Kumar, "Emperor penguin optimizer: A bio-inspired algorithm for engineering problems," *Knowledge-Based Systems*, Vol. 159, 2018, pp. 20–50.
[67] S. Arora and S. Singh, "Butterfly optimization algorithm: A novel approach for global optimization," *Soft Computing*, Vol. 23, 2019, pp. 715–734.
[68] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell et al., *Experimentation in software engineering: An introduction*. Kluwer Academic Publishers, 2000.