Tackling the Challenges of Evolution in Multiperspective Software Design and Implementation

Steffen Lehnert

Ilmenau University of Technology
Ilmenau, Germany
steffen.lehnert@tu-ilmenau.de

Matthias Riebisch

University of Hamburg
Hamburg, Germany
riebisch@informatik.uni-hamburg.de

Abstract

The design and implementation of software require the usage of different perspectives and views to cope with its static structure, dynamic behavior, and requirements. Artifacts of different views are dependent on each other and subject of frequent changes. Anticipating those changes becomes difficult, as most impact analysis approaches are not designed to work in multiperspective environments. They treat artifacts of different perspectives in isolation, which tends to introduce further inconsistencies and faults. In the related research area of consistency checking, the problem of multiple views has been addressed for a long time and solutions have been developed. We aim on combining the predictive capabilities of impact analysis with the approach of multiperspective consistency checking, to bridge the gap between artifacts of different views. We propose an approach which facilitates impact analysis of different UML models and Java source code. Our approach is based on impact propagation rules, which analyze traceability relations between software artifacts and the type of change applied by a developer.

1 Introduction

Studying the "ripple effects" of changes [6] is in the focus of research in software evolution for many years. Various techniques have been proposed to predict and assess the impacts of changes, e.g. slicing [3], history mining [7] or probabilistic analysis [2]. The majority of those approaches is limited to a certain type of software artifacts only, e.g. source code or static UML models. However, software artifacts do not evolve in isolation and ripple effects spread over the boundaries of the different views of software. Insufficient impact analysis increases the drift between high-level design and implementation, as well as gaps between different views in general. On the other hand, a considerable effort has been invested in studying consistency checking in the light of multiperspective software design, e.g. [5]. Although able to analyze multiple different artifacts, current consistency checking approaches are not able to predict the impacts of future changes, nor do they facilitate cost or time estimations. Thus, merging impact analysis with the concept of multiperspective consistency checking yields several improvements, aiding change prediction and consistency maintenance likewise. Expanding impact analysis beyond isolated views increases its ability to forecast cost estimations, and it helps developers to better understand which artifacts require rework according to a change. Improved impact analysis on the other hand results in fewer inconsistencies, if artifacts of different views are addressed adequately, thus reducing the effort required for consistency checking.

2 Multiperspective Impact Analysis

Impact analysis faces two major challenges when applied in multiperspective environments. First, dependent artifacts of different views must be joined and treated in a unified manner to enable multiperspective analysis. Secondly, it requires an impact propagation technique which is capable of analyzing multiple different artifacts, emerging from the different perspectives. Addressing the first challenge can be accomplished by mapping all artifacts on a unifying framework and using an integrating repository for managing all artifacts, which is further explained in Section 3. Using this common basis allows us to tackle the second challenge. The ripple effects of changes can be computed by impact propagation rules, which are able to address arbitrary artifacts. Our impact propagation rules analyze dependency relations which exist between artifacts to determine if and how a change propagates to related artifacts, according to the type of relation and type of applied change. Likewise, consistency management can be accomplished by applying consistency checking rules [5] on the models. Figure 1 illustrates the steps required for impact analysis, consistency management, and steps required for both activities (dashed lines). To enable this kind of analysis, we first need to elicitate the dependencies which exist between the various models and record them in a structured way. We achieve this by applying traceability mining rules which transform dependency relations into traceability links. Our traceability mining rules analyze the structure, the names, and relations

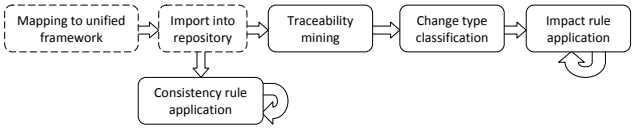

Figure 1: Overview of our approach.

