Automatic Resolution of Model Merging Conflicts Using Quality-Based Reinforcement Learning

Mohammadreza Sharbaf, Bahman Zamani, Gerson Sunyé

Automatic Resolution of Model Merging Conflicts Using Quality-Based Reinforcement Learning

Mohammadreza Sharbaf[a,b], Bahman Zamani[a,*] and Gerson Sunyé[b]

[a]MDSE Research Group, University of Isfahan, Isfahan, Iran
[b]LS2N, University of Nantes, Nantes, France

ARTICLE INFO

Keywords:
Collaborative Modeling
Model Merging Conflict
Conflict Resolution
Reinforcement Learning
Quality Evaluation

ABSTRACT

Modeling is an activity in the software development life cycle in which different experts and stakeholders collaborate as a team. In collaborative modeling, adhering to the optimistic versioning paradigm allows users to apply concurrent changes to the same model. In such a situation, conflicts may arise. To have an integrated yet consistent merged model, conflicts have to be resolved. To this end, automation is currently at its limit or is not supported at all, and user interaction is often required. To alleviate this flaw, there is an opportunity to apply Artificial Intelligence techniques in a collaborative modeling environment to empower the provisioning of automated and intelligent decision-making. In this paper, we propose the use of reinforcement learning algorithms to achieve merging conflict resolution with a high degree of automation. This enables the personalized and quality-based integration of model versions. To evaluate our idea, we demonstrate the resolution of UML class diagram conflicts using a learning process in an illustrative modeling scenario. We also show the applicability of our approach through a proof of concept implementation and assess its accuracy compared to the greedy and search-based algorithms. Moreover, we conducted an experience with five experts to evaluate the satisfaction of actual users with the selection of resolution actions for different conflicts. The result of the assessment validates our proposal with various syntactic and semantic conflicts.

1. Introduction

Model-Driven Engineering (MDE) is a software engineering discipline, which promotes the usage of models as first-class artifacts of the software lifecycle [1, 2]. When the software is complex, the size and complexity of models increase dramatically. This enforces the participation of many developers and stakeholders, who collaborate in large teams to evolve models in an optimistic versioning process [3]. In this context, different participants may work concurrently and independently on the same model from different geographical sites. Each participant focuses on specific aspects of the system and locally modifies only a particular part of the model. When participants deliver the locally modified models, these models need to be integrated into a common and relevant model to continue the software evolution. However, some concurrent updates may be incompatible and engender conflicts during the merge process [4]. Some conflict situations may lead to errors in the merged model, which should be repaired with different merging approaches to produce a valid merged model. However, automatic repairing is not possible for all conflict situations, since some conflictual change operations cannot be resolved easily. Therefore, techniques and tools for automatic conflict management are not optional but necessary to cope with the growing number of conflicts and to ensure the consistency of the merged-models [3].

Conflict management in model merging deals with techniques, activities, and tools for enhancing consistency in the result of the merge process. So far, several approaches have been proposed to support conflict management in model merging. Most of these approaches provide conflict detection techniques such as constraint violation (e. g., [5]) and change overlapping (e. g., [6]) to discover conflictual situations that may occur due to the concurrent changes. Some approaches try to prevent conflicts using a pessimist lock-based versioning strategy (e. g., [7]). Some others avoid conflicts using mechanisms that provide synchronous model updates and inform the user about conflicting situations for concurrent modifications (e. g., [8]). Few approaches propose conflict specification techniques such as formal specification pattern (e. g., [9]) or conflict model (e. g., [10]) to describe conflict situation in a high-level representation. Other approaches focus on conflict resolution techniques, such as operational transformation (e. g., [11]), and try to

*Corresponding author
m.sharbaf@eng.ui.ac.ir (M. Sharbaf); zamani@eng.ui.ac.ir (B. Zamani); gerson.sunye@univ-nantes.fr (G. Sunyé)
ORCID(s): 0000-0001-5113-7689 (M. Sharbaf); 0000-0001-6424-1442 (B. Zamani); 0000-0001-6407-8075 (G. Sunyé)

automate the reconciliation phase as much as possible. Despite these efforts, a common automatic conflict resolution technique that supports conflict resolution for different types of conflict has not yet been provided. On the one hand, most conflict management approaches prefer a manual reconciliation mechanism that completely delegates conflict resolution to the users. On the other hand, existing resolution approaches are limited to particular types of conflict or specific modeling language and involve users in the resolution phase.

We advocate solutions that provide general and automatic conflict resolution techniques that can be adapted and customized for all types of conflicts in any modeling language. In particular, we aim to achieve human performance in resolving conflicts by applying Artificial Intelligence (AI) techniques such as Machine Learning (ML) for automatic conflict resolution in the model merging process. To this end, most of the ML algorithms require a large number of training data to achieve proper result [12]. However, the lack of adequate datasets in the modeling field [13] is a big challenge to use ML in this area. Reinforcement Learning (RL) is a branch of ML which could offer promising results for the situation which lacks historical data because it does not require initial training data. RL techniques work without supplying any target or outcome beforehand to find structures in the domain [14].

In this paper, we propose to use RL as a solution for automatic conflict resolution in the model merging process. We perform RL to resolve possible *syntactic* and *semantic* conflicts that may arise due to *overlapping changes* and *violations of validation rules*, which are two main reasons for conflicts [6]. To achieve this goal, we present a resolution algorithm, which per se can learn how to deal with conflicts to create a consistent merged model based on the available actions and their rewards. We explain that actions for each conflict can be defined according to the predefined resolution patterns or three automatic resolution strategies. Moreover, the reward of applying each action is dynamically specified based on the improvement of quality metrics in models. In our approach, we use the E3MP toolkit [15] to detect a list of possible conflicts and evaluate the success of available actions. We demonstrate the applicability of our approach with a set of conflicts for an illustrative example and provide a proof-of-concept implementation for UML class models in the Eclipse framework, which is called CoReRL. The results show that CoReRL can resolve different syntactic and semantic conflicts detectable by machine. We also assess the accuracy of our resolution algorithm compared to the execution of a greedy algorithm, as well as a search-based algorithm that follows the depth-first-search (DFS) style for resolving conflicts in two running examples. Furthermore, we conducted an experience with modeling experts to investigate the satisfaction of modelers with the selected resolving actions. The initial evaluation shows that our solution can improve the accuracy of conflict resolution in the model merging process without human intervention, and modelers were satisfied with the selected resolution actions.

The rest of this paper is organized as follows. Section 2 briefly provides the theoretical background on reinforcement learning and model merging conflicts. Section 3 introduces a motivating example using an order management system. Section 4 presents the overview of our approach, which focuses on the resolution algorithm, strategies to define resolution actions, and approach demonstration. The evaluation of the proposed approach is depicted in Section 5 and the threats to validity are discussed in Section 6. Section 7 reports related work, and Section 8 concludes the paper and outlines future work.

2. Background

This section introduces basic theoretical notions of Reinforcement Learning (RL) and merging conflict to provide a comprehensive guide for the rest of this paper.

2.1. Reinforcement Learning

RL is a method that does not need a training phase or labeled data. The learning process is based on a sequence of actions [13, 14]. RL deals with *agents* that directly learn from the interaction with the *environment*. RL algorithms assess applying the sequence of actions to the environment to learn from good sequences to generate new sequences. In such cases, a single action is not important, since it is the whole sequence of actions that leads to the goal. In particular, the agent performs actions in the environment that changes its state. Each action gets a reward or a punishment depending on the resulted state. The agent continues performing actions, seeking the highest reward until it reaches its ultimate goal. In the learning process, performance is improved as the agent gains experience [12].

Q-learning is a form of RL that learns which actions are the best overall by evaluating a long-term discounted reward [14]. It evaluates the consequences of the action based on the immediate reward and its estimation of the value of the next state results of the action. In Q-learning, the long-term discounted rewards are stored in a simple table-based structure called Q-table. From a given state, the agent finds the most optimal action by consulting the Q-table. For each

action, the table is initialized with zero at the beginning and updated while the agent interacts with the environment based on the Bellman equation [16]:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha(r + \gamma \, max_a \, Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)) \quad (1)$$

The Bellman equation returns a weight, named *Q-value*. For each action a_t, Q-value presents the maximum future reward (r) the agent receives for entering the current state (s_t), plus the maximum future reward for the next state (s_{t+1}) and next action (α_{t+1}). To avoid falling into a local maximum, the computed score is reduced by a discount factor γ, which determines the importance of future reward against the immediate reward. The Q-value allows inferring the value of the current state s_t based on the calculation of the next one s_{t+1}, which can be used to calculate an optimal policy to select actions. The factor α specifies the learning rate, which determines how much Q-values change from one iteration to the next one. At this point, the algorithm can determine the value of each state and find the optimal action to be applied to the next state until the ultimate goal is achieved. This process is known as *exploitation*, and this calculation is repeated every time t that the algorithm selects an action. Also, to find new and perhaps more optimal solutions, the algorithm can pick actions at random, instead of choosing the optimal action from the Q-table. This process is called *exploration* as it explores alternative, possibly more optimal, solutions [14].

Each iteration is called an *episode*. Inside the episode, the agent combines exploitation and exploration to find a sequence of actions to reach the ultimate goal. The agent may perform several attempts to find an action for solving the current state. Each attempt is called *step*. Note that the updated version of the Q-table will be used to resolve the problem in a new episode. Moreover, the number of episodes and the number of steps per episode are decided concerning the problem size.

2.2. Model Merging Conflicts

The joint creation of models is considered as collaborative modeling, which depends on a set of collaborative means such as model versioning systems and model merging mechanisms for allowing stakeholders to coordinate themselves as a team [3]. Conflicts may arise in the model merging process when concurrent incompatible modifications directly or indirectly affect the same model element. For instance, in updating a UML class diagram, two modelers may rename the same class with different names. As another example, modelers may add a generalization relation in each version of a UML class diagram, which would create an inheritance cycle in the merged model. Conflicts may also happen due to a set of different modifications with the same intention. For instance, when updating a UML state machine to define substates, one modeler adds a composite state containing several substates, while another modeler represents substates as normal states. Dangling reference is another example of a conflict in UML class diagrams that occurs when one modeler adds an association to a class, whereas another modeler removes that class [4]. In such situations, either the modifications cannot be integrated to produce a unique model, or the integration would result in an inconsistent merged version of the model.

According to the origin of conflicts mentioned above, *overlapping changes* and *violations* are two main reasons for conflicts [6]. Conflicts due to overlapping changes result from the existence of contradicting changes (e. g., renaming a class with two different names) or modifications that express the same meaning in varying ways (e. g., state composition in UML state machines). Overlapping changes can easily give rise to false situations and should not be performed concurrently in the merged model. In comparison, conflicts due to violations result from applying changes that impact the conformance of the model to its well-formedness rules (e. g., dangling reference) or violate the validation rules of modeling languages (e. g., inheritance cycle). Apart from the given conflict reasons, different types of model merging conflicts can be categorized into syntactic and semantic conflicts, which respectively are defined by considering the syntax and the semantics of models [17]. Although there exist various categorizations of conflicts, in this paper, we follow the categorization proposed by Altmanninger and Pierantonio [18].

Syntactic conflicts are those which take the modeling language syntax into account, e. g., the UML syntax does not allow dangling reference. Syntactic conflicts do not necessarily produce language syntax violations. For instance, contradicting modifications for an element (e. g., renaming a class with two different names) or updating the same model element by adding different properties constitutes a syntactic conflict [18]. This type of conflict may be detected by checking the structure of models and comparing the similarity of elements.

