Poster: An Automated and Instant Discovery of Concrete Repairs for Model Inconsistencies

Roland Kretschmer, Djamel Eddine Khelladi, Alexander Egyed
Johannes Kepler University Linz
Linz, Austria
<firstName.lastName>@jku.at

ABSTRACT

Developers change software models continuously but often fail in keeping them consistent. Inconsistencies caused by such changes need to be repaired eventually. While we found that usually few model elements need to be repaired for any given inconsistency, there are many possible repair values for any given model element. To make matters worse, model elements need to be repaired in combination. The result is a large and exponentially growing repair space. In this paper we present an approach towards grouping alike repair values if they have the same effect to provide example-like feedback for developers. A preliminary evaluation shows that our approach can more scalably explore the repair space.

CCS CONCEPTS

• **Software and its engineering → Model-driven software engineering**;

KEYWORDS

concrete repair, abstract repair, model inconsistencies, model repair, concrete values

ACM Reference Format:
Roland Kretschmer, Djamel Eddine Khelladi, Alexander Egyed. 2018. Poster: An Automated and Instant Discovery of Concrete Repairs for Model Inconsistencies. In *ICSE '18 Companion: 40th International Conference on Software Engineering Companion, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3183440.3194979

1 INTRODUCTION

Model-Driven Engineering (MDE) has shown to be effective and beneficial in the development and maintenance of large scale and embedded systems [Hutchinson et al. 2011; Liebel et al. 2014]. These benefits, however, hinge on the models being consistent. This is a problem when changes happen. Changes may cause inconsistencies if the changes are wrong and/or incompatible. Once models are inconsistent, all reasoning with them is untrustworthy and likely even causes additional errors. Therefore, inconsistencies must at least be known to developers. Hence they must not only be detected timely but ultimately be repaired [Demuth et al. 2016; Frakes and Kang 2005; Whittle et al. 2014].

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3194979

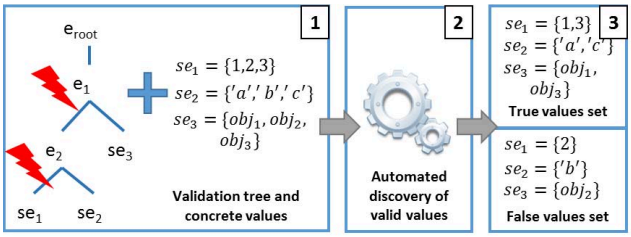

Figure 1: Overall approach.

A repair is a set of one ore more model changes that together resolve a given inconsistency. Literature distinguishes *abstract* and *concrete* repairs [Jackson 2002; Mens et al. 2006; Nentwich et al. 2001, 2003; Reder and Egyed 2012; Xiong et al. 2009]. An *abstract* repair identifies the model element(s) to repair (the location(s) in model) but does not reveal how to change the model elements. Abstract repairs are easily computed. A *concrete* repair identifies how to change the model element with a concrete value. A *concrete* repair, in contrast to an *abstract* repair, can thus be executed on the inconsistent model.

Computing concrete repairs by exploring all possible concrete values is an exponential problem, because there are many concrete repairs per abstract repair and we must consider concrete repairs in combination. So if there are n model elements and m concrete values for each model element to repair then we need to explore m^n combinations. All of these must be explored to check which ones are indeed capable of fixing the inconsistency.

When multiple values are combined to form concrete repairs, one invalid value would result in many invalid combinations (i.e., incorrect concrete repairs). Moreover, even valid values on their own may contradict each other when combined.

This paper proposes a novel approach that combines similar sets of concrete values to find valid combinations of values which can repair given inconsistencies automatically. The concrete repairs are similar if they affect the cause of the inconsistencies alike [Reder and Egyed 2013].

2 APPROACH

Our approach separates possible values for fixing inconsistencies into two sets, one with invalid values and one with valid values for all model elements that could be changed to repair a given inconsistency. This process is depicted in Figure 1.

The *first stage* (1) checks fo inconsistencies in a model based on provided consistency rules. For each inconsistency a validation tree is constructed [Reder and Egyed 2013]. The validation tree identifies all model elements involved (leaves) and shows how their values cause the inconsistency. Every validation tree consists of

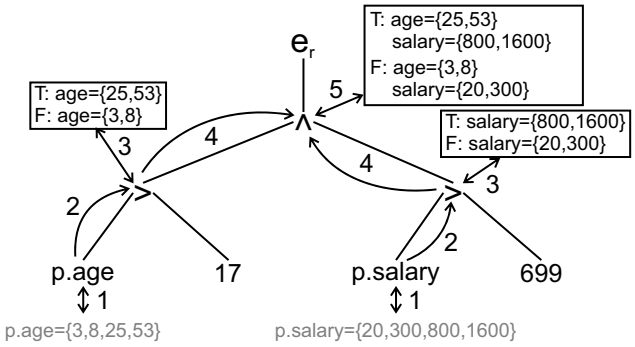

Figure 2: Validation Tree for Consistency Rule.

expressions which in turn can have sub-expressions. The leaves in a validation tree are specific model elements or constant values and contribute to the computation and detection of the inconsistency. Note that by just identifying an inconsistency with its validation tree, it is not clear how to repair it. However, all model elements causing the inconsistency are identified.

For example consider Figure 2, which shows the validation tree for the consistency rule `self.age > 17 and self.salary > 699`. Let us assume that the `p.age` is below 18 and the `p.salary` is also below 700 which leads to an inconsistency. Here (e_r) represents the root expression which every validation tree has to have. The next expression is the AND ($\wedge$) expression which consists of a left and right hand side sub tree. On the left hand side we have the greater than expression ($>$) which checks if `p.age` (expression representing the model element's p value for its property age) is indeed larger than 17 (17 is a constant value defined by the rule itself). In the right hand side sub tree the same is done for the `salary` of a model element (`p.salary`).

