Guiding the use of AspectJ Advice: An Initial Assessment

Henrique Rebêlo
Informatics Center
Federal University of Pernambuco
Recife, Pernambuco, Brazil
hemr@cin.ufpe.br

Márcio Ribeiro
Informatics Center
Federal University of Pernambuco
Recife, Pernambuco, Brazil
mmr3@cin.ufpe.br

***Abstract*—When using AspectJ-like languages, a developer may implement the same concern in different ways, e.g., using different kinds of advice. Despite the equivalence of such implementations, they may be different with respect to characteristics such as bytecode size and running time. In this way, this paper presents an initial assessment towards guiding developers to choose the proper advice for the proper situation.**

***Keywords*-aspect-oriented programming; aspectj advice; empirical analysis;**

I. Introduction

In an aspect-oriented language such as AspectJ [1], a general-purpose aspect-oriented extension to Java, we have special idioms including pointcuts and advice to provide separation of concerns at source code level.

In this context, one may obviously implement concerns using different advice. However, it is important to understand which kinds of impact we have when implementing a crosscutting concern using a specific AspectJ advice. The literature [2], [3] explains better how we can employ AspectJ advice to add behavior to a particular concern. However, we have different ways to implement and different weaving processes [4] to apply in the same concern modularization. For example, Soares *et al*. describe [3] how to implement a transaction concern using *before*, *after-returning*, and *after-throwing* advice. On the other hand, Laddad [2] shows a solution using only an *around* advice.

While the proposed solutions are equivalent, neither assessment nor guidance are discussed elsewhere to show which solution should be more efficient when constraints such as code size and running time are taken into account. In this way, this paper proposes an initial assessment of AspectJ advice using two metrics: (i) bytecode size and (ii) running time. Guiding developers towards which AspectJ advice should be used to implement a concern under some constraints represents the contribution of our work.

II. Assessment Approach

In order to provide an assessment of AspectJ advice, we conducted a case study involving the Heath Watcher system[1]. We considered only the transaction management concern, which was originally implemented by Sergio *et al*. work [3]. As mentioned, they used *before*, *after-returning*, and *after-throwing* advice. We also implemented an alternative solution with *around* advice inspired on Laddad's work [2]. In addition, our case study employed the ajc and abc [5] AspectJ weavers. The abc weaver considers some optimizations techniques for *around* advice during weaving process, being important because we consider such an advice in our study.

Our study consists of the following two scenarios: (i) *before* and *after-returning* versus *around*; and (ii) *before* and *after-throwing* versus *around*. Notice that the first scenario represents the commit accomplishment, whereas the second one represents the rollback. In order to assess the running time, we used the Profiling[2] plugin. For each scenario, we calculated the running time of the transaction concern, which encompass the time of execution spent by the aspect responsible for implementing such concern. Aiming at avoiding outliers, we executed both scenarios ten times using the ajc and abc compilers and the mean was taken into consideration for our assessment.

From the obtained data, considering a scenario with the ajc weaver, the original implementation by Soares *et al*. [3] is better than the alternative one, regarding both bytecode size and running time. However, when using the abc, the alternative implementation based on Laddad's work [2] is better than the original one, with respect to the running time. On the one hand, the bytecode size is smaller in the original implementation. On the other hand, we observed that when using many other advice (not only the transaction ones), the bytecode size of the alternative implementation tends to get smaller when compared to the original one, leading us to conclude that the alternative implementation is really better for real systems, which use several advice.

Based on the results showed in the Table I, we conclude that the *Original/ajc* and *Alternative/abc* are the best ones. But, when considering constrained environments such as

[1]http://www.comp.lancs.ac.uk/computing/users/greenwop/tao/HealthWatcherAO_01_Base.zip

[2]http://www.eclipse.org/projects/project_summary.php?projectid=tptp.performance

Table I
RUNNING TIME AND BYTECODE SIZE RESULTS

	Running time (sec)	Bytecode size (KB)
Original/ajc	**0.89521445**	**291**
Original/abc	1.26414025	264
Alternative/ajc	0.9141467	303
Alternative/abc	**0.7168156**	**273**

J2ME, we recommend the alternative implementation combined with abc weaver.

ACKNOWLEDGMENT

We would like to thank FACEPE and CNPq, Brazilian research agencies, for partially supporting this work. In addition, we thank SPG [3] members for fruitful discussions about this paper.

REFERENCES

[1] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold, "An overview of aspectj," in *ECOOP '01: Proceedings of the 15th European Conference on Object-Oriented Programming*. London, UK: Springer-Verlag, 2001, pp. 327–353.

[2] R. Laddad, *AspectJ in Action: Practical Aspect-Oriented Programming*. Greenwich, CT, USA: Manning Publications Co., 2003.

[3] S. Soares, E. Laureano, and P. Borba, "Implementing distribution and persistence aspects with aspectj," in *OOPSLA '02: Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. New York, NY, USA: ACM, 2002, pp. 174–190.

[4] E. Hilsdale and J. Hugunin, "Advice weaving in aspectj," in *AOSD '04: Proceedings of the 3rd international conference on Aspect-oriented software development*. New York, NY, USA: ACM, 2004, pp. 26–35.

[5] P. Avgustinov, A. S. Christensen, L. Hendren, S. Kuzins, J. Lhoták, O. Lhoták, O. de Moor, D. Sereni, G. Sittampalam, and J. Tibble, "abc: an extensible aspectj compiler," in *AOSD '05: Proceedings of the 4th international conference on Aspect-oriented software development*. New York, NY, USA: ACM, 2005, pp. 87–98.

APPENDIX

A. Online Appendix: We invite researchers to replicate our case study. Both implementations and our results are available at: http://www.cin.ufpe.br/~hemr/lawasp09.

[3] http://twiki.cin.ufpe.br/twiki/bin/view/SPG