Semantic conflicts go beyond syntactic ones, as they need to consider the system behavior and the modeling language semantics. Indeed, the integration of concurrent changes may result in a syntactically correct merged version, yet semantically invalid [4]. The parallel control flow in UML state machine is a semantic conflict that can be revealed by considering the semantics of the used modeling concepts. Moreover, the inheritance cycle is a

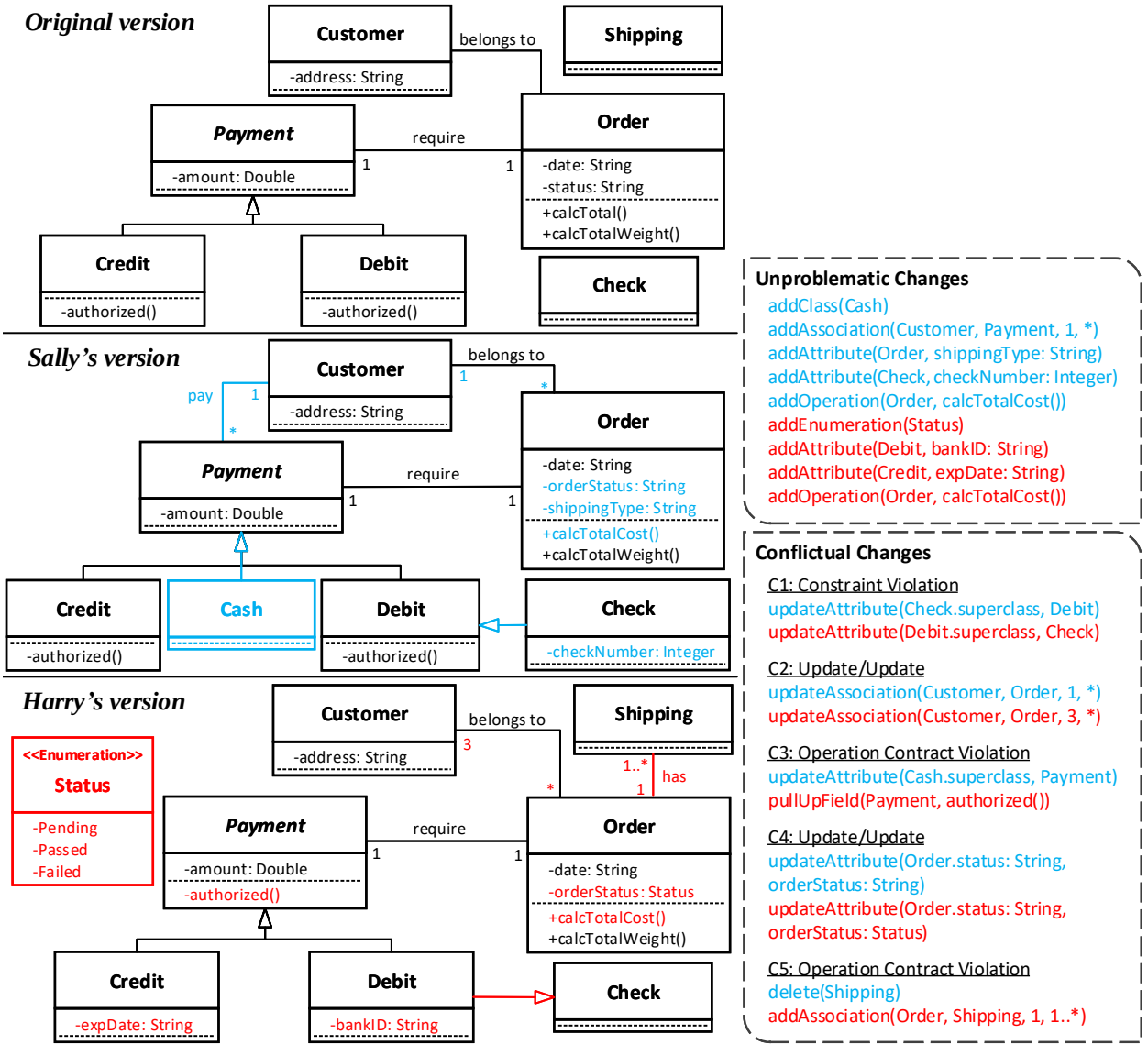

Figure 1: Motivating example: evolution of an order management system

semantic conflict in which the language constraint "there must not be any inheritance cycle in UML class diagrams" is violated [18]. Semantic conflicts are divided into three categories: Static semantics, behavioral semantics, and semantic equivalence [4]. We omit their definition for the sake of conciseness.

3. Motivating Example

The focus of this paper is on conflict resolution. In this context, the detection of conflicts is a central prerequisite, and the origin of conflicts acts as input for the resolution approach presented in this paper. Therefore, we briefly introduce various origins of conflict with the help of the example shown in Fig. 1, which presents a collaborative modeling scenario for the structural model of an order management system that aims to provide one centralized place to manage orders from all sales channels. To achieve this, two modelers, Sally and Harry, concurrently work on their local copies of the original version. Each one applies several changes to create her/his version. Fig. 1 shows their versions.

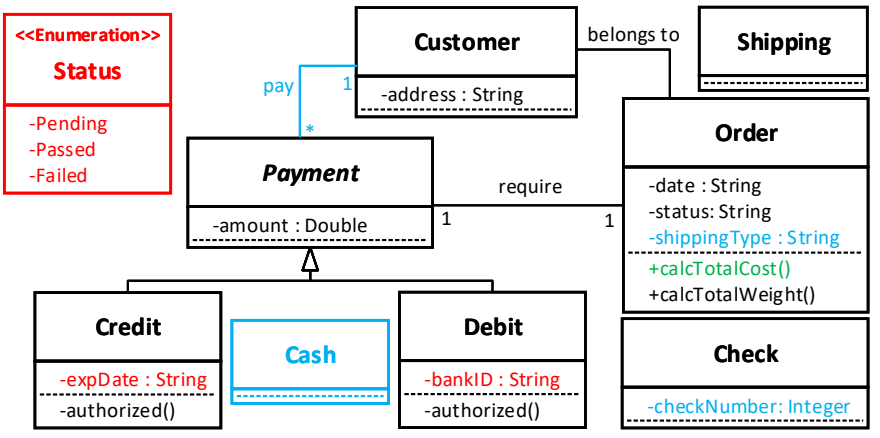

Figure 2: Initial merged version by applying unproblematic changes to original version

In the merge process, modelers' changes can be categorized into *unproblematic* and *conflictual* changes. All unproblematic changes can be easily integrated into the original version to create an initial merged version, as shown in Fig. 2. However, applying conflictual changes leads to conflicts and an inconsistent merged model that requires conflict resolution.

A conflict is a general or a domain-specific situation with different origins. General conflicts origin can be categorized based on the equivalent changes or contradicting changes, such as Add/Add, Update/Update, and Delete/Update, which are applied to the same model element [4]. While domain-specific conflicts mainly occur based on the conflictual changes over different model elements leading to the operation contract violations (i. e., Delete/Use, Update/Use, and Add/Forbid) or the parallel changes that violate constraints related to syntax or semantics of a modeling language [6]. Note that the occurrence of some conflicts depends on the order in which the change operations are applied. Therefore, to detect conflicts in our example, we assume that Sally's changes will be applied first in such situations. In our example, conflictual changes lead to five conflict situations that are summarized as follows:

C1 : A constraint violation conflict results from inheritance cycle between the classes *Debit* and *Check*

C2 : An association Update/Update conflict results from different multiplicities for the association *belongs_to*

C3 : A Update/Use conflict due to operation contract violation results from incorrect inheritance of method *authorized()* for the class *Cash*

C4 : An attribute Update/Update conflict results from different modifications to update the attribute *status* in the class *Order*

C5 : A Delete/Use conflict due to operation contract violation results from a dangling reference for the association *has*

According to the suggestion of Gerth et al. [19], these conflicts can be resolved in multiple ways to restore the consistency of the merged version. Each way might simply ignore one or more changes to resolve the conflict situation or reconcile that by modifying conflictual changes to a new combination of operations. Furthermore, conflicts are related to different model elements might be resolved by changing the order of applying conflictual change operations in the merged model. Thereby, a possible set of actions can be undertaken to resolve each conflict. In the following, we introduce the available actions for each conflict of our example. To prepare the actions, we suppose that Sally's changes were applied before those from Harry.

We start from semantic conflict C1, a constraint violation due to an inheritance cycle. Where the combination of modification leads *Check* to inherit *Debit* (indicated by *Sally*), and *Debit* inherits *Check* (indicated by *Harry*). As mentioned in Section 2.2, C1 is a semantic conflict [18] since it can be detected only by considering the validation rules of the UML class diagram. Available actions for C1 are as follows:

A1 : Reverse the order of changes application by applying Harry's change at first.

A2 : Only apply Sally's change in which she adds the inheritance from *Debit* to *Check*.

A3 : Only apply Harry's change in which he adds the inheritance from *Check* to *Debit*.

A4 : Apply none of them.

In this specific case, the inheritance cycle can be resolved with A2 where we only add the inheritance from *Debit* to *Check* or A4 where we do not add either of the changes. However, with A1 the inheritance cycle conflict will remain and with A3, a semantical inconsistency arises, the incorrect inheritance of attribute *checkNumber*.

The next conflict is C2, a syntactic conflict where modelers add two different multiplicity values for the end property of class *Customer* in the association *belongs_to*. C2 occurs due to the concurrent application of contradicting changes to edit the same property with different values that would report a syntactic conflict for the association [18]. The available actions for C2 are as follows:

A1 : Reverse the order of changes application by applying Harry's change at first .

A2 : Only apply Sally's change that sets the association ends with * and 1.

A3 : Only apply Harry's change that sets the association ends with * and 3.

A4 : Apply none of them.

A5 : Apply the resolution pattern where sets the association ends with * and 1..3.

For conflict C2, actions A2 to A5 can result in resolving this conflictual situation. However, A1, where we reverse the order of applying operations, does not resolve conflicts that arise when two applied operations are for the same element.

In the semantic conflict C3, the new class *Cash* would inherit an unrelated method (i. e., *authorized()*) from its superclass after applying changes. C3 might not be reported by detecting conflicts on the syntactical level, since that occurs due to the application of one operation (pull-up *authorized()*), which violates the preconditions of another operation (add subclass *Cash*). A precondition of pull-up field operation is that each subclass must contain an operation same as the operation that is moved to the common superclass. Therefore, C3 is a semantic conflict situation in that language knowledge is needed to be detected [18]. The available actions for C3 are as follows:

A1 : Reverse the order of changes application by applying Harry's change first.

A2 : Only apply Sally's change in which she adds the inheritance from *Payment* to *Cash*.

A3 : Only apply Harry's change in which he moves the method *authorized()* from *Credit* and *Debit* to *Payment*.

A4 : Apply none of them.

A5 : Apply the resolution pattern which adds a new class *Credit_Debit*, which inherits from *Payment*, then change the superclass of classes *Debit* and *Credit* from *Payment* to *Credit_Debit*, and finally, pull up method *authorized()* from classes *Debit* and *Credit* to class *Credit_Debit*.

