A Delta-Oracle for Fast Model Merge Conflict Estimation using Sketch-Based Critical Pair Analysis

Karl Kegel
Technische Universität Dresden
Dresden, Germany
karl.kegel@tu-dresden.de

Andreas Domanowski
Technische Universität Dresden
Dresden, Germany
andreas.domanowski@tu-dresden.de

Kevin Feichtinger
Karlsruhe Institute of Technology
Karlsruhe, Germany
kevin.feichtinger@kit.edu

Romain Pascual
Karlsruhe Institute of Technology
Karlsruhe, Germany
romain.pascual@kit.edu

Uwe Aßmann
Technische Universität Dresden
Dresden, Germany
uwe.assmann@tu-dresden.de

ABSTRACT

Conflicting changes are a major challenge in branch-based development and modeling. State-of-the-art research proposes continuous analysis via attempted three-way merges to find potential merge conflicts early on. These approaches are computation-heavy due to the necessity of comparing all variant combinations, ideally for each change. This work proposes a conflict approximation algorithm (oracle) for quick feedback. The oracle approximates conflicts using critical pair analysis on tracked delta sequences, providing a quick feedback loop. The oracle is paired with a classical slow-but-precise full model comparison algorithm, which is run occasionally to validate the oracle's results. This work contributes the *Sketch-based Critical Pair Analysis (SCPA)* approach for fast merge conflict estimation. SCPA's runtime depends only on the number of changes and not the model size. We evaluate SCPA against *EMFCompare* in different simulated model evolution scenarios. We found that for the investigated model sizes, SCPA is faster by a magnitude while the number of found conflicts strongly correlates with *EMFCompare*.

CCS CONCEPTS

• **Software and its engineering** → *Software creation and management*; *Development frameworks and environments*; • **Theory of computation** → **Theory and algorithms for application domains**.

KEYWORDS

merge conflict estimation, critical pair analysis, oracle algorithm

ACM Reference Format:
Karl Kegel, Andreas Domanowski, Kevin Feichtinger, Romain Pascual, and Uwe Aßmann. 2024. A Delta-Oracle for Fast Model Merge Conflict Estimation using Sketch-Based Critical Pair Analysis. In *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24), September 22–27, 2024, Linz, Austria.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3652620.3688341

MODELS Companion '24, September 22–27, 2024, Linz, Austria

ACM ISBN 979-8-4007-0622-6/24/09
https://doi.org/10.1145/3652620.3688341

1 INTRODUCTION

Agile, distributed development workflows are widely established in today's software engineering landscape. Many variations of branch-based workflows are known in the literature[9]. These workflows propose various solutions to merge individually edited branches. While merge and versioning strategies and techniques are well-studied [1, 2, 4, 21], merge conflicts remain an issue that must be solved manually.

We distinguish between *online* and *offline* workflows and tasks. Offline refers to everything happening on the local system and online to everything happening in a synchronized or commonly accessible context. We use the more general term *variant* to describe variations in space, e.g., *branches*. The merge problem is typically addressed by timely detection and reporting of conflicts:

(1) A manual conflict analysis is conducted once a change is committed to the repository. Open (draft) merge/pull requests show conflicts with other branches. These requests are managed manually or by a CI pipeline.
(2) IDE-supported feedback, e.g., an (online) conflict detection tool, continuously collects changes made by the collaborators. Merge attempts are executed in the background, and conflicts are reported in the IDE. Guimarães and Silva designed such a system for software development [12].
(3) Metric-based, e.g., the Drift metric proposed in [17] measures merge conflict potential. These metrics are computed after specific events, e.g., in the CI after a commit, or continuously. The actual computation typically requires a merge attempt between all branches for each change.

The literature studies branch-based workflows for developing and maintaining code. These workflows are also applicable and used in modeling. Since models have a structured representation often based on a graphical syntax, modeling environments (language workbenchs) are more constrained than programming environments (code editors). Modeling relies on tools that support the modeling language, understand the metamodel, and provide a graphical modeling interface. For instance, *EMF*-based tools in *Eclipse*[1] comprise graphical and DSL editors, version control via *EGit*[2], and model comparison and merging with *EMFCompare*[3].

[1] https://eclipseide.org/
[2] https://eclipse.dev/egit/
[3] https://eclipse.dev/emf/compare/

1.1 Problem Statement

The approaches for detecting and reporting merge conflicts by consecutive merge attempts are resource-intensive. In the worst case, a full conflict detection cycle requires merge attempts for all variant pairs, leading to $n^2 - n$ comparisons for n variants or $n^2/2 - n$ assuming symmetrical merge operations. Ideally, a full conflict detection cycle is executed for each change. Consequently, a huge number of three-way merges must be conducted.

The default *Git* algorithm for distinguishing and merging is line-based and textual. In this algorithm, two documents are matched line by line using string comparison. These algorithms have well-known downsides when comparing documents with complex syntax. The unawareness of the syntax leads to a high number of false positives [2]. However, an unstructured difference is fast, and its results remain usable for practitioners during manual analysis. Having a fast algorithm for model comparison would solve the online-analysis problem. However, unstructured but fast difference algorithms are not feasible for comparing models. For example, *Ecore* models are typically saved as XMI files. XMI is hardly human-readable and model elements frequently change their position in the underlying XML tree. Position-based indices are used to define references. A line-based difference between two XMI files is not suitable for conflict detection.

For models, structured difference algorithms exist, e.g. *EMFCompare*. They are aware of the models' metamodel and work on the abstract syntax tree (AST). These algorithms are slow compared to line-based algorithms. In particular, finding a subgraph matching, i.e., a subgraph isomorphism between two graphs is NP-complete [7, 10]. Practical model comparison algorithms mostly rely on graph comparisons, e.g., the extension of the *VF2* algorithm in [8].

This makes approaches like the *Drift* for projects with large models and many variants unsuitable for providing live feedback. There is a need for a fast comparison approach.

1.2 Research Questions

We therefore aim to answer the research question *RQ1*:

RQ1 What is a suitable design for a language-agnostic approach for fast but dependable merge conflict counting for three-way model merges?

In the context of answering this main research question, we ask two subsequent questions RQ1.1 and RQ1.2:

RQ1.1 What is a suitable language-agnostic oracle algorithm for conflict estimation trading accuracy with speed?

RQ1.2 How well does such an estimation algorithm perform regarding runtime and accuracy compared to full model comparison?

1.3 Approach

We argue that, in the short term, e.g., during active collaboration, fast feedback is more important than minor inaccuracies in the metric. However, in the long term, exact conflict reports are required for strategic decisions. Of course, quantifying "major" and "minor" is subjective and should be discussed in a scenario-specific context. Given the above restrictions, the problem of locating merge conflicts can be reduced to estimating the number of merge conflicts. Therefore, we propose an oracle approach combining a slow, accurate model merging (dependable results) with a fast conflict estimation (fast feedback). We propose a variation of *Critical Pair Analysis* [13, 19, 20] for the fast lane. We analyze the simplified representation of the changes (deltas) made to the models rather than the models themselves. Thus, we propose the *Sketch-based Critical Pair Analysis (SCPA)* algorithm. In parallel, we use the standard *EMFCompare* algorithm for the slow lane.

