JOT
JOURNAL OF
OBJECT TECHNOLOGY

Journal of Object Technology | **RESEARCH ARTICLE**

Conflict-based Change Awareness for Collaborative Model-driven Software Engineering

Edvin Herac[*], **Luciano Marchezan**[*], **Wesley K. G. Assunção**[†], **and Alexander Egyed**[*]
[*]Institute of Software Systems Engineering, Johannes Kepler University, Austria
[†]Department of Computer Science, North Carolina State University, USA

ABSTRACT

Collaborative Model-driven Software Engineering (CoMDSE) involves multiple engineers working together on modifying domain-specific models. This collaborative effort, however, can lead to conflicts during merges due to differing perspectives and goals among engineers. Solutions primarily focus on resolving conflicts during merges, but these approaches can be cumbersome and time-intensive. Thus, researchers also propose secondary strategies such as change awareness to notify engineers of each other's model changes mitigating conflicts before they happen. Change awareness strategies, however, often overwhelm engineers with a bulk of change notifications, many of which are irrelevant to their immediate tasks. Additionally, they are required to manually analyze these notifications for conflicts, which adds significant overhead, leading to decreased productivity and increased frustration. Thus, in this paper, we propose a novel approach for conflict-based change awareness. It utilizes an incremental growing operation tree data structure to promptly detect potential conflicts after a model change. It specifically targets conflicts detectable through the analysis of changes in property-value-based models, such as syntactic conflicts. The obtained results are utilized to filter corresponding change notifications, which are then sent to the engineers. This empowers engineers to proactively identify and address conflicts before merging their work. Moreover, by filtering out irrelevant or non-critical notifications, engineers can focus their attention on changes that are most relevant to their work, reducing cognitive overload and improving efficiency, as demonstrated by related work. With this approach, we aim to enhance collaboration and productivity in CoMDSE environments. We evaluate the completeness of our approach by applying it to a variety of conflict scenarios, as defined by the EMFCompare completeness tests. We also apply our approach in a feasibility CoMDSE scenario using Visio to test the conflict-based notifications. Lastly, we evaluate our approach's performance evidencing that it can detect a large number of conflicts in a reasonable time as it detected almost 20K conflicts in less than 20 seconds.

KEYWORDS model-driven development, collaborative modeling, change awareness, conflict prevention

1. Introduction

Collaborative Software Engineering (CoSE) relates to the practice of having multiple individuals working together to develop large and complex software systems (Mistrík et al. 2010). This development process entails the collaboration of stakeholders from various domains to create and refine artifacts that represent different aspects of a software system (Tröls et al. 2022). To ensure effective collaboration, it is crucial to establish a shared understanding of each artifact throughout the development process (Whitehead 2007). Thus, to support project work, engineers have adopted various communication and collaboration technologies. One of these technologies is model-based collaboration tools (David et al. 2021; Di Ruscio et al. 2017). These tools allow engineers to collaborate in the context of a model-based software representation, such as UML models.

JOT reference format:
Edvin Herac, Luciano Marchezan, Wesley K. G. Assunção, and Alexander Egyed. *Conflict-based Change Awareness for Collaborative Model-driven Software Engineering.* Journal of Object Technology. Vol. 23, No. 3, 2024.

http://dx.doi.org/10.5381/jot.2024.23.3.a7

Model-driven Software Engineering (MDSE) is a software development approach that employs a model-centric paradigm to manage complexities in software system development (Hutchinson et al. 2011; Franzago et al. 2017). Collaborative Model-driven Software Engineering (CoMDSE) is an extension of MDSE that involves artifacts represented as models. CoMDSE enables the early detection of potential collaborative issues during the development process, thereby reducing the cost of fixing problems later on. The ability of CoMDSE to improve efficiency, quality, and stakeholder satisfaction throughout the development life cycle has made it an indispensable practice in modern collaborative software engineering (Muccini et al. 2018).

Proper methods, tools, and standards are a major prerequisite for the widespread adoption of CoMDSE (Hutchinson et al. 2011; Franzago et al. 2017). In CoMDSE, process tasks can be cumbersome and may result in such inevitable change conflicts due to errors in communication and coordination (Young Bang et al. 2018). Change conflicts (either syntactic or semantic, definitions are detailed in (Sharbaf et al. 2022)) are challenges associated with collaborative modeling that arise from multiple engineers working concurrently at the same model (Muccini et al. 2018). To resolve conflicts in CoMDSE, a well-defined conflict-handling process needs to be in place. However, conflicts must be detected first.

Most approaches and tools detect conflicts during model merging (David et al. 2023). There are, however, change awareness approaches that aim at detecting conflicts and notifying users of them as early as possible (Sarma et al. 2011; Guimarães & Silva 2012; Y. Brun et al. 2013; Kanagasabai et al. 2018; Sharbaf et al. 2022). These approaches, however, are either domain-specific, i.e., focusing only on a subset of conflicts, or rely on traditional change awareness, i.e., all changes made by an engineer are notified to others. While the first strategy is limited to specific tools and scenarios, the second is not ideal because it can lead to the disturbance of the engineer's workflow (Guimarães & Silva 2012; J. Y. Bang & Medvidovic 2015). Moreover, notifying engineers about all changes performed by their colleagues requires them to manually analyze these notifications for conflicts. This adds significant overhead considering the large number of changes, leading to decreased productivity and increased frustration.

In this paper, we propose a novel approach that uses potential syntactic conflicts as a means to filter change notifications (i.e., change awareness). Our approach involves an incremental growing operation tree data structure that enables the immediate detection of potential conflicts after a change. This potential conflict detection mechanism works among an arbitrary number of model versions within a reasonable time. The identified potential conflicts are notified to the engineers immediately. This allows them to decide what actions to take.[1] Engineers have then the opportunity to communicate or perform additional changes to prevent conflicts before merging their work with other team members. This approach likely leads to more effective decision-making and problem-solving during the modeling process, rather than reacting to conflicts after designs are finished (J. y. Bang et al. 2018).

For the evaluation of our approach, we implemented a prototype, analyzing it in three aspects. The first aspect assesses the *completeness of our approach*, by determining if it is able to detect the different types of conflicts that can appear when merging model versions. To accomplish this, we apply our approach to detect various conflict situations adopting the conflict categories of the unit test suite of EMFCompare (C. Brun & Pierantonio 2008). The results show that our approach can be used to detect all the conflicts that EMFCompare specifies, with the exception of special cases that cannot be represented by our infrastructure. The second aspect is a *feasibility* study, where we implement a conflict-based change awareness scenario using Visio[2] to show how our approach can be applied in practice as Visio is used in industry for software modeling purposes (Marchezan et al. 2023). The third aspect involves a *performance evaluation* of our approach by showing that the potential conflicts are notified in a reasonable amount of time after model modifications. For example, to detect and notify almost 20k conflicts considering multiple changes and users, our approach requires less than 20 seconds.

The main contributions of our approach are the following: (i) Preventing frequent disruptions to engineers by filtering change notifications using potential conflicts; (ii) Minimizing the workload involved in conflict resolution by providing immediate potential conflict detection; (iii) Mitigating the occurrence of cascading problems by identifying and notifying users about potential conflicts. The remainder of this work is structured as follows: Section 2 details the motivation and definitions of this work by introducing a CoMDSE scenario. Section 3 describes our conflict-based change awareness approach. Section 4 presents the evaluations performed considering the aforementioned aspects. Section 5 discusses the results of the evaluation, considering limitations and future work. Related work is presented in Section 6. Lastly, we conclude this work in Section 7.

2. Background & Motivation

In this section, we introduce an illustrative example of a CoMDSE scenario that describes the problems that motivate this work. Figure 1 presents the overall context of a software project (Bug-Tracking-System) composed of both source code and UML models.

The primary objective of the project is to specify and design a Bug-Tracking-System, focusing on its structure and behavior. This system will determine how bugs are tracked and resolved within a specific IT project, with requirements including managing bugs, identifying responsible engineers, and tracking bug states. To achieve these objectives, stakeholders have agreed to use the Unified Modeling Language (UML), resulting in design models based on the UML metamodel and artifacts consisting of UML state and class diagrams. The example illustrated in Figure 1 demonstrates a scenario where engineers (Alice, Bob, and Carol) collaborate on modifying a state & class diagram and a corresponding implementation (in C#) of the Bug-Tracking-

[1] A demonstration of our approach in use is presented in a video available at (Herac et al. 2024).

[2] https://www.microsoft.com/en-us/microsoft-365/visio/

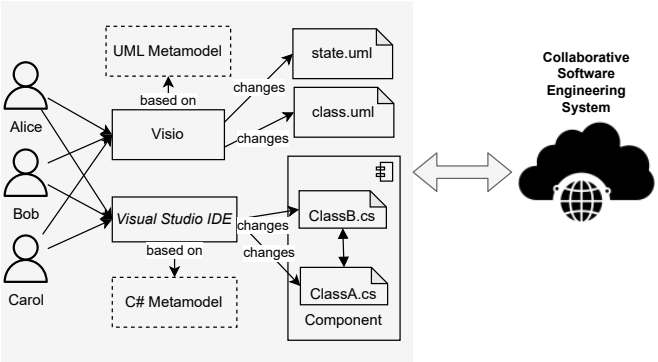


Figure 1 Running example: Bug-Tracking-System.