Conflict C3 can be resolved with A2, where we only apply Sally's change, A3, where we only apply Harry's change, A4, where we skip both modelers' changes, or A5, where we try to represent intents of both modelers on the merged model. But action A1 cannot resolve conflict C3 because the reversed order similarly leads to the same conflict.

In conflict C4, there is a syntactic conflict again due to the contradicting changes, where modelers update the attribute *status* by adding different data types (i. e., *String* and *Status*). In the following, we present five available actions for conflict C4. While actions A2, A3, or A4 can result in resolving concurrent *status* updates, action A1 is not applicable for this situation.

A1 : Reverse the order of changes application by applying Harry's change at first.

A2 : Only apply Sally's change in which she updates the attribute *status* with *orderStatus:String*.

A3 : Only apply Harry's change in which he updates the attribute *status* with *orderStatus:Status*.

A4 : Apply none of them.

The last conflict is the syntactic conflict C5, where Harry adds a new association that uses a class which is concurrently deleted by Sally. More precisely, based on the syntax of the UML modeling languages, we can find that applying these modifications would arise a syntactic conflict that violates the well-formedness of the merged model [4]. In this specific case, the four following actions can resolve this conflict. With A1, we first add the association *has* then we delete the class *Shipping* which also results in removing its relationships without arising any conflicts. Moreover, with A2 or A3 only one change will be applied in the merged model, and with A4, we skip both changes.

A1 : Reverse the order of changes application by applying Harry's change first.

A2 : Only apply Sally's change, which deletes the class *Shipping*.

A3 : Only apply Harry's change, which adds the association *has*.

A4 : Apply none of them.

We could propose other alternative actions for each conflict, but to avoid increasing the complexity of our example, we only consider four or five available actions per conflict. We can automatically generate some actions (e. g., actions A1 to A4), but others (e. g., action A5) should be added manually. In the next section, we explain different scenarios to create actions following the strategies proposed by Gerth et al. [19].

4. Proposed Conflict Resolution Approach

In this section, we present our approach that applies quality-based RL to automatically resolve conflicts in model merging. This approach uses an RL algorithm based on Q-learning to find and apply the best possible resolution action for detected conflicts in the merging process. The algorithm per se can learn which sequence of resolution actions can create a consistent merged model with the maximum total reward. Our approach works based on the quality preferences which are selected before beginning the resolution algorithm and does not require any supervision when resolving conflicts. When the resolution algorithm finishes, it lists consistent merged models with their rewards. We choose the merged model with the highest reward as output by default. However, modelers can select another compatible merged model according to their intentions. Fig. 3 shows the workflow of our approach, which comprehends of three categories of functions. The first category (marked with a ⊖) consists of the optional functions that language engineers should perform *before starting* the resolution process, including *Conflict Specifications* and definition of *Resolution Patterns*. The second category (marked with a ◉) consists of the mandatory functions that must be performed just once, *at the start* of the resolution process, including *Preference Selection* by modeling engineer, *Change Calculation* according to the input models, *Conflict Detection* regarding the computed changes, and automatic preparation of actions based on the *Resolution Strategies*. The third category (marked with a ↻) consists of the repetitive functions that must be performed *during* the resolution process and, more precisely, in each episode of the resolution algorithm. The functions in this category include the execution of the *Resolution Algorithm* based on the extracted *Learning Experiences*, and applying *Model Preparation*, *Conflict Resolution Checking*, and *Quality Evaluation* for selected resolution actions.

In the *Modeling Module* of this workflow, we use the Conflict Pattern Language (CPL) [9] to support *Conflict Specification*. We also compare input models using the Epsilon Comparison Language (ECL) [20] to identify change operations in the *Change Calculation* function. Moreover, we apply resolution action using the Eclipse Modeling Framework (EMF) [21] to create the provisional resolved model as well as output merged model in the *Model Preparation* function. In the *Consistency Checking Module*, we use the E3MP toolkit [15] to detect a list of possible conflicts in the *Conflict Detection* function and evaluate that a specific conflict is resolved by applying a resolution action in *Conflict Resolution Checking*. In the *Action Generation Module*, we use the Epsilon Validation Language (EVL) [22] to define resolving situations in the *Resolution Patterns* function. Moreover, we developed three general strategies in the *Resolution Strategies* function to automatically generate actions for the detected conflicts. In the *Learning Module*, we customize the Q-learning algorithm [14] in the *Resolution Algorithm* function to learn and resolve model merging conflicts. We also implemented the corresponding Q-table using Java to store learning results in the *Learning Experience* function. Finally, in the *Quality Evaluation Module*, we prepared a graphical user interface

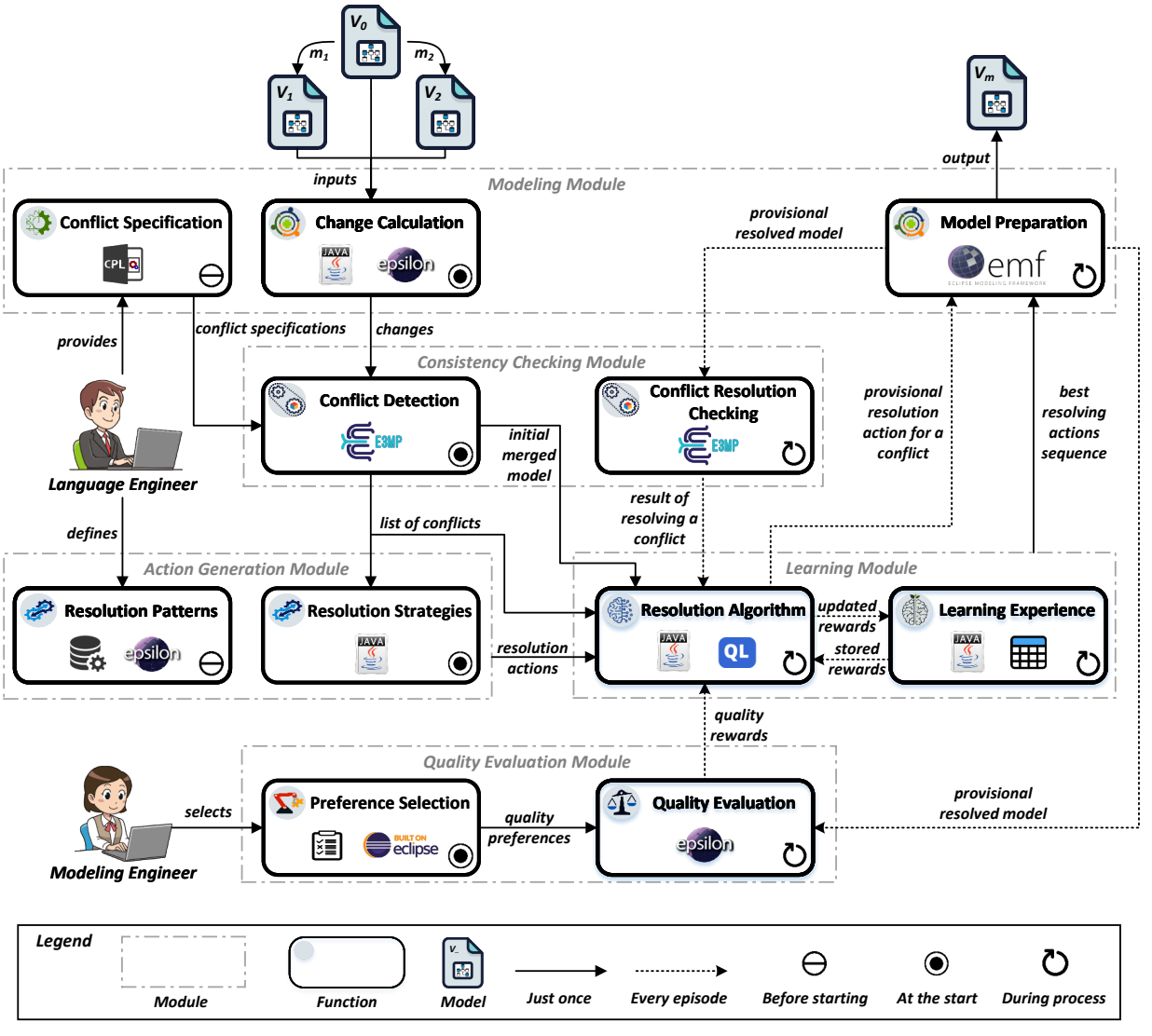

Figure 3: Workflow of the resolution process using RL

built on the Eclipse framework for *Preference Selection* and used Epsilon Operation Language (EOL) [23] to implement the specification of quality preferences for measuring the reward of applied actions in the *Quality Evaluation* function.

In the following, we focus on the *Resolution Algorithm*, which is the main contribution of this paper. To this end, we first explain the learning algorithm for conflict resolution. Then, we introduce our quality-based rewarding system that is used in the learning process. After that, we represent the available scenarios to create the possible resolution actions for each conflict. Finally, we apply our approach to resolve the conflicts described in the motivating example. We emphasize that our approach applies to any EMF-based modeling language and conflicting situations that are supported by the E3MP toolkit. However, we only implemented the *Consistency Checking* and *Quality Evaluation* modules for UML class models.

4.1. Algorithm

RL-based algorithms learn how to interact in an environment, given a situation, a set of available actions, and a reward for each action [12]. In our approach, we use a slightly different RL technique from the standard to resolve conflicts. The resolution algorithm starts when we find a list of conflicts using the E3MP toolkit in the *Conflict Detection*

function. Thereby, we know the conflicts that arise from the model merging process. In our RL algorithm, we consider conflicts as states of the environment. Therefore, a consistent merged model is produced when the algorithm reaches the final state and resolves conflicts completely. To this end, our RL algorithm requires resolving one conflict before moving to the next one. By targeting one conflict at a time, we reduce the state space of the algorithm. In our conflict resolution process, the objective is not only to resolve the conflicts in the model merging process. We also aim to find the best possible sequence of resolution actions that maximizes the total reward, which is computed based on the selected quality preferences by the modeling engineer. As a result, the resolution algorithm can lead to a compatible merged model with high quality that supports most of the changes applied by modelers. To achieve this goal, we propose a Q-learning-based resolution algorithm, presented in Algorithm 1.

Algorithm 1 Resolution Algorithm

Input: $ConflictList$, $Actions$, V_{merged}
Output: *Q-table*, V_{merged}
procedure RESOLVING METHOD
  Initialize: *Q-table*, *NOEpisodes*, *NOSteps*, $V_M Collection$
  while *NOEpisodes* <> ∅ **do**
    Initialize: *ConflictList*, V_{merged}, $Reward_{V\,merged}$
    for each *conflict* ∈ *ConflictList* **do**
      while *NOSteps* <> ∅ **do**
        action ← *selectAction(conflict, Actions)*
        result ← *checkResolutionAction(V_{merged}, conflict, action)*
        reward ← *qualityEvaluation(V_{merged}, conflict, action)*
        Update $Q(conflict, action)$ in *Q-table* by *reward*
        NOSteps ← *NOSteps* - 1
        if *isConflictResolved(result)* **then**
          Apply action to V_{merged}
          Update $Reward_{V\,merged}$
          Break while
        end if
      end while
    end for
    if *isConsistent*(V_{merged}) and $V_{merged} \notin V_M Collection$ **then**
      Add (V_m, $Reward_{Vm}$) to $V_M Set$
    end if
    NOEpisodes ← *NOEpisodes* - 1
  end while
  V_{merged} ← *Select* the best from $V_M Collection$ based on the maximum $Reward_{V\,merged}$ or with the *User's opinion*
end procedure

The resolution algorithm receives as input, the initial merged model, a list of detected conflicts and possible actions for each conflict. We explain how the functions of the *Action Generation Module* define resolution actions for each conflict in Section 4.3. The resolution algorithm updates the initial merged version and creates a consistent merged model as well as the learned Q-table as output. This algorithm starts with initializing the Q-table for conflicts. All possible actions for each conflict are used to create a Q-table. This is a two-dimensional structure to store a weight for a combination of conflict and action in the *Learning Experience* function. The weight indicates how good an action is for a given conflict. Therefore, the Q-table leads to finding the best action that can be applied to resolve each conflict. The weights are initialized with zeros and updated based on the Q-value computed for each action according to the obtained reward. If the action can resolve the conflict, the Q-value is calculated with a positive reward, and contrarily it is penalized.

