

Grade

Date of acceptance

Instructor

# **Aspect-Oriented Programming**

Jyri Laukkanen

1.2.2008

UNIVERSITY OF HELSINKI

Department of Computer Science

Tiedekunta/Osasto – Fakultet/Sektion – Faculty/Section		Laitos – Institution - Department	
Faculty of Science		Department of Computer Science	
Tekijä – Författare - Author			
Jyri Laukkanen			
Työn nimi – Arbetets titel - Title			
Aspect Oriented Programming			
Oppiaine – Läroämne - Subject			
Computer Science			
Työn laji – Arbetets art - Level	Aika – Datum – Month and year	Sivumäärä – Sidoantal – Number of pages	
Seminar paper	1.2.2008	13	
Tiivistelmä – Referat - Abstract			
<p>This seminar paper introduces aspect-oriented programming in general. Defines the key terminology used and covers some areas where it can be applied. Also the benefits and drawbacks of the aspect-oriented programming are discussed.</p> <p>Aspect-oriented programming is a complementary programming technique to the object-oriented programming. It provides tools for managing so called cross-cutting concerns. Log writing, authorization and transaction management can be taken as examples of this kind of cross-cutting functionality.</p> <p>In aspect-oriented programming the system is divided into a two halves: the base program and the aspect program. The base program will contain the main functionality of the system and can be implemented using object-oriented programming. The aspect program will consist of the cross-cutting functionality that has been modularized away from the base program.</p> <p>ACM Computing Classification System (CCS):</p> <p>D.1.5 Object-oriented Programming</p>			
Avainsanat – Nyckelord - Keywords			
Aspect Oriented Programming, AOP			
Säilytyspaikka – Förvaringställe – Where deposited			
Muita tietoja – Övriga uppgifter – Additional information			

## Index

1	Introduction .....	1
2	Terminology .....	2
3	Applications .....	5
4	Benefits and drawbacks.....	8
4.1	Benefits.....	9
4.2	Drawbacks .....	9
5	Summary .....	11
	References.....	12

## 1 Introduction

Traditionally in the history of the programming, different kinds of programming paradigms have evolved from the need of managing the complexity of the software engineering. Since the size and overall complexity of the software projects have been growing, the industry has been forced to develop and adapt different approaches to the problem, in order to keep up with the progress. The procedural and the object-oriented programming paradigms are the results of this continuous evolution. Especially the object-oriented paradigm can be used efficiently to model the structure of the software and to abstract away the low level details of implementation.

Although the object-oriented paradigm provides a rich set of tools for abstraction and modularization, it can not address the problem with so called cross-cutting concerns. One can think the cross-cutting concerns as functionality that when implemented, will scatter around the final product in different components. Since this kind of functionality will cut through the basic functionality of the system, it is hard to model even with the object-oriented programming [Kic97, s. 1]. Good examples of the cross-cutting concerns are authorization, synchronization, error handling and transaction management [KiM05, s. 49].

Aspect-oriented programming tries to address the problem by modularizing the cross-cutting functionality into more manageable modules – aspects. Unlike the object-oriented programming, aspect-oriented programming does not replace previous programming paradigms. Therefore it can be seen as a complementary to the object-oriented paradigm rather than a replacement. In aspect-oriented programming the system is divided into a two halves: the base program and the aspect program. The base program will contain the main functionality of the system and can be implemented using object-oriented

programming. The aspect program will consist of the cross-cutting functionality that has been modularized away from the base program. This leads to a more concise structure since the functionality of the cross-cutting concerns is contained within well defined modules.

## 2 Terminology

In order to fully understand the concepts of aspect-oriented programming, we have to define some basic terminology. In this chapter we will define the key terms used with the aspect-oriented programming and explain their meaning.

### **Cross-cutting concerns**

Hridesh Rajan and Kevin Sullivan define the concern as a dimension where a design decision is made. It becomes cross-cutting if its realization in traditional object-oriented designs leads to scattered and tangled code [RaS05, s. 60]. The code scattering results from the implementation of cross-cutting concerns within the base program. For example log writing functionality is needed in most of the components, and thus the implementation will disperse among them. Also the responsibilities of the different components will get vaguer as they are forced to take care of non-core functionality. In other words, the components are responsible for some additional functionality – namely the cross-cutting functionality. This is what we call as code tangling. Examples of the cross-cutting concerns can be seen in the figure

1.