System. Their concurrent changes, however, can result in conflicts, emphasizing the challenges of collaborative modeling in a real-world setting.

According to Sharbaf et al. (2022) there are two categories of conflicts: syntactic and semantic. Syntactic conflicts are detected when changes have an impact on the syntax of the model (although this may not be true for all cases), i.e., conflicts where two engineers change the name of the same element to different values (Sharbaf et al. 2022). Semantic conflicts, however, refer to conflicts that, while theoretically mergeable, result in a semantically invalid model. For instance, if user A renames class C1 to C and simultaneously user B renames class C2 to C, it leads to the creation of two classes with the same name, violating the programming language syntax and rendering the code/model invalid. These conflicts can be identified through various methods, such as mathematical bidirectional transformations (Diskin et al. 2018), which analyze transformations between models to pinpoint conflicts, or by employing straightforward model views comparison (Reiter et al. 2007). In this paper, however, we focus specifically on syntactic conflicts of property-value-based metamodels (details in Section 5) detected by analyzing changes made directly to the models (e.g., changes that can not be merged or co-exist together in one model). Henceforth, when we refer to "conflicts", we are specifically addressing this category.

The scenario depicted in Figure 1 represents a CoMDSE environment, where multiple engineers work together on creating and updating domain-specific models. Without proper coordination, however, conflicts can arise during merges due to different views, goals, and understandings among engineers. These conflicts not only interrupt engineers' workflows during conflict resolution but also lead to increased time consumption, frustration, and potential errors (Nelson et al. 2019).

Version Control Systems (VCS) like Git or Subversion, however, have simplified this collaborative process by the use of branching and merging mechanisms. Although such systems offer numerous benefits, they also introduce several challenges (Bird et al. 2009), such as issues related to merging and integration (McKee et al. 2017).

Consider Alice and Bob, who concurrently modify the same part of a state diagram that represents the different states during the bug resolution process, as depicted in Figure 2.

They opt against real-time collaboration methods such as Google Docs, which prevent conflicts altogether, as they prioritize leveraging the benefits of working on model features in isolation. This approach allows them to focus on individual aspects of the model without interference and merge their contributions only when they are fully complete. Such a method aligns with the workstyle preferences of engineers and offers distinct advantages in terms of efficiency and workflow organization (Jackson et al. 2024). This collaboration method, however, can benefit from conflict-based change awareness to predict future conflicts.

Alice's task involves extending the state diagram of a BugReport by adding two states, namely `In Progress` and `Rejected`, with corresponding transitions (these changes are labeled as adds A_1 and A_2 in Figure 2). Additionally, Alice decides to rename the `close` transition and `Closed` state to `fix` and `Fixed`, respectively (labeled as renames R_1 and R_2).

Meanwhile, Bob independently initiates changes: renaming the `close` transition and `Closed` state for clarity. However, Bob goes a step further by deleting the `Closed` state and introducing a new one named `Resolved` (delete D_1 and add A_3 in Figure 2). Furthermore, Bob renames the `close` transition to `resolve` (rename R_3).

It is important to highlight Bob's method for renaming the final state: instead of merely renaming the element, he chooses to delete it and then create a new one with a different name. This approach is facilitated by the functionality of Visio tools, which permits the creation of semantically incorrect diagrams during the workflow. For instance, it allows transitions to remain after shapes are deleted. This distinction is significant because it affects the way conflicts are detected and categorized in model-based approaches (e.g., element ID changes after a delete).

When they attempt to merge their state diagrams, a traditional VCS (e.g., Git) may struggle to determine which changes to accept, resulting in textual conflicts. Resolving such conflicts manually can be a daunting and error-prone task, often consuming significant time and resources (Nelson et al. 2019). To mitigate the impact of merge conflicts and integration difficulties research has been carried out to prevent conflicts at all. Strategies such as change awareness, i.e., notifying Alice of changes performed by Bob and vice versa (Sarma et al. 2011; Guimarães & Silva 2012; Y. Brun et al. 2013; Kanagasabai et al. 2018; Sharbaf et al. 2022) can be useful for detecting potential conflicts before they happen (Estler et al. 2014).

The overall idea is that when engineers are notified of changes, they will proactively engage in identifying and analyzing these changes, thereby resolving or avoiding potential conflicts early, while they remain manageable in size. Additionally, the team would develop a deeper understanding of their shared work, reducing frustration and the potential for conflicts overall. The disadvantages of change awareness, however, are that Alice and Bob can be overwhelmed by the constant stream of incoming change notifications that are often irrelevant to their work, e.g., Bob gets notified by the changes A_1 and A_2, which do not cause conflicts with his work. Furthermore, they have to analyze the notifications to see if there are any conflicts, which is often highly time-consuming (Guimarães & Silva 2012).

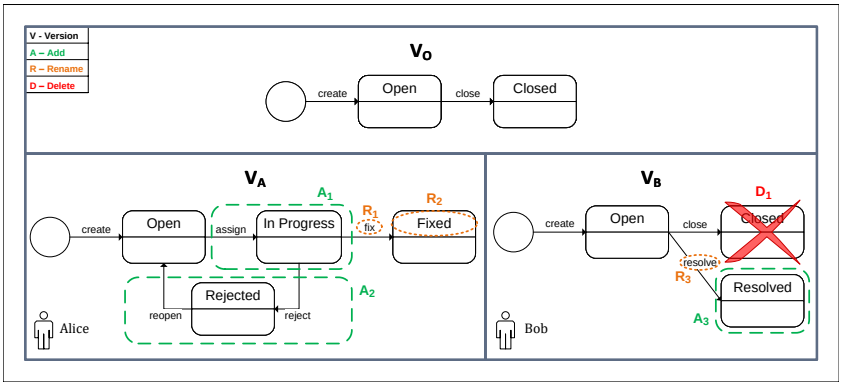


Figure 2 Running example: a BugReport UML state-machine with Alice's and Bob's changes.

To avoid the aforementioned disadvantages of change awareness, researchers proposed change awareness tools that are tailored to notify engineers about filtered changes (e.g., filtering by various features) to aid in analyzing potential conflicts. For instance, proposed change notifications filters can vary from line-based, dependency-based, or branch-based awareness (Guimarães & Silva 2012; Sarma et al. 2007; Dewan & Hegde 2007; Schümmer & Haake 2001) which are domain-specific cases. Nevertheless, tools dedicated to preventing conflicts in collaborative model-driven software engineering are still scarce and often customized for specific domains. Moreover, a mapping study on model-based conflict management techniques conducted by (Sharbaf et al. 2022) revealed that only one study by (Kanagasabai et al. 2018) offers model-based change awareness for conflict prevention, which is currently highly desired in the industry (David et al. 2023).

In summary, *change awareness methods often overwhelm engineers* with a bulk of change notifications, many of which are irrelevant to their immediate tasks. Furthermore, the need to *manually analyze these notifications for conflicts* adds significant overhead, leading to decreased productivity and increased frustration. In the next section, we propose our approach of *conflict-based change awareness* for CoMDSE, designed to tackle the problem of change notifications overwhelming engineers and the problem of domain-specific solutions for change awareness.

3. Proposed Approach

In response to the challenges posed by overwhelming change notifications and the lack of non domain-specific solutions for change awareness in CoMDSE, we introduce a novel approach called *conflict-based change awareness*. Our approach focuses on leveraging potential conflicts as indicators of changes that require attention. By using possible conflicts to filter changes during the engineering process, engineers can proactively address them before they escalate into larger, more challenging problems. Moreover, our approach incorporates a conflict filtering mechanism to streamline the notification process. By filtering out irrelevant or non-critical notifications, engineers can focus their attention on changes that are most relevant to their work and potential conflicts, reducing cognitive overload

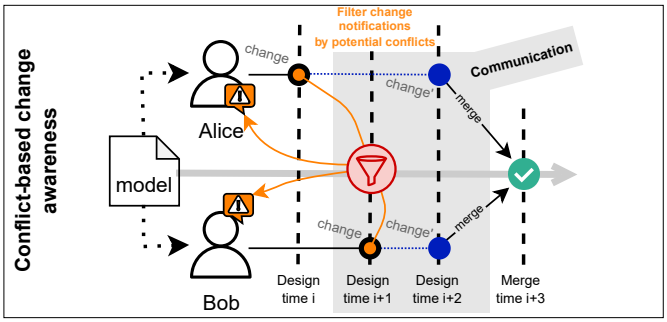


Figure 3 Conflict-based change awareness workflow: Filtering change notifications for engineers with potential conflicts detected through an incremental change analysis.

and improving efficiency (Guimarães & Silva 2012; J. Y. Bang & Medvidovic 2015).

In the following sections, we delve into the principles of our conflict-based change awareness approach, emphasizing its benefits and potential impact on improving collaboration and productivity in CoMDSE environments.

