Taming Aspects with Managed Data

Theologos A. Zacharopoulos
theol.zacharopoulos@gmail.com

April 4, 2016, 17 pages

Supervisor: Tijs van der Storm
Host organisation: Centrum Wiskunde & Informatica, http://www.cwi.nl

Universiteit van Amsterdam
Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Master Software Engineering
http://www.software-engineering-amsterdam.nl

Contents

Abstract

Chapter 1

Introduction

Cross Cutting Concerns (CCC) is a problem for which the classic programming techniques can not tackle with sufficiently. This results in scattered and tangled code, which affects the system's modularity and it's ease of maintenance and evolution. Since Object Oriented Programming (OOP) and Procedural Programming (PP) techniques can not solve this problem, Aspect Oriented Programming (AOP) presented [KLM+97] in order to provide a solution by introducing the notion of *aspects*.

AOP results in a modular and *single-responsibility* design whose properties must be implemented as *components* (cleanly encapsulated procedure) and *aspects* (not clearly encapsulated procedure), both separate concepts that are combined for the result through a process called *weaving*. However, relying on AOP, paradoxically, does not improve the evolution of a project even with the modularity that it provides since it introduces tight coupling between the aspects and the application. As a result the way to tackle with this problem we need a more sophisticated and expressing crosscut language. Consequently, CCC could be handled in a higher level of the language such as the data structuring and management mechanisms.

Managed data [LvdSC12] allows programmers to take control of important aspects of data as reusable modules. Using managed data a developer can build *data managers* that handle the fundamental data manipulation primitives that are usually hard-coded in the programming language, by introducing custom data manipulation mechanisms. Managed data have been researched and implemented under the Enso project[1], which is developed in Ruby[2] (a dynamic programming language) using Rubys reflection capabilities. Furthermore, managed data are considered less able to be supported in static languages directly which makes it more challenging for this thesis since it is going to be implemented in Java. In this thesis I am going to use the Java reflection capabilities to implement managed data and focus on specific aspects and design patterns implementations using the data managers concept of managed data.

1.1 Initial Study

In their study on managed data, Loh et al. [LvdSC12] present an implementation of managed data in Ruby and they use as a case study a web development framework from the Enso project to reuse database management and access control mechanisms across different data definitions.

This thesis is an extension of their work; we implement managed data in Java (a static programming language) using the Java reflection API[3] and dynamic proxies[4]. Although proxies in static programming languages can not implement the full range of managed data [LvdSC12]. Java provides a strong implementation of the meta-object protocol [KDRB91], which can be used though the Java Reflection API [FFI04]. Additionally, this project will focus on aspects and will provide a solution to the CCC problem by using managed data.

[1] http://enso-lang.org/
[2] https://www.ruby-lang.org/en/
[3] https://docs.oracle.com/javase/tutorial/reflect/
[4] https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Proxy.html

1.2 Problem statement

The problem we study regards the CCC that are scattered around the application, resulting to a hard to maintain system by tangling implementation logic and concerns code together. Even though, AOP provides new modularization mechanisms, which should result in easier evolving software, it delivers solutions that are as hard and sometimes even harder to evolve than before [TBG03]. The problem lays on the aspects, which have to include a crosscut description of all places in the application where this code yields an influence. Thus, the aspects are tightly coupled to the application and this greatly affects the evolvability of the overall system.

Additionally, Friedrich Steimann [Ste05] argues that modeling languages are not aspect ready. The problem that arises is located at the level of software modeling. More specifically, in *roles modeling*, whereas in OOP roles are tied to the collaborations, collaborations rely on interactions of objects, and aspects on the other hand are typically defined independently of one another.

Furthermore, in terms of order, it has been observed that aspects are not elements of the domain, they describe the order rather than the domain. Finally, aspects invariably express non-functional requirements, but if the non-functional requirements are not elements of domain models then neither are aspects.

In order to solve the aforementioned problems, we implement manage data, lifting the data management up to the application.

1.2.1 Research Questions

Managed data has not been practically implemented in a static language before, therefore my first research questions states "Can managed Data be implemented in a static language like Java?". Based in the previous argumentation about the relevance of AOP and the solutions that managed data can provide in CCC, my second research question is "Can managed data solve CCC and to what extend does it improve the software evolution problems that AOP introduces in a modular solution?". Finally by using a software showcase, the JHotDraw framework, as well as its AOP implementation AJHotDraw [MM], I am going to evaluate the implementation of managed data on an inventory of aspects and design patterns. As a result the third research question states "To what extent can managed data tame an inventory of aspects and design patterns in the JHotDraw framework, in contrast with the original and the AOP implementation."