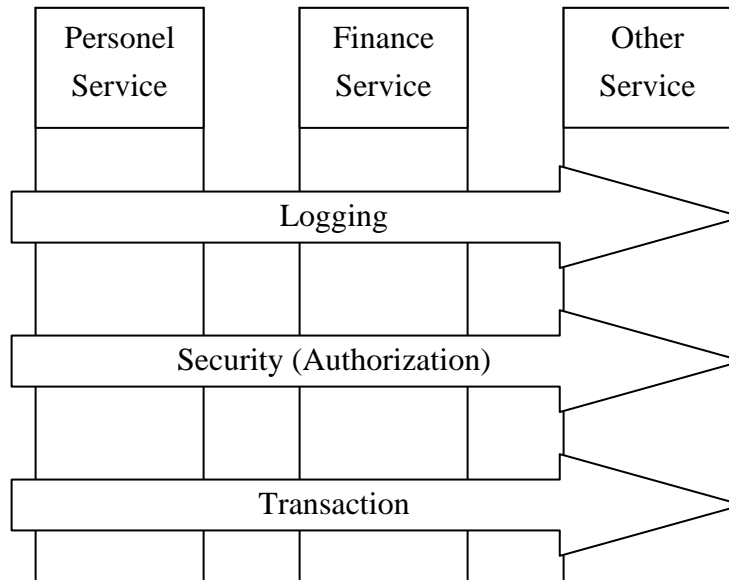


Figure 1. Cross-cutting concerns [see JuB07, s. 904, figure 1]

### **Aspect**

Aspect can be seen as a class-like construct. It consists of set of pointcuts and related advices. The basic idea of the aspect is to encapsulate the cross-cutting functionality away from the base program into separate well defined modules [RaS05, s. 60].

### **Join point**

The events during execution at which advice may execute are called join points [WKD04, s. 890]. Depending on the aspect language implementation the join points can be for example method calls, exceptions thrown or modifications of classes' attributes.

## **Pointcut**

The pointcut is a pattern that binds an advice to a single or multiple join points [AVG07, s. 11]. Usually the pointcut is defined with some kind of pointcut expression language. The most notable of these languages is the AspectJ's pointcut expression language.

## **Advice**

An advice is an action to be taken at the join points in a particular pointcut [WKD04, s. 890]. In other words, the advices contain all the functionality of the cross-cutting concerns. There are different kinds of advices that are related to method calls. Depending on the advice it can be called before the target method, after it or both before and after, thus around the method.

## **Weaving**

The process of causing the relevant advice at each join point to be executed is called weaving [WKD04, s. 891]. Basically this means that advices are bound to join points according to the pointcuts. Depending on the aspect language implementation, the weaving process can take place at compile-, load- or runtime [KiM05, s. 53-54]. In compile-time weaving, a separate aspect compiler is used. In load-time weaving the classloader is responsible for the weaving process while loading the classes into the virtual machine. The runtime weaving utilizes proxy classes and code generation libraries like the CGLIB, to perform the binding. One of the most notable compile-time aspect language implementation is the AspectJ, while the Spring Framework offers the most well known runtime implementation [Joh05, s. 109].

### 3 Applications

Aspect-oriented programming has many applications. Even though the main usage lies in managing the cross-cutting concerns, it is by no means restricted to that. Adding new or modifying existing functionality in the base program is another area where aspect-oriented programming can be applied.

As we have stated earlier, the object-oriented programming itself does not provide enough tools to manage the cross-cutting concerns. This leads to the code scattering and tangling which is illustrated in figure 2.