3.1. Workflow

The proposed approach, referred to as the Conflict-based Change Awareness and illustrated in Figure 3, works in the following manner: Consider Alice and Bob, who are independently working on their own versions of the same model. At `Design time i`, Alice modifies the transition name from `close` to `fix`. This change goes unnoticed by Bob, as the conflict detection filter does not identify any potential issues and filters out all change notifications. However, at `Design time i+1`, Bob changes the transition name from `close` to `resolve`. This alteration is recognized as a conflict, triggering the change notification where the filter allows both Alice and Bob's changes to pass (i.e., Alice is notified about Bob's rename, and vice versa).

Subsequently, at `Design time i+2`, Alice and Bob have the opportunity to discuss their respective changes, allowing them to reconcile their differences and agree on a unified approach. This step could be crucial to avoid future merge conflicts and potential complexities, such as when the transition name is used in other models. Finally, by `Design time i+3`, Alice and Bob are able to merge their models without any conflicts

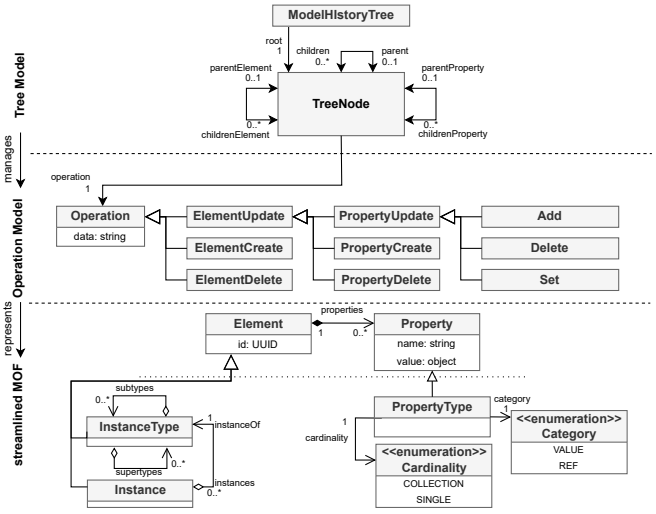


Figure 4 Data model used by our approach, based on (Herac et al. 2023).

due to the preventive resolution of potential conflicts. In the following section, we describe how each step of this workflow is supported by our approach.

3.2. Collaboration mechanism & Data model

Our approach uses our previous work (Herac et al. 2023) as the basis for the collaboration infrastructure. The infrastructure defines a `Workspace` for the collaborative mechanism to give the engineers Alice and Bob their own replicated model locally where models can be modified, pushed, and pulled to a common `Public Workspace`. A `Workspace` provides a replicated isolated model from the other engineers. Hence, while a model is being modified inside a *Workspace* no other engineers can see its state until it is pushed. Thus, each `Workspace` can have its own metamodels and models that can be shared through pushes and pulls to or from a public workspace, forming a star topology with the `Public Workspace` in its center. This kind of collaboration mechanism permits parallel/concurrent work on models and avoids turn-taking, while requiring conflict management (e.g., detection and resolution). In this approach, we are mainly focusing on raising awareness about changes to prevent conflicts during future merges, thus engineers can decide how to tackle conflict management itself.

Similar to the generic infrastructure on which we base our approach on (Herac et al. 2023), we rely on the streamlined MOF for our datamodels, as illustrated in Figure 4. This streamlined MOF enables the creation, modification, or deletion of metamodels and models, instantiating corresponding models within a private `Workspace`. For our research purposes, this streamlined MOF can be used to represent the necessary metamodels and models required by our approach. Additionally, it offers a simpler and more generic way to represent various models than more complex solutions such as EMF (Steinberg et al. 2008). This results in fewer distinct operation types, making conflict detection easier to perform.

The approach uses a history of operations, that conforms to our operation model (center part of Figure 4). These operations can represent changes made to the metamodels and their corresponding models generated from the streamlined MOF. This technique is commonly referred to as change-based persistence (CBP) (Yohannis et al. 2017). Unlike state-based persistence approaches, such as EMF, which stores the current state of the model, CBP records all changes as a finite sequence of operations. Operations are atomic changes that represent as a sequence the data model state. CBP offers rapid change detection as demonstrated in (Ráth et al. 2012; Ogunyomi et al. 2015; Yohannis et al. 2019). However, CBP has the potential drawback of accumulating extensive and ever-growing change history files. This accumulation can lead to longer loading times and require a complex reconstruction of the current model state from the model history (Yohannis et al. 2017).

Our defined operations are displayed in the *Operation Model* section of Figure 4). They include `ElementUpdate`, `Create`, or `Delete` and `PropertyUpdate`, `Create` or `Delete`. `PropertyUpdate` can be of types `Add` (for collections only) and `Set` or `Delete` (for both collections and single properties). The aforementioned history of operations is managed in a tree-like data structure called `Model History Tree`, described in the following section.

3.3. Model History Tree & Conflict Detection

We use an infrastructure implementation of CBP that employs a history of change operations to represent the state of the entire model, including metamodels of all versions (i.e., branches). However, simple lists do not suffice for illustrating competing versions since concurrent changes on the same model cannot be persisted. To address this limitation, we used incremental sequences of operations in a tree-like structure to represent diverging model versions, represented as an `Operation Tree` (see Figure 5).

Each node within the `Operation Tree` encapsulates an operation, with regular edges indicating the historically preceding change operation node, ultimately tracing back to the root node, which represents the initial change. A branch delineates a sequence in the model's history, with each gray dotted circle surrounding a node denoting a model state token of the labeled workspace. A branch is created through a new change on a predecessor node, independent of other branching nodes. The exact build-up algorithm of the operation tree is explained in more detail in our prior work (Herac et al. 2023).

For the purpose of finding change operations of corresponding elements or properties faster, we added additional edges to the `Operation Tree`. These are edges for element operations and edges for property operations, shown as associations in the `Tree Model` section of Figure 4. Following those edges from the roots to the leaves, we only get a list of operations that represent a specific element or a specific property of an element. Thus, an element operation sequence is a sub-list of a whole model history. Henceforth, a sequence of property operations is a sub-list of a sequence of element operations. Those sequences, however, can also diverge to a tree and represent different branches of element versions or property versions. The `Element Tree` is illustrated in Figure 5 for the specific element `x` and the `Property Tree` for a specific property `y` of an

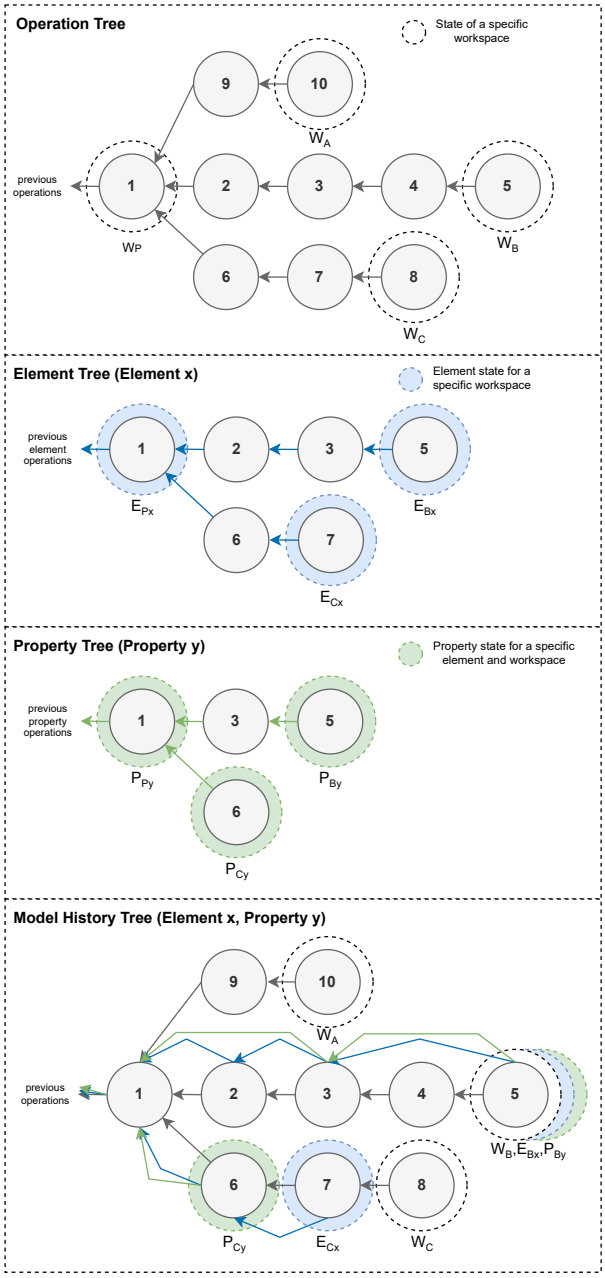


Figure 5 Operation Tree, Element Tree and Property Tree encoded together in the Model History Tree. This figure shows a segment of the Tree, concealing prior operations that lead to the root node due to space limitations.

