Iterative Development and Execution of Consistency-Preserving Rule-Based Refactorings

Background

Refactorings[1] are widely accepted as a technique to restructure a program or model in order to improve its readability, extensibility, or reduce its complexity. For example, the object-oriented refactoring “Pull Up Method” is a special variant of the “Move Method" refactoring and moves a method from a sub- to a super-class. According to Fowler [1], this refactoring is used to remove duplicate behavior, which occurs in all sub-classes of a common super-class.

An important property of refactorings is that they should be behavior-preserving. However, another property that refactorings should demonstrate is *consistency preservation*[2], in the sense that specific well-formedness constraints of the language under consideration should be preserved. For example, after performing the refactoring “Pull Up Method” on an object-oriented program, the following well-formedness constraint should hold: “no two methods sharing the same signature are contained in the same class”.

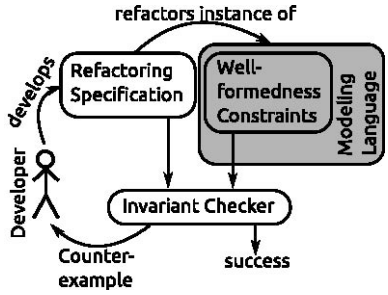
Consistency preservation of refactorings can be ensured by *runtime checks*. However, this means that not the developer of the refactorings but the user is confronted with the violation of well-formedness constraints. For example, if the ‘’Pull up Method” refactoring is wrongly implemented, the user might end up with a refactored program holding a class containing two methods sharing the same signature.

In order to avoid the user to be confronted with consistency problems, it is most desirable to be able to build on an approach capable of checking consistency preservation of refactorings *at design time*. In this way, for example, it can be ensured that after performing the refactoring “Pull Up Method”, it always holds that no two methods sharing the same signature are contained in the same class. Checks at design time become possible if refactorings are specified by means of *rules* describing which side-effects may take place under which conditions on the program or model to be refactored.

Description

The goal of this project is to investigate the *iterative development of consistency-preserving rule-based refactorings* and its *integration with a refactoring execution environment*, which is safe by construction.

To achieve this goal, the students will build on an *available back-end verification technique* [2], where rule-based refactorings are formalized using graph transformation and consistency is formalized using graph constraints [3]. This verification technique, called invariant checking, is capable of proving at design time that refactoring rule applications preserve consistency. Moreover, it automatically computes meaningful counterexamples paving the way for the iterative development of refactoring rules guaranteeing consistency preservation as sketched in the figure below.



First, the students will be responsible for implementing an *iterative development environment* for consistency-preserving rule-based refactorings based on Eclipse and the Eclipse Modeling Framework (EMF) [5]. This development environment builds on the back-end verification technique described above. Moreover, the students will develop a *refactoring execution environment,* which is safe by construction, since it is generated from consistency-preserving rule-based refactorings constructed in the iterative development environment. When developing and *evaluating* their approach, the students will focus on common Java refactorings [1], although the approach should work for all kinds of EMF-models. JaMoPP (Java Model Parser and Printer) [4] can be used to derive EMF models from Java programs.
We expect the students to have knowledge in Java programming and Eclipse. Knowledge of the Eclipse Modeling Framework is not required, but we expect the students to familiarize themselves with the framework in the course of the project.

References

1. Fowler, M.: *Refactoring: Improving the Design of Existing Code.* Object Technology Series, Addison-Wesley (1999)
2. Becker, B., Lambers, L., Dyck, J., Birth, S., Giese, H.: *Iterative Development of Consistency-Preserving Rule-Based Refactorings.* In: Proceedings of 4th International Conference on Model Transformations, ICMT 2011, Vol. 6707, p.123-137, Lecture Notes in Computer Science, Springer / Heidelberg (2011)
3. Giese, H., Lambers, L., Becker, B., Hildebrandt, S., Neumann, S., Vogel, T., Wätzoldt, S.: *Graph Transformations for MDE, Adaptation, and Models at Runtime.* To appear in: Vol. 7320, Lecture Notes in Computer Science, Springer / Heidelberg (2012)
4. Heidenreich, F., Johannes, J., Seifert, M., Wende, C.: *Closing the Gap between Modelling and Java*. In: Proceedings of 2nd International Conference on Software Language Engineering, SLE 2009, Vol. 5969, p.374-383, Lecture Notes in Computer Science, Springer / Heidelberg (2010)
5. F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T. J. Grose*: Eclipse Modeling Framework* (The Eclipse Series). Addison-Wesley (2003) (http://www.eclipse.org/emf)

Contact

Fachgebiet Systemanalyse und Modellierung

- Prof. Dr. Holger Giese
- Dr. Leen Lambers, Basil Becker, Johannes Dyck, Stephan Hildebrandt