Towards a Formalism for Specifying N-way Model Merging Rules

Mohammad-Sajad Kasaei, Mohammadreza Sharbaf, Bahman Zamani
MDSE Research Group, University of Isfahan, Isfahan, Iran
{kasaei, m.sharbaf, zamani}@eng.ui.ac.ir

***Abstract*— Integrating multiple versions of a model into a unified yet consistent model, which is called N-way merging, is a key challenge in collaborative modeling. Recently, several approaches have been proposed that construct a merged model by incorporating all elements of parallel versions. Despite these efforts, techniques for the high-level specification of N-way merging have hardly been addressed, and research on proposing a formalism for this problem is scarce. Such a formalism can be used to better understand and analyze the behavior of merging real-world systems. This paper presents a formalism to facilitate the specification of the logic behind merging N concurrent versions. The proposed formalism can be used in different merging scenarios to help domain experts analyze their integration requirements more precisely. We introduced three types of merging rules to empower our formalism for specifying all possible scenarios in merging N model versions. We also provide a proof-of-concept implementation in which the N-way merging formalism is equipped with a syntax-aware editor and a parser to promote N-way merging rules for EMF-based models. We conducted a case study to evaluate the applicability of our formalism via a real-world versioning scenario. The results are promising for specifying merging rules in various situations.**

Keywords— Model Merging, N-way Merging, Specification Language, Formal Language Definitions.

I. Introduction

Model-Driven Engineering (MDE) aims to tackle the complexity of software systems [1]. MDE uses models as firstclass artifacts in the process of developing software systems. Models allow developers to describe and share a potential solution at a high level of abstraction by providing a simple view of the system [2]. Like the traditional code-centric development approaches, MDE requires collaboration between different experts. That means modeling a system involves several developers organized in teams. Each team may apply some change operations and create different versions of the same model. Managing different versions of an artifact can be challenging, particularly when different teams apply incompatible modifications to the same model. In this situation, model management techniques, such as model merging and conflict management, are not an option but a necessity to provide an integrated yet consistent merged version [3].

Model merging is defined as a systematic process to manipulate model versions and prepare an integrated model. The process includes the following phases: comparison, conformance checking, merging, and reconciliation [4], [5]. Existing approaches for model merging mainly concentrate on pairwise comparison and two-way merging of models. In this context, Epsilon Merging Language (EML) is a well-known rule-based language, which is built on top of the Epsilon platform to merge two arbitrary EMF models using a pairwise mechanism [6]. However, increasing the number of developers who work in parallel on the same model leads to N concurrent versions that must be merged simultaneously to create a compatible and integrated model. Recent research has shown that pairwise merging is not an appropriate technique to integrate N models because it increases the cost of integrating and maintaining models in the software lifecycle [3], [7]. Pairwise merging may miss some changes that result in an invalid merged model [8]. N-way merging is introduced as a solution, which concurrently considers N models in the merge process [7]. However, there are also important specifying the integration logic for determining the target model elements. But specifying integration rules for N-way model merging is a challenging task, which still needs to consider.

This paper presents a formalism for specifying N-way model merging rules to solve the aforementioned issue. To this end, we first focus on the merging phase of the merge process and introduce different merging mechanisms for integrating N concurrent model versions. Then, we investigate all possible scenarios for integrating matched or unmatched elements. Finally, we outline N-way merging formalism according to the three different types of merging rules required for the concurrent integration of N models. Our proof-of-concept implementation provides a parser and a syntax-aware editor for merging rules specifications based on the proposed formalism. We also performed a case study that illustrates the applicability of our formalism using a real-world versioning scenario.

The remainder of this paper is organized as follows. Related work is discussed in Section II before briefly providing the background information on the model merging process, merging strategies, and N-model integration strategies in Section III. Section IV presents N-way merging mechanisms, then outlines N-way merging formalism. Section V introduces the case study to demonstrate the applicability of our formalism. Section VI presents the implementation details, and Section VII concludes the paper and outlines future work.