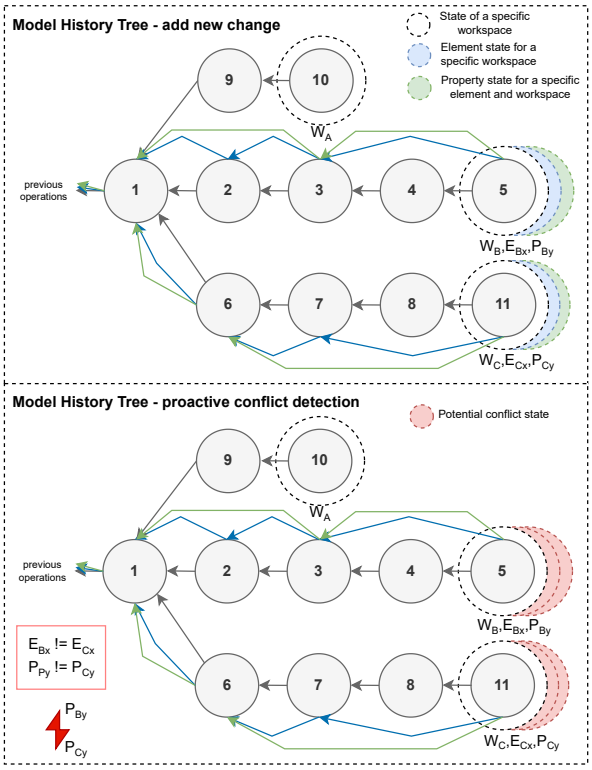


Figure 6 Model History Tree with a potential conflict.

element `x`. Encoding all the mentioned trees into one tree, gives us the `Model History Tree`, with the corresponding states of all workspaces, elements, and properties as tokens. This is illustrated at the bottom part of Figure 5, for the workspaces P (Public), A, B, and C, as well as for their corresponding element `x` and property `y`.

To get a specific model state, all operations of the corresponding workspace from the root to its state node have to be executed. For example, let us consider the tree at the top of Figure 5, the current model for workspace `B` (i.e., W_B) operation sequence would be (1, 2, 3, 4, 5). For the workspace `A` (i.e., W_A), the operation sequence would be (1, 9, 10). The same algorithm applies for elements and properties, but following their corresponding edges.

The creation of a new change adds a new child node with the corresponding operation to the current workspace's state node. A workspace state is depicted as a token (i.e., gray circle around a node) on a node in the `Model History Tree`. Additionally, an edge for element and an edge for property changes are added to the new operation node, to connect their last state to the next change. The operation tree enables the support of branches and merging of branches (i.e., merging of model versions) by reattaching the branches to other leaf nodes. However, our work focuses solely on providing an approach for conflict-based change awareness to prevent conflicts before merges occur. The process of merging and conflict resolution is beyond the scope of this work.

In summary, when elements or properties are changed, a new node is added to the `Model History Tree` with three distinct

edges attached to the leaf state of the corresponding workspace, element, and property, as depicted in the top part of Figure 6. This mechanism is utilized to detect potential conflicts after every change. It is achieved by comparing the leaf nodes of each workspace of the modified element or property. If the leaf node is not the same (e.g., not the common branching node), it can be inferred that a potential conflict has been created (e.g., new change node was added). Subsequently, we compare the modified property leaf node, to detect if the element or one of its properties was changed. In the case of different property nodes, we can conclude that a potential conflict has been created on the changed property, illustrated in the bottom part of Figure 6.

Once a potential conflict is detected due to overwritten change nodes, we take the corresponding changes (e.g., filtered changes by a potential conflict) and notify the engineers of those. The engineers receiving the notifications retain their autonomy to determine their responses. They may choose to promptly make changes to preempt conflicts, engage in communication with the colleague responsible for the potential conflict, or delay conflict resolution for a later time. Alternatively, the implementation can be customized to deactivate messages for specific users or refrain from detecting conflicts for certain users altogether, if deemed necessary. However, these implementation details fall outside the scope of this research paper.

3.4. Filtered Change Awareness

The proposed approach detects conflicts on each new change incrementally (e.g., just comparing the workspace element nodes, where the element was changed), leaving out nonconflicting changes. Figure 7 summarizes the process of conflict-based change awareness. Alice and Bob are changing a state model through an arbitrary tool (e.g., Visio). The tool generates change operations and sends them to the `Model History Tree`, which persists the new change and checks for potential conflicts, as formerly illustrated in Figure 6. In a conflict, the change notifications are sent to the engineers to make them aware. Alice and Bob can now communicate on a common resolution during design time before the conflict emerges and prevent it. Here it is also important to state that the filtering mechanisms can be exchanged by different mechanisms for change-awareness (e.g., filter changes by a certain user, or by a certain element). Moreover, our approach does not focus on ways to detect conflicts, but on how to detect them during design time before they happen during a merge.

Finally, after Alice and Bob are notified, their tools can decide what to do with the new awareness data. For example, Alice's and Bob's state model gets a highlighted transition label, marked as $\texttt{N}_1$, as exemplified in Figure 8. This indication occurs because Alice changed the transition name to `fix`, which was subsequently altered to `resolve` by Bob. The changed values, for instance, can be seen as *tooltips* inside the tool. Furthermore, Alice's last state `Fixed` was highlighted too, marked as $\texttt{N}_2$. It was deleted by Bob, because he created a new state and named it `Resolved`, see Figure 2. The filter, however, was implemented to neglect renamed elements that were deleted by Bob. Thus, no notification $\texttt{N}_2$ was displayed to Bob. This is one possibility of how the final change awareness could be visualized. It depends on the implementation of the tool and how it reacts to the conflict-based change notifications. In the following section, we present a feasibility evaluation that demonstrates one example of visualization using Visio.

4. Evaluation

We evaluated our approach based on three aspects. The first aspect assessed the completeness of our approach by determining its ability to detect different types of potential conflicts that can occur when engineers work concurrently. The second aspect is related to our approach's performance, aiming at measuring if it can notify potential conflicts in a reasonable amount of time across multiple workspaces. The last aspect is related to the feasibility of the approach. Hence, we implemented a proof-of-concept version of our approach as a server. We then implemented a plugin for MS Visio that connects to the server, demonstrating how immediate conflict awareness could be utilized in the industry. In the following sections, we detail each aspect of the evaluation.

4.1. Prototype Implementation

To evaluate our approach, we implemented a prototype tool in Java as a server that supports a generic infrastructure established by prior work (Herac et al. 2023). Further implementation details can be found in that publication for a more in-depth understanding. This infrastructure allows the connection of different engineering tools by implementing tool plugins that transform artifacts created in these tools to a model representation based on the data model presented in Figure 4. Although multiple artifact types are supported by the server in this evaluation we only adapted a Visio plugin (see Section 4.3) to receive and display potential conflict notifications.[3] The specifications for the execution environment of our evaluation are an Intel Core i7-7700 CPU @3.6GHz with 16GB (8GB available for the tool) RAM and Windows 10 x64-based. The generated results are available at our online repository (Herac et al. 2024).

4.2. Completeness

For completeness, we adopt a strategy similar to the evaluation by (de la Vega & Kolovos 2022). They utilized the external test suite of the EMFCompare framework to verify the completeness of their conflict detection approach, which comprises 60 unit test cases. Those tests detect conflicts that could occur while merging EMF models. The difficulty of adapting their completeness evaluation lies in the difference between the EMF model and our data model (Figure 4). This is the case because EMFCompare uses XMI files[4] to load models for three-way merge (Lindholm 2004) conflict detection. Thus, we implemented a parser to transform the XMI models from the EMFCompare test suite and use them with our streamlined MOF data model. For instance, we used our instances (`Instance`) instead of XMI nodes, and instance properties (`Property`) instead of node attributes. Consequently, we mapped attribute

[3] Details about the implementation are available at https://isse.jku.at/designspace.

[4] https://git.eclipse.org/c/emfcompare/org.eclipse.emf.compare.git/

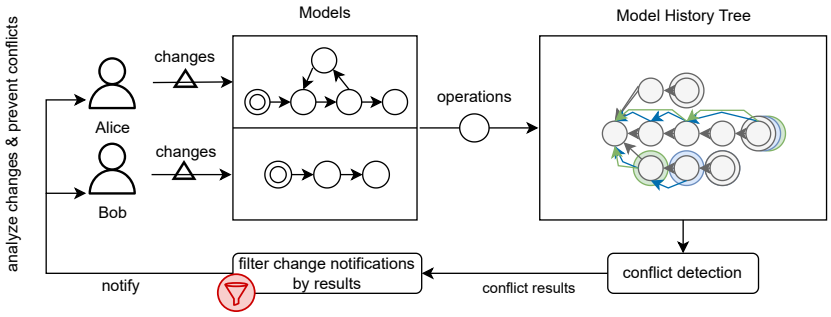


Figure 7 Conflict-based change awareness approach with the model history tree.