We conduct time and accuracy measurements based on simulated models and delta sequences showing the feasibility of SCPA. Therefore, we conduct a series of experiments using simulated datasets. We correlate the conflict counts found by SCPA with the conflict counts found by *EMFCompare*. The results show strong correlations with SCPA being significantly faster.

1.4 Method

First, we describe and abstract our problem space in Section 2. We present the concept of the hybrid approach in Section 3 and the design of the oracle algorithm in Section 4. Section 5 presents the evaluation and Section 6 discusses related work. We conclude the paper and outline the next steps in our research in Section 7.

This work contributes the abstract concept of a hybrid conflict estimation pipeline. Particularly, we contribute the novel *Sketch-Based Critical Pair Analysis (SCPA)* for fast merge conflict estimation.

2 FOUNDATIONS

This section formalizes the problem space and then discusses concepts relevant to our solution. We describe the delta-based model evolution scenario according to the formalization and definitions of *Abstract Delta Modeling*[6]. The basis for an evolution scenario is a model $m \in \mathcal{M}$. The metamodel $\mathcal{M}$ is defined as the set of all its valid instances. The empty model ϵ is a mandatory element of $\mathcal{M}$.

Models are modified via atomic edits, called *delta operations*, d applicable to models in $\mathcal{M}$. The finite set of delta operations is called a *delta language* and written $\mathcal{D}$. Applying a delta operation often requires pre-existing structures within the model, which could only be added by prior delta operations. For instance, if d_n adds a node n and d_e adds an edge e incident to n, then d_n must precede d_e. This requirement is denoted by the strict partial order $\prec$, making the delta language a partially ordered set $(\mathcal{D}, \prec)$.

Delta operations may not always produce models $m \in \mathcal{M}$. For example, applying d_e to a model without the node n results in an ill-formed model. Thus, *deltas* aggregate delta operations into model transformations $\delta : \mathcal{M} \to \mathcal{M}$, ensuring that applying all operations of a delta to a well-formed model yields another well-formed model. Formally, a delta acts as a partial function applicable only to some models. For instance, a delta containing the operation removing the node n but lacking the operation that adds n is only applicable to models already containing n.

Deltas inherit the poset structure from $\mathcal{D}$. Applying a delta involves sequentially applying all its delta operations while respecting $\prec$. Formally, the application of δ is a linear extension $\prec^*$ of $\prec$ to δ, i.e., as a sequence $d_0, d_1, \ldots, d_n$ of all $d \in \delta$ such that $d_i \prec d_{i+1}$ for all i. Such linear extensions typically come from logging change operations made by a collaborator. Deltas can be composed into

sequences, called *delta sequences*, and written $\Delta = \delta_1, \ldots, \delta_n$. We define the evolution scenario according to [17]. We write m_0 for the base model of an evolution scenario. This model is evolved by a collaborator $c \in C$ via a delta sequence to reach the desired model variant m_c^τ. Conflicts may exist between the delta sequences of two collaborators. These pairs of operations are called *Critical Pairs*. For example, if $d_a \in \delta_a$ moves an element in one variant and $d_b \in \delta_b$ deletes it in another variant, then a merge conflict occurs when merging the two model variants. Consequently, (d_a, d_b) forms a critical pair.

Critical pair analysis is a well-known concept that has been studied in various works. In the context of term or graph rewriting, critical pair analysis is conducted on a rewriting system to analyze its confluence, i.e., whether the order in which the rule is applied matters [19]. It corresponds to a minimal example of a conflict application for two rules. It is intuitively computed by overlapping all elements of the rule left-hand sides modified by the two rules. Then, showing that the applications of the two rules to these minimal examples are joinable is sufficient to ensure the confluence of the system [14]. Hausmann et al. [13] use critical pair analysis to find conflicting actions between use cases. In delta-based approaches for software product line engineering, the notion of *Critical Pair* is defined as a pair of delta operations that cannot be applied together (because of a contradiction) or their application is ambiguous (different results depending on the application order) [20]. For instance, Pietsch et al. [20] use critical pair analysis to detect critical modifications in delta-based software product lines.

3 HYBRID CONFLICT MEASUREMENT

We propose a two-lane approach for merge conflict estimation. In our approach, two algorithms work in parallel. One algorithm gives complete and correct results but is consequently slow. We call this algorithm the *Validator*. The other algorithm gives an approximated result but is fast. We call this algorithm the *Oracle*.

During active development, collaborators require fast feedback to counter (or be aware of) merge conflicts timely. In this case, the oracle provides feedback as fast as possible. Each collaborator's edit operations (delta) are sent to an analysis facility. An analysis facility can be, for example, a remote service, a local service, or a CI pipeline. This facility continuously runs the oracle and occasionally runs the validator to provide dependable results. State-of-the-art model comparison tools like *EMFCompare* can serve as validator algorithms. The activation of the validator happens according to a predefined strategy:

(1) *Ad-hoc:* The validator only runs if manually triggered.
(2) *Threshold:* The validator runs when a certain threshold - number of changes or conflicts - is reached or based on a metric estimated by the oracle, such as the *Drift* metric [17].
(3) *Synchronous:* The validator runs with a specified frequency.

4 CRITICAL PAIR ANALYSIS WITH DELTA SKETCHES

The hybrid estimation approach requires an oracle algorithm for fast merge conflict estimation. Model difference algorithms typically rely on complex graph comparison. As we envision a performant yet simple algorithm, we decided against optimizing a graph comparison approach. Instead, we base our algorithm on delta sequences and critical pair analysis (CPA). In other words, the oracle counts conflicts by evaluating the changes themselves instead of the changes' manifestations in the models. We consider δs retrieved by logging the operations performed by a collaborator and assume the underlying sequence $\prec^*$ to be a valid extension of $\prec$. In the sequel, we identify $\prec^*$ and $\prec$. However, we aim to find an agnostic algorithm operating on general deltas $(\delta, \prec)$.

We consider an algorithm to be a suitable oracle if it is: (A) Language-agnostic; (B) Performant; (C) Simple. Language-agnostic means agnostic of a concrete set of delta operations. Performant means that the algorithm is significantly faster than a full model comparison. Simple means low implementation complexity. These three requirements improve the ability to embed the algorithm in existing tools and to support modifications and customization.

CPA, as studied by Mens et al. [19] or Heckel et al. [14], is not directly applicable to our problem space because we work with deltas rather than rewrite rules. A related approach is the change-based conflict detection algorithm by Yohannis [21]. The algorithm constructs *Operation Trees* from logged change operations and analyzes them for conflicts. Yohannis [21] shows that the algorithm is faster than *EMFCompare* while finding the same conflicts. However, this precision comes with the tradeoff of requiring a sophisticated, memory-intensive data structure. We argue that relaxing the requirement for precise conflict detection can lead to an algorithm that is less complex than comparable approaches and significantly faster than full model comparison.

