Identification of Differences between Aspect-Oriented Programs

Marija Katic and Kresimir Fertalj

University of Zagreb, Faculty of Electrical Engineering and Computing, Department of Applied Computing, Unska 3, 10000 Zagreb, Croatia
{marija.katic, kresimir.fertalj}@fer.hr

Abstract. In order to support various aspect-oriented software evolution tasks, differencing algorithms are needed. Identification of textual differences might be useful, but can miss changes that are specific to object-oriented and aspect-oriented behavior (e.g. method call in case of dynamic binding or set of advice at join-points). We present a new algorithm for identification of differences between the two aspect-oriented programs which accounts for aspect-oriented specific behavior.

Keywords: aspect-oriented program differencing, aspect-oriented program representation, aspect oriented program evolution

1 Intorduction

Aspect-oriented programming paradigm (Kiczales and Lamping 1997) has been introduced as a complement to traditional programming paradigms such as procedural and object-oriented paradigm in order to improve the overall quality of programs through the isolation of cross-cutting concerns into separate modules. In such a way program code is divided into two parts: base code which is more purpose-specific and aspect code represented with aspects. The aspect contains an advice (before, after, around) which is an additional code that has to be executed at a join point - a particular point in the execution of the program such as method invocation (call join-point), method execution (execution join-point), field access (field get join-point) or field modification (field set join-point). Due to its isolation mechanisms, aspect-oriented programming has found its place in the contemporary world among both academics and practitioners

Evolution of aspect-oriented programs might require identification of differences between the two aspect-oriented program versions. For example dynamic program evolution can be based on version differences (Katic and Fertalj 2011). However, only syntactic differencing might not be sufficient since it may miss object or aspect-related specific behavior. In order to support identification of such behavior we defined a new graph representation of an aspect-oriented program as a modification of a traditional control-flow graph and an enhanced control-flow graph proposed by Apiwattanapong (Apiwattanapong et al. 2007) and tailored to object-oriented pro-

grams. The representation is called aspect-oriented control-flow graph (AOCFG). The main difference between aspect-oriented and object-oriented programs' representation is in representation of interactions between base and aspect code. Such interactions are intrinsic to AO programs only. AOCFG provides representation of these interactions. In the following text we briefly describe our algorithm for identification of differences between the two aspect-oriented programs.

2 Differencing Algorithm for Aspect-Oriented Programs

Our algorithm for identification of differences between the two aspect-oriented programs is an extension of a differencing algorithm for object-oriented programs (Apiwattanapong et al. 2007). It works on AOCFG representation of methods in aspect-oriented programs and is based on identification of single-entry single-exit subgraphs i.e. minimal hammocks (Laski and Szermer 1992). Additionally, it is based on identification of aspect-related hammocks (artificially defined and not necessarily minimal) that are used for identification of aspect-related constructs. Since AOCFG allows for representation of before, after and around advice at method call and execution join-points as well as field get and set join-points, the algorithm allows for identification of all related differences at those places. Apart from changes related to base code only (i.e. non-join-point statements), the algorithm classifies identified aspect-related changes into added, deleted, modified and unchanged sets for each join-point statement.

3 Research Status

The algorithm is implemented and tested on three aspect-oriented programs with several versions. The work is under development since we plan to finish last tests and publish results afterwards.

References

1. Apiwattanapong, T., Orso, A., Harrold, M. J.: JDiff: A differencing technique and tool for object-oriented programs. Automated Software Engineering 14(1), 3-36 (2007)
2. Katic, M., Fertalj, K.: Model for Dynamic Evolution of Aspect-Oriented Software. In: 15th European Conference on Software Maintenance and Reengineering, pp. 377-380. IEEE Press (2011).
3. Kiczales, G., Lamping, J., et al.: Aspect-oriented programming. In: 11th European Conference on Object-Oriented Programming, pp. 220-242. Springer-Verlag (1997).
4. Laski, J., Szermer, W.: Identification of Program Modifications and its Application in Software Maintenance. In: International Conference on Software Maintenance, pp. 282-290. IEEE Press (1992).