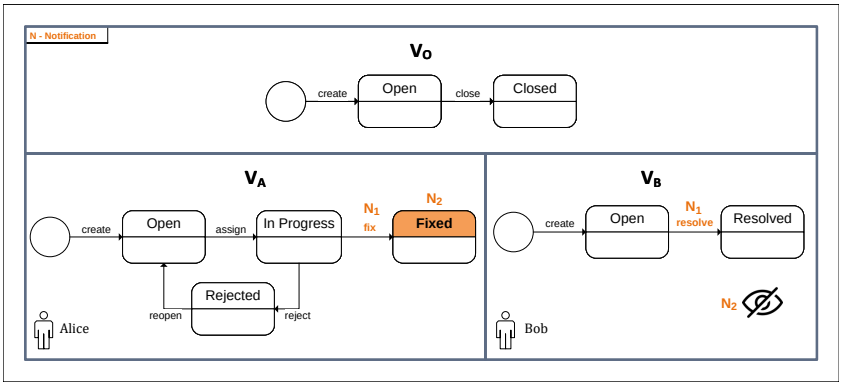


Figure 8 Running example: BugReport UML state-machine with filtered change notifications for Alice and Bob.

deletions to property deletions, and so on. After transforming 54 out of the 60 EMFCompare unit tests, we were able to detect all those conflicts.

Table 1 summarizes the results of the 54 tests grouped into 11 categories. As can be seen, all 54 tests passed, showing that our approach can detect all the same types of conflicts that EMFCompare supports. Furthermore, our approach detected even more conflicts than EMFCompare in some cases, as we analyzed fine-grained atomic changes of our data model, resulting in a higher resolution of changes during conflict detection.

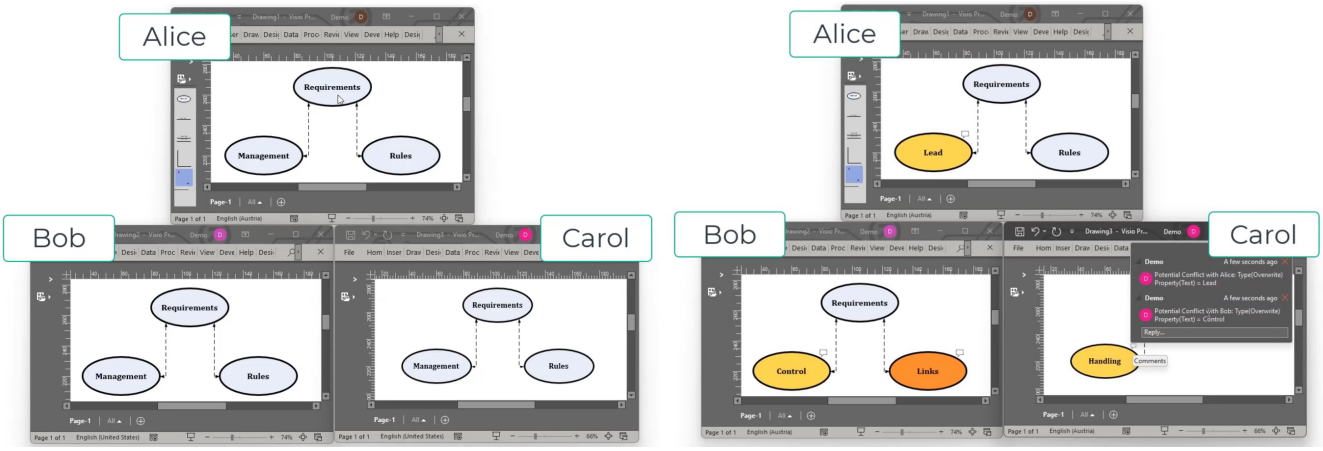
However, it is challenging to determine whether these results are beneficial or not because we did not compare the types of conflicts, as EMFCompare only uses definitions such as *real* or *pseudo* conflicts, whereas we implemented *overwrite*, *order* and *delete* conflict types for this evaluation. The types of conflicts should be adapted to the requirements of the scenario. At this point, finding a common ground truth for the conflict types becomes increasingly difficult. The modeling community lacks of a consensus on a possible generalization of conflicts in versioning (Altmanninger & Pierantonio 2011). Initial approaches for categorizing conflicts in versioning cannot be fully applied to model artifacts (Mens 2002). Moreover, it is often challenging to categorize conflicts into specific categories (Altmanninger et al. 2009). Thus, we plan to investigate this in future work as detailed in Section 5.

4.3. Feasibility

We conducted a feasibility evaluation of our approach by implementing a two-tier client/server system. The server acts as a centralized node responsible for managing all workspaces and changes, while clients are connected through our software development kit (SDK). To ensure the necessary data modifications, our solution includes a connection handler and a synchronization layer of our streamlined MOF. The model history tree has been integrated into the server, operating according to the workflow depicted in Figure 7. In addition, we developed a plugin for Visio that connects to our system through the client SDK and receives the potential conflict notifications. This plugin also defines a metamodel for Visio shapes, generated from the streamlined MOF (Figure 4).

In our feasibility scenario, visible in Figure 9, three engineers, namely Alice, Bob, and Carol, collaborate on a brainstorming diagram.[5] In this scenario, Bob starts by renaming the element `Management` to `Control` (on the left side of Figure 9a). Then, Alice renames the same element (notice that at this point, Alice does not see Bob's change) to `Lead`. Once Alice's change is applied, a potential conflict is immediately detected because both Bob and Alice changed the same element to different values but have not pushed or pulled any changes. Once this happens, both Bob's and Alice's Visio are notified of the changes from our approach. The implemented plugin can now decide how the notifications should be displayed. In this example, we implemented the Visio plugin to highlight the conflicting elements (e.g. yellow for simple overwrite conflicts and orange for delete conflicts), additionally adding a message from the conflict notification as a shape comment. This is one

[5] This scenario is demonstrated in a video available at (Herac et al. 2024).



(a) Initial state of the diagram.

(b) Example of visible notifications of potential conflicts.

Figure 9 Collaboration between three instances of Visio working with a brainstorm diagram.

example of how tools could handle change notifications about potential conflicts. Nevertheless, different stakeholders may want to handle them differently. Thus, the visualization of the notifications here serves only as an illustrative example and is part of future work.

Next, Bob renames element `Rules` to `Links` while Carol deletes that element. This creates a conflict on Bob's side since the element he just modified has been deleted by another engineer. A notification is sent to Bob telling exactly who caused the conflict and why. We decided, for simplicity and preference reasons, to not notify Carol if there is a conflict with her deleted element. At this point, Carol changes `Management` to `Handling` producing two conflicts for everyone (displayed in Carol's Visio window as a comment of the shapes in Figure 9b). Now the problem cascaded, and they decided to meet in person and find a common problem-solution. After communication, they decide to go with the name `Handling`. Then, they all change `Handling` until the conflict disappears. They decided to leave the deletion conflict to be handled for the conflict resolution during the merge (delaying the resolution). Their awareness process could foster a deeper understanding of each other's work and decisions, likely resulting in a positive influence on all subsequent model designs they produced. In summary, our approach gives engineers the autonomy to determine how to handle predicted conflict changes. We have intentionally designed it to be minimally intrusive, allowing plugins to respond to change messages as needed.

4.4. Performance

To evaluate the performance of our approach, we aim to measure the time required to detect potential conflicts using a varied number of changes, properties, and workspaces. For that, we relied on a random change generator where we simulated a scenario to collect the results. Unique values for the changes are created with a UUID generator. We run through diverse settings to measure the conflict detection time via our model history tree. For example, if we use 40 workspaces, 5 instances, 50 properties, and 50 changes, we have 40 times 5 instances with 50 properties overall in the system. That means that we have 250 unique properties encoded in our model history tree. We also generate, for each workspace and each property, 50 changes to random UUIDs. This creates an overall of 250 conflicts in between every workspace leading to a total of 9,750 conflicts. When we generate the last workspace changes, we measure the time from the last change until all potential conflicts are identified and notified. In this example, the notification consists of 9,750 conflicts and takes 4.3 seconds to be analyzed.

A summary of the runtime results is presented in Figure 10, demonstrating that even with 40 workspaces, 100 changes, 50 instances, and 10 properties (leading to 19,500 conflicts detected) the calculation time remains under 20 seconds. This indicates that the runtime grows linearly. This efficiency is particularly good considering the typical scenarios in traditional CoMDSE. Furthermore, the results were collected using a prototype implementation of the approach (i.e., they can be improved when implementing a professional version). Generally, the number of changes and the quantity of workspaces being concurrently modified are lower than those tested in our performance scenario (Aldndni et al. 2023; Brindescu et al. 2020; Dias et al. 2020). In addition to that, (Yohannis et al. 2019) demonstrated that the use of change-based comparison of operation sequences is a highly efficient method for detecting conflicts.