II. Related Work

There exist a few efforts to support N-way model merging. Most existing approaches introduced a method to merge models. Kolovos et al. [6] introduced the Epsilon Merging Language (EML), a rule-based language to support model merging based on the two-way merging strategy. We also found an approach that introduced a configurable three-way model merging [9]. However, none of those approaches proposed a language or formalism to specify N-model integrations. In this section, we consider all approaches that have been proposed for merging N concurrent models and the most important works that introduced a formalism in the model merging area.

Rubin and Chechik [7] and Schultheiß et al. [10] proposed the NwM and RaQun algorithm for N-way merging. NwM creates a collection of tuples from N models and picks out

tuples with the maximal weight to create the merged model. Schultheiß et al. [10] proposed an N-way model matching algorithm, called the RaQun, which uses multi-dimensional search trees for efficiently finding suitable match candidates through range queries. Mansoor et al. [11] introduced a multi-objective model merging approach based on the NSGA-II algorithm. They identify the change operations applied in parallel and decide which operation has to be omitted and applied on the original model to construct a valid merged model. Reuling et al. [8] proposed a precise N-way model merging approach by improving an arbitrarily imprecise matching. They create a tentative merged model based on the input versions and an initial matching list. Then incrementally apply model refactoring operators to identify and unify further similarities among unmatched elements to construct the merged model.

Nejati [12] outlined a formal approach to merging and negotiation over behavioral models. Westfechtel [13] presented a formal approach for two-way and three-way merging of EMF models. However, both do not support N-way merging. Moreover, Sharbaf et al. [14] introduced a formalism to represent model merging conflicts, which does not focus on specifying merging rules.

III. Background

This section provides the principles of model merging, which inspired our work and help the comprehension of the N-way merging approach. More specifically, we first introduce the foundation of the model merging process in Section III.A. Then, we define basic strategies for merging two versions of the same model in Section III.B. Finally, in Section III.C, we discuss two main strategies that can be used in dealing with the integration of N concurrent models to provide a comprehensive guide for the rest of this paper.

A. Model Merging Process

Different versions of a model must be merged to create a unique and consistent model for the evolution of the software system. The model merging process can be decomposed into four distinct and complementary phases, including comparison, reconciliation, merging, and conformance checking [4], [5]. The comparison phase tries to identify equivalent or similar elements between model versions. The reconciliation phase is adopted to handle conflicts between contradicting modifications. The merge phase specifies the elements that must appear in the tentative merged version. Finally, the conformance checking phase is used to examine modeling constraints to ensure the correct final model is created.

B. Merging Strategies

There are three general strategies to integrate two different versions of a model at the same time [15]. In the following, we describe each strategy.

In *Raw Merging strategy*, the merged version will be built by applying a sequence of all change operations that both designers performed to modify the original version, as long as there are no desired change operations left. Based on this strategy, first, all change operations that are performed by the first designer are applied to the original version to create a tentative merged model. Then, all change operations performed by the second designer are applied to the tentative merged model to make the final merged version.

Two-way Merging strategy starts by comparing input models to identify corresponding elements based on the similarity criteria without considering the original model. Then based on the comparison result, the merge process adds relevant elements of both models to the merged version so that the duplicated elements do not appear in the final merged model. Two-way is not able to identify the type of modifications applied to the original model, which may lead to conflict and an inconsistent merge model.

In *Three-way Merging strategy*, the common ancestor version is involved in the merge process that helps precise identification of similar elements as well as the type of modifications. Three-way merging starts with two comparisons are conducted between new versions and their common ancestor model, which determines a list of corresponding elements. Given that, three-way merging attempts to pick up elements of each version that should be appeared on the merged model by resolving contradicting modifications (e.g., [11]). Three-way merging is more powerful than two-way merging since it provides the proper support to handle conflicts and automate the merging process by consulting the common ancestor.