The *Quality Evaluation* function calculates the reward of each action, based on the quality characteristics that were selected in the *Preference Selection* function. The resolution algorithm executes several episodes to learn how to select the best sequence of resolution actions for conflicts in a specific modeling language. The adequate number of episodes

is reached when the algorithm has sufficient time to search the most desirable sequence of resolution actions to arrive at the final state. It is difficult to infer how many episodes are needed in a given context and thus, it must be defined empirically through experimentation [14]. According to our empirical experiments, we initialize the maximum number of episodes with ten because repeating additional episodes is useless. However, when the Qtable has been learned for conflict resolution in a particular modeling language, the maximum number of episodes can be reduced.

Each episode is an iteration in which the algorithm that resolves all detected conflicts within a certain number of steps. There is no setup policy to specify what number of steps are adequate for a given problem [14]; however, the maximum number of steps can be based on the number of conflicts and the size of the models. Thus, in line with our empirical experimentation, the maximum number of steps inside the episodes is set to two times the number of conflicts. Each step tries to resolve one conflict by applying one of the existing actions. The action is either a random action to avoid a local maximum or the most optimal action from the Q-table chosen by weight. The *Model Preparation* function performs the selected resolution action on the merged version to create a provisional model. Then, it evaluates the consistency of the provisional merged model using E3MP in the *Conflict Resolution Checking* function to check if that action could resolve the relevant conflict. Applying an action to a conflict returns a reward by the *Quality Evaluation* function, which leads to learning the environment and updating the Q-table for that action based on the Bellman equation. When an action resolves a conflict, the algorithm updates the merged version and moves to find a resolution action for the next conflict.

This process is repeated until the episode terminates by resolving all conflicts or reaching the maximum number of steps. The resulting merged version is added to the collection of possible solutions if all conflicts have been resolved and lead to a valid and unique model. The solution reward starts as zero and is accumulated only over actions that have been applied on the merged model and lead to resolving the conflict. By repeating episodes, the algorithm reuses the learned Q-table to improve action selection over time, and the chances for finding the optimal sequence of actions with maximum rewards increases. The algorithm returns the Q-table that has learned from resolving conflicts and the merged version with the best reward as the final result. Moreover, we support an optional interaction at the end of the resolution process, in which the user is able to choose a solution from the collection of merged versions. This feature allows the user to select the best consistent merged model compatible with the intention of the modeler and other criteria that may not be possible to capture in quality metrics.

4.2. Rewarding system

The definition of the reward is the most crucial element of reinforcement learning. Using rewards, the RL algorithms can learn which are the best actions to interact with an environment [14]. We propose the calculation of rewards based on the quality of the provisional merged models. Thereby, the resolution algorithm must search to find resolution actions that boost the quality characteristics in the merged models. Multiple quality characteristics may be considered in order to evaluate qualitative aspects of models. In this work, we use characteristics like maintainability, understandability, complexity, reusability, and completeness. These characteristics are adapted from the quality model introduced by Bettini et al. [24] and merging properties proposed by Chechik et al. [25]. The first four characteristics are a series of metrics that the modeling community has developed to measure the quality of models, and completeness is the metric that personalizes the degree of data is lost along the merging process. The *Preference Selection* function offers these quality characteristics to the modeling engineer as preferences to guide conflict resolution in the merged model. Model engineers can choose their preferences before the resolution algorithm starts and can prioritize them by quantifying the impact of each quality characteristic in the computation of rewards. For measuring the quality metrics in the provisional merged model, we used EOL that allows defining each quality characteristic for different modeling languages as an operational expression. The *Quality Evaluation* function calculates the rewards aligned with user preferences and expressions, defining how the value of quality has to be measured. In the following, we investigate the quality characteristics considered in our rewarding system and explain their definitions for UML class models based on some of the metrics shown in Table 1.

The *maintainability* quality characteristic used in our work has been defined according to the definition introduced by Genero and Piattini [26]. This definition focus on the structural complexity of class diagrams to predict maintainability early in the software development life-cycle. In the considered definition, the lower values indicate that the models have better maintainability. In particular, maintainability can be defined as follows:

$$Maintainability = \left(\frac{NC + NA + NR + DIT_{Max} + HAGG_{Max}}{5}\right) \tag{2}$$

Table 1
Excerpt of the metrics used in the definition of quality characteristics for UML class models

Metric	Acronym
Number of Class	NC
Number of Total Reference	NR
Number of Opposite Reference	NOPR
Number of Total Containment Reference	NCR
Number of Total Attribute	NA
Number of Unidirectional Reference	NUR
Max generalization hierarchical level	DIT_{Max}
Max aggregation hierarchy path	$HAGG_{Max}$
Number of Total Features	NTF
Sum of inherited structural features	INHF
Attribute inheritance factor	AIF
Number of predecessor in hierarchy	PRED
Number of Total change Operations	NO
Number of Applied change Operations	NAO

We take the definition of *understandability* quality characteristic from Sheldon et al. [27], where they measure the level of understandability of a given class diagram by measuring the total number of ancestor classes that affect a class. Thus, the understandability can be defined as Equation 3, in which class models with better understandability level have the lower values.

$$Understandability = (\frac{\sum_{k=1}^{NC} PRED + 1}{NC}) \tag{3}$$

The definition of *complexity* quality characteristic is adopted from Sheldon and Chung [28]. They introduce the complexity of class diagrams in terms of the number of static relationships between classes, such as the number of references. Therefore, the complexity of the model should be measured considering the relationships, such as association and generalization. The association relationships are counted as the number of direct connections. Also, the generalization relationship is counted as the number of all the ancestor and descendant classes. Equation 4 indicates the definition of the complexity characteristic, in which UND is the understandability value calculated as defined in Equation 3. According to such a definition, class models with better complexity quality have lower values.

$$Complexity = (NR - NUR + NOPR + UND + (NR - NCR)) \tag{4}$$

The *reusability* of a class diagram is computed based on various metrics in different ways. Inheritance helps reuse of already-designed classes when designing a new class. The reusability quality characteristic considered in this work was defined based on the attribute inheritance factor (AIF) as proposed by Genero et al. [29]. AIF is defined as a quotient between the sum of inherited attributes in all classes of the system under consideration and the total number of available attributes for all classes. Thus, the reusability quality characteristic can be defined as Equation 5, where a model with a higher level of reuse has a higher value.

$$Reusability = (\frac{INHF}{NTF}) \tag{5}$$

The *completeness* characteristic that is used in our work has been defined according to the definition of merging properties introduced by Chechik et al. [25]. According to the given definition, each model element from the input

versions of the model should be represented in the target merged model. Thus, the completeness characteristic can be defined as Equation 6, where it is stated that a model with a higher degree of completeness has a higher value.

$$Completeness = (\frac{NAO}{NO}) \tag{6}$$

The reward of applying one resolution action is introduced as the quality improvement for the resulting merged model. To this end, we define Equation 7 to calculate the reward according to the selected preferences and their priorities expressed by the modeling engineer. Each chosen preference determines which characteristic should be investigated, and priorities (i. e., P_i) define the impact of preferences on the reward. To investigate four quality characteristics, we need to calculate the delta for identifying the quality improvement. The delta for the reusability metric in which a higher number is more valuable is computed based on $\Delta Reusability = R_2 - R_1$. But to calculate the delta for the other criteria whose lower number is more valuable (i. e., maintainability, understandability, and complexity), we use multiplicative inverse for each variable. For instance, $\Delta Maintainability = \frac{1}{M_2} - \frac{1}{M_1}$. For each model with at least five elements, the parameters of the reward function would be compatible since the completeness and all delta parameters are less or equal to one, and the reward is calculated as the more is better. Thus, we measure the reward considering the boost of characteristics chosen and their impact collectively.

$$\begin{aligned} Reward = {} & (\Delta Maintainability * P_1) + (\Delta Understandability * P_2) + (\Delta Complexity * P_3) + \\ & (\Delta Reusability * P_4) + (Completeness * P_5) \end{aligned} \tag{7}$$

Note that to define the quality characteristics for a new modeling language, we first need to update and redefine quality metrics presented in Table 1 according to the structure of the target modeling language. Then, we should follow the definitions of quality characteristics (i. e., maintainability [26], understandability [27], complexity [28], and reusability [29]) to update their equations based on the new quality metrics.

4.3. Preparing possible actions

More than one action is used to resolve conflicts that arise in the merge process between conflictual changes. We provided two different scenarios in the *Resolution Patterns* and *Resolution Strategies* functions of the *Action Generation Module* to prepare appropriate actions for each conflict. The reward for each resolution action depends on the quality improvement in the resulting model, as well as the number of change operations applied to the merged version.

4.3.1. Scenario 1: Create actions using resolution patterns

A conflict might be resolved and engender a consistent model by applying a particular, user-definable, resolution pattern [15] specified for that conflict. To this end, users must first specify conflict situations and their preconditions. Then they can define the appropriate resolution patterns according to the specified elements. We follow the structure of the CPL template [9] to specify a conflict situation that enables us to detect conflict for any instance of different modeling language concepts. In the first scenario, we have a pattern repository that can contains possible resolution patterns for the well-known conflicts in a modeling language. Each resolution pattern is a fixing option for a conflictual situation that is optionally defined by a language engineer using the EVL script, before starting the resolution process. At the beginning of the resolution process, we search the pattern repository for the detected conflicts and if we find any resolution pattern for them, we create relevant actions according to the extracted patterns.

4.3.2. Scenario 2: Create actions using resolution strategies

In the second scenario, we explain three strategies based on the suggestion of Gerth et al. [19] to provide available actions for each conflict. We emphasize that the proposed strategies do not ensure conflict resolution and therefore, we must check the success of each action after the application. The mentioned automatic resolution strategies for any conflict are defined as follows:

- Strategy 1: Given two conflicting change operations that are applied sequentially and modify different model elements that are completely independent, the conflict might be resolved by applying operations in the reverse order. This strategy might avoid conflict because changing the order of modifications may lead to the preconditions not being violated.

Table 2
Step by step execution of Episode 1 of Resolution Algorithm for Motivating Example

Motivating Example	Step1	Step2	Step3	Step4	Step5	Step6	Step7	Step8	Step9
Episode 1	C1 ⇐ A1	C1 ⇐ A3Ⓡ	C1 ⇐ A2	C2 ⇐ A3Ⓡ	C3 ⇐ A1	C3 ⇐ A2	C4 ⇐ A1	C4 ⇐ A4Ⓡ	C5 ⇐ A1

- Strategy 2: Given two conflicting change operations for any model element, the conflict might be resolved and created a consistent model by applying only one of the change operations and discarding the other.
- Strategy 3: Given two conflicting change operations for any model element, the conflict is resolved and most likely create a consistent model by applying none of the conflicting operations in the merged model.