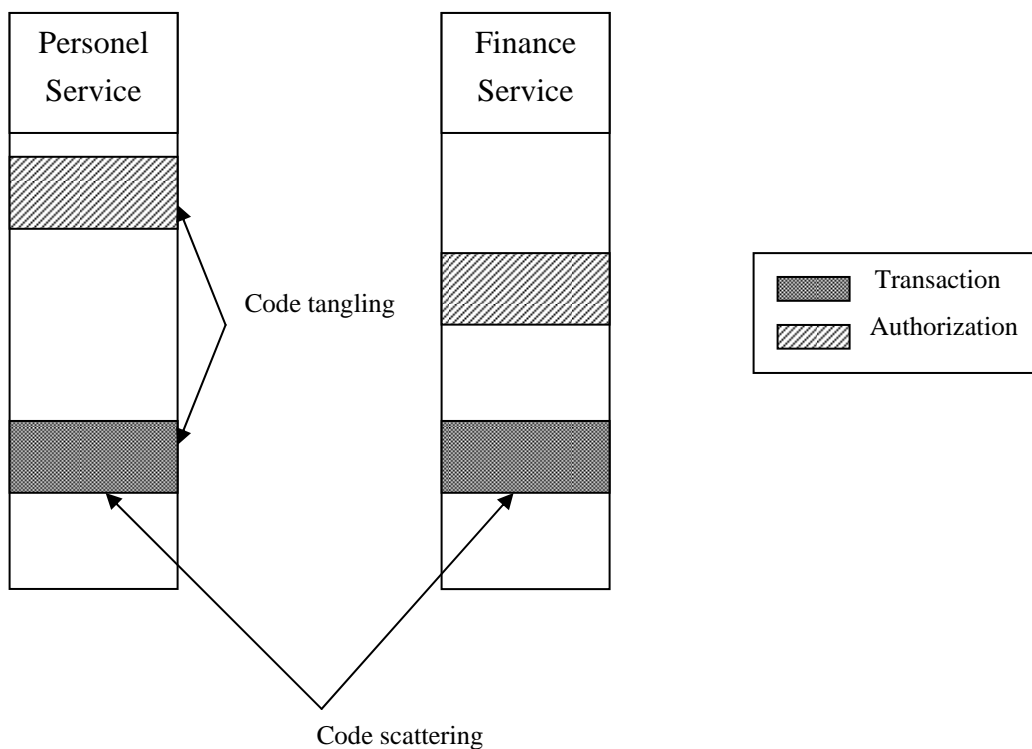


Figure 2. Code scattering and tangling

By using aspect-oriented programming the code scattering and tangling can be avoided. Since the cross-cutting functionality is now encapsulated within well defined modules, the impact on the different components decreases. This is illustrated in figure 3.

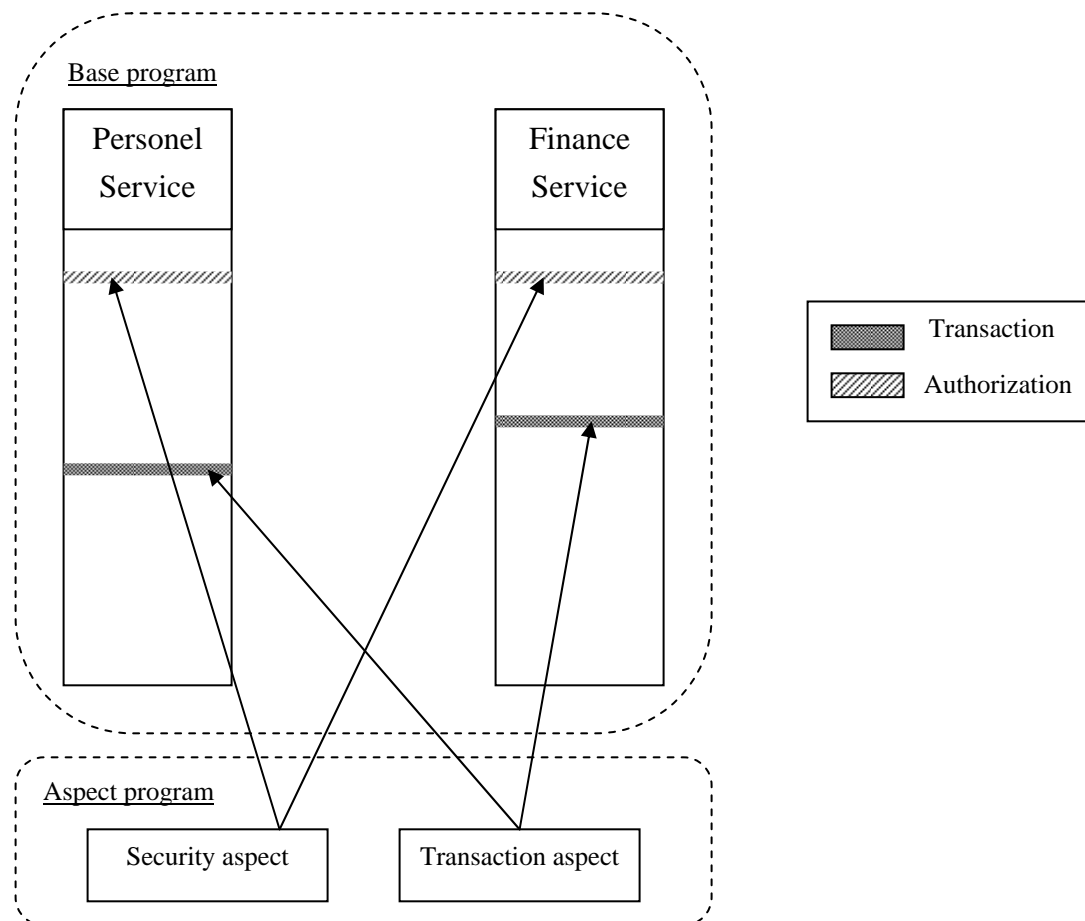


Figure 3. Cross-cutting concerns implemented as aspects

Introduction of a caching database access can be taken as an example of existing functionality modification. Let say that we have a layered architecture based service that utilizes database. This is illustrated in figure 4.