We propose the *Sketch-based Critical Pair Analysis (SCPA)* to cover requirements A-C. The input of the SCPA is a delta sequence Δ, a *Redundancy Heuristic* H_R, and a *Critical Pair Heuristic* H_C. First, we extend the foundations from Sec. 2.

We define a domination function $\alpha : \mathcal{D} \to \mathcal{P}(\mathcal{D})$ that outputs the set of dominated delta operations for each delta operation. A concrete $d_1 \mapsto \{d_2, d_3\}$ denotes that d_2 and d_3 are dominated by d_1, meaning that d_2 and d_3 exist as a consequence of d_1. This is not to be confused with the order $\prec$ on delta operations. For example, deleting a graph node also leads to (automatically) deleting all adjacent edges. Thus, the node deletion dominates the edge deletion. Additionally, every delta operation dominates $\emptyset$. In practice, α is built for a concrete delta tracked by the associated modeling tool.

An exemplary delta operation may be $d_{ex} \in \mathcal{D}$ corresponding to: *"move node N1 from subgraph A to subgraph B"*. It contains an operation name (*move*), a main target element (*node N1*) and contextual information about the operation (from where to where). We write Σ for the set of possible operation names and consider a function $o : \mathcal{D} \to \Sigma$ providing the name of each delta operation. For example, $o(d_{ex}) = move$. We assume that model elements have a (unique) identifier and consider Ω as the set of all identifiers. We further assume that each delta operation knows the main model element that it modifies, such that we obtain an identification function $i : \mathcal{D} \to \Omega$. For example, $i(d_{ex}) = N1$. More precisely, we define a *delta operation sketch*, or simply operation sketch, $\tilde{d}$ as a pair $(o(d), i(d)) \in \Sigma \times \Omega$, reducing a delta operation to its operation name and its main target element's identifier. The key idea is to work with $o(d)$ and $i(d)$ instead of d. For example, $\tilde{d_{ex}} = (move, N1)$. While mapping the delta operations to

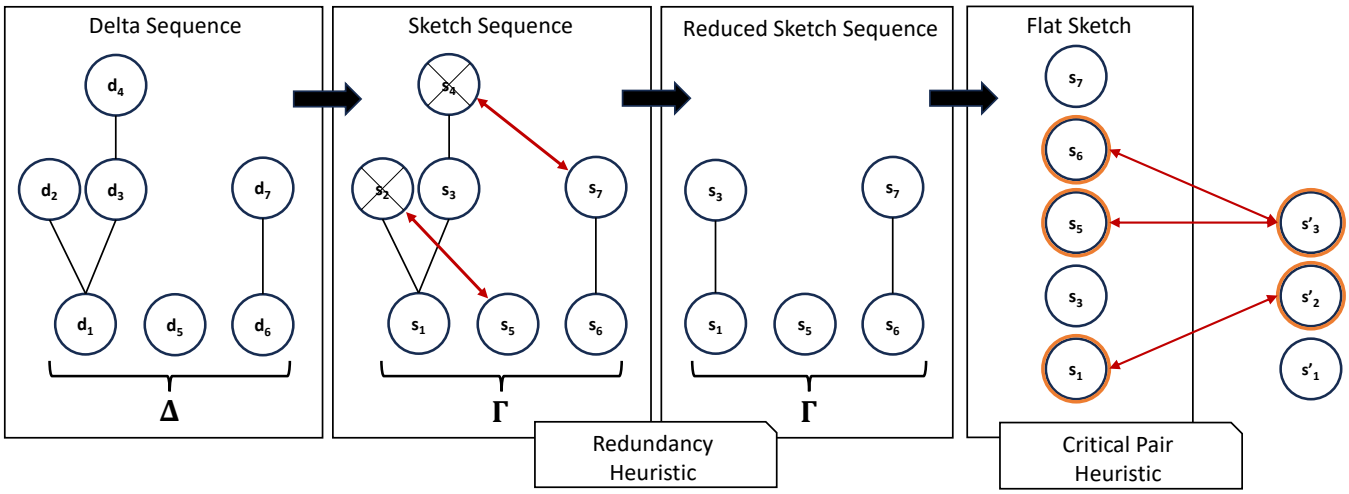

Figure 1: The abstract principle of the *Sketch-based Critical Pair Analysis*

delta operations sketches, we lift α to $\tilde{\alpha}$ and $<$ to $<$. Then, *delta sketches* are tuples $(\tilde{\delta}, <)$. When building sketches, two different delta operations may result in the same operation sketch due to the simplifications made, which would violate the order $<$. An additional identifier - e.g., based on information from the delta - is added, ensuring the uniqueness of operation sketches. A sequence of delta sketches form a *delta sequence sketch* Γ.

The *Redundancy Heuristic* H_R is a set of pairs in $\Sigma \times \Sigma$ encoding internally redundant operation sketches within a delta sequence sketch. An operation sketch $\tilde{d_1} = (o_1, i_1)$ is internally redundant if there is an operation sketch $\tilde{d_2} = (o_2, i_2)$ such that $\tilde{d_1} < \tilde{d_2}$, $i_1 = i_2$, and $(o_1, o_2) \in H_R$. For example, *(add, delete)* can belong to H_R as these operations would cancel each other out if applied to the same model element. Similarly, the *Critical Pair Heuristic* H_C is a set of pairs in $\Sigma \times \Sigma$ encoding conflicting operation sketches between two delta sequence sketches. A pair $(\tilde{d_1}, \tilde{d_2})$ is conflicting if $i_1 = i_2$ and $(o_1, o_2) \in H_C$. For example, *(move, move)* can belong to H_C as these operations may cause merge conflicts when applied to the same model element in different variants.

Based on these definitions, the algorithm works as follows. Fig. 1 depicts the SCPA process from left to right.

(1) Transform the input delta sequence into a delta sequence sketch. This reduces the size (memory) of the delta. Technically, the result is a sequence of posets of string pairs according to $\tilde{\alpha}$. This is shown by the transition from the first to the second box in Fig. 1.
(2) Eliminate internally redundant operation sketches via H_R while traversing the data structure once. Eliminating an operation sketch implies eliminating the dominated delta operations according to $\tilde{\alpha}$. Fig. 1 shows the elimination by crossing out redundant elements.
(3) Flatten the reduced data structure to a flat delta sketch, dismissing $\tilde{\alpha}$ and $<$. Technically, the result is a set of string pairs. The reduced data structure is depicted in the third box of Fig. 1, and the flatten one in the fourth box.
(4) Compare two flat delta sketches by counting the number of critical pairs via H_C for every $(\tilde{d_a}, \tilde{d_b}) \in \Gamma_1 \times \Gamma_2$.

The proposed SCPA process meets the requirements A-C. It is language agnostic in principle as it works on the generic sketch representation. The only language-specific step required is the preprocessing of Δ to Γ. After sketch construction, all operations use string comparisons. This makes the approach agnostic of technical spaces, i.e., it does not require a specific (formal/technical) framework for implementation. The approach is performant as no complex matching procedures are required. After sketch construction, the algorithm consists of only two simple traversals of the sketch structure. The first tree traversal eliminates redundant operations. The second set traversal counts the critical pairs. The algorithm has a linear space complexity and a quadratic time complexity regarding the delta sequences.