The first strategy can only be used for specific syntactic conflicts, while other strategies, as well as actions that are generated using resolution patterns, can be applied to provide various actions for syntactic and semantic conflicts. Note that the reward for an action that is created by the third strategy is zero, since it does not apply any change and improvement in the model. But this strategy is important for situations that other strategies are unable to provide a final valid and consistent merged model.

4.4. Demonstration

In this section, we demonstrate how the resolution algorithm resolves all conflicts from the motivating example. Our algorithm starts without any knowledge about the environment. According to our experiments, we found that for examples of the UML class model, better results are obtained with 0.9 for the learning rate (α). Regarding other parameters, we consider the random probability starts with 80 %. When the algorithm learns from the actions, it should be less and less influenced by the discoveries. Hence, we decrease the random exploration by a factor of 0.9 in each episode. Moreover, based on the results of our empirical experimentation, we use 0.8 for the discount factor (γ) since future rewards are as important as the immediate one. Note that the algorithm has a random element and might not produce the same result for each execution.

After applying the above settings, the learning algorithm starts the first episode, considering a maximum of 10 steps. Table 2 shows the execution of Episode 1, step by step . Since it is the first time the algorithm processes each conflict, it tries the available actions one by one or randomly. First attempt is to resolve C1 using A1 without success. Next, the algorithm randomly chooses A3, which does not resolve the current conflict. A2 is the next action that resolves C1. The next conflict is C2 and the algorithm randomly chooses A3 with success. For C3, it finds the solution by applying A2. The first action for the next conflict is picked randomly, and it resolves C4. In step 9, Episode 1 terminates after resolving C5 with A1. After each step, the algorithm learns from actions and updates their weight. The resulting Q-table is shown in Fig. 4. The first episode received a total reward of 50.86. For next episodes, the algorithm might take a random action or the optimal action based on the Q-table.

Q-table

C#	A#	Weight	C#	A#	Weight	C#	A#	Weight	C#	A#	Weight	C#	A#	Weight
C1	A1	-20.19	C2	A1	0	C3	A1	-23.37	C4	A1	-19.38	C5	A1	17.48
	A2	10.20		A2	0		A2	9.10		A2	0		A2	0
	A3	-8.62		A3	8.99		A3	0		A3	0		A3	0
	A4	0		A4	0		A4	0		A4	0		A4	0
				A5	0		A5	0						

Figure 4: Resulted Q-table after the execution of Episode 1 of resolution algorithm for motivating example

Fig. 5 shows the learned Q-table, after terminating the execution of the algorithm for the motivating example. In the best episode for this demonstration, the algorithm starts based on the Q-table and chooses the optimal actions A2 for C1, A5 for C2, A5 for C3, and A1 for C5. It also randomly picks A3 for C4. This sequence of actions results in the merged version shown in Fig. 6 and receives the best total reward of 85.41. All the modifications of Sally and 78 %

of Harry's modifications are represented in this merged model. According to the unproblematic changes, adding the inheritance form *Check* to *Debit* in the only action can provide a consistent model. Moreover, to resolve conflict C2 and C3, the pattern-based actions are chosen that support both modeler's changes. The conflict C4 has been resolved by updating the attribute *status* with *orderStatus:Status* that obtain a higher reward by improving the *understandability* and *reusability* of the merged model. Furthermore, the class *Shipping* and the linked association (i. e., *has*) are deleted according to Sally's change to resolve conflict C5. This resolution action is reasonable and improves the *maintainability* of the merged model, particularly where Sally added the attribute *shippingType* for the *Order*.

Q-table

C#	A#	Weight	C#	A#	Weight	C#	A#	Weight	C#	A#	Weight	C#	A#	Weight
C1	A1	-20.19	C2	A1	-11.44	C3	A1	-23.37	C4	A1	-19.38	C5	A1	19.45
	A2	44.60		A2	0		A2	10.01		A2	21.59		A2	8.52
	A3	-8.62		A3	17.84		A3	0		A3	26.95		A3	7.60
	A4	0		A4	0		A4	0		A4	0		A4	0
				A5	51.82		A5	44.01						

Figure 5: Learned Q-table after terminating the execution of resolution algorithm for motivating example

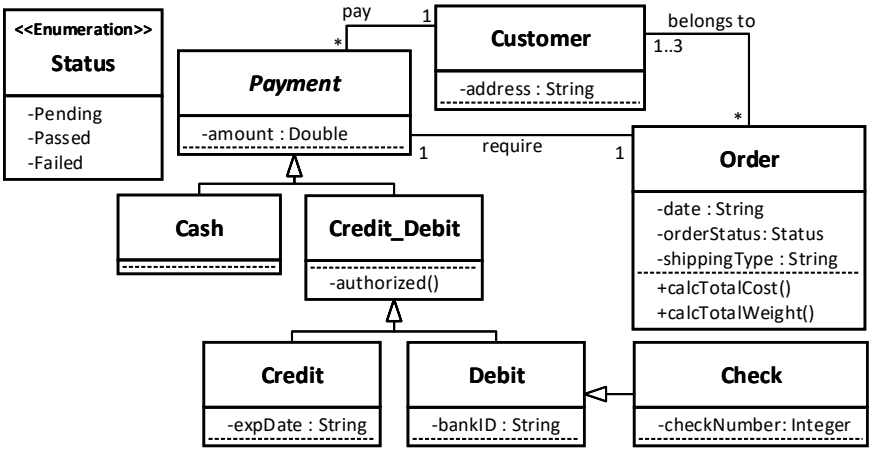

Figure 6: Merged version with the best total reward for motivating example

5. Implementation and Evaluation

Once the resolution algorithm learns the reconciliation of conflicts for a particular modeling language, we must evaluate its flexibility for resolving conflicts in other examples, assess its accuracy compared to similar algorithms, and investigate the satisfaction level of actual modelers for the proposed solution. To this end, we first present a proof of concept implementation of our approach. Then, we performed a comparative study on two existing resolution algorithms to resolve conflicts for two model versioning cases that contain UML class diagrams. Finally, we conducted an online workshop with modeling experts and gathered their experience after working with our approach. In the following, we describe our experiments and discuss the corresponding results.

5.1. CoReRL Eclipse Plugin

We assess the applicability of the proposed approach through an initial implementation in the Eclipse framework, which is available as the CoReRL Eclipse plugin (see Fig. 7). CoReRL uses the E3MP toolkit [15] for consistency checking. In particular, we first execute the E3MP conflict detection module to provide the list of conflicts. Then, we

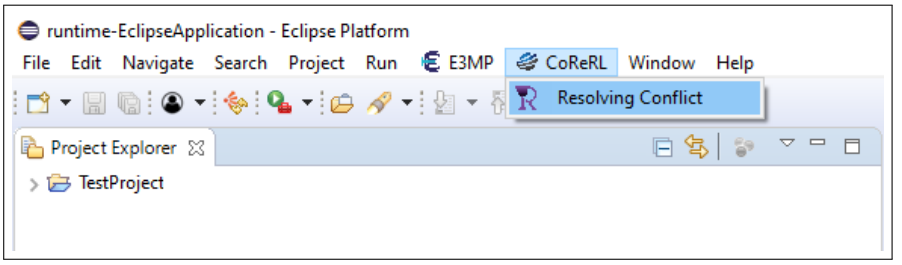

Figure 7: Screenshot of CoReRL Eclipse plugin

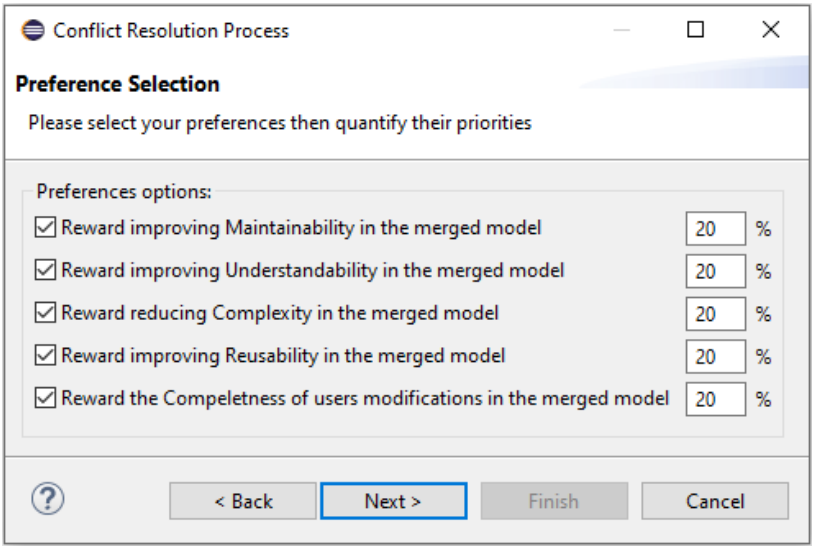

Figure 8: Screenshot of Preference Selection Window

generate the available actions and initialize Q-table automatically. After that, the resolution algorithm is simulated to update Q-table and creates different merged versions by applying resolution actions. We wrote some evaluation scripts based on the Epsilon framework to measure the quality metrics and calculate the reward of resolution actions for UML class models according to the preferences chosen by modeling engineers. Fig. 8 shows the preference selection window in CoReRL. We also use the E3MP consistency module to check the result of resolving a conflict in the provisional merged model as well as the validity and compatibility of the resulting merged version with consistency rules. The implementation is available on GitHub[1] under the Apache 2.0 license. While our approach is generic, the implementation is based on the Epsilon framework and on the E3MP toolkit, which are limited to UML and EMF-based models. However, we only address the evaluation scripts for measuring the quality metrics in the UML class models. We performed manually spot-checking to ensure the correctness of the resolution algorithm and validity of merged models.

5.2. Comparative Study

In this section, we present a study that assesses the performance of our algorithm in comparison to similar conflict resolution algorithms. To this end, we select the search-based algorithm presented by Dam et al. [30] and a greedy algorithm shown in Algorithm 2. The search-based algorithm follows the depth-first search style to find a solution path among multiple valid ways of resolving conflicts. The greedy algorithm also searches for a sequence of available resolution actions with maximum rewards, which can resolve conflicts. In our experiments, we choose two model versioning examples that contain modeling scenarios for UML class diagrams as test cases. The first test case includes five conflicts, and the second case contains three conflicts. We used search-based and greedy algorithms to resolve the

[1] https://github.com/MSharbaf/CoReRL

arisen conflicts. We also ran our resolution algorithm for only one episode to resolve conflicts in the first case using the Q-table that was built in the motivation example (see Fig. 5). Then, we performed our resolution algorithm with the updated Q-table for the second case. In the following, we report the obtained results for each example and compare the accuracy of algorithms.

