Maintenance Support for Aspect-Oriented Programs: Opportunities and Challenges

Jianjun Zhao
School of Software
Shanghai Jiao Tong University
800 Dongchuan Road, Shanghai 200240, China
zhao-jj@cs.sjtu.edu.cn

1 Introduction

Aspect-oriented programming (AOP) [1, 4, 5, 10] has been proposed as a technique for improving the separation of concerns in software design and implementation. AOP provides explicit mechanisms for capturing the structure of crosscutting aspects of the computation such as exception handling, synchronization, performance optimizations, and resource sharing. Because these aspects crosscut the dominant problem decomposition, they are usually difficult to express cleanly using standard languages and structuring techniques. AOP can eliminate the code tangling often associated with the use of such standard languages and techniques, making the program easier to develop, maintain, and evolve.

The field of AOP has, so far, focused primarily on problem analysis, language design, and implementation. Even though the importance of program comprehension and software maintenance is known, it has received little attention in the AO paradigm.

AOP languages present unique opportunities and challenges for program comprehension and software maintenance community. An AO program is typically composed by two parts: base code, which concentrates on the main business logic of the software, and aspect code, which concentrates on universal requirements that crosscut the base code. When compiling the program, the aspect code will be woven into the base code in the phase of weaving [3]. Because of weaving mechanism, AOP languages are different largely from procedural or object-oriented languages. In order to support program comprehension and software maintenance tasks, new analysis and testing techniques are strongly needed for AO software.

We believe that many program comprehension and software maintenance techniques which have been well developed for procedural and object-oriented programming languages are applicable, with more or less modification, to AOP languages. For example, many program comprehension and software maintenance techniques for procedural or object-oriented programming languages rely on control flow and data flow information, which is typically gathered by the control flow and data flow analysis and represented by a call graph, a control-flow graph (CFG), or a program dependence graph (PDG).

2 Open Research Issues

To support the comprehension and maintenance of AO software, several open research issues should be addressed, which include:

- How do we develop some program representations for modeling the control and data flows in AO programs?
- How do we develop some static and dynamic analysis techniques to support analyzing the behavior of an AO program?
- How do we develop testing techniques, especially regression testing techniques for AO programs?
- How do we develop some debugging tools for AO programs?
- How do we have some metrics for measuring the maintainability of AO systems?

3 Work in Progress

Encouragingly, work is emerging in examining different approaches to the open research issues mentioned in Section 2.

Recently, several researchers consider the control and data flow analysis for AO programs. Sereni and de Moor [7] propose a simple call graph for AO programs without considering more detailed control flow analysis. Rinard et al. [6] propose a control flow graph in the classification system for AO programs. Other approaches, which provide partial solutions for analyzing control flows in AO programs, have been proposed by Zhao [16], Xu and Rountev [11], and Bernardi and Lucca [2]. Moreover, Zhao and Rinard [15, 17] define the system dependence graphs for AO programs for slicing, and Xu and Rountev [12] extend this
work by handling more complex situations such as multiple advices with a join point and dynamic advices in AO programs.

Change impact analysis is a useful technique for software maintenance. Zhao [14] presents an approach to support change impact analysis of AO programs based on program slicing. Stoerzer and Graf [9] focus on the semantic modification of base code and introduce a delta analysis to deal with the fragile pointcut problem, based on a comparison of the sets of matched join points for two program versions. Shinomi and Tamai [8] discuss the impact analysis of aspect weaving and its propagation through the base program and aspects. They focus on the change impact caused by the weaving aspect. Recently, Zhang et al. [13] present the atomic changes for aspect-related constructs to capture program semantic changes in the source code level. Their change impact model can be used to determine the affected program parts, affected tests and their responsible changes effectively.

Regression testing is another useful technique for software maintenance. Regression testing of AO programs has been studied by Xu and Rountev [11] and Zhao et al. [18].

4 Conclusion

We discussed some open research issues that should be addressed in the comprehension and maintenance of AO software, and gave a brief overview of the work in progress in this area. Some research results are quite encouraging, but many work still needs to be done to support the maintenance of AO software.

Acknowledgements

This work was supported in part by National High Technology Development Program of China (Grant No. 2006AA01Z158), National Natural Science Foundation of China (NSFC) (Grant No. 60673120), and Shanghai Pujiang Program (Grant No. 07pj14058). We would like to thank members of Software Theory and Practice (STAP) Group at Shanghai Jiao Tong University for their discussions on this topic.

Biography

Jianjun Zhao is a professor at Shanghai Jiao Tong University since 2005. He was a co-organizer of several workshops including Asian Workshop on Aspect-Oriented Software Development (2005-2008). He was also served on program committees of International Conference on Aspect-Oriented Software Development (2003-2005). His research interests include software engineering and program analysis, with emphasis on the application of program analysis techniques to problems in program comprehension, software maintenance, software testing, debugging, and aspect-oriented software development. He received a B.E. in Computer Science from Tsinghua University in 1987, and a Ph.D. in Computer Science from Kyushu University in 1997. Prior to joining Shanghai Jiao Tong University, he was employed as an Associate Professor in Computer Science at Fukuoka Institute of Technology.

References