between model elements, and are able to determine the type of relation which exists between them [1]. The second part of our propagation approach requires a structured treatment of change types. In previous work [4] we defined a set of change types on which real reengineering activities can be mapped upon. Using this set of changes, we defined a catalog of more than 140 change operations of different granularity, e.g. changing the return type of a method or merging UML components. Having both, traceability links and change types, we define a set of impact propagation rules which are used to compute the propagation of changes. Our rules are then applied in a recursive manner, until all impacted elements have been identified. An exemplary impact propagation rule is shown in Listing 1. Impacted elements can then be handed over to developers to perform a deeper inspection, or they are used to prepare cost and time estimations based on the amount and types of affected elements.

```
impact rule(Model a, Model b, Change c)
   report b as affected when
      a.type == "UML:Component"
      b.type == "UML:Class"
      c.type == "Rename_component"
      c.target == a
      b.relationTo(a) == "Refinement"
end
```

Listing 1: A simple impact propagation rule

3 Tool Support and Evaluation

As we are concerned with impact analysis between different UML models and Java source code, we chose the EMF-framework[1] as the common basis for our approach. We extended our prototype case tool EMF-Trace[2] for the purpose of multiperspective analysis. The tool is based on the EMFStore[3] repository and provides means for executing SQL-like rules to mine traceability relations among models [1]. Our rule-concept has been extended to carry out the task of rule-based impact analysis as explained in Section 2. Importing the required UML models and Java source code is achieved as depicted by Figure 2. We import UML models from two different CASE tools and convert their XMI-based output to an EMF-based Ecore representation using XSLT. Java source code is imported using the Ecore-mapping features provided by the MoDisco[4] tool environment. We are currently conducting a case study to evaluate our approach with a system comprised of 16500 UML model elements and Java source code artifacts. We compare the results of our approach against manual inspections, to obtain the figures for recall and precision.

[1] http://eclipse.org/modeling/emf/
[2] http://sourceforge.net/projects/emftrace/
[3] http://www.eclipse.org/emfstore/
[4] http://www.eclipse.org/MoDisco/

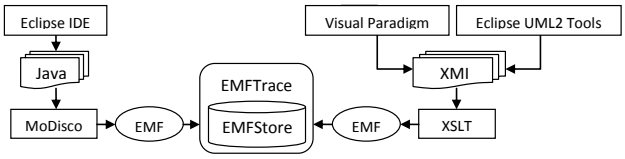

Figure 2: Our tool environment.

4 Conclusion

We outlined typical problems of impact analysis approaches when confronted with multiperspective software designs, and proposed to merge impact analysis with multiperspective consistency checking to tackle the challenges of software evolution. Using the EMF framework, we are able to map different static and dynamic UML models and Java source code on a common base to allow for multiperspective analysis. We extract dependencies from the models, which in combination with the type of applied change, serve as input for our rule-based impact propagation approach. Depending on the type of relation between two models and the change type at hand, our rules then determine further change propagation. We implemented our approach in a prototype tool and are currently performing a first case study to evaluate its performance.

References

[1] S. Bode, S. Lehnert, and M. Riebisch. Comprehensive model integration for dependency identification with EMFTrace. In *Workshop on Model-Driven Software Migration (MDSM 2011)*, pages 17–20, 2011.

[2] M. Ceccarelli, L. Cerulo, G. Canfora, and M. Di Penta. An eclectic approach for change impact analysis. In *32nd ACM/IEEE Int. Conf. on Software Engineering*, pages 163–166, 2010.

[3] K. B. Gallagher and J. R. Lyle. Using program slicing in software maintenance. *IEEE Transactions on Software Engineering*, 17(8):751–761, August 1991.

[4] S. Lehnert, Q.-U.-A. Farooq, and M. Riebisch. A taxonomy of change types and its application in software evolution. In *19th Annual IEEE Int. Conference on the Engineering of Computer Based Systems*, 2012.

[5] T. Sunetnanta and A. Finkelstein. Automated consistency checking for multiperspective software specifications. In *Workshop on Advanced Separation of Concerns (ICSE2001)*, 2001.

[6] S. S. Yau, J. S. Collofello, and T. M. McGregor. Ripple effect analysis of software maintenance. In *Computer Software and Applications Conf.*, pages 60–65, 1978.

[7] T. Zimmermann, P. Weissgerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, 31(6):429–445, June 2005.