Runtime Considerations. To be useful, SCPA must be faster than a full three-way model comparison. SCPA is necessarily faster for empty delta sequences since nothing needs to be computed. A non-empty delta sequence that transforms the empty model into empty models will result in longer comparison times for the delta sequences than the model comparison of the empty models. Loosely speaking, there is a necessary trade-off between the two approaches and specific situations where one comparison is faster than the other. We assume standard sizes of models and delta sequences (given as the number of model elements involved in the model, resp. the sequence) where the sizes of the delta sequences generally remain negligible compared to the model sizes. With the smaller size of delta sequences and the use of simple representations for deltas, namely sketches, the runtime of the SCPA is faster than the full model comparison, as shown in Section 5.

Limitations. The literature in the context of graph rewriting states that a CPA finds all potential conflicts [13]. This does not apply to SCPA. (1) During sketch construction, an operation loses its context. As SCPA is unaware of the models, reconstructing the context is impossible. For example, if we add an edge between two graph

nodes, the sketch of this operation loses the knowledge about the nodes. It just knows that we added an edge with a certain ID. If an operation modifying the context induces a merge conflict, the SCPA algorithm remains unaware of the change. (2) Without looking at the underlying models in a merge scenario, it is impossible to reason about violations of the validity of the merge results. The merge of two models m_a and m_b, resulting in m_c may be conflict-free according to a perfect CPA, but the m_c is not a valid instance of the metamodel $\mathcal{M}$. The reason is that SCPA does not know the impact of deltas on constraints. For example, consider a graph model that does not allow the empty graph and a scenario where two collaborators delete a different node of a two-node graph. SCPA will not detect any problem as it only knows about the operations, not the graph. However, the merge is not possible as the merge result is not a valid graph. This problem holds in all operation-based approaches.

5 EVALUATION

This work proposes combining a fast conflict estimation algorithm (oracle) with a slow but exact model comparison tool (validator) to answer RQ1. We propose the SCPA algorithm for answering RQ1.1. This section aims to evaluate the feasibility of the SCPA approach compared to the classical model comparison. We consider the overall hybrid approach feasible if the SCPA algorithm is faster than the classical model comparison and sufficiently accurate. We implemented the SCPA algorithm in Kotlin.

This section evaluates the proposed approach by conducting a series of experiments. We evaluate the approach using a single metamodel and a corresponding delta language as an example. The delta language uses a strict total order $\prec$ for the operations. As this corresponds to logged deltas, this is a valid simplification. An evaluation spanning several modeling/delta languages is beyond the scope of this work, but we consider such an evaluation for our future work.

5.1 Objectives

This evaluation aims to accept or reject the following hypothesis.

- H1 The number of merge conflicts found by SCPA strongly correlates with the number of merge conflicts found by a full model merge via *EMFCompare* for models with uniquely identifiable model elements.
- H2 SCPA is significantly faster than a full model merge via *EMFCompare*.

Accepting H1 shows the suitability of SCPA as a conflict estimator. Accepting H2 shows the feasibility of SCPA as the fast algorithm in the proposed hybrid approach. Accepting H1 and H2 shows that SCPA is sufficiently accurate yet fast and thus suitable as an oracle algorithm. We accept H1 if the experiments show a strong correlation with a correlation coefficient > 0.9 for all investigated non-degenerate cases. We accept H2 if the critical pair analysis's runtime is significantly faster than the full model comparison's. We do not state a specific time factor at this point. However, the runtime difference must be visible under the experiment setup.

Additionally, this evaluation aims to investigate the following statements without accepting or rejecting them. We discuss the results for further research.

- S1 How is the accuracy of the delta analysis influenced by model properties, e.g., structuredness of the model, ratio between model elements, and models with or without IDs?
- S2 How is the accuracy of the delta analysis influenced by edit behavior, e.g., focused edits vs. random edits?
- S3 How does the accuracy of the delta analysis change with very long edit sequences?

5.2 Method

We conduct this evaluation by designing and executing a series of experiments. The experiments are split into experiment groups. We present the experiment design and the experiment groups in Section 5.5. We present and discuss the results in section 5.6. Additional experiments serve to discuss statements S1, S2, and S3. We require data comprising models, evolved models, and delta sequences to design and conduct the experiments. Furthermore, we require the data to have particular properties, e.g., model size, edit behavior, etc. For this purpose, we extend and use the *GraphGentool* model generator introduced in [17]. The extensions include the generation of models with and without IDs for each element and the generation of delta sequences for each simulated edit.

We develop the *Model Comparison Workbench* to compare the models and delta sequences. The workbench is a Kotlin application that directly works with the models generated by the *GraphGentool*. The application performs a full comparison via *EMFCompare* or uses our SCPA implementation, depending on its configuration. We present the tooling in the following Sections 5.3 and 5.4.

5.3 Tooling: GraphGentool

We use the *GraphGentool* to generate the model data required for the experiments. The *GraphGentool* is a model generator that generates variant sets based on an *Ecore* metamodel for hierarchical graphs. First, the tool generates a base model according to a configuration. The configuration comprises properties such as model size, average edges per node, depth vs. width of the graph, etc. The tool generates a set of variants by applying edit operations to copies of a base model. The variant generation is controlled by a configuration comprising the number of variants, the number of edits (deltas) per variant, and a probability distribution for choosing the edit operations. The export format is XMI. We extended the *GraphGentool* for this work. The default graph and delta metamodels used by the tool do not use IDs for the model elements. The only means of identification is the unique name of each node. Edges are identified by their source and target. We extended the metamodels to use IDs for each element. Fig. 2 and 3 respectively show the graph and delta metamodels with IDs. The composition relations in the delta metamodel between certain operations implement the domination relation α.

A detailed description of the *GraphGentool* can be found in its openly-available repository[4]. The provided reproduction package contains a copy of the used version of the tool [16].

5.4 Tooling: Model Comparison Workbench

We developed the *Model Comparison Workbench* to compare the models and delta sequences generated by the *GraphGentool*. The

[4] https://github.com/convidev-tud/emf-graph-gen

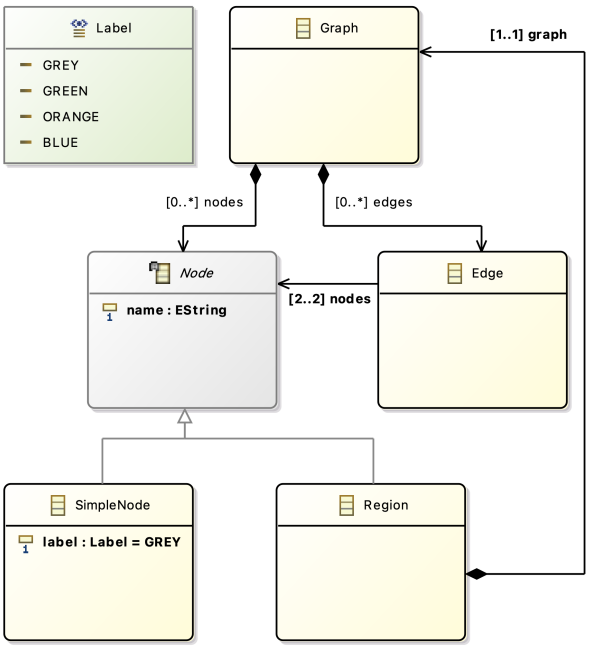

Figure 2: *Ecore* graph metamodel with full ID support used by the *GraphGentool*

reproduction package provides a copy of the source code [16]. Based on the input configuration, the workbench compares a pair of graph models (and their base) using *EMFCompare* or analyzes a pair of delta sequences using our SCPA implementation. The results, i.e., the number of conflicts found, are written in a result file. The workbench also measures the algorithm's runtime separated into load and compare time. The load time includes opening the model files, parsing the content, setting up the required *Ecore* objects, etc. The compare time includes the actual comparison task, e.g., calling *EMFCompare's* compare method or running the SCPA algorithm.