1.2.2 Solutions Outline

1.2.3 Research Method

1.3 Contributions

1.4 Related Work

1.5 Document Outline

Chapter 2

Background

2.1 Cross Cutting Concerns

2.2 Aspect Oriented Programming

2.2.1 Aspect Oriented Programming Showcases

2.2.2 Design Patterns in Aspect Oriented Programming

2.2.3 Aspect Oriented Programming Evaluation

2.2.4 Evolvability

2.3 Managed Data

2.3.1 Schemas

2.3.2 Data Managers

2.4 Internal DSLs

2.5 Java Reflection and Proxies

2.5.1 Reflection

2.5.2 Reflection and MetaObject Protocol

2.5.3 Dynamic Proxies

Uniform Proxies

2.6 JHotDraw And AJHotDraw

2.6.1 Refactoring of Crosscutting Concerns

Role-based Refactoring of Crosscutting Concerns.

Evaluation

2.6.2 The Undo Concern of JHotDraw

Evaluation

AspectJ Drawbacks in the Undo Solution

2.6.3 The Persistence Concern of JHotDraw

Chapter 3

Theory

3.1 Self Describing

3.1.1 Reuse

3.1.2 Malleability

3.1.3 Java runtime

3.2 Model Driven Development

3.2.1 Object and Schemas

3.3 Schema

3.3.1 Description of Schema

3.3.2 Schema Schema

3.3.3 Metadata

3.4 Factories

Chapter 4

Implementation

4.1 Managed Data

4.1.1 Schema

Schema Definition

4.1.2 Data Managers

Data Managers Definition

4.2 Bootstrapping

4.2.1 Cutting the umbilical cord

4.3 Self-describing schema (SchemaSchema)

4.4 Schema Loading

4.4.1 Forward

4.4.2 Wire the Cross-References

4.5 Typing

4.5.1 Primitives

4.5.2 Collections

4.6 Implementation Issues

4.6.1 Methods ordering

4.6.2 Hash-code of Managed Objects

4.6.3 Default methods of Managed Objects

4.6.4 Collections of Managed Objects

4.6.5 Transparent equivalence

Chapter 5

Evaluation

5.0.1 Research Questions and Answers

5.0.2 Evidence

Design Patterns

Undo Concern of JHotDraw

Persistence Concern of JHotDraw

5.0.3 Results

5.0.4 Claims

Chapter 6

Conclusion

Chapter 7

Further Work

Acknowledgments

Appendix A

How to Use the Framework

Appendix B

Example Application

B.1 Schemas definition

B.1.1 Point Schema

B.1.2 Line Schema

B.2 Data managers definition

B.2.1 Basic Data Manager

B.2.2 Lockable Data Manager

B.2.3 Observable Data Manager

B.3 Tame Aspects

B.3.1 Immutability

B.3.2 Logging

B.3.3 More

Appendix C

Refactoring of JHotDraw's Undo Concern

Bibliography

[FFI04] Ira R Forman, Nate Forman, and John Vlissides Ibm. Java reflection in action. 2004.

[KDRB91] Gregor Kiczales, Jim Des Rivieres, and Daniel Gureasko Bobrow. *The art of the metaobject protocol*. MIT press, 1991.

[KLM+97] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In *ECOOP'97Object-oriented programming*, pages 220–242. Springer, 1997.

[LvdSC12] Alex Loh, Tijs van der Storm, and William R Cook. Managed data: modular strategies for data abstraction. In *Proceedings of the ACM international symposium on New ideas, new paradigms, and reflections on programming and software*, pages 179–194. ACM, 2012.

[MM] Marius Marin and Leon Moonen. Ajhotdraw: A showcase for refactoring to aspects.

[Ste05] Friedrich Steimann. Domain models are aspect free. In *Model Driven Engineering Languages and Systems*, pages 171–185. Springer, 2005.

[TBG03] Tom Tourwé, Johan Brichau, and Kris Gybels. On the existence of the aosd-evolution paradox. *SPLAT: Software engineering Properties of Languages for Aspect Technologies*, 2003.