IET Software

Research Article

UML2Merge: a UML extension for model merging

ISSN 1751-8806
Received on 6th April 2018
Revised 16th May 2019
Accepted on 19th July 2019
E-First on 26th September 2019
doi: 10.1049/iet-sen.2018.5104
www.ietdl.org

Kleinner Farias[1] ✉, Toacy Cavalcante de Oliveira[2], Lucian José Gonçales[1], Vinicius Bischoff[1]
[1]Applied Computing Graduate Program (PPGCA), University of Vale do Rio dos Sinos, São Leopoldo, RS, Brazil
[2]COPPE, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
✉ *E-mail: kleinnerfarias@unisinos.br*

Abstract: Model merging plays a chief role in many software engineering activities, e.g. evolving Unified Modelling Language (UML) models for adding new features. Software developers may evolve UML models using merge relationships. However, given the growing heterogeneity of merge strategies and the UML limitations for expressing merge relationship, it is particularly challenging for them to specify merge relationships. Consequently, developers, end up expressing improperly merge relationships. Today, the UML can neither specify the semantics nor the order in which merge relationships must be performed. Developers are unable to specify the semantics and order in which merge relationships must be performed. This study, therefore, proposes UML2Merge, which is a UML extension for expressing merge relationship. The UML2Merge was evaluated through an empirical study with 10 participants for investigating its effects on the merge effort, the correctness of merge relationships, and the participant's acceptance. The collected data suggest that the UML2Merge is proper to express merge relationships by requiring a low merge effort, producing elevated correctness of merge relationships, and having a high acceptance of the participants. The results are encouraging and show the potential for using UML2Merge to express the evolution of UML models through merge relationships.

1 Introduction

The Unified Modelling Language (UML) is the de facto standard for object-oriented software modelling [1–3] and has been widely used to represent evolving UML models (e.g. UML class diagrams), especially in scenarios where model merging is required. In collaborative software development, for example, separate virtual teams may concurrently work on a partial model of the overall architecture to allow developers to concentrate more effectively on parts of the architecture relevant to them. At some point, it is necessary to combine these models to produce a ‘big picture’ view of the overall architecture. For this reason, a significant body of research has been done in the field of collaborative software modelling [4], merge conflicts [5], model synchronisation [6], integration of heterogeneous models [7], and integration of feature models [8].

Regardless of the application field, developers usually need to specify how UML model elements should be combined. Bischoff *et al.* [8] highlight that there may be many different ways of combining design models. However, it is particularly challenging to specify how to merge input models, given the growing heterogeneity of the merge strategies, and mainly the limitations of UML to express specific merge semantics, or even the order of model merging. Hence, developers end up being unable to express how the parts of the input models should be combined.

To date, the UML lacks constructs that help developers to express merge relationships. Previous empirical studies [9, 10] have identified a series of ambiguous and missing rules in the UML built-in merge mechanism, namely UML package merge [3]. Although many UML extensions have been proposed over the past years [11–13], this literature still fails, for example, to parameterise merge relationships with specific semantics. Guessi *et al.* [11] provide a thematic analysis of studies on UML extensions for supporting aspect-oriented modelling. Still, overcoming these challenges is crucial to address broader challenges in the context of software development in practice. Rubin and Rinard [14] point out some challenges related to efficient integration of software artefacts to support ‘building a large system while controlling interactions between all its different parts.’ Rubin and Chechik [15] also highlight those merging design models is an NP-hard problem. To sum up, developers end up having critical problems in employing UML constructs to support the evolution of architecture models using merging relationship.

This paper, therefore, presents UML2Merge, which is a UML extension to express merge relationship. For this, we extend the UML metamodel with a set of constructs following the open–closed design principle [16] to ensure a conservative extension, thereby allowing developers to extend it with specific matching and merge semantics, e.g. override, merge, and union (but not limited to). The UML2Merge specification is based on the UML metamodelling standard for being more intuitive and pragmatic in real-world settings [3]. Note that our study introduces a UML extension rather than matching and merging strategies.

The UML2Merge was evaluated through an empirical study with 10 participants for investigating its effects on the merge effort, the correctness of merge relationships, and the participants' acceptance. The collected data suggest that the UML2Merge is appropriate to express merge relationships as it required a low merge effort, produced a high correctness of merge relationships, and having an elevated high acceptance of the participants. Our initial evaluation has also shown that the extension is (i) proper to express merge relationship and its execution order, as well as (ii) flexible to support new match and merge strategies by making the extension more adherent to the open–closed principle [16]. The results are encouraging and show the potential for using UML2Merge to express the evolution of UML models through merge relationship.

In addition, it is important to highlight that this study is not an attempt to address all contexts in which model merging can be applied, since this would be extremely burdensome given the several meanings that it can assume in different contexts, e.g. merging of independently developed models, merging of models with common ancestors, merge of behavioural components (parallel and sequential merge), and weaving of aspects. Still, this study does not seek to provide expressiveness for supporting, e.g. join points for aspect models, even for the simple case where there is an individual diff model. Rather, this study is motivated by the need for specifying *how* UML models should be combined in

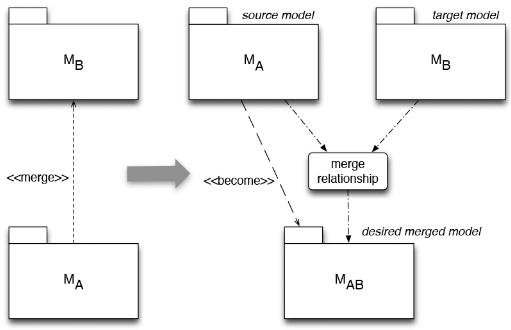


Fig. 1 *Conceptual view of the merge relationship semantics (adapted from [3])*

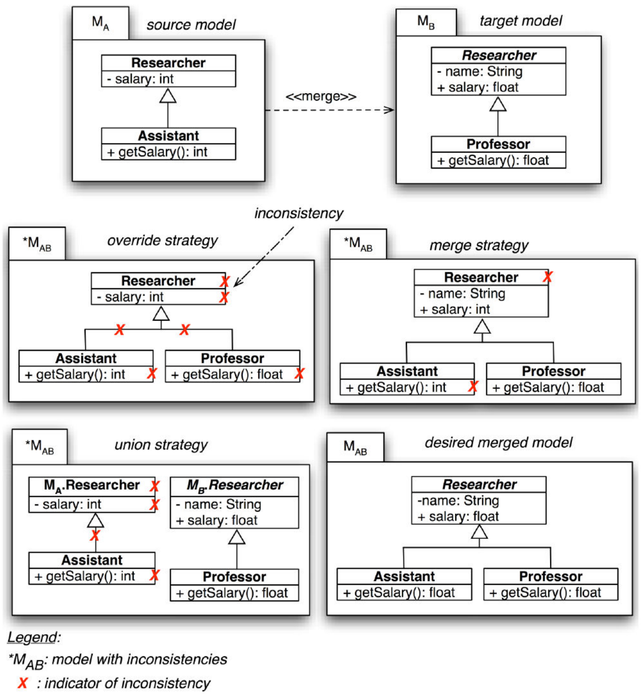


Fig. 2 *Illustrative example of model merging*

evolution scenarios. Based on this merge relationship specification, the current tools can automate merge tasks, for instance.

The remainder of the paper is organised as follows. Section 2 introduces the main concepts and knowledge that are going to be used and discussed throughout the paper. Section 3 presents the proposed UML extension for representing model merging. Section 4 describes the methodology used for evaluating the UML2Merge, and presents some results. Section 5 contrasts our work with the current literature. Finally, Section 6 presents some concluding remarks and future work.

2 Background

This section discusses the main terms used throughout this paper. Section 2.1 introduces key concepts by presenting a comprehensive example of model merging. Section 2.2 outlines the metamodelling pattern.

2.1 Model merging

2.1.1 Merge relationship: The term *merge relationship* might be briefly defined as a specification of an operation in which a set of tasks should be performed over two input models, M_A and M_B, to produce a desired merged model, M_{AB}. M_A and M_B represent the *source* and *target* of a merge relationship, respectively. M_B has a set of increments that should be accommodated into M_A to transform it into M_{AB}. Fig. 1 illustrates a merge relationship between M_A and M_B, and the role assumed by them (source and target model). Merging M_A and M_B means to match M_A and M_B (Section 2.1.2) for identifying their commonalities and differences. After that, model elements of M_A and M_B that are equivalent should be combined using a merge strategy (Section 2.1.3). Evidence from previous studies (such as [17]) has shown that M_A and M_B frequently suffer from conflict problems (Section 2.1.4). Also these conflicts are often solved improperly, giving rise to inconsistencies in the merged output model. This becomes the practice of merging models a highly error-prone task.