5. Discussion & Limitations

While it is essential to note that a streamlined MOF (Figure 4) may lack certain advanced features and capabilities found in more sophisticated solutions (e.g., EMF), it is equally important to recognize that this could curtail the expressiveness of the models that can be created, potentially limiting their flexibility and richness. Additionally, the simplified nature of streamlined MOF may introduce constraints on the complexity and size of models that can be effectively managed, with large-scale or intricate models potentially encountering performance issues or becoming unwieldy to handle. Nevertheless, our approach can

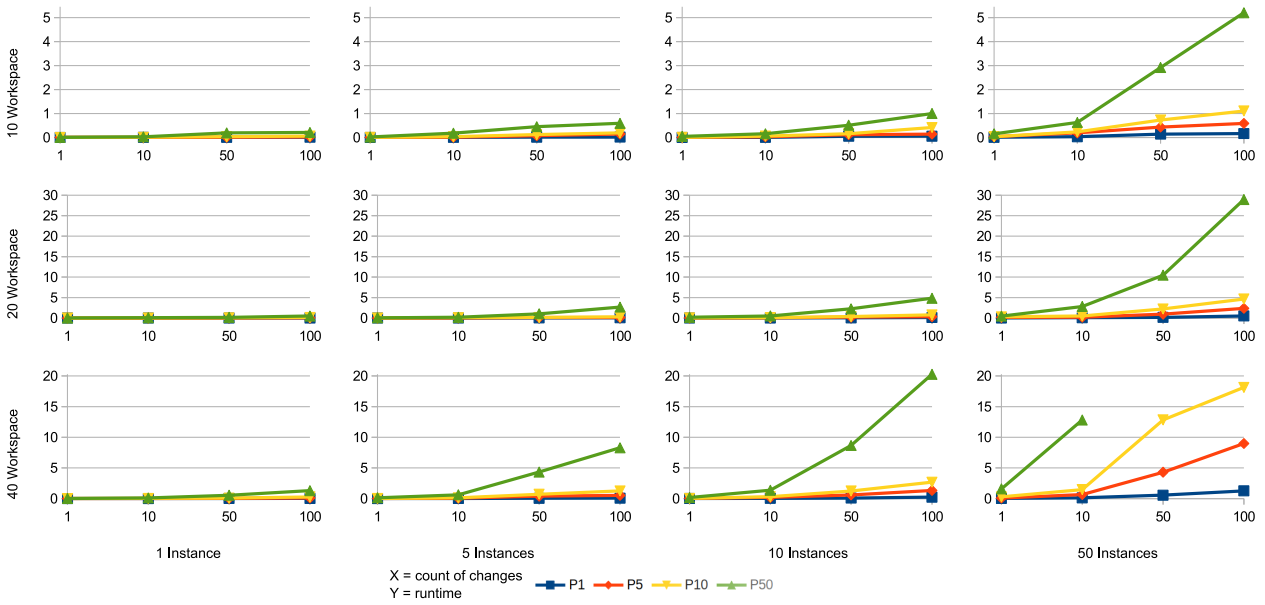


Figure 10 Runtime (*s*) required to detect potential conflicts in different configurations of changes, instances, and workspaces.

theoretically be applied to more complex MOFs using a broader range of operation types. In this paper, however, we focus on presenting the concept of utilizing a history tree and operations to find simple property-value-based conflicts in almost real-time to filter changes for users.

Evaluating the correctness of change awareness approaches is a challenge mainly because of the lack of a proper baseline for comparison in the CoMDSE field. As known, conflict detection (especially resolution) research and practice has mainly focused on source-code/textual artifacts that rely on traditional VCS for merging (Costa & Murta 2013). Thus, change awareness considering conflicts is either delayed or not possible. Recent mapping studies reported approaches for conflict detection that support change awareness (Sharbaf et al. 2022). Among these approaches, only one is not designed specifically for a subset of conflicts (Brosch et al. 2011). Their approach, however, is not evaluated as only an illustrative application scenario (similar to our feasibility study) is used. Thus, the correctness of our approach is argued in terms of completeness. Considering the completeness, however, we must clarify that in this work we have focused on one category of conflict only, i.e., syntactic conflicts (Sharbaf et al. 2022). As mentioned in Section 2, there are still semantic conflicts.

Furthermore, the streamlined MOF (Figure 4) may lack certain advanced features needed to express complex models fully. The six missing EMF-specific conflict cases fell into this category, as our streamlined MOF could not accommodate them without significant technical implementations. Nonetheless, our approach focuses on detecting property-value-based conflicts by the help of model changes and forwarding the filtered changes to the user. While we acknowledge that our approach may not cover all possible conflicts, especially those beyond property-value definitions, *it effectively predicts conflicts arising from merges involving properties defined within our streamlined MOF* (e.g., the 54 conflicts of EMFCompare). Moreover, it is important to clarify that the streamlined MOF provided is not the primary contribution of this work. Our primary objective is to efficiently address a broad spectrum of property-value-based conflicts, which are common in model merging scenarios. This focus ensures a high degree of utility in typical use cases, albeit with some limitations in special cases.

We plan to extend the current approach to also identify semantic conflicts. These conflicts, however, require semantic knowledge of the models (as their name suggests). Related work on this topic, relies on rules (e.g., defined in OCL [6]) to analyze the models semantically and detect these possible conflicts (J. Y. Bang et al. 2017; Tröls et al. 2022). Thus, we plan to make use of OCL-based rules to evaluate changes that cause potential conflicts. We had proposed similar approaches using rules for consistency checking, showing prominent results (Tröls et al. 2022; Marchezan et al. 2022, 2023). Adapting these approaches to support our conflict-based change awareness remains future work.

The feasibility and the performance evaluations can be used to better understand how our approach can be applied in a real scenario. While we can extend the approach to work in additional tools, the Visio plugin is a good example of the generalization of our data model (Figure 4) as Visio itself supports multiple types of models (UML, BPMN, Requirements). As mentioned, we use a generic infrastructure for collaborative work (Herac et al. 2023) as the basis for our conflict-based awareness approach. Thus, extending our approach is straightforward as the infrastructure was designed and evaluated to be applied with different tools and models. We thus argue that *our approach can be extended to a variety of models, allowing conflict-based change awareness to be adapted by different tools.* Moreover, unlike domain-specific tools that cater to specific types of changes or models, our conflict-based change

[6] https://www.omg.org/spec/OCL/

Table 1 EMFCompare conflicts test results, grouped by categories.

Cat.	Conflict Description (Left ⇔ Right)	# of tests/passing
a	change single value ⇔ delete element	6/6
b	change single value ⇔ change same single value	13/13
c	change values of collection ⇔ deletes containing element.	10/10
d	change values of collection ⇔ change same collection in a conflicting way	12/12
e	add reference to deleted element ⇔ delete element	3/3
f	change containment feature to two features ⇔ change containment feature to different two features	1/1
g	change containment feature to two features ⇔ change containment feature to two distinct features	1/1
h	change container of element ⇔ delete container	2/2
i	delete a set of elements ⇔ delete a different set of elements	1/1
j	change multiple features ⇔ delete its containing element	1/1
k	add elements ⇔ add elements, some of them have the same id as left	4/4

awareness method is applicable across various domains and modeling contexts, making it a versatile solution for CoMDSE projects and highlighting the novelty of our work. Still, implementing such extensions requires effort and is restricted to models/tools that can be represented in a property-value manner. Details about these requirements are discussed in our previous work (Herac et al. 2023).

Despite the benefits that the evaluation performed in this work brings to understanding our approach's applicability, they are still limited in terms of being a real environment. Thus, a limitation that we plan to address in future work is conducting an empirical evaluation with engineers. This evaluation would allow us to analyze how the potential conflict notifications can be fine-tuned, e.g., communicated to the engineers. Although not evaluating with engineers, however, our approach still contributes to the CoMDSE field. As evidenced by literature (Nelson et al. 2019; Vale et al. 2022) conflict awareness can bring different benefits for engineers, as the more conflict detection is delayed, the more effort and costs are required for the resolution. Hence, our conflict-based change awareness can bring practical benefits such as: (i) *Avoiding frequent disruptions to engineers* by filtering change notifications based on potential conflicts; (ii) *Reducing the workload associated with conflict resolution* through instant detection of potential conflicts; (iii) *Mitigating the occurrence of cascading problems* by identifying and promptly notifying users about potential conflicts.

Lastly, our approach focuses on the use of conflicts to filter change notifications and only send to users those changes that can cause potential conflicts. Our approach, however, can be extended to incorporate other types of filters besides conflicts. For example, we can modify the filtering mechanism to consider the engineer's role in the company, e.g., a manager is notified of the changes from their subordinates, or based on types of model elements, e.g., a change in a state is notified but not a change in a transition. Thus, in the future, we plan to employ our change awareness approach for other contexts that are not necessarily conflict-related.

6. Related Work

As mentioned in Section 2, despite the importance of potential conflict detection and change awareness for CoMDSE, there have been only a few approaches proposed to address these issues. This lack of existing support for detecting potential conflicts before they happen is highlighted in the studies of (Sharbaf et al. 2022), (Franzago et al. 2017), (David et al. 2021, 2023), and (Kanagasabai et al. 2018).

Furthermore, there are many sophisticated collaborative approaches like MONDO (Debreceni et al. 2017) or variED (Kuiter et al. 2021) that only provide reactive conflict handling techniques. However, (Kanagasabai et al. 2018) propose a form of change awareness for multi-view modeling in a UML environment. As part of their work, they utilize UML metamodels to visualize changes made by remote engineers in a multi-view modeling environment. Their algorithm highlights the elements that are affected by the change. In contrast, we propose a change-based approach that produces fine-grained change notifications that are filtered by conflicts. Our proposed technique enables end-users to use the notifications in any way they deem appropriate to enhance their understanding of their work, such as highlighting artifacts that contain potential conflicts.