5.4.1 SCPA Implementation. First, the implementation loads the two delta sequences from XMI into *EObjects*. The *EObjects* objects are then transformed into objects of our Kotlin implementation of the delta metamodel. These objects are further transformed into the sketch data structure. The sketch construction is a language-specific transformation. An operation sketch is implemented as a data class comprising two string attributes: the element identifier and the operation name. Before the flattening, each operation sketch also knows the set of operation sketches according to the domination relation $\tilde{\alpha}$. The two heuristics H_R and H_C are implemented as lists of language-specific operation name pairs. We use the class names of the operation classes. Checking whether an operation sketch is redundant or critical is implemented as a lookup in the heuristics. The sketch data structures, reduction, and conflict detection algorithms are language-agnostic.

Table 1: Edit strategies and delta operation probabilities in the experiments.

Probability	*Balanced*	*Drag-Drop*	*CRUD*
ADD SIMPLE	15	15	20
ADD REGION	5	5	10
DELETE NODE	5	10	20
MOVE NODE	5	35	0
CHANGE LABEL	25	0	30
ADD EDGE	25	20	10
DELETE EDGE	20	15	10

Table 2: Overview of experiment group 1.

Experiment	Metamodel with IDs	Depth	Edit Strategy
E1	yes	flat	balanced
E2	no	flat	balanced
E3	yes	deep	balanced
E4	no	deep	balanced
E5	yes	flat	drag-drop
E6	no	flat	drag-drop
E7	yes	deep	drag-drop
E8	no	deep	drag-drop
E9	yes	flat	crud
E10	no	flat	crud
E11	yes	deep	crud
E12	no	deep	crud

5.5 Experiment Design

We use *GraphGentool*'s ability to vary the model size, delta length (edit size), edit behavior, and structuredness for the individual experiments. We use the graph and delta metamodels with and without IDs. Preliminary explorative experiments showed a high accuracy of the delta oracle for small deltas with zero and close to zero conflicts. We decided to "stress" the approach in the conducted experiments and not include deltas with < 20 operations in the comparison. Each delta sequence in groups 1 and 2 has up to 200 top-level elements while the base model size is 1000 elements. An experiment group is executed as follows:

(1) Generate a set of model variants according to the group's parameter specification. Each model variant set comprises two variants.
(2) For each variant set, conduct n comparisons. Each comparison selects a random evolution step (delta length) from each variant. The comparison is done using both the SCPA and *EMFCompare*. A results file is created.
(3) Collect the results and calculate the measures for analysis. This includes the correlation coefficient between the number of conflicts found by the SCPA and *EMFCompare*, and average runtimes.

We conduct 22 experiments over three experiment groups. Each experiment in groups 1 and 2 comprises 20 samples. Each experiment in group 3 comprises 10 samples. We conduct 10 comparisons per sample. This leads to 200 comparisons per experiment (E_) per comparison strategy for groups 1 and 2. It leads to 100 comparisons per experiment (E_) per comparison strategy for group 3.

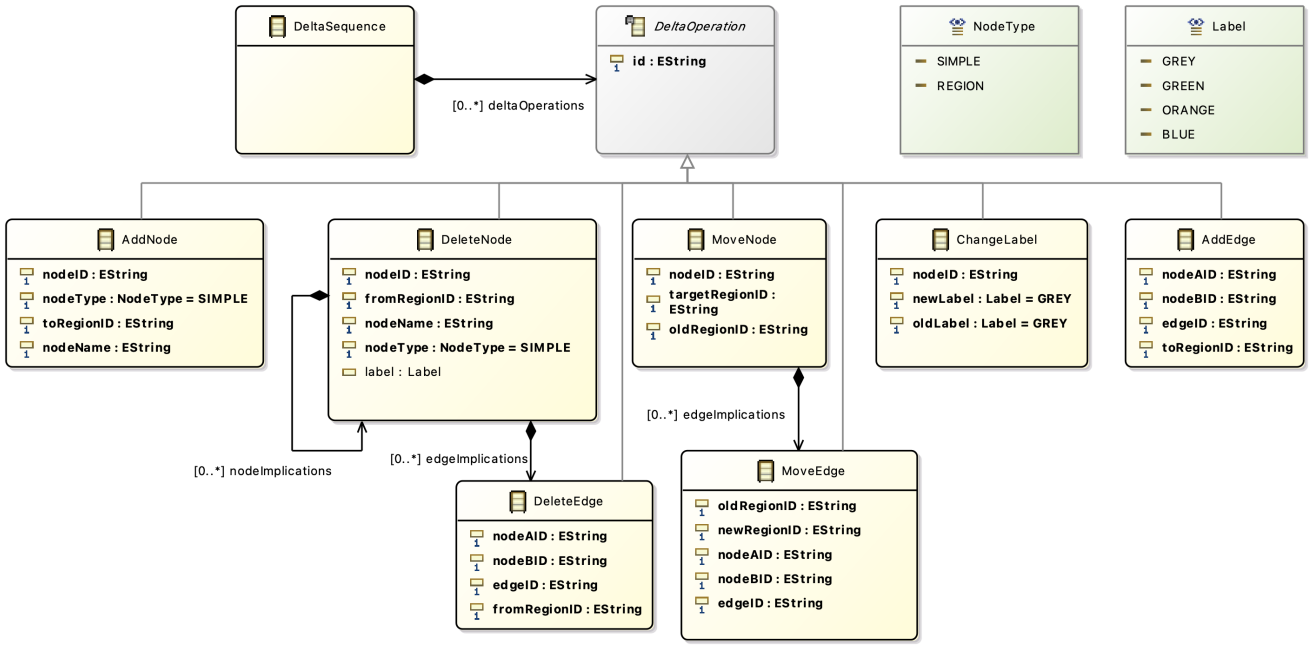

Figure 3: *Ecore* delta metamodel with full ID support used by the *GraphGentool*

Table 3: Overview of experiment group 2.

Experiment	Inherits From	Edit Focus
E13	E1	random
E14	E3	random
E15	E1	strict
E16	E3	strict

Table 4: Overview of experiment group 3.

