Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 99

A Summary of ReVision: History-based Model Repair Recommendations

Manuel Ohrndorf[1] Christopher Pietsch[2] Udo Kelter[3] Lars Grunske[4] Timo Kehrer[5]

Abstract: This work reports recent research results on history-based model repair recommendations in Model-Driven Engineering (MDE), originally published in Reference [Oh21]. Models in MDE are primary development artifacts that are heavily edited in all software development stages and can become temporarily inconsistent during editing. Model repair tools can support developers by proposing a list of the most promising repairs. Such repair recommendations will only be accepted in practice if the generated proposals are plausible and understandable and the set as a whole is manageable.

Our interactive repair tool ReVision [Oh18], aims at generating repair proposals for inconsistencies introduced by past incomplete edit steps. Such an incomplete edit step is either undone or extended to the full execution of a consistency-preserving edit operation. We evaluate our approach using histories of real-world models from popular open-source modeling projects. Our experimental results confirm our hypothesis that most of the inconsistencies can be resolved by complementing incomplete edits. In fact, 92.2% of the proposed complementations could be observed in the model history.

Keywords: model-driven software engineering; model repair; consistency; recommendations; history analysis

1 Summary

Model-Driven Engineering (MDE) raises the level of abstraction in software engineering by using models as primary artifacts. Thus, models in MDE are subject to continuous evolution and heavily edited during all stages of development and maintenance. As a consequence, models may get inconsistent for various reasons, e.g., due to misunderstandings when being edited collaboratively in teams. Technically, one main reason for consistency violations is the isolated editing of interrelated views or model fragments.

Inconsistencies are detected as violations of consistency rules defined for a specific modeling language. Violations of these rules can be automatically obtained using inconsistency detection techniques. While these techniques are widely established in practice, how to optimally support developers in resolving inconsistencies is still being actively discussed.

[1] Universität Bern, Switzerland manuel.ohrndorf@unibe.ch
[2] University Siegen, Germany cpietsch@informatik.uni-siegen.de
[3] University Siegen, Germany kelter@informatik.uni-siegen.de
[4] Humboldt-Universität zu Berlin, Germany grunske@informatik.hu-berlin.de
[5] Universität Bern, Switzerland timo.kehrer@unibe.ch

Repairing all inconsistencies in a single step often leads to solutions whose rationale is hard to grasp for developers. Following the generally accepted debugging strategy of fixing a single defect at a time, we iteratively repair each single violation of a consistency rule.

We assume that inconsistencies are introduced by past, incomplete editing processes that require additional changes to achieve a new consistent state. In general, there are many alternatives to resolve such inconsistencies. In such cases, recommender systems can generate a ranked list of suitable repair proposals from which a developer can choose.

Our repair recommendation tool ReVision requires specifying the transitions between *consistent* states by formal consistency-preserving edit operations (CPEOs). The main idea of our approach is to consider CPEOs as ideal edit operations and to recommend the "gap" between ideal edits and the edits which have caused an inconsistency as model repairs [Oh18, Oh21]. Therefore, incomplete edit steps are detected in the model history and can be either undone or extended to the full execution of a CPEO.

A systematic process supports the specification of CPEOs capturing typical complex edit steps, which are likely to be applied only partially, leading to model inconsistencies. Specifically, we follow an example-driven approach to manually specify sets of minimal yet valid example model fragments, which are then automatically composed into CPEOs.

We evaluate our approach using histories of real-world models obtained from popular open-source modeling projects. Our empirical study shows that, in most observable inconsistency repair cases, it is more likely that a developer wants to catch up on missing changes. In fact, 92.2% (510 complementations, 43 undos) of our repair proposals that could equally be observed (44 not observable) in the original modeling history are complementations.

In general, approaches that are not aware of the change history of a model are likely to propose repairs that just undo former changes that caused an inconsistency. Thus, a developer should be enabled to make an informed decision whether to undo former work or, if this work was just incomplete, retain and complete it.

1.1 Data Availability

ReVision and the evaluation data can be found at `https://repairvision.github.io/`.

Bibliography

[Oh18] Ohrndorf, Manuel; Pietsch, Christopher; Kelter, Udo; Kehrer, Timo: ReVision: a tool for history-based model repair recommendations. In: Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings. ACM, pp. 105–108, 2018.

[Oh21] Ohrndorf, Manuel; Pietsch, Christopher; Kelter, Udo; Grunske, Lars; Kehrer, Timo: History-based model repair recommendations. ACM Transactions on Software Engineering and Methodology (TOSEM), 30(2):1–46, 2021.