J. y. Bang et al. (2018) propose a similar approach for detecting potential conflicts, called FLAME. FLAME uses an event-based version control system and existing modeling and analysis tools to perform proactive conflict detection. It is built on top of the Generic Modeling Environment (Ledeczi et al. 2001) and the XTEAM architecture modeling and analysis framework (Edwards & Medvidovic 2008). FLAME also tracks every operation and detects conflicts after each operation, similar to our approach. In FLAME, however, each received operation is first executed on its local model before the conflict detection is invoked automatically between the final state models by simulating merges. This strategy of executing changes in a local model is called speculative merging (Owhadi-Kareshk et al. 2019) and is very performance intensive. FLAME distributes the conflict detection to multiple engines, each maintaining an internal copy of the model, to improve performance. In contrast to FLAME, we do not execute the operations on local models for conflict detection. Instead, we propose a conflict detection approach based on operations orderly managed in a tree structure, enabling us to detect conflicts with high performance, as evidenced by related work on operation-based comparisons (Yohannis et al. 2019). Numerous approaches exist for potential conflict detection (Y. Brun et al. 2011, 2013;

Guimarães & Silva 2012). These, however, are domain-specific, primarily implemented with text-based VCS such as Git or Subversion. Our approach is model-based and works with any metamodel that can be represented with our data model.

Our approach stands out from existing solutions by offering a unified and domain-agnostic model-based approach to change awareness with a change notifications filter that detects potential conflicts before they happen. Furthermore, it does not use speculative merges (e.g., checking for conflicts by simulating merges). This increases the performance as we do not need to simulate possible merges to identify potential conflicts. In addition, the filtering mechanism can be adapted to other scenarios, that are not related to conflicts. For example, filtering notifications based on user roles. This allows our approach to be extended to other CoMDSE applications not necessarily related to conflict management.

7. Conclusion & Future Work

In this paper, we investigate the use of conflict information as a mechanism to filter change notifications sent to engineers of CoMDSE. This opportunity is used to define our conflict-based change awareness approach. We rely on the use of a model history tree that analyzes operations created by engineer changes to identify conflicting ones. Since these changes were not merged yet, our approach is able to identify potential conflicts in real-time, triggering notifications to the engineers that can be affected by these potential conflicts. Engineers can then reason about the potential conflict, deciding to communicate with their colleagues or perform changes to prevent the conflict from becoming concrete.

We argue that the approach can be used to improve the conflict management process, by allowing engineers to foresee conflicts and thus immediately deal with them (if they decide to do so). We implemented a prototype version of the approach to perform an evaluation considering the completeness, feasibility, and performance. The completeness evaluation shows that our approach is able to detect all the possible conflict types of the used baseline (EMFCompare). The other category of conflicts, i.e., semantic conflicts (Sharbaf et al. 2022), is not part of this work and will be investigated in future research. Considering the feasibility, we implemented a plugin for Visio that is connected to our approach prototype implementation using a client/server architecture. This plugin is able to listen to the filtered change awareness notifications, i.e., potential conflicts, and display them to the users.

Lastly, we simulated a variety of CoMDSE scenarios with a variety of changes, workspaces (i.e., engineers), and potential conflicts. This simulation allowed us to measure the performance of our approach, which only required around 20 seconds to detect and notify almost 20k conflicting changes. This shows that the approach can be applied in practice as this amount of conflicts is much larger than the expected possible conflicts that would happen in a 20-second interval of a CoMDSE workflow.

For future work, besides investigating the detection of semantic conflicts, we also plan to evaluate the approach in scenarios with real engineers. The main goal of this evaluation is to understand how the potential conflicts' visualization can be fine-tuned for specific tools or types of users. Another plan for future work is to implement additional plugins to extend the prototype implementation of the approach to additional tools (e.g., Papyrus, Visual Studio, IntelliJ). In addition, we plan to extend our filtering mechanism to support different heuristics besides conflicts, e.g., filtering notifications based on user roles or model element types.

Acknowledgement

This research was funded in part by the Austrian Science Fund (FWF) [10.55776/P34805], the LIT Secure and Correct System Lab funded by the State of Upper Austria, and the FFG-COMET-K1 Center "Pro^{2}Future" (Products and Production Systems of the Future), Contract No. 881844.

References

Aldndni, W., Meng, N., & Servant, F. (2023). Automatic prediction of developers' resolutions for software merge conflicts. *Journal of Systems and Software*, *206*, 111836. doi: https://doi.org/10.1016/j.jss.2023.111836

Altmanninger, K., Brosch, P., Kappel, G., Langer, P., Seidl, M., Wieland, K., & Wimmer, M. (2009). Why model versioning research is needed!? an experience report. In *Modse-mccm 2009 workshop @ models* (Vol. 9, pp. 1–12).

Altmanninger, K., & Pierantonio, A. (2011). A categorization for conflicts in model versioning. *e & i Elektrotechnik und Informationstechnik*, *11*(128), 421–426.

Bang, J. Y., Brun, Y., & Medvidovic, N. (2017). Continuous analysis of collaborative design. In *2017 ieee international conference on software architecture (icsa)* (p. 97-106). doi: 10.1109/ICSA.2017.45

Bang, J. y., Brun, Y., & Medvidović, N. (2018). Collaborative-design conflicts: Costs and solutions. *IEEE Software*, *35*(6), 25-31. doi: 10.1109/MS.2018.290110057

Bang, J. Y., & Medvidovic, N. (2015). Proactive detection of higher-order software design conflicts. In *12th working ieee/ifip conference on software architecture* (p. 155-164). doi: 10.1109/WICSA.2015.15

Bird, C., Rigby, P. C., Barr, E. T., Hamilton, D. J., German, D. M., & Devanbu, P. (2009). The promises and perils of mining git. In *2009 6th ieee international working conference on mining software repositories* (pp. 1–10).

Brindescu, C., Ahmed, I., Jensen, C., & Sarma, A. (2020). An empirical investigation into merge conflicts and their effect on software quality. *Empirical Software Engineering*, *25*, 562–590. doi: 10.1007/s10664-019-09735-4

Brosch, P., Egly, U., Gabmeyer, S., Kappel, G., Seidl, M., Tompits, H., ... Wimmer, M. (2011). Towards semantics-aware merge support in optimistic model versioning. In *International conference on model driven engineering languages and systems* (pp. 246–256).

Brun, C., & Pierantonio, A. (2008). Model differences in the eclipse modeling framework. *UPGRADE, The European Journal for the Informatics Professional*, *9*(2), 29–34.

Brun, Y., Holmes, R., Ernst, M. D., & Notkin, D. (2011). Proactive detection of collaboration conflicts. In *19th acm sigsoft symposium and the 13th european conference on foundations of software engineering* (p. 168–178). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/2025113.2025139

Brun, Y., Holmes, R., Ernst, M. D., & Notkin, D. (2013). Early detection of collaboration conflicts and risks. *IEEE Transactions on Software Engineering*, *39*(10), 1358-1375. doi: 10.1109/TSE.2013.28

Costa, C., & Murta, L. (2013). Version control in distributed software development: A systematic mapping study. In *2013 ieee 8th international conference on global software engineering* (pp. 90–99).

David, I., Aslam, K., Faridmoayer, S., Malavolta, I., Syriani, E., & Lago, P. (2021). Collaborative model-driven software engineering: a systematic update. In *2021 acm/ieee 24th international conference on model driven engineering languages and systems (models)* (pp. 273–284). doi: 10.1109/MODELS50736.2021.00035

David, I., Aslam, K., Malavolta, I., & Lago, P. (2023). Collaborative model-driven software engineering—a systematic survey of practices and needs in industry. *Journal of Systems and Software*, *199*, 111626. doi: 10.1016/j.jss.2023.111626

Debreceni, C., Bergmann, G., Búr, M., Ráth, I., & Varró, D. (2017). The mondo collaboration framework: Secure collaborative modeling over existing version control systems. In *2017 11th joint meeting on foundations of software engineering* (p. 984–988). Association for Computing Machinery. doi: 10.1145/3106237.3122829

de la Vega, A., & Kolovos, D. (2022). An efficient line-based approach for resolving merge conflicts in xmi-based models. *Software and Systems Modeling*, *21*(6), 2461–2487. doi: 10.1007/s10270-022-00976-4

Dewan, P., & Hegde, R. (2007). Semi-synchronous conflict detection and resolution in asynchronous software development. In *10th european conference on computer-supported cooperative work* (pp. 159–178). doi: 10.1007/978-1-84800-031-5_9

Dias, K., Borba, P., & Barreto, M. (2020). Understanding predictive factors for merge conflicts. *Information and Software Technology*, *121*, 106256. doi: https://doi.org/10.1016/j.infsof.2020.106256

Di Ruscio, D., Franzago, M., Malavolta, I., & Muccini, H. (2017). Envisioning the future of collaborative model-driven software engineering. In *2017 ieee/acm 39th international conference on software engineering companion* (pp. 219–221). doi: 10.1109/ICSE-C.2017.143

Diskin, Z., König, H., & Lawford, M. (2018). Multiple model synchronization with multiary delta lenses. In A. Russo & A. Schürr (Eds.), *Fundamental approaches to software engineering* (pp. 21–37). doi: 10.1007/978-3-319-89363-1_2

Edwards, G., & Medvidovic, N. (2008). A methodology and framework for creating domain-specific development infrastructures. In *2008 23rd ieee/acm international conference on automated software engineering* (pp. 168–177).