C. N-Model Integration Strategies

Nowadays, collaborative modeling has become more pervasive in software development [16]. That leads to the need for fast and low-cost integration approaches resulting in a correct merged model from N concurrent versions. Studies have shown that *subset-based* and *concurrent processing* are two different strategies to integrate N models [7].

The *subset-based* integration processing is a solution for merging the arbitrary number of model versions using customized subsets of models, which are combined in a pairwise manner with a specific order to achieve an integrated model. In this strategy, all input models are grouped according to the integration algorithm or user opinion and merged together to achieve the final result. Moreover, any integration method to simultaneously merge N model versions follows the *concurrent processing* strategy. All model versions are concurrently integrated into the final merged model by adding common elements and appropriate variable parts of each version.

IV. The Proposed Approach

In this section, we present a formalism to specify merging rules for N-way merging. We first introduce different mechanisms for N-way merging. Then, we discuss possibly merging scenarios and propose N-way merging formalism based on that. The proposed formalism includes three different types of merge rules to cover all possible situations for integrating concurrent models. Each rule provides the ability to describe model elements that should be appeared in the integrated model during the merging of N concurrent model versions.

A. N-way Merging Mechanisms

As discussed in Section III, *Raw merging*, *Two-way merging*, and *Three-way merging* are three general strategies to integrate two versions of a model [15]. That would be possible to use all three merging strategies based on the *subset-based processing* to support N-way merging. To achieve this, we should apply several pairwise integration operations to produce the final merged model. However, the *subset-based processing* is too costly and error-prone, which

may arise conflicts and create an inconsistent merged model. In contrast, concurrent processing strategy is a more robust strategy since it attempts to consider all versions instead of inferring only two models. Therefore, we extend all three merging strategies for *concurrent processing* integration to support N-way merging.

Fig. 1 shows the proposed mechanisms to support the integration of N versions of the same model. We consider the starting situation of the models (V_0) has been checked-in by n users ($n \geqslant 2$). Users build their models in parallel by editing the original model. The original model is model V_0 which is resulted into $V_1...V_n$ versions through $C_1...C_n$ changes. In the following, we describe the N-way merging mechanism for each merging strategy.

1) Raw Strategy for N-way Merging

Raw merging strategy for N concurrent versions build the merged model by applying a sequence of change operations to the original model, same as raw merging for two models. However, the sequence should include all change operations made by the modeler one to modeler N, respectively. As shown in Fig. 1.a, the sequence $\{C_1...C_n\}$ is applied on the V_0 to produce version M due to the raw merging strategy for concurrent integrations of N versions of the original model.

2) Two-way Strategy for N-way Merging

Two-way strategy combines models without involving the original model. Hence, the merge mechanism should examine all elements of input versions to specify elements that should be appeared in the merged version. To this end, all input versions are considered together, and a comparison is performed on all versions to identify the list of corresponding elements. That list can be generated manually or automatically based on the different similarity criteria. Then, the merging phase starts to specify the elements that should be appeared in the merged model. In the merging phase, only one element is added to the integration model for all the corresponding elements in the matched list. The value for each property of an added element is determined in the merging rules based on the corresponding property value in other versions. Moreover, all elements without correspondence must be directly transferred from all versions to the merged model. As shown in Fig. 1.b, model $V_1...V_n$ are compared, and based on the results the model M is produced as the merging result.

3) Three-way Strategy for N-way Merging

The three-way merging strategy creates the merged model by considering the common ancestor. In this strategy for integrating N concurrent versions, all versions are separately compared with the common ancestor model. The comparison helps to identify unchanged elements as well as elements that are added, updated, and deleted in each new version. Then, the results of all comparisons are investigated to conclude similar and equivalent elements and reconcile inconsistencies. The inference on comparison results leads to a match table of corresponding elements between all versions, including more details about each element. In the next step, the merge engine examines each row of the match table to specify the rows that should be appeared in the merged model. Finally, for each selected row, only one element is added to the merged model, that the merge rules specify the value of its properties. Fig. 1.c shows an overview of integrating N concurrent versions of a common ancestor model using the three-way merging. Since the common ancestor is used in this strategy, managing conflicts is more precise, making it possible to create a consistent merged model.