2.1.2 Matching strategy: Before producing a desired merged model, developers need to compare the input models so that the commonalities and differences between them may be identified. Matching strategies define the semantics about how the input models should be compared and contrasted, identifying their overlapping parts and differences. Developers can match the input models in different ways, e.g. following a named-based matching strategy [18] (default strategy), considering syntactic properties (partial strategy), or even considering both syntactic and semantic properties (complete strategy). To put all strategies in practice, developers may elaborate *matching rules* to implement and refine them. For example, developers might use the Epsilon Comparison Language [19] for coding such matching rules.

After matching the input models, the next step is to put them together. For this, merging strategies can be used. Some examples of merging strategies are presented as follows. Note that the definition of matching and merging strategies is beyond the scope of this paper. This work just uses a set of merge strategies.

2.1.3 Merge strategy: Based on the overlapping points between M_A and M_B, and their differences, identified by the matching strategies, merge strategy defines how to integrate the matching parts. We adopted (for convenience) in the proposed UML extension (but not limited to) three merging strategies (i.e. override, merge and union described in [20]). The matching and merging strategies were chosen because they have been applied to a wide range of merging scenarios [5] – evolution of design models, ontology merging, and conceptual model merge. Furthermore, they have been recognised as handy, helpful heuristics in evolving architectural models (e.g. [20]). Therefore, we present such strategies as part of the proposed UML extension (Section 3.2). In the following, we briefly introduce these three strategies, and assume M_A and M_B as the input models.

Fig. 2 illustrates an example about how the matching and merging strategies are used to evolve a simple UML class diagram. M_A plays the role of the source model, while M_B plays the role of the target model. M_B has the changes that should be inserted into M_A to produce a new version of a UML class diagram, M_{AB}. To generate this desired new version, M_A and M_B are compared to identify their overlapping parts. The equivalence between the model elements can be obtained by using match-by-name matching strategies (e.g. [18, 21, 22]).

Thus, the classes M_A.*Researcher* and M_B.*Researcher* have equivalent names. However, M_A.*Researcher* is a concrete class (i.e. *Researcher.isAbstract* = false), whereas M_B.*Researcher* is an abstract one (i.e. *Researcher.isAbstract* = true). The equivalent parts and differences between model elements can be combined following a particular merge strategy. The merge strategies considered in our study are briefly discussed as follows:

- *Override:* For all pairs of corresponding elements in M_A and M_B, the elements of M_A should override the similar elements of M_B. The different model elements remain unchanged, i.e. they are just inserted into the merged output model. For example, Fig. 2 shows an example of merged model, M_{AB}, which was produced using the override algorithm. In this case, the class M_A.*Researcher* (concrete class) overrides M_B.*Researcher* (abstract class), producing M_{AB}.*Researcher* (concrete class).
- *Merge:* For all pairs of corresponding elements in M_A and M_B, the elements should be combined. The fusion depends on the

types of model elements, i.e. the only model element of the same type can be combined. If two model elements have same name but different types (e.g. Component and Class), then they cannot be merged. The model elements of M_A and M_B that are not equivalent remain unchanged and are directly inserted into the merged output model. For example, Fig. 2 depicts a merged model, M_{AB}, which was elaborated using the merge algorithm. In this case, the class *M_A.Researcher* (concrete class) and *M_B.Researcher* (abstract class) are merged, producing *M_{AB}.Researcher* (concrete class) with two attributes.

- *Union:* All pairs of corresponding elements in M_A and M_B are manipulated, having their names modified, and inserted into M_{AB} so that their identification can be preserved. This means that they coexist but with different identifiers or names in a merged output model. The elements of M_A and M_B that are not correspondent remain unchanged, being inserted into the merged output model, M_{AB}. In Fig. 2, the merged output model, produced using union strategy, has two classes, *Researcher*, which have the package name from which they come from, i.e. M_A and M_B. The classes, *Assistant* and *Professor*, were just inserted into M_{AB} without any modification.

2.1.4 Merge conflicts and inconsistency: *Merge conflicts* consist of contradictions between values assigned to the properties of design models. Fig. 2 illustrates an example of a merge conflict. In the source model, the class *M_A.Researcher* is defined as a concrete class (i.e. *M_A.Researcher.isAbstract* = false), whereas in the target model it is set as an abstract one (i.e. *M_B.Researcher.isAbstract* = true). These contradicting values assigned to *isAbstract* represent a conflict that must be solved by developers. This implies that developers should answer the following question: should class *Researcher* be abstract or not?

Based on the desired merged model in Fig. 2, the correct answer for this question would be that *Researcher* should be abstract – i.e. *Researcher.isAbstract* = true. However, developers might assign *false*. This would generate an *inconsistency*, which can be briefly defined as divergences between model elements found in the merged output model and desired merged model. Fig. 2 presents two merged output models, which were produced using the override strategy and merge strategy. Both models have inconsistencies.

We can observe that the produced models using the override and merge strategies present more problems as compared with the desired one. First, the model produced following the override algorithm has three inconsistencies: (i) the class *Researcher* is concrete, instead of abstract; (ii) the visibility of attribute *Researcher.salary* is private, instead of public; (iii) the methods *Assistant.getSalary(): int* and *Professor.getSalary(): float* cannot access *Researcher.salary* because the attribute s visibility is private. Second, the model produced using the merge algorithm has, in turn, two inconsistencies: (i) the class *Researcher* is concrete, instead of abstract; and (ii) the method *Professor.getSalary():int* cannot return a float, as would be expected.

For this, developers need to be able to specify which merge strategy should be applied, and in which order they should be performed when two (or more) merge relationships are defined. If the merge semantics and its execution orders are not properly expressed, the model merging can become an error-prone and time-consuming task [20]. In fact, model elements of M_A and M_B can conflict with each other and developers need to be able to decide how the conflicts should be solved. Conflicts cannot be avoided at this stage, they can only be resolved.

Unfortunately, developers tend to overlook conflict problems, given the problem at hand. Consequently, a model with inconsistencies, M_{CM}, is produced instead of the desired merged model, M_{AB}. If $M_{CM} \neq M_{AB}$, then developers should invest some effort so that the inconsistencies in M_{CM} can be repaired, recovering M_{AB}. In practice, developers can be aided by traditional matching and merge strategies to produce a desired output model, or even a merged output model close to a desired output model.

2.2 Metamodelling pattern

This section describes the UML metamodelling pattern [3] used to specify the proposed extension. According to OMG [3], the key role of a metamodel is to define the semantics for how model elements in a model get instantiated. The definition of UML2Merge (Section 3) follows the metamodelling pattern, presented in [3], due to some reasons. First, the UML metamodel specification pattern is the mainstream way to define the meaning of the constructs of the current UML extensions, such as [12, 13]. Secondly, it provides practitioners and researchers with a systematic but easy-to-read specification style [16], enabling an improved understanding of the meanings of the proposed extension, and reducing an inappropriate use in practice. Thirdly, it leverages the compliance with the semantics of UML metaclasses, their structural relationships, and constraints, thereby allowing a seamless integration between the UML specification and the proposed one. Finally, it is a well-established way for specifying the syntax and semantics of metamodel [3]. This can be perceived when reviewing the current literature (Section 5) that customises UML metamodel constructs, mainly ones proposed and maintained by OMG [3] in the past decade.

For this, the following concepts guide the definition of the UML2Merge constructs:

- *Abstract Syntax.* It defines the metaclasses that represent the language constructs, e.g. Class, Attribute, Association, their relationships, multiplicity, and ordering constraints. These constructs and their relationships are described in natural language.
- *Attributes and Associations.* The attributes and associations are enumerated along with a short explanation. The multiplicity of the attributes is suppressed when it assumes a default value, i.e. the value of 1 (default in UML metamodel [3]).
- *Semantics.* It defines the meaning of each well-formed construct, while the static semantics defines how an instance of a construct should be connected to other instances to be meaningful; the dynamic semantics defines the meaning of a well-formed construct [3]. The meaning of a created model using the proposed extension is correctly defined, i.e. it is well-formed if and only if it achieves the rules defined in the static semantics.
- *Well-formedness rules.* It defines constraints that should be fulfilled to get instantiated a valid and meaningful model. The constraints concerning multiplicity and ordering are defined in the metamodel. Thus, the rules specify constraints over attributes and associations defined in the metamodel. For this, OCL (Object Constraint Language) expressions [23], along with an informal explanation of the expression, specify the proposed invariants. However, as far as possible, the invariants are expressed in natural language.

3 Proposed UML extension

This section presents the UML2Merge, a UML extension to support the specification of merge relationship between UML models. The main contribution of this work is the proposed abstract syntax, constructors and notations to express merge relationships. Section 3.1 pinpoints the UML and UML2Merge constructs that may participate in a merge relationship. Section 3.2 specifies the UML2Merge s merge relationship.