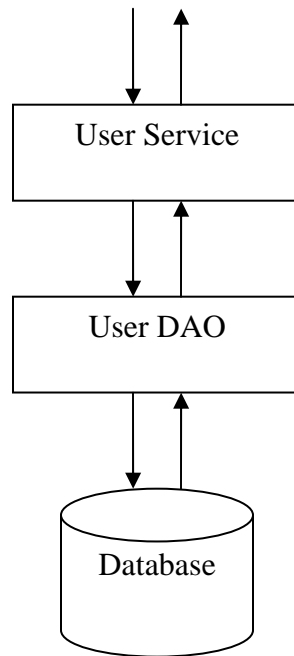


Figure 4. Layered architecture based service utilizing database

We can use aspect-oriented programming to add caching functionality without making any modifications to the original base program. This can be achieved by using runtime weaving and implementing the caching functionality as an aspect. The newly weaved caching aspect is illustrated in figure 5.

Since the caching functionality could be shared in other services too, we can see that it actually is a cross-cutting concern. Of course we could implement a separate cache for every service independently, but better solution is to use an advice. We can implement one generic cache and apply it into all services using aspect-oriented programming. This way we can avoid the unnecessary code duplication and increase the reusability and maintainability of the system.

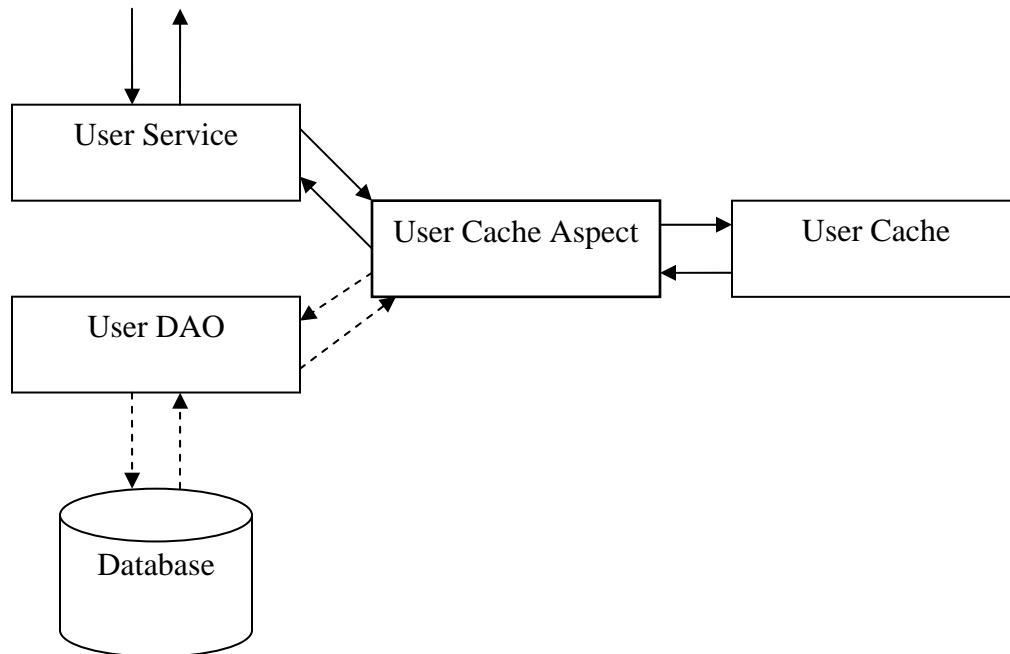


Figure 5. Layered architecture based service utilizing database through a caching-aspect

## 4 Benefits and drawbacks

In this chapter we will discuss some of the benefits as well as the drawbacks related to the aspect-oriented programming. Although the aspect-oriented programming provides powerful tools for managing the cross-cutting concerns, it may also produce some unwanted side effects in software development.

## **4.1 Benefits**

As mentioned earlier, the aspect-oriented programming helps to manage the cross-cutting functionality in the system by encapsulating the otherwise dispersed functionality into well defined modules. This affects the overall structure of the system because the base program does not have to concern about the cross-cutting functionality [AlB04, s. 93-94]. This naturally reduces the amount of duplicated code and decreases the error probability.