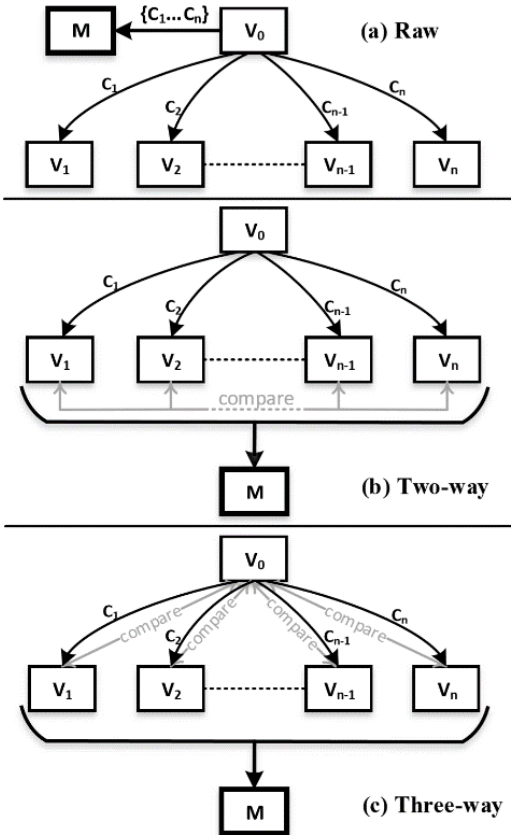

Fig. 1. N-way Merging Mechanisms

B. *N-way Merging Formalism*

In this section, we specify the N-way merging formalism using a set of rules inspired by Rouhi and Zamani specification scheme [17]. These rules are indicated from Rule (1) to Rule (16), which help users to describe formal terms and conditions of model elements integration. The proposed notation satisfies the *Reachability* and *Completeness* conditions by reachability of non-terminal symbol from the *NWayMergingModule* root. There exists one and only one description rule for each non-terminal symbol. Hence, the N-way merging formalism is a well-formed syntax [18].

Possible scenarios for integrating concurrent models can be determined by considering the situation of model elements during the merge process. In the N-way model merging, the situation of model elements in each version can be classified to *Core*, *Shared*, or *Unique*, as elaborated in the following [19].

- **Core** refers to elements that are common in all versions.
- **Shared** refers to elements that are common in two or more versions but not all of them.
- **Unique** refers to elements that exist only in one version.

In the proposed formalism, we introduce *Octopus*, *Ours*, and *Transfer* as three types of merging rules to support specifying possible integration scenarios. Each rule specifies the integration procedure for model elements that are compatible with the source parameters of that rule. The *Octopus* rules are proposed to specify merging details for elements of the *Core* category. The *Ours* rules can be used to describe the integration of elements in the *Shared* category. The *Transfer* rules are introduced to represent the details of integration for elements of the *Unique* category. In the following, we present each rule.

Rule (1) defines that a N-way merging module includes a *name*, the *list* of input models, the *Octopus* rules, the *Ours* rules, and finally, the *Transfer* rules. According to Rule (1), each module starts with the '*module*' keyword, followed by a name, models declarations, and different merging rules definitions that are surrounded by braces ('{' and '}'). In a module, the *inputModels* (Rule (2)) defines the list of model versions that must be integrated. Each input model consists of an '*import*' keyword, a string name (Rule (3)), and an *address* (Rule (4)), which is a URI to access the model.