Algorithm 2 Greedy Algorithm

Input: *ConflictList*, *Actions*, V_{merged}
Output: V_{merged}, *reward*
procedure GREEDY METHOD
  Initialize: *totalReward*, *action*
  for each *conflict* ∈ *ConflictList* **do**
    do
      Select *action* with maximum reward of *Actions* for the *conflict*
      Remove *action* from available *Actions* for the *conflict*
    while *isResolved(conflict)* <> *true*
    Apply *action* to V_{merged}
    Update *totalReward* based on the action *reward*
  end for
end procedure

5.2.1. First Experiment

This experiment is presented by Brocsh et al. [31], where an event management system is concurrently developed by two modelers, Sally and Harry. Fig. 9 shows the original version as well as modifications made by Sally (in blue) and Harry (in red). Some of these modifications are conflictual, leading to five conflicts in the merge process. The first one is an attribute Update/Update conflict and the second one is a constraint violation conflict due to the inheritance cycle. The third one is an association Update/Update conflict and the fourth one is an Update/Use conflict due to operation contract violation. These four conflicts are respectively the same as C4, C1, C2, and C3 conflicts in the motivating example. We also have an Add/Add conflict, called C6, which does not exist in the motivating example, but that can be resolved using actions A1 to A4 based on the resolution strategies.

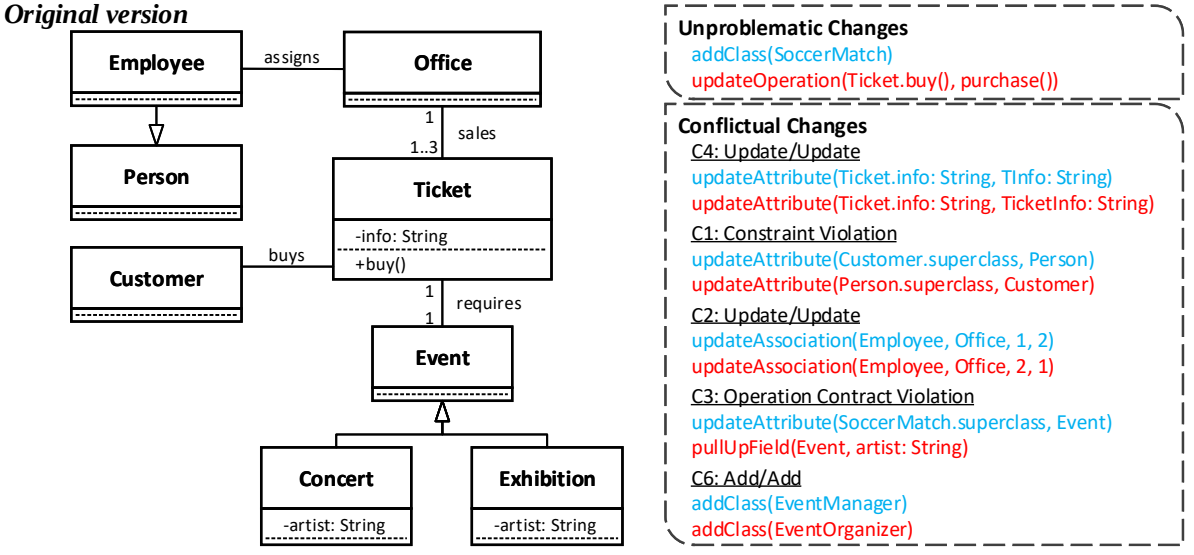

Figure 9: Experiment 1: original version of event management system and changes applied by two modelers (adapted from [31])

Table 3
Step by step execution of algorithms for resolving conflicts in Experiment 1

Experiment 1	Step1	Step2	Step3	Step4	Step5	Step6	Step7	Step8
Resolution Algorithm	C4 ⇐ A3	C1 ⇐ A2	C2 ⇐ A1Ⓡ	C2 ⇐ A5	C3 ⇐ A5	C6 ⇐ A1	C6 ⇐ A2Ⓡ	
Greedy Algorithm	C4 ⇐ A1	C4 ⇐ A2	C1 ⇐ A1	C1 ⇐ A2	C2 ⇐ A5	C3 ⇐ A5	C6 ⇐ A1	C6 ⇐ A3
Search-based Algorithm [30]	C4 ⇐ A3 C4 ⇐ A2 C4 ⇐ A1	C1 ⇐ A3 C1 ⇐ A2 C1 ⇐ A1	C2 ⇐ A3 C2 ⇐ A2 C2 ⇐ A1	C3 ⇐ A3 C3 ⇐ A2 C3 ⇐ A1 C3⇐A0,A3	C6 ⇐ A3 C6 ⇐ A2 C6 ⇐ A1			

Q-table

C#	A#	Weight	C#	A#	Weight	C#	A#	Weight	C#	A#	Weight	C#	A#	Weight	C#	A#	Weight
C1	A1	-20.19	C2	A1	-12.71	C3	A1	-23.37	C4	A1	-19.38	C5	A1	19.45	C6	A1	-18.61
	A2	49.41		A2	0		A2	10.01		A2	21.59		A2	8.52		A2	9.32
	A3	-8.62		A3	17.84		A3	0		A3	43.81		A3	7.60		A3	0
	A4	0		A4	0		A4	0		A4	0		A4	0		A4	0
				A5	54.70		A5	31.93									

Figure 10: Learned Q-table after the execution of resolution algorithm for Experiment 1

Table 3 illustrates the resolution actions that are applied by *Resolution*, *Greedy*, and *Search-based* algorithms for mentioned conflicts. Our resolution algorithm uses the Q-table shown in Fig. 5. This Q-table has been learned from resolving conflicts in the motivating example to find the optimal resolution action for C4, C1, C2, and C3. It also fails to resolve conflict C6 using A1 but randomly chooses A2 with success in the next attempt. Fig. 10 shows the updated Q-table after applying actions for the first experiment. The greedy algorithm tries to resolve each conflict by applying available resolution action with maximum reward, which does not fail in the last steps. But the search-based algorithm attempts to resolve each conflict in one step. It first applies A1, A2, A3, and other permissible combinations of these three actions for each conflict. Then the algorithm uses a depth-first search to find the best way between states, such as actions that are underlined in Step 1 to Step 5 of Table 3.

5.2.2. Second Experiment

This experiment is introduced by Schwägerl et al. [32] to display the evolution of a project management system by two modelers. Fig. 11 shows the original version and modifications that are applied by modeler1 (in blue) and modeler2 (in red). When modelers try to integrate modifications, three conflicts arise. The first one is a Delete/Use conflict due to operation contract violation that results from adding an inheritance relationship to the deleted class, NamedEntity, which is similar to conflict C5 in the motivating example. The second one is an attribute Update/Update, the same as conflict C4 in the motivating example. The last is an Add/Add conflict, which is very similar to conflict C6 in the first experiment.

Table 4 depicts the actions that are applied in each execution step of the mentioned algorithms in our study. For this example, our resolution algorithm uses the learned Q-table shown in Fig. 10, which leads to choosing optimal resolution actions A1, A3, and A2, for conflicts C5, C4, and C6, equivalently. The greedy algorithm chooses A1, which has a maximum reward, in its first attempt for each conflict. But action A1 cannot resolve conflicts C4 and C6, where the algorithm chooses A3 and A2 with success. Moreover, the search-based algorithm starts with applying actions A1, A2, and A3 to resolve conflict C5. Then, it performs actions A1 to A3 for resolving conflict C4 in the tentative merged model resulted from applying A1 for C5. Finally, the algorithm follows action A3 for conflict C4 and finds the action A2 as the best choice for conflict C6.

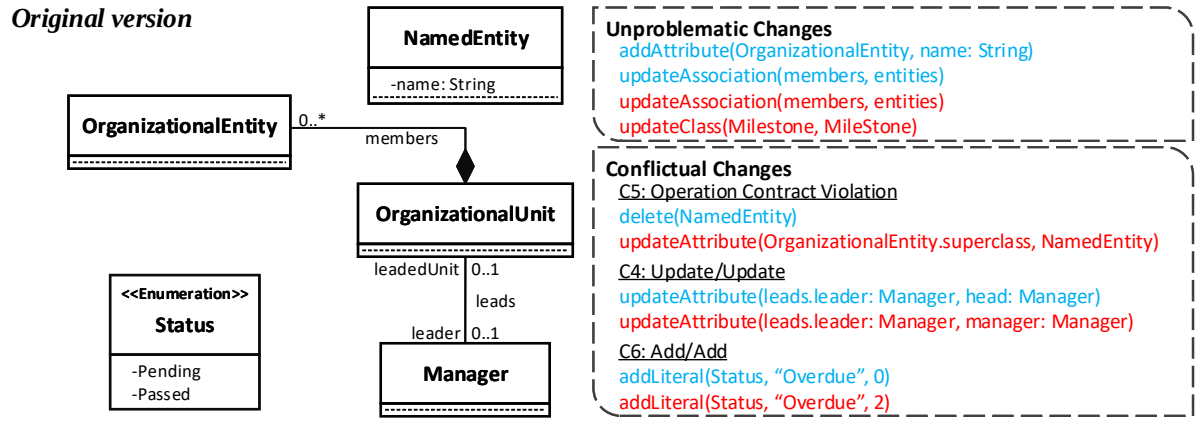

Figure 11: Experiment 2: original version of project management system and changes applied by two users (adapted from [32])

Table 4
Step by step execution of algorithms for resolving conflicts in Experiment 2

Experiment 2	Step1	Step2	Step3	Step4	Step5
Resolution Algorithm	C5 ⇐ A1	C4 ⇐ A3	C6 ⇐ A2		
Greedy Algorithm	C5 ⇐ A1	C4 ⇐ A1	C4 ⇐ A3	C6 ⇐ A1	C6 ⇐ A2
Search-based Algorithm [30]	C5 ⇐ A3	C4 ⇐ A3	C6 ⇐ A3		
	C5 ⇐ A2	C4 ⇐ A2	C6 ⇐ A2		
	C5 ⇐ A1	C4 ⇐ A1	C6 ⇐ A1		

Table 5
Results of Experimental Evaluation for Greedy and Resolution Algorithms

	Experiment 1				*Experiment 2*			
	#Step	#Attempt	Accuracy	Reward	#Step	#Attempt	Accuracy	Reward
Resolution Algorithm	7	7	71.4%	79.25	3	3	100.0%	40.55
Greedy Algorithm	8	8	62.5%	79.25	5	5	60.0%	40.55
Search-based Algorithm	5	16	68.7%	47.27	3	9	77.7%	40.55

5.2.3. Evaluation Results

This experiment shows that our resolution algorithm is able to automatically resolve conflicts in the integration of new models when the algorithm learned the Q-table and obtained enough knowledge. Regarding the actions that are applied in Table 2, the accuracy of our resolution algorithm in the first episode of resolving conflicts for the motivating example was about 55.5 %. But it quickly improves to 71.4 % and 100 %, only after processing a couple of examples.

The results of our study on executing three algorithms for the same versioning examples are depicted in Table 5. Using our resolution algorithm, we could resolve all conflicts by applying the minimum number of resolution actions. The results also show that our resolution algorithm has the best accuracy and leads to maximum rewards.