Experiment	Inherits From	Max Delta Operations
E17	E1	1000
E18	E3	1000
E19	E5	1000
E20	E7	1000
E21	E9	1000
E22	E11	1000

5.5.1 Experiment Group 1. The first group of experiments aims to evaluate the hypothesis H1 and H2. This group also supports the discussion of S1. All experiments in this group have the same model size (1000), edit focus (0.75), and the ratio of nodes and edges (1:2). We vary the model structure (flat vs. deep), the edit strategy, and the metamodels (with and without IDs). Flat models are generated with a low probability that nodes are regions (0.05) and deep models with a high probability (0.25).

We designed three different edit strategies: *balanced*, *drag-drop*, and *crud*. These strategies define the probabilities used to choose the delta operations during the edit simulation in the *GraphGentool*. The probabilities are presented in Table 1. We argue that these strategies represent real-world behavior in model editing. As we did not find suitable empirical data, we base these assumptions on our own modeling experiences and observations. The overview of experiment group 1 is presented in Table 2.

5.5.2 Experiment Group 2. This group of experiments aims to analyze S2 further. Therefore, we repeat the experiment runs E1 and E3 from group 1 with different values for the edit focus. We use E1 and E3 as a base because of the balanced edit strategy and the more accurate ID variation of the models. The edit focus defines the probability that the next edit will be made in the same *Region* where the previous edit started. E1 and E3 use a focus of 0.75, which we consider a balanced focus. In this group, we use additional values of 0.0 (random edits) and 0.9 (strict focus). We do not choose a strict focus of 1.0, which would lead to very few merge conflicts, preventing a meaningful comparison. Table 3 shows the experiment descriptions.

5.5.3 Experiment Group 3. This group of experiments aims to investigate S3, i.e., the behavior of the accuracy for long delta sequences. Therefore, we repeat E1 and E4 with delta sequences up to 1000 (top-level) deltas. For this group of experiments, we reduce the sample size to 10, which leads to 100 comparison points per experiment. Table 4 shows the experiment descriptions.

5.6 Results

This section presents the results of the experiment groups. Table 5 contains the results of all experiments. Values of particular interest are printed in bold. The following sections discuss the presented results in detail.

5.6.1 General Observations. The results shown in Table 5 allow for several observations. First, the correlation coefficient between the number of conflicts found by the SCPA and *EMFCompare* is

Table 5: Experiment results of all groups. The C values show the correlation coefficient. The T values show the runtime in milliseconds. Values in brackets show the standard deviation in milliseconds.

Experiment	C Graph/δ	C Graph/δ (> 0)	C δ/Length	T Setup Graph	T Compare Graph	T Setup δ	T Compare δ
E1	**0.917**	0.912	0.367	254 (24)	125 (19)	167 (13)	14 (1)
E2	0.773	0.773	0.461	174 (3)	2460 (438)	158 (3)	14 (1)
E3	0.857	0.831	0.425	164 (16)	**116** (15)	163 (11)	14 (1)
E4	0.550	0.550	0.545	177 (3)	1536 (533)	158 (3)	14 (1)
E5	0.879	0.869	0.635	147 (18)	120 (16)	166 (13)	15 (1)
E6	0.236	0.236	0.115	174 (3)	1840 (399)	160 (3)	16 (1)
E7	0.865	0.858	0.722	267 (18)	126 (18)	169 (12)	15 (1)
E8	0.537	0.537	0.424	176 (3)	**2407** (392)	161 (3)	16 (1)
E9	**0.918**	0.914	0.457	246 (20)	127 (19)	165 (12)	14 (1)
E10	0.641	0.641	0.512	173 (4)	2183 (376)	158 (3)	14 (1)
E11	0.896	0.876	0.464	261 (20)	129 (20)	166 (13)	15 (2)
E12	0.740	0.740	0.566	175 (3)	1414 (483)	158 (3)	15 (1)
E13	0.891	0.881	0.299	235 (6)	**109** (9)	155 (3)	13 (1)
E14	0.819	0.805	0.442	253 (8)	**109** (10)	155 (3)	13 (1)
E15	**0.928**	0.921	0.335	236 (8)	**106** (9)	155 (3)	13 (1)
E16	0.863	0.803	0.230	256 (9)	**104** (9)	155 (3)	13 (1)
E17	**0.952**	0.952	0.331	240 (9)	146 (12)	168 (4)	29 (7)
E18	**0.948**	0.948	0.708	153 (8)	147 (16)	169 (3)	29 (7)
E19	0.827	0.827	0.638	221 (10)	152 (11)	172 (5)	34 (8)
E20	0.874	0.874	0.621	230 (12)	153 (14)	174 (6)	**36 (9)**
E21	**0.929**	0.929	0.330	219 (6)	147 (12)	167 (4)	29 (7)
E22	0.842	0.842	0.424	219 (10)	157 (13)	168 (4)	31 (8)

generally higher for models with IDs than those without IDs. This indicates the SCPA's unsuitability for models and deltas without IDs. Second, the correlation between SCPA and *EMFCompare* is higher than between SCPA and the delta length. This indicates a clear benefit of SCPA compared to inferring the number of conflicts alone from the number of changes. Third, the setup time for SCPA is just slightly lower than the setup time for the full model comparison. This indicates that keeping the models in memory instead of loading them for each comparison leads to a runtime benefit for both approaches. Fourth, the compare time for SCPA is lower by a magnitude compared to the full model comparison via *EMFCompare*. Consequently, we accept H2.

Notably, the setup time and compare time of SCPA only depend on delta size. The setup and comparison time of the full model depends on the model size. Therefore, the time benefit of SCPA is necessarily more pronounced for larger models. As this follows per construction, we did not investigate this in our experiments.

5.6.2 Results Group 1. Group 1 shows an overall high correlation (> 0.85) between the number of conflicts found by SCPA and *EMFCompare* for models with IDs. For experiments E1 and E9, the correlation is > 0.9. The correlation is not strongly influenced by the model parameters (S1), apart from the existence of IDs. A slight decrease in correlation is observed for the cases using the *drag-drop* edit strategy. Moving an element leads to many effects on the surroundings (context) of this element. SCPA's sketch construction loses this contextual information. This may lead to a higher number of inaccuracies. Fig. 4 shows the scatterplot of experiment E5. It shows an increase in variance with the number of conflicts.

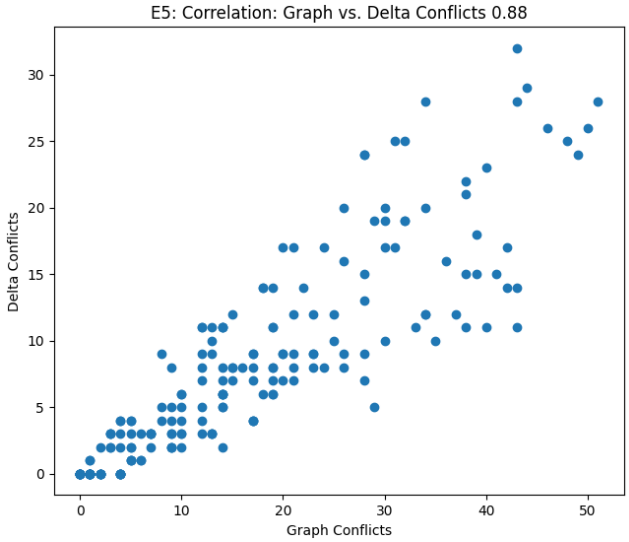

Figure 4: Scatterplot experiment E5.