NWayModule ::= 'module', **moduleName** *:String, '{',*
defineModel *:inputModels+,*
octopusRule *:octopusExp+,*
oursRule *:oursExp+,*
transferRule *:transferExp+,*
'}'

(1)

inputModels ::= 'import', modelName,' : ', address

(2)

modelName ::= name : String

(3)

address ::= ' " ', uri : String, ['/', id : String], ' " '*

(4)

Each octopus expression (Rule (5)) consists of the '*rule*' keyword, followed by a name, a '*mergeOctopus*' keyword, and several model element declarations that one should define before '*with*' keyword and others can specify after '*with*'. The rule followed by an optional '*withBase*' keyword, and the declaration of base model element. It also contains the '*into*' keyword, and a declaration of target model element. Each rule can be extended using the '*extends*' keyword, followed by the name of rule. The last part of octopus rule is the statements block that specifies the details of model element integration, surrounded by braces. In the body of an octopus expression, the *modelDefinition* (Rule (6)) defines the model elements that participate in the integration process. Each model definition consists of *nameParameter* (Rule (7)), which is a name, an *elementExp* (Rule (8)), which is a model name (Rule (3)) followed by an element type (Rule (9)).

Each ours expression (Rule (10)) consists of the '*mergeOurs*' keyword, and same as *octopus* expression started by a *rule* name, and input and target models definitions. But it after the base model declaration followed by the '*existsIn*' keyword, and an integer numberOfVersion to define the minimum number of versions that must contain the deleted element for applying the rule. This expression is followed by '*priority*' keyword, and declaration of the priority list (Rule (11)) to define the order of versions, which can be used in the statement block to populate the property values of the target model elements.

octopusExp ::= 'rule', **ruleName** *:String,*
'mergeOctopus', **model** *: modelDefinition+,*
'with', **model** *: modelDefinition+,*
*(',' ,*__model__ *: modelDefinition+)*,*
('withBase', **model** *: modelDefinition+)?,*
'into', **model** *: modelDefinition+,*
'extends', **ruleName** *:String, '{',*
statementBlock *:statement,*
'}'

(5)

modelDefinition ::= nameParameter, ' : ', elementExp

(6)

nameParameter ::= name : String

(7)

elementExp ::= modelName, '!', elementType

(8)

elementType ::= EDataType | EClass | 'P',Type | id : String

(9)

oursExp ::= 'rule', **ruleName** *:String,*
'mergeOurs', **model** *: modelDefinition+,*
'with', **model** *: modelDefinition+,*
*(',' ,*__model__ *: modelDefinition+)*,*
('withBase', **model** *: modelDefinition+,*
('existsIn', **numberOfVersion** *:Int)?)?,*
'priority', **list** *: priorityList+,*
'into', **model** *: modelDefinition+,*
'extends', **ruleName** *:String, '{',*
statementBlock *:statement,*
'}'

(10)

The *priorityList* (Rule (11)) defines the list of models parameter according to the input models. Each priority list consists of a string name (Rule (12)) and a *listParam* (Rule (13)), which is a queue of input model names to determine property values for the target elements based on the statement block.

Each transfer expression (Rule (14)) consists of the '*rule*' keyword, followed by the name of the rule, a '*transfer*' keyword, and the declarations of source model elements (Rule (15)). It is followed by a '*from*' keyword that specifies the list of source models. It also defines the target model element after a '*to*' keyword, and like the *Octopus* and *Ours* exprssions, it can be finished by an optional '*extends*' keyword and a

statement block that expresses assignments and relationships between elements.