5.3. User Survey Experiment

The proposed conflict resolution approach introduced in Section 4 aims at enabling automatic and quality-based resolution of merging conflicts for users who are in charge of merging models but are unfamiliar with resolving conflictual situations for a specific modeling language. Therefore, we have conducted a preliminary evaluation with modeling experts at different universities to assess the user satisfaction and effectiveness of our approach. One of the

Table 6
Workshop Questions Regarding Usability

Question	P1	P2	P3	P4	P5
Q1. Which action is the best solution to resolve the conflict C1?	A4	A2	A2	A2	A2
Q2. Which action is the best solution to resolve the conflict C2?	A5	A2	A5	A5	A5
Q3. Which action is the best solution to resolve the conflict C3?	A5	A5	A5	A2	A5
Q4. Which action is the best solution to resolve the conflict C4?	A3	A3	A3	A3	A3
Q5. Which action is the best solution to resolve the conflict C5?	A3	A3	A3	A3	A3
Q6. How satisfied are you in general with the final result?	(5)	(4)	(4)	(4)	(5)

most problematic hurdles for conducting a survey is finding a sufficient number of suitable subjects who agree to participate. Luckily, we were able to gain five participants, including one third-year master's student, three second-year Ph.D. students, and one assistant professor. All subjects are experts in modeling and had enough knowledge in collaborative MDE, and had prior experience in model merging. In the following, we first introduce the objective of our survey. Then, we elaborate on the design of this experiment. Subsequently, we present the results and discuss our findings concerning the objectives.

5.3.1. Objectives

The experiment consists of guiding participants with a cooperative scenario during the resolution of conflicts for a model versioning example. Our main objective is to evaluate the participant satisfaction with the resolution actions that are chosen by our resolution algorithm. We also aim to assess the effectiveness of our algorithm in comparison to the manual resolution of conflicts. In particular, our objectives are described as follows:

- ***User satisfaction:*** are the resolution actions chosen by the proposed approach satisfactory?
- ***Effectiveness:*** is the proposed approach effective in creating a consistent merged model?

5.3.2. Experimental design

We conducted an empirical user study consisting of two phases with five participants. In the first phase, we held an online workshop in which we spent about 30 min presenting our approach using a model merging example. Then, participants were asked to merge different versions of the motivating example (see Fig. 1). Each person started with the original version and applied unproblematic changes to create an initial merged model. They continued with working on conflictual changes and iteratively resolved conflicts to produce a consistent merged model. After the participants completed the task, we displayed five merged models created with our algorithm, which led to the maximum rewards (without showing their rewards) and asked them to choose the best option. Furthermore, we asked how much time the participants spent preparing the merged model and resolving conflicts. In the second phase, we created a questionnaire to ask participants which available resolution action is the best solution for each conflict in the motivating example. If a participant did not agree with available resolution actions, she/he had to select "none". We also asked a question concerning the level of satisfaction with the output of the resolution algorithm. Where participants should have indicated that the result of our algorithm is either (1) Very dissatisfied, (2) Dissatisfied, (3) Neutral, (4) Satisfied, or (5) Very satisfied. The precise questions are presented in Table 6 when discussing the results in Section 5.3.3.

5.3.3. Results and discussion

We grouped the results of our user survey based on the objectives. The results of the questionnaire have questions with responses to the level of user satisfaction are summarized in Table 6. The obtained results with actual modelers that are corresponding to the effectiveness of our approach are shown in Table 7. In the following, we analyze the results and draw some conclusions for the objectives of our experiment: *user satisfaction* and *effectiveness*.

User satisfaction. The first objective aims to assess the level of user satisfaction and determine if the actions chosen by the resolution algorithm are acceptable for actual modelers. To this end, we investigate the resolution actions that each participant preferred to use for resolving conflicts in the motivating example. Based on the answers of the

Table 7
Results of Executing Merge Process by the Participants

Metric	P1	P2	P3	P4	P5
Merging Time (minutes)	16	15	10	10	9
Similar Elements in Merged Model (% compared to our result)	83%	86%	86%	83%	93%
Selected Merged Model Among Options	Option3	Option3	Option3	Option3	Option3

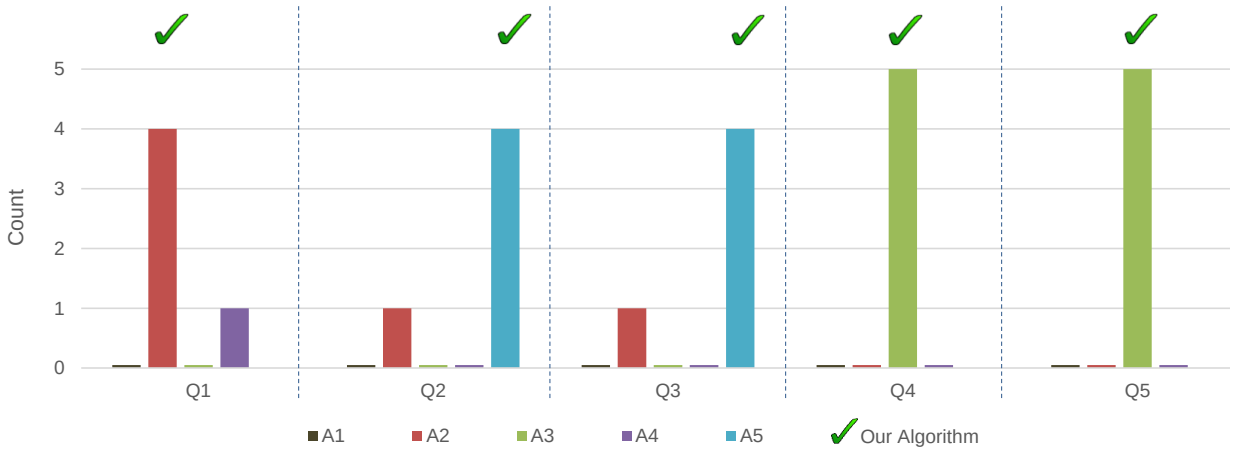

Figure 12: Actions chosen by participants Vs final actions chosen by our algorithm

participants to question 1 to 5 in Table 6, we realize that there is a strong acceptance rate (80 % or 100 %) for the resolution actions chosen by our algorithm. Fig. 12 summarizes the selected actions by the actual modeler compared with the resolution actions applied by our algorithm as the best sequence of actions for creating the final merged model in the motivating example. This figure shows that for conflicts C1 to C3, our algorithm and four of the five participants have chosen the same actions. Furthermore, all five participants and our algorithm selected action A3 to resolve conflicts C4 and C5. These answers are also confirmed by the results for the control question (question 6 in Table 6) that aim to evaluate the satisfaction of participants in general. As a consequence, three participants stated that the results were satisfied, and two others reported that they were very satisfied. After the end of the resolution algorithm, users can optionally review other solutions in the collection of merged versions. This feature allows users to choose another consistent merged model when the merged model with the highest award does not meet their intentions.

Effectiveness. The second objective concerns the benefits of using the proposed resolution algorithm to resolve conflicts and create a consistent merged model in contrast to the manual reconciliation of conflicts by actual modelers. As depicted in Table 7, the minimum required time to fix conflicts and prepare the merged model is 9 min, while this time could be decreased to less than one minute using our approach. Moreover, we compared each of the merged models created by the participants with the merged model shown in Fig. 6 as the final result of our algorithm for the motivating example. Nearly the fraction of the model elements that appear in the participants' model and our final merged models are between 83 % to 93 %. This indicates a strong similarity among merged results created by actual modelers and the final model produced automatically by our approach. We also displayed five merged models with maximum reward to select the best one as merging result. Interestingly, all participants selected *"Option 3"*, which was the final result of our algorithm that led to the maximum total rewards. Consequently, the benefits of applying reinforcement learning algorithms to fix conflicts for a specific modeling language could be effective in automating the resolution phase and preparing a consistent merged model with high quality.

6. Threats to validity

Although experiments provide positive results, we face some validation issues. In the following, we discuss the potential threats that are related to the validity of our experiments and results.

We try to minimize the selection bias by following the participation of modeling experts with different backgrounds, including master students, Ph.D. students, and professors. However, our five participants are not industrial experts, and two of them had a few prior empirical experiences in model merging. Therefore, they might have been less demanding when evaluating the effectiveness of our approach. In this regard, we try to minimize this effect by conducting an empirical workshop to demonstrate model merging examples.

Another threat is associated with the generalizability of our approach across different models. To minimize this risk, we consider three model versioning examples to assess the applicability of our approach. However, all selected models were UML class diagrams implemented based on EMF and Ecore metamodels. We should also evaluate the proposed resolution algorithm with other modeling languages to have an intuition of the generality of our approach.

Another threat to the generalizability is the preparation of quality characteristics for calculating the action rewards. It is obvious that providing a universal set of quality characteristics that work for any modeling language might not be applicable. We used the user-definable EOL scripts to measure the quality metrics to mitigate this threat. This ability supports the possibility of providing further interactions with users to define language-specific quality metrics for different modeling languages.

Moreover, our study demonstrates the resolution process for conflicts arising in merging two concurrently modified models. Hence, our findings cannot be generalized to situations where several modelers work in parallel at the same model. Although, it seems that the proposed workflow and our resolution algorithm can be used for more than two modelers if we can extend the action generation module to prepare appropriate available actions.

7. Related Work

Conflict resolution for model merging is a research field that attracted many researchers in recent years. The researchers propose new conflict detection approaches and build different tools to formulate and (semi-)automatically resolve conflicts that arise during the merge process. The main feature that distinguishes our approach from others is the capability to learn from actions applied to resolve conflicts to find the best available solution.

In the literature, we could not find any research applying RL to resolve the merging conflicts. The most similar work to ours we found is the approach proposed by Barriga et al. [13], who use reinforcement learning algorithms to achieve personalized and automatic model repairing for broken models. Iovino et al. [33] extended this approach to introduce PARMOREL as an extensible framework for repairing the broken domain models. They integrated a tool to measure the quality of Ecore class diagrams based on different quality characteristics, such as maintainability and relaxation. These quality characteristics are used to improve the quality of the repaired models. Although this approach has proposed a technique to repair the broken model, we can also use PARMOREL to restore an invalid model obtained by merging all modifications. However, sequential application of concurrent modifications which are conflictual might lead to a valid merged model. In this situation, some of the conflicts that should be reported due to overlapping changes will remain hidden. More problematic, we cannot apply some of the conflictual change operations (e.g., updating an element that has been removed) to obtain an initial as well as invalid merged model and then repair possible issues. Moreover, to feed the rewarding system compared to PARMOREL, we have considered one more quality characteristic (i.e., the completeness metric), which has been defined based on the merging properties.

There are some recent search-based proposals to resolve conflicts. In this context, Dam et al. [30] present an approach to consistently merge model versions. They define conflicting changes as state space and search for all the possible ways of resolving conflicts using a depth-first search algorithm. This approach provides a systematic proposal to generate a consistent merged version; however, it does not guarantee to find the optimal merged version. Furthermore, Brosch et al. [34] propose a recommender system that, conversely to our work, suggests executable resolution patterns to the users. They use an algorithm based on similarity-aware graph matching to lookup incorporating information from the metamodel. The result allows the recommender to retrieve resolution patterns matched to the given conflict situation from a resolution repository. Gerth et al. [19] also propose to resolve conflicts using the resolution process and suggesting possible resolution actions. They present a method to guide users to manually resolve conflicts by providing an order in which conflicts should be fixed and suggesting appropriate strategies for resolving individual conflicts. In contrast to Brosch et al. [34] and Gerth et al. [19], our approach automatically finds the best sequence of resolution

actions according to quality-based preferences chosen by the user. Moreover, Edded et al. [35] introduced preference-based conflict resolution approach for collaborative configuration of product lines. They first allow users express their preferences through a set of substitution rules. Then, they perform a Minimal Correction Subsets (MCSs) computing algorithm to detect conflicting situations. Finally, they delete the identified situations based on the preferences to avoid conflicts. However, contrary to our work, this approach is limited to a minimal set of conflicting configuration decisions against the feature models.