However, no extreme outliers are visible apart from this range of insecurity.

We accept H1 as E1 and E9 show a correlation > 0.9. Although E3, E5, and E7 only show a correlation > 0.85 we assess this as still strong considering the intentionally large deltas.

5.6.3 Results Group 2. The results of group 2 show no clear evidence of whether the edit focus (random vs. local edits) influences

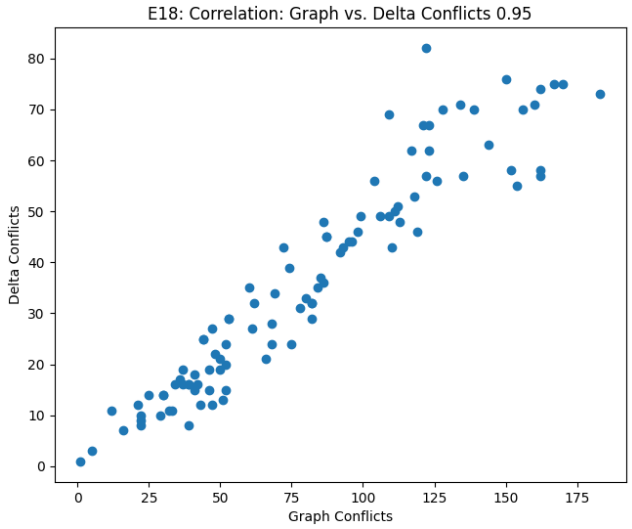

Figure 5: Scatterplot experiment E18.

the accuracy of the SCPA (S2). As visible in the raw data, the number of conflicts is lower for the strict focus. This provides weak evidence that the number of conflicts has a lower impact on SCPA than the size of the delta.

5.6.4 Results Group 3. The results show that SCPA's accuracy remains high for the investigated large delta sequences (S3). Fig. 5 shows the scatterplot of experiment E18. The scattering of the comparison points remains surprisingly low, even for larger conflict counts. We explain this by the fact that SCPA's inaccuracy is not random but systematically depends on the delta. The graph comparison time is close to the other experiments, as the graph size was unchanged. Although the average delta size is five times larger than in the other experiments, the SCPA comparison time increased only by a factor of two. We explain this the following: first, the quadratic factor of SCPA's time complexity considers the worst case; second, the quadratic factor's impact becomes only significant for delta sizes that are even larger than the analyzed ones. However, we preliminarily excluded this case from our problem space.

5.7 Summary

The experiments show that SCPA is a suitable oracle algorithm within the scope of our problem space. SCPA's conflict estimations strongly correlate with the conflict count found by full model comparison using *EMFCompare* for models with uniquely identified model elements. Furthermore, SCPA's runtime is significantly faster than the full model comparison for the investigated cases. From this, we infer the feasibility of the proposed hybrid merge conflict estimation approach. In summary, the evaluations support our answers to research questions 1.1 and 1.2.

5.8 Threats to Validity

The first threat is the possibility of systematic errors in the experiment environment. We conducted the experiments on a freshly set-up Ubuntu system. No memory- or CPU-intensive tasks were running in parallel. The machine ran below 20% CPU and 5% memory load while executing the experiments. Repetitions of the experiments on the same machine showed only minimal runtime deviations. Repetitions on a slower machine led to slower comparison times. The correlation coefficients showed only minimal deviations. Consequently, we assess the influence of the experiment environment as negligible

The second threat is the possibility of systematic errors in the experiment automation and tooling. The experiment relies on data generated by the *GraphGentool* and a fully automated comparison pipeline. Therefore, the generator or experiment code may contain bugs, leading to wrong results. A test suite covers critical parts of the *GraphGentool*. Samples of the generated data and comparison results were manually reviewed. Overall, the experiment results are plausible, and no inexplicable behavior was observed. We provide the full code of the model generator and experiment as a reproduction package for evaluation and use by other researchers.

The third threat is the bias from analyzing only two graph- and delta metamodels. The presented results are obtained using a single metamodel and delta language (with and without IDs). We countered this threat by varying the model properties and editing behavior in the experiments. Furthermore, we argue that the metamodel of a hierarchical graph used represents a wide range of practical metamodels. However, we cannot generally state that the results are transferable to other metamodels and delta languages. Therefore, further research is required.

5.9 Verifiability

We provide a reproduction package containing copies of the software tools used, the experiment scripts, and the documentation in [16]. As parts of the experiment rely on indeterministic sampling, the reproduction runs' results may vary slightly from the presented results. During multiple repetitions, we observed no deviations invalidating the drawn conclusions.

6 RELATED WORK

There is considerable related work which we discuss in this section.

Apel et al. [2] present an approach to cope with the shortcomings of text-based conflict resolution in version control systems. For example, ordering conflicts not handled automatically by a textual difference detector and merger can be resolved by a structured merger. Apel et al. [1] optimize the said approach with *auto-tuning*. Cavalcanti et al. [4] assess how semi-structured merge strategies relate to fully structured merging regarding accuracy and performance.

Kolovos et al. [18] present a survey of approaches for model differencing. They report about common steps in the differencing process. Usually, differences must be calculated, represented, and visualized. Besides matching based on the signature, i.e., the features of a model element, there are similarity-based approaches, approaches that take the semantics of a model into account, and identifier-based matching. It is stated that there is no single optimal solution to solve the problem of model matching and that trade-offs are required.

Dig et al. [11] improve syntactical differencing and merging by assigning unique identifiers to parse tree elements. Text-based

difference comparators track source code entities by their name. Extensive refactoring operations like renaming can result in version control systems being unable to track changes correctly anymore. The name-independent identification of source elements enables semantics-based refactoring, differencing, and merging. Yohannis [21] presents a change-based conflict detection algorithm based on *Operation Trees*. It works by constructing a tree of dependent change operations and finding conflicts using a traversal algorithm. The algorithm is used by Herac et al. [15] to realize a collaborative modeling environment with built-in conflict detection.

Cicchetti et al. [5] describe an approach that automatically derives a metamodel that specifies the structure of differences between model instances for a given metamodel. Models conforming to this metamodel specify usual evolution steps like adding, deleting, and changing model elements.

Barkowsky and Giese [3] propose to detect merge conflicts and check the well-formedness of model versions by encoding models as graphs with unique identifiers for both nodes and edges, such that model modifications become graph rewriting. All model versions are incorporated into a multi-version model encoded as a single graph, which is used for detecting merge conflicts with a quadratic algorithm (concerning the size of the multi-version model).