The *sourceModel* (Rule (15)) defines the elements of input models that should participate in the integration process. Each source model consists of a string name (Rule (7)) and an *sourceExp* (Rule (16)), which is a string name (Rule (3)) and element type (Rule (9)) for model elements retrieval.

$$priorityList ::= listName, \text{`:'}, listParam \quad (11)$$

$$listName ::= name : String \quad (12)$$

$$listParam ::= nameParameter, (\text{`,'}, nameParameter)^* \quad (13)$$

transferExp ::= 'rule', **ruleName** *:String,*
'transfer', **model** *: sourceModel+,*
'from', '(', **name** *: modelName+,*
(',', **name** *: modelName+)*, ')',*
'to', **model** *: modelDefinition+,*
'extends', **ruleName** *:String, '{',*
statementBlock *:statement,*
'}' (14)

$$sourceModel ::= nameParameter, \text{`:'}, sourceExp \quad (15)$$

$$sourceExp ::= name : String, \text{`!'}, elementType \quad (16)$$

V. Case Study

In this section, we evaluate the applicability of the proposed formalism by running a descriptive case study. Our evaluation aims at illustrating the conditions of integration expressiveness by examples. To this end, we chose a set of models inspired by the same model that indicate different types of merging, including *octopus*, *ours*, and *transfer*, presented in Section IV.B.

As an appropriate input for this case study, we need a real-world model with more than two concurrent versions. School management is a system to organize and manage educational programs for institute members. Fig. 2 illustrates the original version of the UML class diagram for a school management system and three derived versions, which are modified by modelers. The original model of this system contains one class and two attributes. The modifications that each modeler applies to the original model to create its version are indicated in Fig. 2 In the following, we sketch the specification of merging rules for *class* elements using the proposed formalism. According to Rule (1), the specification consists of five parts that are presented separately. We added next to each predicate a comment to show which rule is applied for each part.

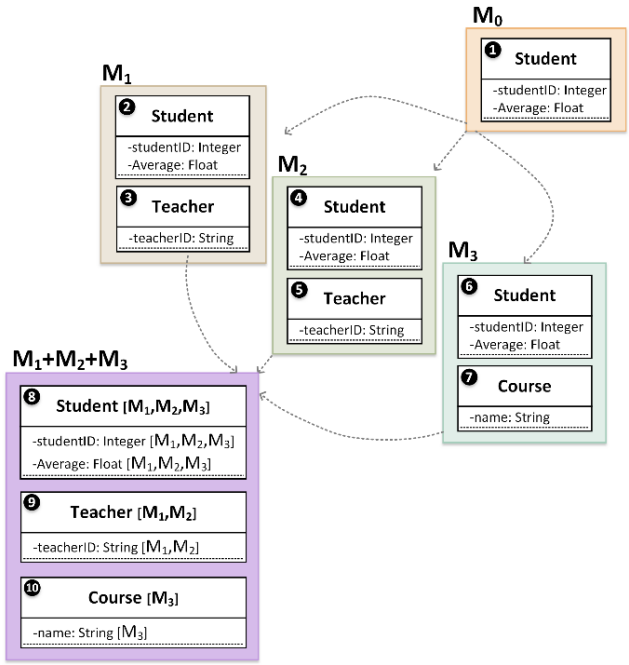

Fig. 2. N-way Merging Example for UML Class Diagrams

moduleName: *//Rule (1)*
module *CDNWayMergeRules* *//Rule (1)*
defineModel: *//Rule (1)*
import *M0 : 'Models/Base.uml'* *//Rules (2)-(4)*
import *M1 : 'Models/Sally.uml'* *//Rules (2)-(4)*
import *M2 : 'Models/Harry.uml'* *//Rules (2)-(4)*
import *M3 : 'Models/Alice.uml'* *//Rules (2)-(4)*
import *M123 : 'Models/Merged.uml'* *//Rules (2)-(4)*
octopusRule: *//Rule (1)*
rule *CLOctopusA* *//Rule (5)*
mergeOctopus *V1 : M1!Class* *//Rules (5)-(9)*
with *V2 : M2!Class* *//Rules (5)-(9)*
, V3 : M3!Class *//Rules (6)-(9)*
withBase *V0 : M0!Class* *//Rules (5)-(9)*
into *Vt : M1+M2+M3!Class* *//Rules (5)-(9)*
vt.name = v1.name *//Rule (5)*
oursRule: *//Rule (1)*
rule *CLOursA* *//Rule (10)*
mergeOurs *V1 : M1!Class* *//Rules (6)-(10)*
with *V2 : M2!Class* *//Rules (6)-(10)*
, V3 : M3!Class *//Rules (6)-(9)*
priority *P1 : [V1, V2, V3]* *//Rules (7)-(13)*
into *Vt : M123!Class* *//Rules (6)-(10)*
vt.name = P1.name *//Rule (10)*
transferRule: *//Rule (1)*
rule *CLTransfer* *//Rule (14)*
transfer *Vs : Source!Class* *//Rules (9),(14)-(16)*
from *(M1, M2, M3))* *//Rules (3),(14)*
to *Vt : M123!Class* *//Rules (14),(6)-(9)*
vt.name = vs.name *//Rule (14)*