The *second stage* (2) suggests values for changing the individual model elements that cause the inconsistency. Some of these values may fix those inconsistencies most do not. There are several possible sources for those values (values from the model itself [Egyed et al. 2008; Kretschmer et al. 2017], and probabilistic generated values [Hegedüs et al. 2011; Xiong et al. 2009]).

This is shown in Figure 2. Assume we have values for `p.age`={3, 8, 25, 53} and `p.salary`={20, 300, 800, 1600} provided by one of the approaches mentioned above (denoted by an arrow labeled with 1).

The *third stage* (3) starts at the leaf expressions in the validation tree. For each expression, there is a `true` and a `false set`. These sets are then recursively combined to higher level expressions and their true and false sets until the root is reached.

This process is shown in Figure 2. From the previous stage we got possible values for `p.age` and `p.salary` (arrow denoted with 1). From this values sets we determine which value is able to satisfy the condition given in the consistency rules (the two greater than expressions denoted with 2). At the greater then expressions we are able to construct the corresponding `true` and `false sets` (denoted with 3) and at the AND expression we combine the sets from the two subtrees into one `true` and one `false set` (denoted with 5). Finally, the values in the `true set` at the root expression e_r can be used to form concrete repairs that can be executed automatically on the inconsistent model.

Please note that for simplicity we chose to use simple numbers to illustrate our approach. However, complex values like classes, packages, model element attributes, etc. are supported.

3 CONCLUSION AND FUTURE WORK

This paper presented a novel approach for discovering and validating values for repairing inconsistencies automatically to get relevant concrete repairs for those model inconsistencies.

For future work, we plan to validate our approach on models from industry and academia. Furthermore we plan to further group valid values based on different criteria. We also plan to consider constraints on the valid values to further reduce the size of the `true` set (e.g. among positive integers consider only odd values). Finally, as alternative repairs are proposed per inconsistency, we plan to provide ranking heuristics to support the developers in choosing repairs.

ACKNOWLEDGMENTS

This research was funded by the Austrian Science Fund (FWF): P 25513-N15

REFERENCES

Andreas Demuth, Roland Kretschmer, Alexander Egyed, and Davy Maes. 2016. Introducing Traceability and Consistency Checking for Change Impact Analysis across Engineering Tools in an Automation Solution Company: An Experience Report. In *Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on*. IEEE, 529–538.

Alexander Egyed, Emmanuel Letier, and Anthony Finkelstein. 2008. Generating and Evaluating Choices for Fixing Inconsistencies in UML Design Models. In *ASE*. 99–108. https://doi.org/10.1109/ASE.2008.20

William B Frakes and Kyo Kang. 2005. Software reuse research: Status and future. *IEEE transactions on Software Engineering* 31, 7 (2005), 529–536.

Ábel Hegedüs, Ákos Horváth, István Ráth, Moisés Castelo Branco, and Dániel Varró. 2011. Quick fix generation for DSMLs. In *VL/HCC*. 17–24. https://doi.org/10.1109/VLHCC.2011.6070373

John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. 2011. Empirical assessment of MDE in industry. In *Proceedings of the 33rd International Conference on Software Engineering*. ACM, 471–480.

Daniel Jackson. 2002. Alloy: a lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.* 11, 2 (2002), 256–290. https://doi.org/10.1145/505145.505149

Roland Kretschmer, Djamel Eddine Khelladi, and Alexander Egyed. 2017. From Abstract to Concrete Repairs of Model Inconsistencies: an Automated Approach. *APSEC* (2017).

Grischa Liebel, Nadja Marko, Matthias Tichy, Andrea Leitner, and Jörgen Hansson. 2014. Assessing the state-of-practice of model-based engineering in the embedded systems domain. In *Model-Driven Engineering Languages and Systems*. Springer, 166–182.

Tom Mens, Ragnhild Van Der Straeten, and Maja D?Hondt. 2006. Detecting and resolving model inconsistencies using transformation dependency analysis. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 200–214.

Christian Nentwich, Wolfgang Emmerich, and Anthony Finkelstein. 2001. Static Consistency Checking for Distributed Specifications. In *ASE*. 115. https://doi.org/10.1109/ASE.2001.989797

Christian Nentwich, Wolfgang Emmerich, and Anthony Finkelstein. 2003. Consistency Management with Repair Actions. In *ICSE*. 455–464. http://computer.org/proceedings/icse/1877/18770455abs.htm

Alexander Reder and Alexander Egyed. 2012. Incremental Consistency Checking for Complex Design Rules and Larger Model Changes. In *MODELS*. 202–218. https://doi.org/10.1007/978-3-642-33666-9_14

Alexander Reder and Alexander Egyed. 2013. Determining the Cause of a Design Model Inconsistency. *IEEE Trans. Software Eng.* 39, 11 (2013), 1531–1548. https://doi.org/10.1109/TSE.2013.30

Jon Whittle, John Hutchinson, and Mark Rouncefield. 2014. The state of practice in model-driven engineering. *IEEE software* 31, 3 (2014), 79–85.

Yingfei Xiong, Zhenjiang Hu, Haiyan Zhao, Hui Song, Masato Takeichi, and Hong Mei. 2009. Supporting automatic model inconsistency fixing. In *ESEC FSE*. 315–324. https://doi.org/10.1145/1595696.1595757