An Approach for Refactoring Planning

Javier Pérez[1,2], Yania Crespo[2]

[1] University of Mons; Software Engineering Lab
[2] University of Valladolid; Department of Computer Science
javipeg@gmail.com, yania@infor.uva.es

Refactorings are source code transformations that change the system's internal design while preserving its observable behaviour [**?**]. Refactorings can be used to improve, in general, certain quality factors such as reusability, understandability, maintainability, etc. More specifically, refactorings can help to achieve a particular system structure or consolidating the system's architecture [**?**].

Refactoring operations are meant to be executed in small steps, so that more complex refactorings can be executed by the composition of simpler ones. Behaviour preservation is also easier to check in the case of simpler refactorings. When a refactoring process aims to solve a complex problem, such as the correction of design smells [**?**], a significant amount of changes is needed. Refactorings' preconditions can help to assure behaviour preservation, but at the same time they hinder the application of complex transformation sequences because they restrict the applicability of refactoring operations. If any precondition of any operation in the intended transformation sequence, is not fulfilled at the time of its application, the whole sequence can not be applied. This makes it hard for the developer to perform complex refactoring processes.

We have developed a technique that uses Hierarchical Task Network (HTN) Planning [**?**] to tackle this problem [**?**]. The proposal is based at the definition of refactoring strategies and refactoring plans. A Refactoring Strategy is a heuristic-based, automation-suitable specification of a complex behaviour-preserving software transformation which is aimed at a certain goal. It can be instantiated, for each particular case, into a Refactoring Plan. A Refactoring Plan is a sequence of instantiated transformations, aimed at achieving a certain goal, that can be effectively applied over a software system in its current state, while preserving its observable behaviour. It can be an instance of a Refactoring Strategy.

To develop this proposal we have focused in how refactorings are used for design smell correction. We have first analysed the current correction specifications from different authors, identified the main characteristics of these specifications and unified them in a single model. We have then identified the current problems and the requirements that correction specifications have to meet in order to be automation-suitable. We have defined a model for refactoring strategies that fulfills these requirements and defined a language in order to ease writing these specifications. We have identified the requirements an underlying approach has to meet to support the computations of refactoring plans from refactoring strategies and we have selected HTN automated planning for this purpose. We have implemented this proposal as a reference prototype and it has been evaluated by performing two case studies over a set of open-source systems.

The assembled prototype is composed of a small HTN domain, which is our main contribution to this prototype, and some third-party tools. The refactoring planning domain we have written, addresses *Feature Envy* and *Data Class* design smells and comprises the specifications of a set of refactorings, refactoring strategies, other transformations and system queries, all of which have been represented as task networks. Regarding the other tools, we have used JTransformer, a program transformation tool based in Prolog, to obtain the predicate-based representation of Java systems. We have also used iPlasma, a design smell detection tool, to obtain reports of the two design smells we have addressed. The logic-based representation of a Java program, and the information about which entities are affected of which smells constitute the initial state of the system for the planner. A set of scripts compiles the refactoring planning domain and all these inputs as a refactoring planning problem for JSHOP2, the planner we had selected. The planner searches refactoring sequences, for applying the requested strategies and produces refactoring plans.

To conclude, two case studies have been carried out to evaluate our approach, and to test the reference prototype in terms of effectiveness, efficiency and scalability. The case studies used are addressed for removing the *Feature Envy* and *Data Class* design smells and have been performed over 9 software systems of different sizes ranging from small to medium size. The results of the study confirm that our approach can be used to automatically generate refactoring plans for complex refactoring processes in a reasonable time. The studies performed also demonstrate that the efficiency of the HTN family of planners and the expressiveness of the JSHOP2 domain specification language makes it the appropriate planner to support the refactoring planning problem.

Aknowledgements

This work has been partially funded by the spanish government (*Ministerio de Ciencia e Innovación*, project TIN2008-05675).

References

1. Kent Beck and Martin Fowler. *Bad Smells in Code*, chapter 3. Refactoring: Improving the Design of Existing Code. Addison-Wesley, 1 edition, June 1999.
2. Dana S. Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN Planning System. *Journal of Artificial Intelligence Research (JAIR)*, 20:379–404, 2003.
3. Colin J. Neill and Phillip A. Laplante. Paying down design debt with strategic refactoring. *IEEE Computer*, 39(12):131–134, 2006.
4. W.F. Opdyke. *Refactoring Object-Oriented Frameworks*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1992. also Technical Report UIUCDCS-R-92-1759.
5. Javier Pérez. *Refactoring Planning for Design Smell Correction in Object-Oriented Software*. PhD thesis, University of Valladolid, 2011.