VI. Implementation

We have implemented a syntax-aware editor and a parser using Xtext [20] for our formalism in the Eclipse framework. Thus, users can specify merging rules for integrating concurrent versions of any EMF-based models using the N-way merging programs. The implementation is available from

GitHub[1] under the Apache 2.0 license. The editor facilitates writing merge rules using various features, such as keyword highlighting, error awareness, and smart terms completion. Therefore, users can specify conditions and the logic of integration for populating property value for target model elements, in a faster and simpler way using our formalism.

Fig. 3 shows an excerpt of the specified merging rules for the UML class diagram in the implemented editor. As displayed in Fig. *3*, the program is organized as a module, which defines five *imported* models and a number of *Octopus*, *Ours*, and *Transfer* rule blocks.

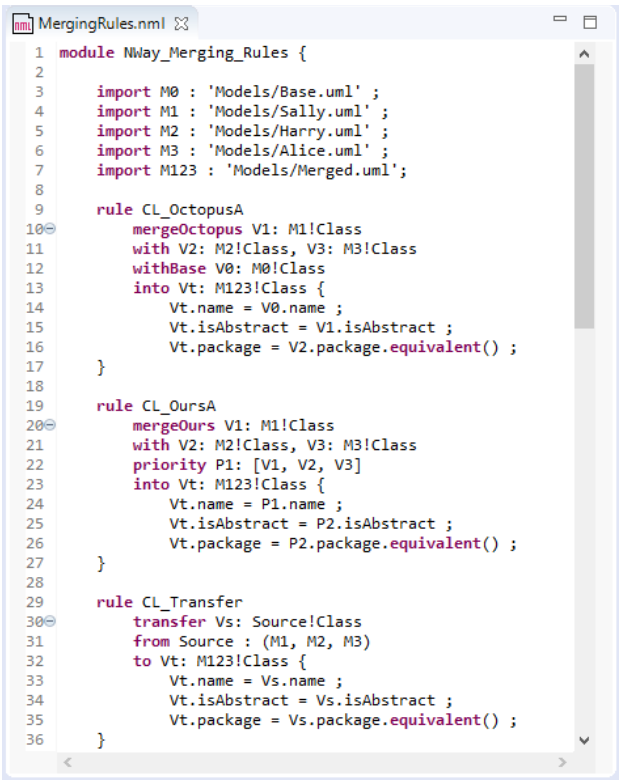

Fig. 3. Specification of Merging Rules for UML Class Diagrams in our Editor

VII. Conclusion and Future Work

This paper introduced a formalism for specifying the logic of integration and merging rules for N concurrent model versions based on the three-way strategy. We provided tool support, including a syntax-aware editor and a parser that allows users to write merging rules for EMF-based models. We assessed the applicability of NML via an experiment using a real-world modeling scenario. The results show that our formalism enables specifying integration rules for all possible scenarios in merging any number of input versions for the same model.

As future work, we aim to extend the current formalism and introduce a language for N-way model merging. Thereafter, we plan to implement the execution semantics of it by an N-way merging engine that can execute the N-way merging programs. We also intend to conduct a complete and comprehensive evaluation to investigate the integration of the heterogeneous models using our engine.