3.1 Composable and composite

Fig. 3 shows the abstract syntax of the UML2Merge regarding the constructs that are able to participate in the merge relationship. It shows which constructs are from UML, such as *NamedElement*, *Parameter*, and *Property*, and those constructs proposed in our extension, such as *ComposableElement* and *CompositeElement*. We highlight that the way that UML metamodel specifies the participants of the inheritance relationship in [3] was the basis to describe these elements. Moreover, we have used the composite design pattern [16] to organise the model elements. We have chosen this pattern for the following reasons. The first would be to

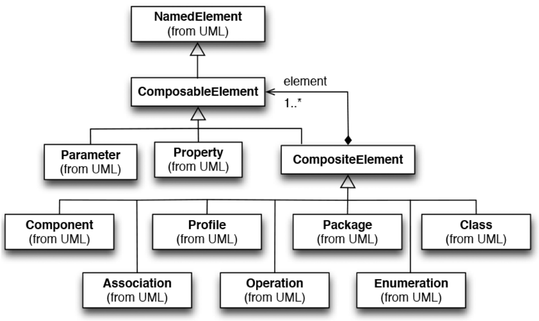


Fig. 3 *Constructs that participate in a merge relationship*

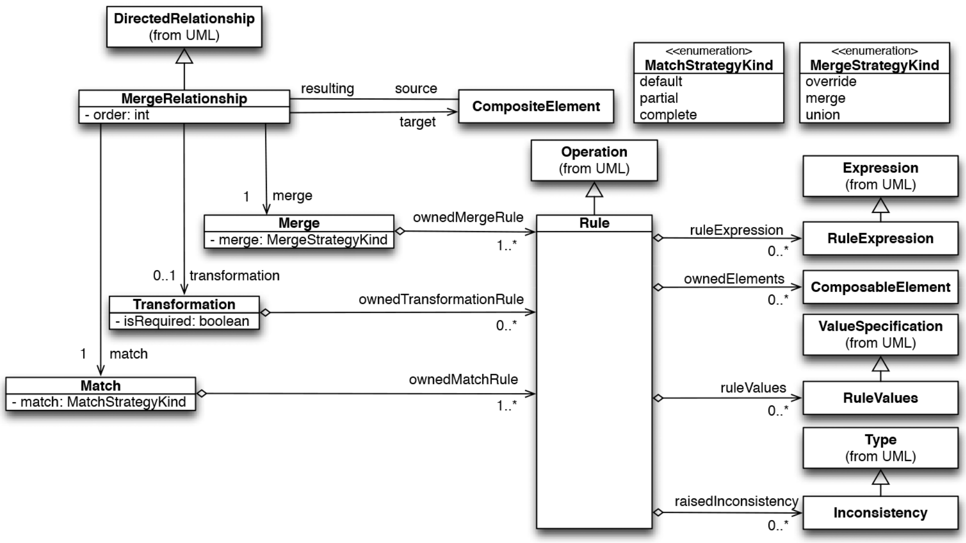


Fig. 4 *Proposed UML extension for expressing model merge*

represent part-whole hierarchies of objects, which might be to participate of a merge relationship. Next, merge relationship can treat uniformly all objects in the composite structure. Thirdly, new constructs, extending our approach, can be supported by just accommodating them in the composite structure; these constructs would need to extend the *CompositeElement* metaclass in Fig. 3.

CompositeElement arranges the *ComposableElements* into tree structures by using a whole-part relationship. Usually, merge techniques are able to combine a set of UML diagrams, not all. Therefore, defining the model elements that can participate in a merge relationship is pivotal to create an alignment between the specification and the practice to combine UML diagrams. Merge techniques can read the specification, and then combine the input models based on this specification.

ComposableElement. It is an abstract class and subclass of *NamedElement*, a central construct in the UML metamodel. It determines the basic semantics for constructs so that they may take part of a merge relationship. On the other hand, the *CompositeElement* is an abstract class that works as a container, i.e. bringing together the composable constructs seamlessly. It represents the UML constructs that contain other elements.

Some constructs need to be analysed individually during the definition of similarity and integration of the model elements, including *Parameter*, *EnumerationLiteral* and *Property*. Thus, this extension defines construct hierarchies, which comprise primitive and composite constructs. While the composite can be decomposed into simpler ones, which in turn can be decomposed, recursively. Hence, merge techniques can deal with composite and primitive constructs indistinctly. This offers two advantages: (i) merge techniques can have more specific matching and merge rules for each construct, rather than generic ones, as for each model element the merge technique should have a rule describing how they should be combined; and (ii) the ease of adding new kinds of constructs, as well as extending merge techniques. In the following section, we present the merge relationship construct.

3.2 Merge relationship and its constructs

This section presents the merge relationship itself and its constructs. Fig. 4 shows the abstract syntax of the proposed merge relationship.

3.2.1 Merge relationship: The *MergeRelationship* metaclass is a directed relationship between two *CompositeElements*. It extends the *DirectedRelationship* metaclass from the UML metamodel, and has two associations with the *CompositeElement* metaclass. This inheritance from *MergeRelationship* to *DirectedRelationship* represents one of the points where the UML2Merge is accommodated into the UML metamodel. The relationship is formed by two models – source (M_A) and target (M_B) –, which are *CompositeElement* constructs. A *merge relationship* produces a resulting merged model, or merged output model. It specifies how the content of a source model can be extended by the target model. The content of the target model is added to the source model to generate a merged output model.

This relationship may be used to combine the content of different models that should represent a same concept, or even supply different meaning of a model element for different purposes. Each relationship aims at accommodating increments into the model elements found in the source model. With this in mind, using the merge relationship developers can evolve model elements with increments for different purposes, where each increment is defined in a particular target model. Creating a merge relationship means to establish a semantic interplay between the input models. This semantics may be defined by a merge strategy (Section 2.1.3), which can use, for example, a set of merge rules so that a desired merged model can be produced, M_{AB}.

By using a merge relationship, developers can mitigate a critical problem previously mentioned – the UML incapacity of representing multiple merge relationships. Given that it is a

DirectedRelationship, its navigability means that the merging should be established in a particular direction.

3.2.2 Associations: The associations of *MergeRelationship* are introduced as follows:

- *target: CompositeElement* [1]. It specifies the *target* of a merge relationship, which is the model to be combined with the *source* one, the origin of the merge relationship. The multiplicity ‘1’ indicates that each merge relationship must have an instance of *CompositeElement* playing the role of target model. It also represents a subset of *DirectedRelationship::target* defined in the UML metamodel.
- *source: CompositeElement* [1]. It determines the *source* of a merge relationship. It is a *CompositeElement*, which is being extended with the contents of the target *CompositeElement*, and represents a subset of *UML::Element::owner* and
- *UML::DirectedRelationship::source* defined in the UML metamodel.
- *merge: Merge* [1]. It specifies the merge semantics must be applied to merge relationship. In Section 2.1.3, we suggest some strategies available in literature.
- *match: Match* [1]. It specifies the match strategy should be applied to merge relationship. The multiplicity ‘1’ determines that each merge relationship must have an instance of *Match* identifying how the input models (source and target ones) should be compared.
- *transformation: Transformation[0..1]*. It specifies a set of transformations that may be applied to resolve inconsistencies found in the merged output models. These transformations play an important role as they transform an output model with inconsistencies into a desired resulting model. The multiplicity ‘0..1’ indicates that the use of transformation is optional. The manner how these transformations are represented is out of the scope of this paper.

3.2.3 Semantics: As previously mentioned (Section 2.1.3), a merge strategy is responsible for determining the semantics through which the content of the target model may be accommodated into the source model – in the same way that a subclass in a generalisation relationship consists of the aggregation of all features of all of its superclasses, and not only the increments attached by it. As a result of that, any reference to the content of the source model means to refer to the merged output model, rather than to the increment derived from the merged model.

Fig. 5 shows this case by demonstrating that the packages M_A and M_B attach different contents to a particular model element of the merged output model. For example, the packages M_A (source model) and M_B (target model) determine different elements to the class *Researcher*, identified as M_A.*Researcher* and M_B.*Researcher*, respectively. Package M_A merges the contents of package M_B, which means to combine M_A.*Researcher* and M_B.*Researcher*. The package M_C imports the contents of M_A to define the subclass of *Researcher*, so-called *Professor*.

Thus, the class M_A.*Researcher* represents the result of merging M_B.*Researcher* and M_A.*Researcher*, not just the model elements found in M_A.*Researcher* upfront. The package M_D imports the contents of M_B. However, it refers to M_B.*Researcher* rather than the M_A.*Researcher*, resulting from the merge of M_A.*Researcher* and M_B.*Researcher*. As a result, the class M_D.*University* has an attribute *-people: Researcher[1..*]*, where the *Researcher* is derived from M_B instead of M_A.