Estler, H. C., Nordio, M., Furia, C. A., & Meyer, B. (2014). Awareness and merge conflicts in distributed software development. In *2014 ieee 9th international conference on global software engineering* (p. 26-35). doi: 10.1109/ICGSE.2014.17

Franzago, M., Di Ruscio, D., Malavolta, I., & Muccini, H. (2017). Collaborative model-driven software engineering: a classification framework and a research map. *IEEE Transactions on Software Engineering*, *44*(12), 1146–1175. doi: 10.1109/TSE.2017.2755039

Guimarães, M. L., & Silva, A. R. (2012). Improving early detection of software merge conflicts. In *2012 34th international conference on software engineering (icse)* (p. 342-352). doi: 10.1109/ICSE.2012.6227180

Herac, E., Assunção, W. K. G., Marchezan, L., Haas, R., & Egyed, A. (2023, July). A flexible operation-based infrastructure for collaborative model-driven engineering. *Journal of Object Technology*, *22*(2), 2:1-14. (The 19th European Conference on Modelling Foundations and Applications (ECMFA 2023)) doi: 10.5381/jot.2023.22.2.a5

Herac, E., Marchezan, L., Assunção, W., & Egyed, A. (2024, February). *Conflict-based Change Awareness for Collaborative Model-driven Software Engineering (Evaluation Data).* Zenodo. doi: 10.5281/zenodo.10699447

Hutchinson, J., Rouncefield, M., & Whittle, J. (2011). Model-driven engineering practices in industry. In *33rd international conference on software engineering* (pp. 633–642).

Jackson, V., Prikladnicki, R., & van der Hoek, A. (2024). Co-creation in fully remote software teams. In *46th ieee/acm international conference on software engineering* (pp. 1–12). doi: 10.1145/3597503.3623297

Kanagasabai, N., Alam, O., & Kienzle, J. (2018). Towards online collaborative multi-view modelling. In *System analysis and modeling. languages, methods, and tools for systems engineering: 10th international conference* (pp. 202–218). doi: 10.1007/978-3-030-01042-3_12

Kuiter, E., Krieter, S., Krüger, J., Saake, G., & Leich, T. (2021). varied: an editor for collaborative, real-time feature modeling. *Empirical Software Engineering*, *26*, 1–47. doi: 10.1007/s10664-020-09892-x

Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., Thomason, C., … Volgyesi, P. (2001). The generic modeling environment. In *Workshop on intelligent signal processing* (Vol. 17, p. 2001).

Lindholm, T. (2004). A three-way merge for xml documents. In *2004 acm symposium on document engineering* (p. 1–10). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/1030397.1030399

Marchezan, L., Assunção, W. K. G., Herac, E., Keplinger, F., Egyed, A., & Lauwerys, C. (2023). Fulfilling industrial needs for consistency among engineering artifacts. In *45th international conference on software engineering - software engineering in practice* (pp. 1–12).

Marchezan, L., Kretschmer, R., Assunção, W. K., Reder, A., & Egyed, A. (2022). Generating repairs for inconsistent models. *Software and Systems Modeling*, 1–33.

McKee, S., Nelson, N., Sarma, A., & Dig, D. (2017). Software

practitioner perspectives on merge conflicts and resolutions. In *2017 ieee international conference on software maintenance and evolution (icsme)* (pp. 467–478).
Mens, T. (2002). A state-of-the-art survey on software merging. *IEEE transactions on software engineering*, *28*(5), 449–462.
Mistrík, I., Grundy, J., Van der Hoek, A., & Whitehead, J. (2010). *Collaborative software engineering: challenges and prospects*. Springer.
Muccini, H., Bosch, J., & van der Hoek, A. (2018). Collaborative modeling in software engineering. *IEEE Software*, *35*(6), 20–24.
Nelson, N., Brindescu, C., McKee, S., Sarma, A., & Dig, D. (2019). The life-cycle of merge conflicts: processes, barriers, and strategies. *Empirical Software Engineering*, *24*, 2863–2906.
Ogunyomi, B., Rose, L. M., & Kolovos, D. S. (2015). Property access traces for source incremental model-to-text transformation. In *European conference on modelling foundations and applications* (pp. 187–202).
Owhadi-Kareshk, M., Nadi, S., & Rubin, J. (2019). Predicting merge conflicts in collaborative software development. In *2019 acm/ieee international symposium on empirical software engineering and measurement* (pp. 1–11).
Ráth, I., Hegedüs, Á., & Varró, D. (2012). Derived features for emf by integrating advanced model queries. In *European conference on modelling foundations and applications* (pp. 102–117).
Reiter, T., Altmanninger, K., Bergmayr, A., Schwinger, W., & Kotsis, G. (2007). Models in conflict-detection of semantic conflicts in model-based development. In *International workshop on model-driven enterprise information systems* (Vol. 7, pp. 29–40).
Sarma, A., Bortis, G., & Van Der Hoek, A. (2007). Towards supporting awareness of indirect conflicts across software configuration management workspaces. In *22nd ieee/acm international conference on automated software engineering* (pp. 94–103).
Sarma, A., Redmiles, D. F., & Van Der Hoek, A. (2011). Palantir: Early detection of development conflicts arising from parallel code changes. *IEEE Transactions on Software Engineering*, *38*(4), 889–908.
Schümmer, T., & Haake, J. M. (2001). Supporting distributed software development by modes of collaboration. In *Seventh european conference on computer supported cooperative work* (pp. 79–98).
Sharbaf, M., Zamani, B., & Sunyé, G. (2022). Conflict management techniques for model merging: a systematic mapping review. *Software and Systems Modeling*, 1–49.
Steinberg, D., Budinsky, F., Merks, E., & Paternostro, M. (2008). *Emf: eclipse modeling framework*. Pearson Education.
Tröls, M. A., Marchezan, L., Mashkoor, A., & Egyed, A. (2022). Instant and global consistency checking during collaborative engineering. *Software and Systems Modeling*, *21*(6), 2489–2515.
Vale, G., Hunsen, C., Figueiredo, E., & Apel, S. (2022). Challenges of resolving merge conflicts: A mining and survey study. *IEEE Transactions on Software Engineering*, *48*(12), 4964-4985. doi: 10.1109/TSE.2021.3130098
Whitehead, J. (2007). Collaboration in software engineering: A roadmap. In *Future of software engineering (fose '07)* (p. 214-225). doi: 10.1109/FOSE.2007.4
Yohannis, A., Kolovos, D., & Polack, F. (2017). Turning models inside out. In *Ceur workshop proceedings 1403* (pp. 430–434).
Yohannis, A., Rodriguez, H. H., Polack, F., & Kolovos, D. (2019). Towards efficient comparison of change-based models. *Journal of Object Technology*, 1–21.
Young Bang, J., Brun, Y., & Medvidović, N. (2018). Collaborative-design conflicts: Costs and solutions. *IEEE Software*, *35*(6), 25–31.

About the authors

Edvin Herac is a PhD student at the Institute of Software Systems Engineering (ISSE) at the Johannes Kepler University Austria, supervised by Prof. Dr. Alexander Egyed. He obtained his master degree in Computer Science from the Johannes Kepler University Linz (JKU). His research focuses on Model-based Software Engineering, particularly on Model-based Conflict Management. You can contact the author at edvin.herac@outlook.com or visit https://edvher.github.io/.

Luciano Marchezan is currently a University Assistant at the Institute of Software Systems Engineering (ISSE) at the Johannes Kepler University Austria. He has completed his Ph.D. in Computer Science (2023) at the Johannes Kepler University (JKU), Austria. During the research of his thesis, he investigated how to improve the consistency maintenance process of a collaborative software system engineering environment, investigating, designing, and evaluating solutions that support automated consistency checking and the generation of repair recommendations. His research interests include Collaborative Model-Driven Software Engineering, Software Reuse and Empirical Software Engineering. You can contact the author at lucianomarchp@gmail.com or visit https://lucianomarchezan.github.io/.

Wesley Klewerton Guez Assunção is an Associate Professor with the Department of Computer Science at North Carolina State University. Wesley received his M.Sc. in Informatics (2012) and Ph.D. in Computer Science (2017) both from Federal University of Paraná (UFPR) - Brazil. His areas of interest are Software Modernization, Variability Management, Collaborative Engineering of Complex Systems, Software Testing, and Search Based Software Engineering. You can contact the author at wguezas@ncsu.edu or visit https://wesleyklewerton.github.io/.

Alexander Egyed is a Full Professor for Software-Intensive Systems at the Johannes Kepler University, Austria. He received his Doctorate from the University of Southern California, USA and worked in industry for many years. He is most recognized for his work on software and systems design – particularly on variability,consistency, and traceability. You can contact the author at alexander.egyed@jku.at or visit http://www.alexander-egyed.com/.