

Aspect-Oriented Programming and Modularity

Gina Häussge

February 2, 2006

Abstract

Aspect oriented programming is a rather new paradigm in software engineering, presenting a possibility to further aid in the separation of concerns in software architectures. In this essay I want to give a short introduction to what aspect oriented programming is and how using it can improve the modularity of software systems.

1 Introduction

When constructing a software system there are always some requirements you have to meet but only can realise by implementing them redundantly and scattered across multiple modules throughout the system. Say for example you have to create some security model, implement this into a module and now want to combine an existing system consisting of several modules with this newly created security module. You now have to change your system in several places to properly use the security subsystem:

- Insert calls to the security modules interface around security critical points in several modules
- Add new kinds of exceptionhandlers to allow the handling of unallowed access situations

Even worse, whenever you have to change something in the security module, you might need to change the related modules of your original system again. The small change originating from the security subsystem the ripples through a lot of modules. Another example would be the addition of a logging system to your existing modules, which makes it necessary to change every module in the whole system, and change it again with each change to the loggers interface.

Such requirements to the software system, like enforcement of some security model or logging certain parts of the systems execution, are so called *crosscutting concerns*, as they exist in many parts of the architecture. They crosscut many or even all modules.

Modularising such crosscutting concerns is impossible using only the object oriented approach, they are simply spread throughout the whole codebase.

This is where *aspect oriented programming* can be of great help. Aspect oriented programming primarily focuses on modularising crosscutting concerns. The programmer can define so called *aspects*, which consist of the definition of one or many *pointcuts* - points where the crosscutting concern hooks into the system - and *advices* implementing the functionality that needs to be applied at the pointcuts. Additionally, in some aspect oriented languages it is possible to insert inter-type declarations via an aspect, expanding the functionality of an existing module on the fly.

We will now take a look at how aspect oriented programming - or AOP - can improve our systems design, primarily focusing on how it may improve modularity continuity, direct mapping and linguistic modularity.

2 Modularity Continuity

When a system has good modularity continuity it means that a small change in the problem specification will only cause changes in one or a small number of modules.

Especially when it comes to crosscutting concerns though, objectoriented programming makes it necessary to weave the crosscutting code deep into other modules instead of simply defining it as a standalone module. A change in the crosscut then triggers changes throughout quite a number of other modules in the system.

Aspect oriented programming on the other hand allows to define the crosscutting concerns in a modular way at a central point. A change in the specification then triggers only the change of the aspect implementing the changed part of the specification and does not ripple throughout the whole system. We see, aspect oriented programming improves the modular continuity of a software system by making it possible to wrap even cross cutting concerns into a module which can easily be maintained and changed independent of the rest of the system.

Taken our previous example of a logger added to the system, aspect oriented programming allows us to simply redefine the pointcuts and advices of the logger mechanism, e.g. adding a logger statement to each method call with only the addition of a very small amount of code into the logger aspect.

3 Direct Mapping

A software structure fulfilling the Direct Mapping rule is a structure whose implemented modular structure reflects the modular structure designed during modeling the problem domain. As you are normally taking into account **what** a system should do and not how it should do it when designing a system, the resulting systems structure should also reflect the components of the systems functionality, not implementation specific tweaks.

This is where software design can profit from aspect oriented programming.

Instead of being forced to split crosscutting concerns among several models in the system and therefore ignoring the designed modular structure which states the crosscutting concern as a module on its own which simply is needed by many other modules, we can now embed the necessary code into one module, an aspect, and weave it into the other modules by defining pointcuts and advices.

Taking a look at the examples given in the Introduction, instead of having to scatter the hooks of the security subsystem and the logger into the existing system all across the systems modules, aspect oriented programming allows us to wrap the subsystem into a module of its own, an aspect, and then defining the hooks of this module in the module itself instead of throughout the whole system. In the systems specification, the logger and the security subsystem appear as a single module, so aspect oriented programming has helped us here to improve the direct mapping of our software system by isolating the subsystems.

4 Linguistic Modular Units

The Linguistic Modular Units principle is fulfilled by a system when its modules correspondent to syntactic units in the language used to describe them, and - in case of a programming language being used - are compilable separately.

Using object oriented programming one usually cannot strictly this principle, as object orientation simply lacks a concept to separate cross cutting concerns into a single module. The programmer is forced to translate the single cross cutting module into several smaller parts scattered throughout the codebase, making it impossible to compile that module on its own. Such a situation is excluded by the principle of linguistic modular units. Aspect oriented programming though allows to treat even cross cutting concerns as single modules and - taken aside implementation specifics that limit this in the real world¹ - compile them on their own.

The logger and the security subsystem from our example could be extracted into one aspect each, compiled separately and then connected with the other modules during runtime.

5 Conclusion

We have seen that aspect oriented programming can greatly improve certain aspects of software design, namely the implementation of cross-cutting concerns and the flexibility of the design itself. With great power comes great responsibility though: As with every programming paradigm, it takes a certain amount of skill and thoughtfulness to create a solid, understandable system. Aspect oriented programming is not the universal remedy of software design.

¹AspectJ[2] for example uses a preprocessor to produce java bytecode executable on any JavaVM, interweaving the defined aspects and the object oriented parts of the code during compiletime. The aspects are not compilable on their own, as they are simply preprocessor directives. However, there do exist other approaches at AOP engines that do not have such restrictions.

References

- [1] Wikipedia. *Aspect-oriented Programming*.
http://en.wikipedia.org/wiki/Aspect-oriented_programming
Accessed: February 1st, 2006
- [2] *AspectJ*
<http://aspectj.org>
- [3] Bertrand Meyer. *Object-Oriented Software Construction*. Prentice Hall 1997.