In contrast, Fig. 6 exhibits two merge relationships to demonstrate the need to express the order in which the merge relationships should be performed. This case shows that the execution order is de fact a problem in the field of model merge. In this case, packages M_B and M_E are being merged with the package M_A. Package M_D imports the content of the package M_B, as previously mentioned. However, now it is hard to determine which content the package M_C will import from the package M_A, given

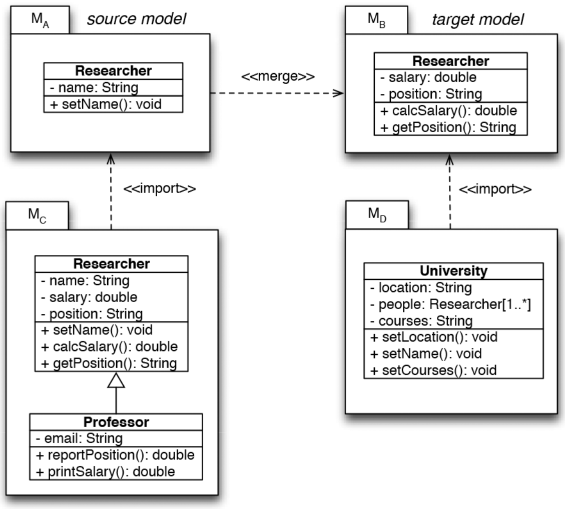


Fig. 5 *Illustrative example of the merge relationship semantics. M_A and M_B represent the source and target models of the relationship, respectively. While M_A is the source model, M_B has a set of increments, which will be accommodated into M_A to transform it into M_{AB}*

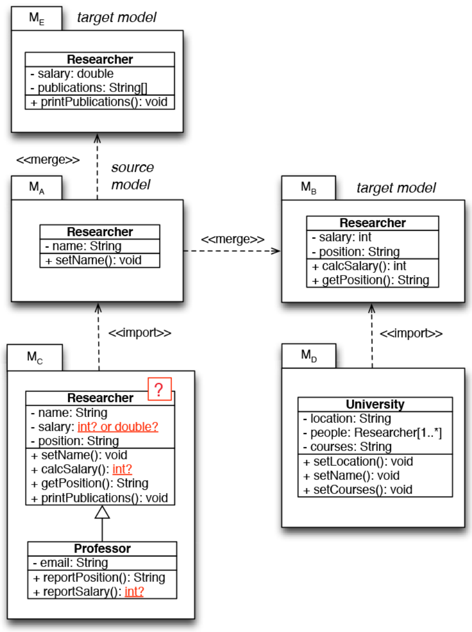


Fig. 6 *Example demonstrating the need to express the order of merge relationship*

the different orders to merge the package M_A with the packages M_B and M_E. We can merge the package M_A with the package M_B and after with the package M_E, or even first with the package M_E and after with the package M_B. In M_B, the attribute *Researcher.salary* is defined as *int*. In contrast, in M_E this attribute is defined as *double*. Thus, developers need to properly answer the question: should the attribute *Researcher.salary* be *int* or *double*?

Given that the merge order has not been defined yet, this conflict must be solved manually so that these attributes can be merged. Establishing a merge order, we can specify a merge semantics that leads, for example, *Researcher.salary: int* to override *Researcher.salary: double*, or vice-verse. If M_C.*Researcher.salary* is defined as *double*, then the method *Researcher.calcSalary():int* and *Professor.reportSalary(): int* should be changed, i.e. their return types changed to *double*. The desired merged model can be obtained setting *Researcher.salary* as *int*. In case this question is not properly answered, this conflict will be transformed into an inconsistency in the merged output model.

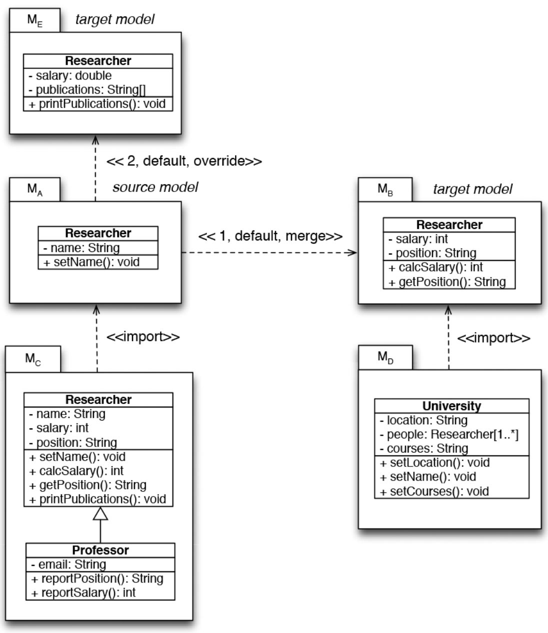


Fig. 7 *Solving the order problem in expressing merge relationships*

3.2.4 Notation: The proposed merge relationship is represented using a dashed line with an open arrowhead pointing from the source model to the target model. Besides specifying the source and target models, three keywords need to be defined, including order, matching strategy, and merge strategy. Code 1 represents the grammar used to define the UML2Merge notation. The first establishes in which order the merge relationship should be considered, avoiding the *execution order problem* of the merge relationships. The second specifies the matching strategy, which designates how the model elements should be compared. The third determines the merge strategy, which delineates how the model elements considered equivalent should be combined. Moreover, keywords are displayed close to the dashed line, which are defined as a text string according to the BNF (Backus Normal Form) defined in Code 1.

$$
\begin{aligned}
&< \texttt{merge} - \texttt{relationship} >:: \\
&= \grave{}\ <\ <'\ < \texttt{order} > \grave{},' \\
&< \texttt{matching} - \texttt{strategy} > \grave{},' \\
&< \texttt{merge} - \texttt{strategy} > \grave{}\ >\ >' \qquad (1)\\
&< \texttt{order} >::= [0-9]* \\
&< \texttt{matching} - \texttt{strategy} >::= [\mathrm{A}-\mathrm{Za}-\mathrm{z}]* \\
&< \texttt{merge} - \texttt{strategy} >::= [\mathrm{A}-\mathrm{Za}-\mathrm{z}]*
\end{aligned}
$$

Fig. 7 shows how the problem reported in Fig. 6 might be solved using the notation proposed. Firstly, M_A.*Researcher* is merged with M_B by using the default name-based matching strategy and the override strategy. This is represented by the notation ≪1, default, merge≫ over the relationship between the M_A and M_B, where 1 is the execution order, *default* is the default match strategy, and *merge* is the merge strategy. Next, the resulting model M_{AB} *Researcher* is merged with M_E. *Researcher*. This is also represented by the notation ≪2, default, merge≫ over the relationship between the M_{AB} and M_E, where 2 is the execution order, *default* is again the default match strategy, and *merge* is the merge strategy.

Merge. This metaclass defines the semantics of merging by specifying a merge strategy to be assigned to a merge relationship. Moreover, it also defines the merge rules that implement the meaning of the merge relationship itself. The following presents its property and association:

– *merge: MergeStrategyKind* represents the merge strategy to be assigned to the merge relationship. The *MergeStrategyKind* defines assignable values for this attribute.

+ *ownedMergeRule: Rule[1..*]* consists of a set of Rules (merging rules). The multiplicity '1..*' specifies that *Merge* always have a reference to *Rule*, which defines the rules that are responsible for combining the model elements considered equivalents. The developers can create such merging rules using OCL [23] or Epsilon Merging Language (EML) [19], for example.

Match. It represents match strategy to be applied to a merge relationship:

−*match: MatchStrategyKind* defines the different types of match strategies that can be associated with the merge relationship. Moreover, its possible values are those found in *MatchStrategyKind*.
ownedMatchRule: Rule[*] consists of a collection of Rule (match rules). The multiplicity '1. *' specifies that Match will always have a reference to Rule, which defines the match rules.

Transformation. It expresses the changes that should be done to overcome inconsistencies found in the merged output model. Transformation rules can be assigned to merge relationship so that inconsistencies in M_{CM} can be solved. The multiplicity '0..1' indicates that a merge relationship may (or not) have model transformation rules (i.e. optional). Its property and association are presented as follows:

- *ownedTransformationRule*: *Rule[*]* consists of a set of Rules (model transformation rules). The multiplicity '*' indicates that can have a set of rules. *isRequired: Boolean* is equal to true when the specifications of model transformation rules should be applied to a merge relationship.

Rule. It adds to the UML metamodel the capability of representing rules to be used in the model merge. It extends the *Operation* metaclass defined in the UML metamodel. This extension allows developers to specify the match, merge, and transformation rules using natural language or even some language, such as OCL [23] or Epsilon [19]. This inheritance relationship represents the point where the UML2Merge is inserted into the UML metamodel. An important observation is that a rule will be executed if its preconditions are satisfied. This construct has some associations that are described as follows:

- *ruleExpression: RuleExpression[*]* represents a set of *RuleExpressions*. The multiplicity '*' means that it may (or not) have a reference to *RuleExpression*. Developers use it to define the rules to be used to implement the merge relationship.
- *ruleValues: RuleValues[*]* consists of a collection of *RuleValues*. The multiplicity '*' specifies that it may (or not) have a reference to a *RuleValues*. It is important to highlight that a rule to be valid it must satisfy the values specified in *ruleValues*.
- *ownedElements: ComposableElement[*]* is the operands of the rules.
- *raisedInconsistency: Inconsistency[*]* represents the conflicts can raise from the merges.