[1] https://github.com/MSharbaf/N-wayMergingEditor

References

[1] M. Brambilla, J. Cabot, and M. Wimmer. *Model-Driven Software Engineering in Practice: SecondEdition*. Morgan & Claypool, 2017.

[2] R. Pressman and B. Maxim. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education, 9th ed., 2019.

[3] S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, and P. Zave, "Matching and Merging of Variant Feature Specifications," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1355–1375, 2012.

[4] C. Batini, M. Lenzerini, and S. B. Navathe, "A Comparative Analysis of Methodologies for Database Schema Integration," *ACM Computing Surveys (CSUR)*, vol. 18, no. 4, pp. 323–364, 1986.

[5] R. A. Pottinger and P. A. Bernstein, "Merging models based on given correspondences," *Proceedings - 29th International Conference on Very Large Data Bases*, pp. 862–873, 2003.

[6] D. S. Kolovos, R. F. Paige, and F. A. Polack, "Merging models with the epsilon merging language (EML)," in MODELS, O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, Eds. LNCS, vol. 4199, Springer, 2006, pp. 215–229.

[7] J.Rubin and M.Chechik, "N-way model merging," in Proceedings of the 9th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2013. New York, USA: ACM Press, 2013, p. 301.

[8] D. Reuling, M. Lochau, and U. Kelter, "From Imprecise N-Way Model Matching to Precise N-Way Model Merging.," *The Journal of Object Technology*, vol. 18, no. 2, p. 8:1, 2019.

[9] M. Sharbaf and B. Zamani, "Configurable three-way model merging," *Software - Practice and Experience*, vol. 50, no. 8, pp. 1565–1599, Aug. 2020.

[10] A. Schultheiß, P. M. Bittner, L. Grunske, T. Th¨um, and T. Kehrer, "Scalable n-way model matching using multi-dimensional search trees," in 24th International Conference on Model Driven Engineering Languages and Systems (MODELS). IEEE, 2021, pp. 1–12.

[11] U. Mansoor, M. Kessentini, P. Langer, M. Wimmer, S. Bechikh, and K. Deb, "Momm: Multi-objective model merging," Journal of Systems and Software, vol. 103, pp. 423–439, 2015.

[12] S. Nejati, "Formal support for merging and negotiation," in Proceedings of the 20th international Conference on Automated software engineering, 2005, pp. 456–460.

[13] B. Westfechtel, "Merging of EMF models," *Software & Systems Modeling*, vol. 13, no. 2, pp. 757–788, May 2014.

[14] M. Sharbaf, B. Zamani, and G. Suny´e, "A formalism for specifying model merging conflicts," in Proceedings of the 12th System Analysis and Modelling Conference, 2020, pp. 1–10.

[15] K. Altmanninger, M. Seidl, and M. Wimmer, "A survey on model versioning approaches," *International Journal of Web Information Systems*, vol. 5, no. 3, pp. 271–304, Aug. 2009.

[16] M. Franzago, D. DiRuscio, I. Malavolta, and H. Muccini, "Collaborative model-driven software engineering: a classification framework and a research map," IEEE Transactions on Software Engineering, vol. 44, no. 12, pp. 1146–1175, 2017.

[17] A. Rouhi and B. Zamani, "Towards a formal model of patterns and pattern languages," *Information and Software Technology*, vol. 79, no. November, pp. 1–16, 2016.

[18] H. Zhu, "On the theoretical foundation of meta-modelling in graphically extended bnf and first order logic," in International Symposium on Theoretical Aspects of Software Engineering. IEEE, 2010, pp. 95–104.

[19] S. Duszynski, V. L. Tenev, and M. Becker, "N-way diff: Set-based comparison of software variants," in VISSOFT. IEEE, 2020, pp. 72–83.

[20] Bettini and Lorenzo, *Implementing domain-specific languages with Xtext and Xtend*. Packt Publishing Ltd, 2016.