

Programming AOPish without an AOP language

Arno Schmidmeier
AspectSoft
Arno@aspectsoft.de

Aspect Oriented Software Development (AOSD) is a new programming paradigm. Aspect Oriented Programming (AOP) is the most prominent technology of this new approach. AOP is very well supported by aspect oriented programming languages. AspectJ as a superset of the Java programming language is the most prominent one. Modern AOP languages proved quite successful to modularize several crosscutting concerns like tracing, auditing, exception and transaction handling, etc. Unfortunately GPALs like AspectJ may not be applicable for several reasons in a project (e.g. (Management) acceptance, different programming language, legacy code bases, reliability).

How can you program in an AOP style without using an AOP language?

I see often following solutions to this common problem:

- Use the middleware infrastructure to implement aspects
- Use an aspect oriented framework
- Use a combination of well known patterns to implement several aspects or implement an aspect oriented architecture. (e.g. Command pattern)
- Use meta elements of the programming language to implement aspects or an aspect oriented framework
- Use a code generation to “weave” code in the existing code or model base

All off these approaches have some common drawbacks:

- They require additional work in architecture, code and design.
- They add additional complexity to the code based and the architecture.
- They are not applicable to intra class concerns.

Which one to choose depends from the project, the used languages, the aspects, the requirements, etc. It is quite likely that a combination of the different approaches is used.

Use the middleware infrastructure to implement aspects

Quite a lot of middleware tools provide a lot of AOSD support at the design or architecture level. First off all modern middleware like (J2EE) provides often reusable implementations for common crosscutting concerns like persistence and transactions. Most modern middleware provide a kind of Interceptors mechanism. It is sometimes called transformer, dynamic proxies, filter or even directly interceptor. This mechanism provides a good base to emulate advices. They are often used to implement popular crosscutting concerns like logging, load balancing, transaction handling, caching, etc.

Middleware infrastructure is best used to implement infrastructure aspects, like caching, transaction handling, etc.

Use an aspect oriented framework

Aspect oriented frameworks like AspectWerks, Nanning or JbossAOP provide sufficient support for common infrastructure concerns. AOP frameworks offer nearly the same possibilities of

modularisation of common crusscutting code as an GAPL. However AOP frameworks do not suffer from the lack of acceptance, which GAPLs often suffer. AOP Frameworks are for the management often a small and cheap new library.

Use a combination of well known patterns to implement several aspects or implement an aspect oriented architecture. (e.g. Command pattern)

“Old” patterns like command or strategy are often used to architect and implement an AO architecture. e.g. All business code must be implemented as a command. The crosscutting code can be modularized in these scenarios as a command handler. The biggest drawback of this approach is the additional crosscutting code, which is necessary to convert the existing business code to participants of the patterns.

use meta elements of the programming language to implement aspects or an aspect oriented framework

Quite a lot of OO languages have support for meta object facilities. In languages like CLOS, Smalltalk or Python it is quite easy and straightforward to implements aspects just with the core features of the language. AspectS or Pythius are some examples.

On the Java platfform a combination of some creational patterns (e.g. Factory method) and dynamic proxies can also help.

use a code generation to “weave” code in the exisiting code or model base

If code generation is already used in the project, generating the weaved aspects in the codebase, is a common approach in this case.

Conclusion

The best approach to program in an AOP way is the use of a GPAL. However GPALs are often not applicable. In these cases there are several options, each one with specific requirements, advantages and problems. Which one to choose depends from case to case. Often they are combined or several approaches are used for several different crosscutting concerns. In any case we try to implement AOP by hand.