7 CONCLUSION

This work proposes a hybrid approach for estimating merge conflicts in collaborative scenarios where response time is crucial. We propose to combine a slow but accurate full model comparison with a fast but less accurate conflict estimation algorithm. We conceptualize the *Sketch-Based Critical Pair Analysis* algorithm for fast conflict estimation. The algorithm drastically simplifies the idea of change-based conflict detection using operation sketches. The experimental evaluation shows SCPA's high accuracy while significantly faster than the full model comparison. Therefore, we conclude the feasibility of the proposed hybrid approach using SCPA as the oracle algorithm. Future work has to focus on realizing the hybrid approach in a collaborative modeling environment. Furthermore, SCPA should be evaluated on larger real-world metamodels and delta sequences.

ACKNOWLEDGMENTS

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - CRC 1608 - 501798263.

REFERENCES

[1] Sven Apel, Olaf Leßenich, and Christian Lengauer. 2012. Structured merge with auto-tuning: balancing precision and performance. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering* (Essen, Germany) *(ASE '12)*. Association for Computing Machinery, New York, NY, USA, 120–129. https://doi.org/10.1145/2351676.2351694

[2] Sven Apel, Jörg Liebig, Benjamin Brandl, Christian Lengauer, and Christian Kästner. 2011. Semistructured merge: rethinking merge in revision control systems. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering* (Szeged, Hungary) *(ESEC/FSE '11)*. Association for Computing Machinery, New York, NY, USA, 190–200. https://doi.org/10.1145/2025113.2025141

[3] Matthias Barkowsky and Holger Giese. 2022. Towards Development with Multi-version Models: Detecting Merge Conflicts and Checking Well-Formedness. In *Graph Transformation (Lecture Notes in Computer Science)*, Nicolas Behr and Daniel Strüber (Eds.). Springer International Publishing, Cham, 118–136. https://doi.org/10.1007/978-3-031-09843-7_7

[4] Guilherme Cavalcanti, Paulo Borba, Georg Seibt, and Sven Apel. 2019. The Impact of Structure on Software Merging: Semistructured Versus Structured Merge. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE Press, San Diego, California, 1002–1013. https://doi.org/10.1109/ASE.2019.00097

[5] Antonio Cicchetti, Davide Di Ruscio, Alfonso Pierantonio, et al. 2007. A metamodel independent approach to difference representation. *Journal of Object Technology* 6, 9 (2007), 165–185.

[6] Dave Clarke, Michiel Helvensteijn, and Ina Schaefer. 2010. Abstract delta modeling. *ACM Sigplan Notices* 46, 2 (2010), 13–22.

[7] Stephen A. Cook. 1971. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing (STOC '71)*. Association for Computing Machinery, New York, NY, USA, 151–158. https://doi.org/10.1145/800157.805047

[8] L.P. Cordella, P. Foggia, C. Sansone, and M. Vento. 2004. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26, 10 (Oct. 2004), 1367–1372. https://doi.org/10.1109/TPAMI.2004.75 Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[9] Julio César Cortés Ríos, Suzanne M. Embury, and Sukru Eraslan. 2022. A unifying framework for the systematic analysis of Git workflows. *Information and Software Technology* 145 (2022), 106811. https://doi.org/10.1016/j.infsof.2021.106811

[10] Colin de la Higuera, Jean-Christophe Janodet, Émilie Samuel, Guillaume Damiand, and Christine Solnon. 2013. Polynomial algorithms for open plane graph and subgraph isomorphisms. *Theoretical Computer Science* 498 (Aug. 2013), 76–99. https://doi.org/10.1016/j.tcs.2013.05.026

[11] Danny Dig, Tien N. Nguyen, Kashif Manzoor, and Ralph Johnson. 2006. MolhadoRef: a refactoring-aware software configuration management tool. In *Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications* (Portland, Oregon, USA) *(OOPSLA '06)*. Association for Computing Machinery, New York, NY, USA, 732–733. https://doi.org/10.1145/1176617.1176698

[12] Mário Luís Guimarães and António Rito Silva. 2012. Improving early detection of software merge conflicts. In *2012 34th International Conference on Software Engineering (ICSE) (ICSE '12)*. IEEE Press, Zurich, Switzerland, 342–352. https://doi.org/10.1109/ICSE.2012.6227180

[13] Jan Hendrik Hausmann, Reiko Heckel, and Gabi Taentzer. 2002. Detection of conflicting functional requirements in a use case-driven approach: a static analysis technique based on graph transformation. In *Proceedings of the 24th International Conference on Software Engineering* (Orlando, Florida) *(ICSE '02)*. Association for Computing Machinery, New York, NY, USA, 105–115. https://doi.org/10.1145/581339.581355

[14] Reiko Heckel, Jochen Malte Küster, and Gabriele Taentzer. 2002. Confluence of Typed Attributed Graph Transformation Systems. In *Graph Transformation (ICGT 2002) (Lecture Notes in Computer Science)*, Andrea Corradini, Hartmut Ehrig, Hans Jörg Kreowski, and Grzegorz Rozenberg (Eds.). Springer, Berlin, Heidelberg, 161–176. https://doi.org/10.1007/3-540-45832-8_14

[15] Edvin Herac, Wesley K. G. Assunção, Luciano Marchezan, Rainer Haas, and Alexander Egyed. 2023. A flexible operation-based infrastructure for collaborative model-driven engineering. *Journal of Object Technology* 22, 2 (July 2023), 2:1–14. https://doi.org/10.5381/jot.2023.22.2.a5 The 19th European Conference on Modelling Foundations and Applications (ECMFA 2023).

[16] Karl Kegel. 2024. SCPA Evaluation Reproduction Package. https://doi.org/10.5281/zenodo.12734989

[17] Karl Kegel, Sebastian Götz, Ronny Marx, and Uwe Aßmann. 2024. A Variance-Based Drift Metric for Inconsistency Estimation in Model Variant Sets. *Journal of Object Technology* 23, 3 (July 2024), 1–14. https://doi.org/10.5381/jot.2024.23.3.a2 The 20th European Conference on Modelling Foundations and Applications (ECMFA 2024).

[18] Dimitrios S. Kolovos, Davide Di Ruscio, Alfonso Pierantonio, and Richard F. Paige. 2009. Different models for model matching: An analysis of approaches to support model differencing. In *2009 ICSE Workshop on Comparison and Versioning of Software Models (CVSM '09)*. IEEE Computer Society, USA, 1–6. https://doi.org/10.1109/CVSM.2009.5071714

[19] Tom Mens, Gabriele Taentzer, and Olga Runge. 2005. Detecting Structural Refactoring Conflicts Using Critical Pair Analysis. *Electronic Notes in Theoretical Computer Science* 127, 3 (2005), 113–128. https://doi.org/10.1016/j.entcs.2004.08.038 Proceedings of the Workshop on Software Evolution through Transformations: Model-based vs. Implementation-level Solutions (SETra 2004).

[20] Christopher Pietsch, Udo Kelter, Timo Kehrer, and Christoph Seidl. 2019. Formal Foundations for Analyzing and Refactoring Delta-Oriented Model-Based Software Product Lines. In *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A* (Paris, France) *(SPLC '19)*. Association for Computing Machinery, New York, NY, USA, 207–217. https://doi.org/10.1145/3336294.3336299

[21] Alfa Yohannis. 2020. *Change-Based Model Differencing and Conflict Detection*. Ph. D. Dissertation. University of York.