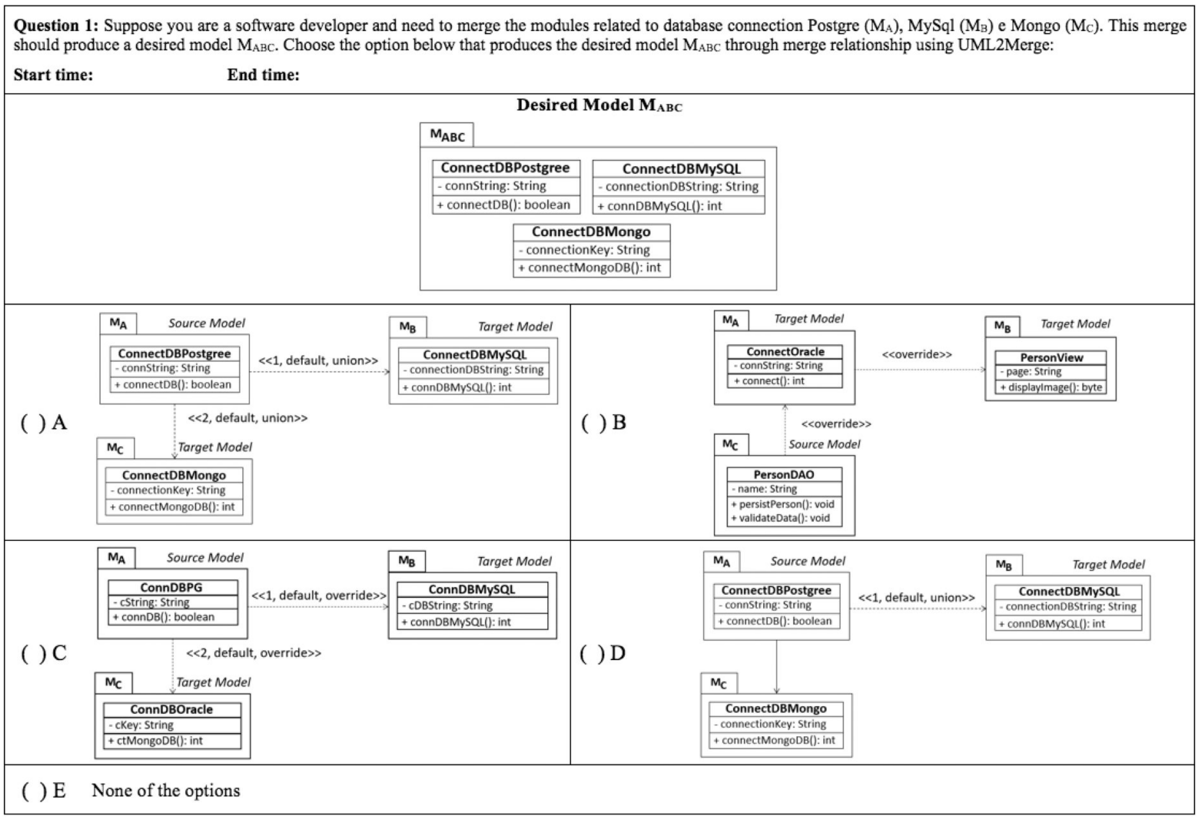
Inconsistency. It represents the contradicting changes found in the input model elements that were improperly resolved.

RuleExpression. It defines a structured-tree of symbols, which denotes a particular value for a rule. Developers use it for elaborating the rules to be applied to merge relationship, including the match rules, merge rules and model transformation rules. For example, *MatchEnumeration*(*Enumeration rcv*, *Enumeration mrgd*) would represent an expression of a match rule to be applied to *Enumeration*.

RuleValues. It represents the values that are manipulated by the rules and is an instance of *ValueSpecification* (from UML).

MatchStrategyKind. It is an *Enumeration* that defines the match strategy to be applied during the merge. It can assume the following values: default, partial and complete, representing the default, partial and complete match strategy, respectively.

MergeStrategyKind. It consists of an *Enumeration* that defines a literal for specifying the types of strategies that can be used for


Question 1: Suppose you are a software developer and need to merge the modules related to database connection Postgre (M_A), MySql (M_B) e Mongo (M_C). This merge should produce a desired model M_{ABC}. Choose the option below that produces the desired model M_{ABC} through merge relationship using UML2Merge:

Start time: **End time:**

Desired Model M_{ABC}

( ) A

( ) B

( ) C

( ) D

( ) E None of the options

Fig. 8 *First question used in our empirical evaluation*

merging two *CompositeElements*. This element has the following literals (but not limited to) 'override', 'merge', and 'union'. These literals represent the override, merge, and union merge strategy, respectively. We use this element to specify the semantics to be applied in a merge relationship, being this meaning passed by parameter.

4 Evaluation

This section focuses on describing the methodology to evaluate the UML2Merge, and presents the collected results. Section 4.1 describes our objective and research questions (RQ). Section 4.2 introduces the formulated questionnaire to evaluate the UML2Merge. Section 4.3 explains the context and participant selection. Section 4.4 introduces the experimental design. Section 4.5 presents the analysis of the collected data. All these methodological steps were followed based on well-established practical guidelines about empirical studies presented in [24].

4.1 Objective and research questions

This empirical study seeks to evaluate the effects of UML2Merge on two variables: the developers effort, and the correctness of such merge relationships. These effects were explored from realistic scenarios by expressing evolutions of UML class diagrams. In this sense, we use the GQM template [24] to state the objective of this evaluation, as follows:

Analyse *the proposed UML2Merge*
for the purpose of *investigating their effects*
with respect to *effort, correctness and acceptance*
from the perspective of *developers*
in the context of *evolving design models.*

For software developers in industry, maximising productivity and not wasting time using techniques that are not cost-effective are essential things. So, we evaluate the effects of the UML2Merge on the effort invested by developers to express merge relationship, and the correctness of such merge relationships. In addition, we also assess the UML2Merge applicability and acceptance from the perspective of software developers. Thus, we focus on exploring three RQ, as follows:

- *RQ1*: What is the effort of expressing merge relationship using the UML2Merge notation?
- *RQ2*: Does the proposed extension favour the production of correct merge relationships?
- *RQ3*: What is the acceptance of the proposed extension by software developers?

4.2 Questionnaire

To answer these RQ, we elaborated a questionnaire [Our questionnaire is available at this link: https://kleinnerfarias.github.io/pdf/studies/uml2merge-quest.pdf.], which was answered by our participants (introduced in Section 4.3). This questionnaire was written in Portuguese because our study was focused on Brazilian developers. Our questionnaire consists of three parts, which are detailed below.

Part 1: participant profile: The first part of the questionnaire sought to collect data related to the characteristics and opinions of the participants. Creating a participant profile with such data is important to select only those who are potential users of the UML2Merge. Without this profile, participants with inadequate profile might generate incoherent evaluation. To do this, we ask the participants for more general information, such as age, gender, current position, profession, level of education. Information regarding the level of experience has also been considered, including academic background, education at university, experience with software development and modelling, and UML knowledge level. We need to be sure that the proposed technique is being evaluated by people whose profile would be of future users of the UML2Merge.

Part 2: usage scenarios of UML2Merge: The second part has six usage scenarios to properly investigate the RQ1 and RQ2. In total, six multiple-choice questions were elaborated. Fig. 8 shows the first question used in our empirical study (*option A* is the correct answer). Each question was composed of a description of evolution, a desired merged model, and five options. For each multiple-choice question, four possible answer options (A–D) had merge relationships using the UML2Merge, while one answer option (E) allowed participants to inform that no answer could be given because of problems in the merge relationship. One of the

Table 1 Tasks of the evolution scenarios

Task	Models	Required changes to the base model
1	database connection	add class to establish different database connections.
2	management	change attribute type, add methods, add attributes.
3	eCommerce	add attributes and methods.
4	health care	remove methods, add classes (*control question*).
5	ERP	add classes, methods, attributes and relationships.
6	calculator	change classes, add classes.

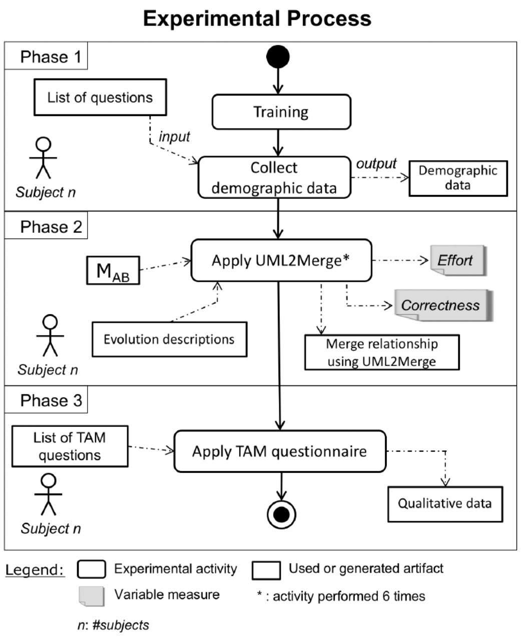


Fig. 9 *Experimental process*

four options (A–D) had the merge relationship that would generate the desired model, i.e. the correct option. The correct option was randomly assigned to one of the four options (A–D). Three options were incorrect merge relationships. Participants were then encouraged to choose the option that had the merge relationship that would generate the desired model. For this, they should consider the evolution descriptions, mentioning the changes that need to be made using the UML2Merge to produce the desired model.