Synchronous modeling [36] and personalized change propagation [37] are important dimensions of collaborative modeling. Therefore, a few of existing researches are focused on resolving the conflicting change operations to enable consistent change propagation within models. In this regard, Rossini et al. [38] proposed a formalization of the copy-modify-merge approach to support optimistic collaborative modeling, where each modeler can access a repository to create and modify a local replica of model. Then, they introduced a synchronization procedure which is divided into several steps to calculate difference model and detect conflicts. However, the resolution of conflicts requires the manual intervention. To cope with this problem, Rossini et al. [39] proposed a constraint-aware model versioning approach based on the Diagram Predicate Framework (DPF). This approach is applicable to automatically resolve syntactic and semantic conflicts using several resolution strategies.

Moreover, Mafazi et al. [40] presented a framework to resolve conflict for on-the-fly change propagation in business processes based on domain ontologies. Their strategies for resolving conflicts are to combine the conflicting changes or to apply one of them. Koegel et al. [41] also introduced operation-based conflict resolution based on rational management technique, where by defining the *requires* and *conflicting* relations users can specify a pattern to effectively resolve conflicts. Finally, Chong et al. [42] proposed an approach to resolve conflict on composite level operations in merging versions of UML models. To deal with conflicting changes, they first generate a tentative merged model. Then they automatically generate repair options based on the minimal compatibility pattern to assist the user in the resolution phase. While these approaches have the great advantages, their conflict resolution techniques are neither language-independent nor quality-based. In contrast, the nature of our approach is language-independent, which can reuse the experience learned from conflict resolution based on the model quality characteristics.

8. Conclusion and future work

In this paper, we proposed an approach to automatically resolve merging conflicts based on quality characteristics introduced by the modeling engineers as preferences. We presented a resolution algorithm based on the RL techniques that do not require initial training data to find the best sequence of available resolution actions for a list of conflicts. Available actions can be produced directly based on the resolution patterns or three major resolution strategies. We developed a prototype based on the E3MP toolkit [15] and on the Eclipse framework for EMF-based models to demonstrate the applicability of our approach. We also assessed the flexibility and accuracy of our resolution algorithm due to a case study and an actual user survey.

As future work, we aim to extend our prototype to support any modeling language and make our approach independent of modeling tools and conflict detection approaches. To achieve this goal, we will generalize our actions and the conflict detection phase to atomic change operations for instances of any concepts in the modeling language, which leads our approach can be integrated with any existing conflict detection approach and the tooling provided with different modeling languages. Moreover, we plan to conduct more experiments to investigate whether or not the order of conflict selection has an impact on algorithm performance. We want to check the side effects of applying resolution actions on model consistency. Investigating the need for conflict tolerance to enable a collaborative resolution at a later point in time is another future direction to enhance the proposed algorithm. We also intend to investigate the relationship between conflict types and resolution actions to reuse the obtained knowledge for other modeling languages. Finally, the automatic generation of available actions to extend our approach for resolving conflicts arising in the merging of more than two concurrent versions is another research thread that needs consideration.

References

[1] J. Whittle, J. Hutchinson, M. Rouncefield, The state of practice in model-driven engineering, IEEE software 31 (3) (2013) 79–85.

[2] M. Brambilla, J. Cabot, M. Wimmer, Model-driven software engineering in practice, 2nd Edition, Synthesis Lectures on Software Engineering, Morgan & Claypool Publishers, San Rafael, CA, USA, 2017.

[3] M. Franzago, D. Di Ruscio, I. Malavolta, H. Muccini, Collaborative model-driven software engineering: a classification framework and a research map, IEEE Transactions on Software Engineering 44 (12) (2017) 1146–1175.

[4] K. Altmanninger, M. Seidl, M. Wimmer, A survey on model versioning approaches, International Journal of Web Information Systems (2009).
[5] A. A. Koshima, V. Englebert, Collaborative editing of emf/ecore meta-models and models: Conflict detection, reconciliation, and merging in dicomef, Science of Computer Programming 113 (2015) 3–28.
[6] P. Brosch, G. Kappel, P. Langer, M. Seidl, K. Wieland, M. Wimmer, An introduction to model versioning, in: M. Bernardo, V. Cortellessa, A. Pierantonio (Eds.), International School on Formal Methods for the Design of Computer, Communication and Software Systems, Lecture Notes in Computer Science, vol 7320, Springer, Berlin, Heidelberg, 2012, pp. 336–398. doi:10.1007/978-3-642-30982-3_10.
[7] C. Debreceni, G. Bergmann, I. Ráth, V. Daniel, Property-based locking in collaborative modeling, in: 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS), 2017, pp. 199–209.
[8] G. Sunyé, Model consistency for distributed collaborative modeling, in: A. Anjorin, H. Espinoza (Eds.), European Conference on Modelling Foundations and Applications, Lecture Notes in Computer Science, vol 10376, Springer, Cham, 2017, pp. 197–212. doi:10.1007/978-3-319-61482-3_12.
[9] M. Sharbaf, B. Zamani, G. Sunyé, A formalism for specifying model merging conflicts, in: Proceedings of the 12th System Analysis and Modelling Conference, 2020, pp. 1–10.
[10] M. Sharbaf, B. Zamani, A uml profile for modeling the conflicts in model merging, in: 2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI), 2017, pp. 0197–0202.
[11] M. Derntl, P. Nicolaescu, S. Erdtmann, R. Klamma, M. Jarke, Near real-time collaborative conceptual modeling on the web, in: P. Johannesson, M. L. Lee, S. W. Liddle, A. L. Opdahl, Ó. Pastor López (Eds.), International Conference on Conceptual Modeling, Lecture Notes in Computer Science, vol 9381, Springer, Cham, 2015, pp. 344–357. doi:10.1007/978-3-319-25264-3_25.
[12] M. Mohri, A. Rostamizadeh, A. Talwalkar, Foundations of machine learning, 2nd Edition, MIT Press, Cambridge, MA, 2018.
[13] A. Barriga, A. Rutle, R. Heldal, Personalized and automatic model repairing using reinforcement learning, in: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), 2019, pp. 175–181.
[14] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, MIT Press, Cambridge, MA, 2018.
[15] M. Sharbaf, B. Zamani, Configurable three-way model merging, Software: Practice and Experience 50 (8) (2020) 1565–1599.
[16] R. Bellman, Dynamic programming, Science 153 (3731) (1966) 34–37.
[17] T. Mens, A state-of-the-art survey on software merging, IEEE transactions on software engineering 28 (5) (2002) 449–462.
[18] K. Altmanninger, A. Pierantonio, A categorization for conflicts in model versioning, e & i Elektrotechnik und Informationstechnik 128 (11-12) (2011) 421–426.
[19] C. Gerth, J. M. Küster, M. Luckey, G. Engels, Detection and resolution of conflicting change operations in version management of process models, Software & Systems Modeling 12 (3) (2013) 517–535.
[20] D. S. Kolovos, Establishing correspondences between models with the epsilon comparison language, in: European conference on model driven architecture-foundations and applications, Springer, 2009, pp. 146–157.
[21] D. Steinberg, F. Budinsky, E. Merks, M. Paternostro, EMF: eclipse modeling framework, Pearson Education, 2008.
[22] D. S. Kolovos, R. F. Paige, F. A. Polack, On the evolution of ocl for capturing structural constraints in modelling languages, in: Rigorous Methods for Software Construction and Analysis, Springer, 2009, pp. 204–218.
[23] D. S. Kolovos, R. F. Paige, F. A. Polack, The epsilon transformation language, in: International Conference on Theory and Practice of Model Transformations, Springer, 2008, pp. 46–60.
[24] L. Bettini, D. Di Ruscio, L. Iovino, A. Pierantonio, Quality-driven detection and resolution of metamodel smells, IEEE Access 7 (2019) 16364–16376.
[25] M. Chechik, S. Nejati, M. Sabetzadeh, A relationship-based approach to model integration, Innovations in Systems and Software Engineering 8 (1) (2012) 3–18.
[26] M. Genero, M. Piattini, Empirical validation of measures for class diagram structural complexity through controlled experiments, in: Proceedings of the 2002 International Symposium on Empirical Software Engineering, Citeseer, 2001.
[27] F. T. Sheldon, K. Jerath, H. Chung, Metrics for maintainability of class inheritance hierarchies, Journal of Software Maintenance and Evolution: Research and Practice 14 (3) (2002) 147–160.
[28] F. T. Sheldon, H. Chung, Measuring the complexity of class diagrams in reverse engineering, Journal of Software Maintenance and Evolution: Research and Practice 18 (5) (2006) 333–350.
[29] M. Genero, M. Piattini, C. Calero, Early measures for uml class diagrams, L'objet 6 (4) (2000) 489–505.
[30] H. K. Dam, A. Egyed, M. Winikoff, A. Reder, R. E. Lopez-Herrejon, Consistent merging of model versions, Journal of Systems and Software 112 (2016) 137–155.
[31] P. Brosch, M. Seidl, M. Wimmer, G. Kappel, Conflict visualization for evolving uml models., Journal of Object Technology 11 (3) (2012) 1–30.
[32] F. Schwägerl, S. Uhrig, B. Westfechtel, Model-based tool support for consistent three-way merging of emf models, in: Proceedings of the workshop on ACadeMics Tooling with Eclipse, 2013, pp. 1–10.
[33] L. Iovino, A. Barriga Rodriguez, A. Rutle, R. Heldal, Model repair with quality-based reinforcement learning, Journal of Object Technology 19 (2) (2020) 1–21.
[34] P. Brosch, M. Seidl, G. Kappel, A recommender for conflict resolution support in optimistic model versioning, in: Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion, 2010, pp. 43–50.
[35] S. Edded, S. Sassi, R. Mazo, C. Salinesi, H. Ghézala, Preference-based conflict resolution for collaborative configuration of product lines, in: Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), 2020, pp. 297–304.
[36] A. De Lucia, F. Fasano, G. Scanniello, G. Tortora, Enhancing collaborative synchronous uml modelling with fine-grained versioning of software artefacts, Journal of Visual Languages & Computing 18 (5) (2007) 492–503.
[37] M. Sharbaf, B. Zamani, G. Sunyé, Towards personalized change propagation for collaborative modeling, in: 24th International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), IEEE, 2021, pp. 3–7.

[38] A. Rossini, A. Rutle, Y. Lamo, U. Wolter, A formalisation of the copy-modify-merge approach to version control in mde, The Journal of Logic and Algebraic Programming 79 (7) (2010) 636–658.

[39] A. Rossini, A. Rutle, Y. Lamo, U. E. Wolter, Handling constraints in model versioning, in: 3rd International Workshop on Collaborative Modelling in MDE (COMMitMDE@MODELS), CEUR-WS, 2018.

[40] S. Mafazi, W. Mayer, M. Stumptner, Conflict resolution for on-the-fly change propagation in business processes, in: Proceedings of the Tenth Asia-Pacific Conference on Conceptual Modelling-Volume 154, 2014, pp. 39–48.

[41] M. Koegel, J. Helming, S. Seyboth, Operation-based conflict detection and resolution, in: 2009 ICSE Workshop on Comparison and Versioning of Software Models, IEEE, 2009, pp. 43–48.

[42] H. Chong, R. Zhang, Z. Qin, Composite-based conflict resolution in merging versions of uml models, in: 17th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), IEEE, 2016, pp. 127–132.