

Improving Design Patterns Modularity Using Aspect Orientation

Mario L. Bernardi, Giuseppe A. Di Lucca

dilucca@unisannio.it, marioluca.bernardi@unisannio.it

RCOST -Research Centre on Software Technology, University of Sannio
Palazzo ex Poste, via Traiano, 82100 Benevento, Italy

Abstract

Design Pattern (DP) implementations may suffer of some of the typical problems related to some deficiencies of Object Oriented (OO) languages that may affect the modularity of the system, and thus its comprehensibility, maintainability, and testability.

Aspect Oriented Programming provide patterns' developers with powerful quantification constructs to better handle modularity and composition that can help to overcome some of the OO design tradeoffs and indirections in typical DP implementations.

A set of appropriate metrics to quantitatively evaluate the modularity and the impact of DP adoptions in OO system has to be identified and used to drive the re-development of DPs by aspect-oriented implementations improving modularity.

1. Introduction

Design Patterns implementation may suffer of some of the typical problems related to some deficiencies of OO languages. Indeed, the way a DP is implemented may heavily impact the overall system structure, and in particular it impacts the modularity of the system, thus affecting its comprehensibility, maintainability, testability too.

The invasive nature of pattern implementations and the scattering and tangling of such an implementation with the code of the remaining components of the system (such as the ones related to the application domain entities) may make it hard to distinguish between the patterns instances code and the code of the 'base' system components. In [2, 4, 5] it has been shown how several patterns from GoF catalog [1] introduce crosscutting that OO abstractions are often unable to well modularise.

Aspect Oriented Programming (AOP) provide patterns' developers with powerful quantification constructs to better handle modularity and composition. These constructs can help to overcome some of the OO design tradeoffs and indirections characterising current typical DP implementations. Composition transparency, optionality, and (un)pluggability are example of modularity properties that can be improved by AOP

pattern implementation and that have to be considered when assessing the quality of aspect oriented designs [2, 3].

We propose to identify and use a set of appropriate metrics to quantitatively evaluate the modularity and the impact of DP adoptions on system implementation. This evaluation can be used to drive the design of improved DP by aspect-oriented implementations.

Quantitative comparison between aspect-oriented versions of the patterns and their OO counterparts will let us to validate results but also to explore the wider spectrum of design choices available using AOP languages.

2. Motivations

Object Oriented Design Patterns provide the design of generic solutions to recurring problems [1].

The usage of DPs would increase the quality of OO design, but often developers miss the subtleties of the DP consequences [1] by applying/implementing patterns with wrong variation points to add (unneeded) flexibility. In these cases the usage of such DPs negatively affects the system overall quality.

However, also when the consequences are well taken into account, DPs may introduce in OO systems some problems related to reuse and maintainability that can be hard to solve within the object oriented paradigm.

Usually, these problems are due to the indirection techniques forcing one or more key interfaces to be implemented introducing a greater overhead.

This issue is dealt in [5] where is shown how AOP can help in restructuring GoF patterns to get more effective anticipation of future changes with a lower overhead.

Recent researches have shown that many DPs involve Crosscutting Concerns (CCC). That is mainly due to the poorness of the composition and quantification constructs in OOP languages that do not allow a good modularization of the concerns. Indeed, by using OO DPs, programmers are forced to add classes, interfaces, methods and attributes inside the code of the components (i.e. classes, packages, etc.) derived from the primary decomposition. The introduction of these elements will produce code scattering and tangling by reducing the

quality of some DP attributes such as reusability, traceability, comprehensibility, maintainability.

AOP constructs are able to better modularize DP concerns. In [2, 4, 6] AOP implementations of GoF [2] patterns are provided; they have better values for properties such as locality, (un)pluggability, composability and reusability.

Then a way to improve the modularity of OO DP is re-implement them by AOP techniques.

3. Using AO to improve DP modularity

The modularity of DP implementation can be improved by re-designing DP by AO. The re-design can be driven by a set of appropriate metrics (to be identified) evaluating the crosscutting of DP concerns: DP implementations characterised by bad values of the metrics will highlight the DP to be re-designed.

Metrics based on aspect mining techniques, as well as metrics based on clone analysis [6, 7, 8] can be used to this aim.

Research in this field is not yet mature, and it is related to just some selected types of patterns; in these cases most of the analyzed patterns introduce crosscutting and a high correlation is between superimposed roles [2] and crosscutting.

Thus a deeper investigation and discussion is required about these issues.

We are carrying out some preliminary studies on a suite of selected pattern implementations from different domains (structural decomposition, data access, communication, management and access-control patterns) to get and analyse quantitative data about crosscutting distribution and modularity properties. The DP to analyze were selected from a DP set wider than the GoF catalog.

As a first result we noted that most of the analysed OO pattern implementations introduce CCC at different levels of degree, as well as that AOP-based reengineering is able to improve their quality.

Three main considerations derived from the first results:

1) Patterns become language idiom

In some cases AOP languages can implement directly the patterns with different degree of flexibility. In our experiments we referred to AspectJ language. Interesting comparisons among different AOP language models can be performed.

2) AOP pattern re-implementation produces no benefits (but results in more overhead)

Some pattern implementations are well handled by OO languages thus they gain no actual benefits by AOP constructs. In these cases AOP implementations can be even worse than the OO ones.

3) Patterns involve crosscutting and role superimpositions.

In these cases pattern implementations introduce in OO systems scattered code while AOP implementations presents better modularity.

Of course, also our studies are not yet mature to give satisfactory answers; however they want to provide some quantitative information to solicit a more general discussion about these issues.

References

- [1]. Gamma, E., Helm, R., Johnson, R., Vlissides, J., 'Design Patterns: Elements of Reusable Object-Oriented Software', Addison-Wesley (1995)
- [2]. Hannemann, J., Kiczales, G., 'Design Patterns Implementation in Java and AspectJ', Proc. of Object Oriented Programming Systems Languages and Applications 2002 (OOPSLA '02), Nov 2002, 161-173
- [3]. Hachani, O. , Bardou D., 'On Aspect-Oriented Technology and Object-Oriented Design Patterns' , Proc. of European Conference on Object Oriented Programming 2003 (ECOOP 2003)
- [4]. Monteiro, M.P. , Fernandes J.M. , 'Towards a catalog of Aspect Oriented Refactorings , Proc. of Aspect oriented Software Developmnet 2005 (AOSD '05)
- [5]. Nordberg Martin E., 'Aspect Oriented Indirection - Beyond Object Oriented Design Patterns' , Proc. of Workshop. Beyond De-sign: Patterns (mis)used, Proc. of Object Oriented Programming Systems Languages and Applications 2002, OOPSLA 2002
- [6]. Magiel Bruntink, Aspect Mining using Clone Class Metrics , Proc. of Workshop on Aspect Reverse Engineering 2004 (WARE '04)
- [7]. [7] Garcia, A. et al. Separation of Concerns in Multi-Agent Systems: An Empirical Study. In Software Engineering for Multi-Agent Systems II, Springer, Lecture Notes on Computer Science, 2940, January 2004.
- [8]. [8] Garcia, A., Silva, V., Chavez, and C., Lucena, C. 'Engineering Multi-Agent Systems with Aspects and Patterns'. Journal of the Brazilian Computer Society, 1, 8 (July 2002), 57-72.