Participants could answer the questions sloppily, without attention. This would pose a threat to the quality of our data. To mitigate this threat, a control question (Question 4) was created, in which no merge relationship would produce the desired model. This question had only incorrect options of easy perception. The correct answer of the control question would be the E-option, which indicated that no merge relationship would be able to produce the desired model. If a participant incorrectly answers this question, this may imply that he did not understand the approach or was unattended.

The UML models of our questionnaire had by about three classes and four relationships. Being a first evaluation, having a small size of models was essential to mitigate possible influences of size in relation to the UML2Merge applicability and acceptance. Mens [25] highlights that software changes should be as small as possible so that the number of changes (i.e. delta model) is low, reducing the number of conflicts. Farias *et al.* [18] also point out the need to have a low amount of changes between design models to be merged so that design models can keep stable. Exposing participants to large models could influence the results obtained. Instead, we restrict the use of small models.

Part 3: TAM questionnaire: The third part addressed questions about the usability and acceptance of the technique. This part aims at exploring the RQ3. For this, this part of our questionnaire is based on the widely known technology acceptance model (TAM) [26]. This part had nine questions, which were answered using a Likert Scale, including Totally Agree, Partially Agree, Neutral, Partially Disagree, and Totally Disagree. The formulated questions (Q) dealt with various topics, including the perceived ease of use (Q1–3), perceived usefulness (Q4–6), attitude towards used (Q7 and Q8), and behavioural intention to use (Q9).

4.3 Context and participant selection

The context selection is representative of realistic scenarios in which software developers need to merge UML models to produce a new release. Six scenarios were considered in our study, which are briefly described in Table 1. Each participant was asked to express each evolution scenario using UML2Merge. These scenarios are typical cases where our participants would not be their designers, but need to evolve them by running merge tasks.

Our experiment was conducted with 10 participants, being one student and nine professionals from Brazilian companies and one student with professional experience. The participants were recruited based on convenience [24]. An e-mail was sent to a set of undergraduate and graduate students at University of Vale do Rio dos Sinos, who with experience with software development and modelling. Some participants held a Master's degree and Bachelor's degree (or equivalent) and had a considerable knowledge of software modelling and programming. We chose participants, including students, so that we could have participants with different profiles and levels of expertise. All participants were from a postgraduate program in Applied Computing at the University of Vale do Sinos, Brazil. The experiment was performed similarly to a practical laboratory exercise. Each participant received the same training on the proposed technique and experimental procedures. We discuss the experimental process in the following section.

4.4 Experimental process

Fig. 9 presents the experimental process, which is formed by a set of activities grouped into three phases discussed as follows:

Phase 1 has two activities. The first, *training*, the participants received training on both the UML2Merge and the experimental procedures. A pre-test was also run in which participants performed an activity similar to that found in the experiment. This allowed us to evaluate whether the participants had actually understood the proposed technique, as well as helped to level the knowledge of our sample. The second activity, *collect demographic data*, the participants answered a list of questions (input) so that we could collect their characteristics and opinions about the UML2Merge. The demographic data collected (output) is the result of this activity.

Phase 2 has only one activity, *apply UML2Merge*. Participants were asked to answer six questions. To be correctly answered, each question required the complete understanding of UML2Merge. Participants received the desired model (input) to be produced. This model should be produced from the application of evolution descriptions (input) to be applied in a base model. At the end of this phase, six merge relationships using UML2Merge (output) were generated. We calculated the invested effort (output) to answer each question, and the correctness (output) of the expressed merge relationship. Effort was computed based on the number of minutes to answer each question, while correctness was calculated based on the number of correctly answered question per question.

Phase 3 focused on the application of the TAM questionnaire (input). Participants received a list of questions inquiring about the perception of ease of use, utility perception, attitude, and intent of behaviour, regarding the UML2Merge. Qualitative data (output) were generated, regarding UML2Merge usability and acceptance from the perspectives of software developers. Participants individually performed all activities (in Phases 1–3) to avoid any threat to the experimental process. We discuss the collected data in the following section.

4.5 Analysis of results

Profile data of the participants: Table 2 describes the profile data, reporting the participants' characteristics and opinion. These data were collected from 17 September to 19 October 2018. In total, we had 10 Brazilian participants. Our participants were *aged* between 20 and 39 years. Considering *education*, the majority (70%) had a complete or incomplete postgraduate degree. Only one participant did not take an undergraduate course in computing, but rather in mathematics, subsequently pursuing a master's degree in applied computing. Most participants (90%) took an *undergraduate course* in computing, only one graduated in mathematics but held a master's degree in Applied Computing. Almost all participants (90%) have 5 years (or more) of education at university. Professional experiences of 5 years (or more) with software development and modelling registered 70 and 40%, respectively. Over 40% indicated having 5 years (or more) of professional experience with software development and modelling. A total of 60% of the participants have 2 years (or more) of experience with this activity. All the participants agreed that expressing merge could minimise conflict problems. Therefore, we believe that our sample is small but adequate to carry out an initial evaluation of the proposed approach.

Table 2 Profile data of the participants

Characteristic and opinion (*n* = 10)	Answer	#	%
age	< 20 years	0	0.0
	20–29 years	6	60.0
	30–39 years	4	40.0
	> 39 years	0	0.0
education	high school	1	10.0
	undergraduate*	2	20.0
	Master*	6	60.0
	PhD*	1	10.0
	Others	0	0.0
undergraduate course	information systems	1	10.0
	computer science	4	40.0
	computer engineering	0	0.0
	system analysis	1	10.0
	Others	4	40.0
education at university	< 2 years	1	10.0
	2–4 years	0	0.0
	5–6 years	3	30.0
	7–8 years	4	40.0
	> 8 years	2	20.0
professional experience with software development	< 2 years	2	20.0
	2–4 years	1	10.0
	5–6 years	1	10.0
	7–8 years	1	10.0
	> 8 years	5	50.0
professional experience with software modelling	< 2 years	3	30.0
	2–4 years	3	30.0
	5–6 years	1	10.0
	7–8 years	0	0.0
	> 8 years	3	30.0
merge experience	< 2 years	4	40.0
	2–4 years	3	30.0
	5–6 years	0	0.0
	7–8 years	1	10.0
	> 8 years	2	20.0
expressing integration could minimise conflict problems	totally agree	6	60.0
	partially agree	4	40.0
	neutral	0	0.0
	partially disagree	0	0.0
	totally disagree	0	0.0

Table 3 Collected data concerning effort (RQ1) and correctness (RQ2)

Question	RQ1: effort, min		RQ2: Correctness	
	General	Mean	Correct answers	Percentage, %
1	25	2,7	9	90
2	46	5.4	9	90
3	31	3.1	10	100
4	28	2.8	10	100
5	42	4.2	8	80
6	18	1.8	7	70

RQ1: Effort to express merge relationship. Table 3 presents the collected data related to the formulated RQ. We begin our analysis by exploring the invested effort (RQ1) to express a merge relationship using UML2Merge. The collected data indicate that, the participants invested a low effort in this regard. On average, the participants invested 3 min to answer each question of the experimental evaluation. Question 2 required the most effort (by about 5.5 min), while Question 06 required the least effort (by 1.8 min). This may mean that the proposed notation required little cognitive processing by allowing participants to make a decision (choosing a questionnaire option) in a short amount of time. So, our data suggest that, the effort of expressing merge relationship using the UML2Merge is low. Note that the use of the proposed approach could be questionable in practice, if the low required effort were accompanied by a high number of incorrect answers.

RQ2: Correctness of the merge relationship. The next step was to evaluate whether the participants expressed the merge relationships correctly. Table 3 presents the number of correct answers per question and percentage. The high number of correctly produced relationships means that the UML2Merge was proper. In the worst case (Question 6), 70% of participants expressed the relationship correctly. The high number of correct answers suggests using the UML2Merge there is a high likelihood to express merge relationship of UML models properly. These results might be harmed, if the participants answered each question randomly. However, this threat could be tamed by formulating and analysing the results of our control question. In Table 3, we can also observe that, all participants answered the control question correctly (Question 4). This implies that they understood the experimental activities as well as were attentive. Therefore, our data also suggest that the UML2Merge also contributed to the expression of correct merge relationships.

RQ3: Acceptance of the proposed technique. By using TAM questionnaire, we seek to evaluate the perceived ease of use, perceived usefulness, attitude towards use, and behavioural intention to use, with regards to the UML2Merge. Table 4 displays the obtained data. Our obtained data show that, no one disagreed that UML2Merge is easy to use, learn, and master. On the contrary, 70% (or more) of the participants think that UML2Merge is easy to use (70% totally agree and 30% partially agree), learn (80% totally agree, 10% partially agree and 10% neutral) and master (70% totally agree, 20% partially agree and 10% neutral). The results are even better considering the perceived usefulness. All participants realised that UML2Merge makes it easier to merge UML models (90% totally agree and 10% partially agree), increase productivity (70% totally agree and 30% partially agree), and improve understanding of merge relationships (50% totally agree and 50% partially agree). In particular, we would like to highlight that 90% of them have fully agreed that UML2Merge makes it easy to express merge relationship. Considering the attitude towards use, the participants believe that using UML2Merge is a good idea (90% totally agree and 10% partially agree), as well as are confident about it (50% totally agree and 50% partially agree). Likewise, participants also agree (80%) that they intend to use the approach to express merge relationship (40% totally agree, 40% partially agree, and 20% neutral). These findings show the potential for acceptance by people with profiles similar to those of the participants. The results are encouraging and show the potential of using the proposed approach in a real environment.

Additional analysis. Previous studies [27] highlight that the developers experience can influence the comprehension of design

Table 4 Collected data related to TAM questionnaire

	Totally agree	Partially agree	Neutral	Partially disagree	Totally disagree
Perceived ease of use					
I found the UML2Merge extension easy to use	7	3	0	0	0
I found the UML2Merge extension easy to learn	8	1	1	0	0
I found the UML2Merge extension easy to master	7	2	1	0	0
Perceived usefulness					
UML2Merge would facilitate the merge of UML models.	9	1	0	0	0
using UML2Merge would help increase productivity.	7	3	0	0	0
the use of UML2Merge would provide a better understanding about merge of UML models.	5	5	0	0	0
Attitude towards use					
using UML2Merge is a good idea.	9	1	0	0	0
I am confident about UML2Merge.	5	5	0	0	0
Behavioural intention to use					
I plan to use UML2Merge to merge UML models.	4	4	2	0	0

Table 5 Analysis about the level of experience

Experience	RQ1: effort, min		RQ2: Correctness	
	General	Mean	Correct answers	Percentage, %
low experience (*n* = 4)	72	3	22	91.7
high experience (*n* = 6)	108	3	31	86.1

models. Thus, we seek to grasp whether the level of experience may impact on results produced by participants. The participants who had less than 7 years of experience in software development were classified as Low Experience ($n = 4$), while the remaining ones considered as High Experience ($n = 6$). The participants with high experience answered 36 questions, while those with low experience answered 24 questions. The collected data (in Table 5) indicate that participants with lower experience achieved a higher rate of correctness while investing the same amount of effort, compared with participants with higher experience.

5 Related work

This section compares the proposed extension with some related works. For this, Section 5.1 analyses some related works. Section 5.2 introduces a comparative analysis of the related works. Section 5.3 presents some limitations of our study.

5.1 Analysis of the related works

We surveyed this literature to identify similar works. For this, we used the IEEE Xplore and Google Scholar. In total, six papers were chosen for convenience.

Sharbaf & Zamani [28]. This study introduces a UML profile for modelling conditions in which conflict problems can emerge. The paper presents a set of conflict elements, used to supply descriptions concerning when the contradicting UML models appear. These conditions are used to produce constraints automatically. Papyrus tool was used to support the proposed UML profile. In addition, the usefulness of this profile was evaluated through a case study, in which two conflicts were created. Although the usefulness has been evaluated, the evaluation cases are simple, and few details about the experimental setup are provided. The proposed approach was not evaluated by potential users, neglecting the provision of data related to the perceived ease of use and usefulness, attitude towards use, and degree of acceptance as a whole.

Mansoora et al. [29]. This paper presents a multi-objective formulation of the model-merging problem so that 'the best trade-off between minimising the number of omitted operations and maximising the number of successfully applied important operations' can be found. They evaluated the proposed approach using seven open source systems and compared it with different existing model-merging approaches. This work did not focus on the expression of merge relationship, and did not carry out empirical studies with potential users on this topic.

Dam et al. [30]. It reports that the current merging techniques are limited to textual artefacts, and they tend to be unable to discover and resolve complex merging issues beyond simple conflicts. This study introduces a novel approach for merging versions of models, which addresses emerging conflicts and inconsistencies throughout the merging process. The proposed approach was validated through case studies using industrial design models, so that its scalability might be properly evaluated. Moreover, this study aimed at exploring conflicts, arbitrary syntactic, and semantic consistency issues. The results, collected from an extensive empirical evaluation, suggest that the approach can scale to practical settings. In addition, it is important to highlight that it is not concerned with expressing a merge relationship, but rather with attacking more complex problems, such as the automatic merging of consistent models.

Brun & Pierantonio [31]. This study highlights the need to record the design-level structural changes that modern software systems can typically undergo throughout its life cycle. In this sense, they introduced EMF Compare, which is a metamodel-independent approach to model differencing. This approach, based on similarity techniques, was fully implemented and incorporated into the Eclipse platform, and aimed at 2-way comparisons approach. Some important contrasts were observed in relation to our work. First, its main contribution is not a UML extension to express model merge, but a comparison technique. Secondly, no experimental study was conducted to analyse the approach in practice from the perspective of software developers.

Kolovos et al. [32]. It puts the spotlight on the problem of UML model merging. In particular, they explored requirements for supporting model merging, and introduced the EML. By using EML, software developers can combine models of diverse metamodels and technologies through a rule-based language with tool support. The work understands that model merging can be seen as an activity composed of four distinct phases, such as comparison, conformance checking, merging, and reconciliation (or restructuring). Similar to our study, this study also addresses, but at the code level, the theme of comparison, merge and transformation of models. We believe that the relationships created with UML2Merge could be implemented using the rules found in EML. Although the work has proposed a language to support the model merge, EML does not address the problem of expressing merge relationships at the level of models.

Mens [25]. It highlights that textual software merge 'an essential aspect of the maintenance and evolution of large-scale software systems.' Thus, he presents a comprehensive survey and analyses the available textual merge approaches. The author also points out that several merge techniques has been proposed purely based on

Table 6 Comparative analysis of the related works

Related work	Comparison criteria							
	C1	C2	C3	C4	C5	C6	C7	C8
UML2Merge	●	●	●	●	●	●	●	○
Sharbaf & Zamani [28]	●	◐	●	○	○	○	◐	⊘
Mansoora *et al.* [29]	○	◐	●	●	○	○	○	◐
Dam *et al.* [30]	○	●	●	○	○	○	○	●
Brun & Pierantonio [31]	○	◐	○	○	○	○	○	●
Kolovos *et al.* [32]	○	◐	●	○	○	○	○	●
Mens [25]	○	◐	◐	○	○	○	⊘	⊘

● Meets Fully, ○ Does not meet.

◐ Meets partially, ⊘ Not Applicable .

textual merging, while ones that are more powerful take the syntax and semantics of the software into account. In addition, Mens has reported that there is a tendency in developing operation-based textual merging due to high expressiveness when compared with others, e.g. state-based or change-based merging. Unlike our study, the study does not propose a UML extension, and does not perform empirical studies to evaluate the merging expression of UML models.

5.2 Comparative analysis of the works

This section contrasts the proposed extension with the previously analysed studies. This comparison, based on comparison criteria (C), serves to identify some similarities and differences. The comparison criteria are presented below:

i. *Main contribution (C1)*: Studies that have, as main contribution a UML extension for expressing merge relationships, or approaching research topics related to merge relationships.
ii. *Proposed approach (C2)*: Studies that introduce a new approach that deal with research topics concerning comparison, merge, and/or transformation of models. In addition, we also analyse whether the studies are able to establish the order of the merge relationships.
iii. *Experimental study (C3)*: Studies that evaluate the proposed approach through empirical studies, including case study, controlled experiment, quasi-experiment, or survey.
iv. *Context (C4)*: Studies that were performed with industry professionals or used real-world artefacts in academia.
v. *Participant profile (C5)*: Studies that collected data of the participants to screen and characterise their profile.
vi. *Study variables (C6)*: Studies that analyse the effort to merge UML models, the correctness of merge relationships and the acceptance of the proposed technique.
vii. *User acceptation (C7)*: Studies that evaluate the proposed approach using the TAM questionnaire, so that the perceived ease of use, perceived usefulness, attitude towards use, and behavioural intention to use can be measured.
viii. *Available tool support (C8)*: Studies that have available tool support. Having a support tool is fundamental to make feasible the use of the proposed technique not only in academia, but also in industry.

Table 6 presents the comparison considering these criteria. We emphasise, that the UML2Merge was the one that met most of the criteria (C1–8), highlighting its contributions and limitations.

5.3 Limitations of our study

This work proposes a UML extension by adding new constructors to the UML metamodel. Although the approach has had a good acceptance of the participants, it still suffers from some limitations. The first would be the lack of a domain-specific modelling environment that supports the proposed extension (i.e. tool support). This environment allowed us to use a UML modelling environment with the constructors proposed by the UML2Merge. This environment might be created using the Graphical Modelling Framework (GMF), which is a model-driven development approach to the elaboration of graphical editors. GMF can generate a ready-to-use graphical editor in the Eclipse platform. The second limitation would be related to the way of representing the rules, including matching, merging, and transformation ones. They can be represented through different languages, including OCL [23], Epsilon, among others. However, no language has been created so far. The third limitation is related to empirical evaluation. Although the current evaluation has been well elaborated and the collected results are favourable, new empirical studies are needed.

6 Conclusions and future work

This paper proposed the UML2Merge, which is an extension for expressing UML merge relationship. An empirical study with 10 participants was run to grasp its effects on the merge effort, the correctness of merge relationships, and the participants' acceptance. On average, the participants invested 3 min to indicate the merge relationship in our experiment tasks. At least, 70% of participants expressed the merge relationship correctly. None of the participants disagreed with the usefulness of the UML2Merge; on the contrary, most of them (>70%) agreed on ease of use and usefulness. Still, all participants demonstrated an attitude towards use. The collected data suggest that the UML2Merge is proper to express merge relationships by requiring a low effort, allowing high correctness of merge relationships, and having a high acceptance of the participants. The results are encouraging and show the potential for using UML2Merge to express the evolution of UML models through merge relationship.

As future work, we plan to investigate two RQ. First, how can the proposed UML extension be used by merge techniques as a merge specification language? Secondly, do developers invest less effort to merge real-world UML models using the UML2Merge? In addition, we will perform empirical studies to understand to what extent, we can use it to support the evolution of industrial design models. Moreover, our evaluation focused on the usability and understanding of the UML2Merge. Upcoming evaluations can extend it by proposing new experimental tasks in which merge relationships are manually specified by the stakeholder, and evaluating its impact on affective states [33]. New findings can open doors to quantifying interactions of stakeholders while expressing merge relationships.'

Finally, we hope that this work represents a first step in a more ambitious agenda on better supporting the representation of merge relationships at modelling time. We also hope that the issues outlined throughout the paper encourage other researchers to extend our study with different matching and merge strategies.

7 References

[1] Chaudron, M., Heijstek, W., Nugroho, A.: 'How effective is UML modeling?', *Softw. Syst. Model.*, 2012, **4**, (11), pp. 571–580

[2] Ho-Quang, T., Hebig, R., Robles, G., *et al.*: 'Practices and perceptions of UML use in open source projects'. 39th Int. Conf. on Software Engineering: Software Engineering in Practice Track, 2017, pp. 203–212

[3] OMG: 'UML: Infrastructure specification', Version 2.5.1, https://www.omg.org/spec/UML/2.5.1/, 2018

[4] Jolak, R., Vesin, B., Chaudron, M.R.: 'Octo U.M.L: an environment for exploratory and collaborative software design'. 39th Int. Conf. on Software Engineering Companion Proc., 2017, Vol. 17

[5] Menezes, G., Murta, L.G.P., Barros, M.O., *et al.*: 'On the nature of merge conflicts: A study of 2,731 open source Java projects hosted by GitHub', *IEEE Trans. Softw. Eng.*, 2018, doi: 10.1109/TSE.2018.2871083

[6] Marussy, K., Semerath, O., Varro, D.: 'Incremental view model synchronization using partial models'. 21th MODELS, 2018, pp. 323–333

[7] Bruneliere, H., de Kerchove, F.M., Daniel, G., *et al.*: 'Towards scalable model views on heterogeneous model resources'. 21th MODELS, 2018, pp. 334–344

[8] Bischoff, V., Farias, K., Gonçales, L.J., *et al.*: 'Integration of feature models: a systematic mapping study', *Inf. Softw. Technol.*, 2019, **105**, pp. 209–225

[9] Dingel, J., Diskin, Z., Zito, A.: 'Understanding and improving UML package merge', *Softw. Syst. Model.*, 2008, **7**, (4), pp. 443–467

[10] Knapp, A., Mossakowski, T.: 'Multi-view consistency in UML: A survey'. Graph Transformation, Specifications, and Nets, 2018, pp. 37–60

[11] Guessi, M., Oliveira, L., Nakagawa, E.: 'Extensions of UML to model aspect-oriented software systems', *Clei Electron. J.*, 2011, **14**, (15), pp. 1–18

[12] Jinwala, D., Phalnikar, R.: 'Optimal web service selection using UML profile', *GSTF J. Comput. (JoC)*, 2018, **2**, (1), pp. 244–249

[13] Robles-Ramirez, D., Escamilla-Ambrosio, P., Tryfonas, T.: 'Iotsec: UML extension for internet of things systems security modelling'. Int. Conf. on Mechatronics, Electronics and Automotive Engineering, 2017, pp. 151–156

[14] Rubin, J., Rinard, M.: 'The challenges of staying together while moving fast: an exploratory study'. 38th Int. Conf. on Software Engineering, 2016, pp. 1–12

[15] Rubin, J., Chechik, M.: 'N-way model merging'. 9th Joint Meeting on Foundations of Software Engineering, 2013, pp. 301–311

[16] Martin, R.: '*Clean architecture: a craftsman's guide to software structure and design*' (Prentice Hall Press, Upper Saddle River, NJ, USA, 2017)

[17] Bang, J., Brun, Y., Medvidovic, N.: 'Continuous analysis of collaborative design'. Int. Conf. on Software Architecture, 2017, pp. 97–106

[18] Farias, K., Garcia, A., Lucena, C.: 'Effects of stability on model composition effort: an exploratory study', *Softw. Syst. Model.*, 2014, **13**, (4), pp. 1473–1494

[19] Kolovos, D., Rose, L., Garcia-Dominguez, A., *et al.*: 'The epsilon book', https://www.eclipse.org/epsilon/doc/book/, 2018

[20] Farias, K.: 'Empirical Evaluation of Effort on Composing Design Models', PhD Thesis, Department of Informatics, PUC-Rio, Rio de Janeiro, RJ, Brazil, 2012

[21] Ermel, G., Farias, K., Goncales, L.J., *et al.*: 'Supporting the composition of UML component diagrams'. XIV Brazilian Symp. on Information Systems, Caxias do Sul, Brazil, June 2018, pp. 441–449

[22] Farias, K., Garcia, A., Whittle, J., *et al.*: 'Evaluating the effort of composing design models: a controlled experiment', *Softw. Syst. Model.*, 2015, **14**, (4), pp. 1349–1365

[23] OCL.: 'The object constraint language specification', Version 2.4, https://www.omg.org/spec/OCL/2.4/, 2014

[24] Wohlin, C., Runeson, P., Host, M., *et al.*: '*Experimentation in software engineering*' (Springer, Heidelberg, Germany, 2012)

[25] Mens, T.: 'A state-of-the-art survey on software merging', *IEEE Trans. Softw. Eng.*, 2002, **28**, (2), pp. 449–462

[26] Marangunic, N., Granic, A.: 'Technology acceptance model: a literature review from 1986 to 2013', *Univers. Access. Inf. Soc.*, 2015, **14**, (1), pp. 81–95

[27] Filippo, R., Penta, M.D., Torchiano, M., *et al.*: 'How developers' experience and ability influence web application comprehension tasks supported by UML stereotypes: a series of four experiments', *IEEE Trans. Softw. Eng.*, 2010, **36**, (1), pp. 96–118

[28] Sharbaf, M., Zamani, B.: 'A UML profile for modeling the conflicts in model merging'. 4th Int. Conf. on Knowledge-Based Engineering and Innovation, 2017, pp. 197–202

[29] Mansoora, U., Kessentinia, M., Langerb, P., *et al.*: 'MOMM: multi-objective model merging', *J. Syst. Softw.*, 2015, **103**, pp. 423–439

[30] Dam, H., Egyedb, A., Winikoffc, M., *et al.*: 'Consistent merging of model versions', *J. Syst. Softw.*, 2015, **112**, pp. 137–155

[31] Brun, C., Pierantonio, A.: 'Model differences in the eclipse modeling framework', *Europ. J. Inf. Prof.*, 2008, **9**, (2), pp. 29–34

[32] Kolovos, D.S., Paige, R.F., Polack, F.A.: 'Merging models with the epsilon merging language (EML)'. MODELS, 2006, pp. 215–229

[33] Manica, M., Farias, K., Gonçales, L.J., *et al.*: 'Effects of model composition techniques on effort and affective states: a controlled experiment'. 30th Int. Conf. on Software Engineering and Knowledge Engineering, 2018, pp. 304–307