Aspect-oriented programming can also be used to add new or to modify existing functionality in the base program. Using aspects it is possible to add new functionality without modifying the base program. This is convenient in cases where the source code of the base program is not available. In any case care must be taken when modifying the functionality of the base program, since it can lead to unpredictable side effects. We will discuss more about the drawbacks in the next chapter.

Aspect-oriented programming can also be used with software testing. Advices can be bound to method calls to measure invocation counts and execution time. Pre- and post-conditions of the methods can be verified by using around-style advices that are wrapped around given target methods.

## **4.2 Drawbacks**

Aspect-oriented programming is very powerful programming technique. Depending on the aspect language implementation, it can be used to modify method call parameters and return values; in some cases even classes' attribute values. While this opens new possibilities it may increase the complexity of the system and in worst case, lead to problems that are hard to track.

It may sound contradictory to claim that usage of the aspect-oriented programming could increase the complexity of the system. In object-oriented programming it is often so that one must understand the functionality in the super class in order to fully understand the functionality of its subclass. This applies to aspect-oriented programming maybe even more in some extend. This is because one must understand the functionality of both base and the aspect programs in order to understand the whole system [AIB04, s. 94].

One potential pitfall resides in the pointcuts that bind the advices to the join points. Faulty pointcut definitions may cause advices to be bound to wrong join points or not to any join points at all. Figure 6 shows different types of errors in pointcut definitions.

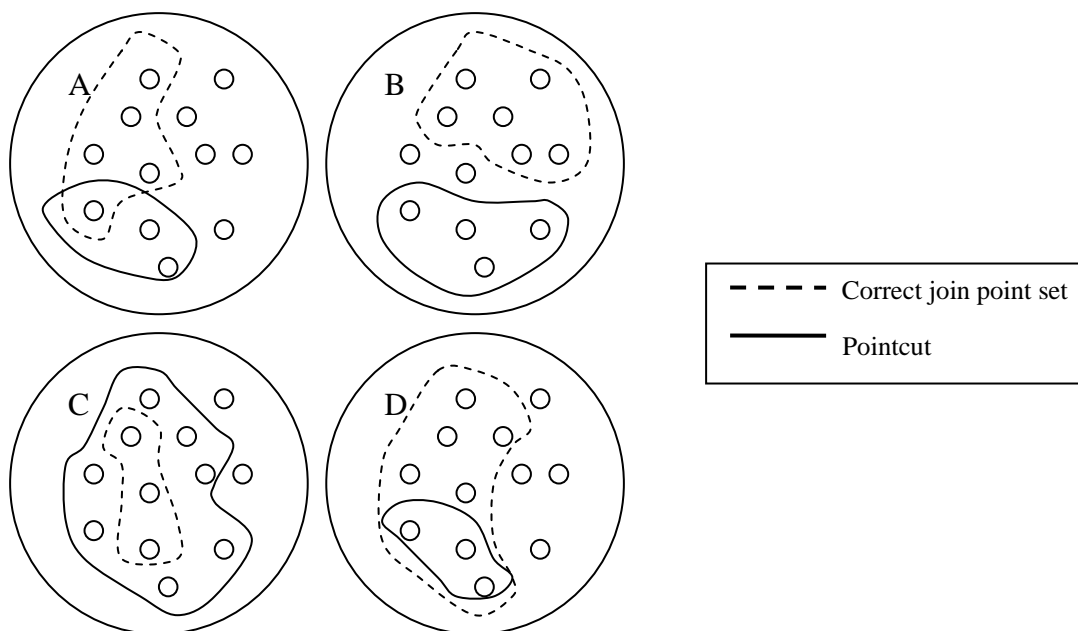


Figure 6. Possible error situations with pointcuts [LFM06, s. 34]

Following error situations are possible with pointcuts (see figure 6) [LFM06, s. 34]:

- A. Pointcut selects some of the intended join points but also some unintended.
- B. Pointcut selects none of the intended join points.

- C. Pointcut selects all the intended and also unintended ones.
- D. Pointcut selects some of the intended but not all.

Errors in the pointcut definitions may cause unpredictable behavior and they can be really hard to find. For example aspect based transaction management is a good example of such functionality that may cause severe problems in the system if the related pointcuts are faulty.

## 5 Summary

Aspect-oriented programming provides a powerful set of tools for managing the cross-cutting concerns. It complements the object-oriented programming and together they form a programming technique that can be used to produce concisely structured and modularized software.

Modularizing the cross-cutting functionality leads to clearer structure and also less error prone implementation. This is because the functionality that would otherwise be scattered around the system can now be implemented in one place. So less code is needed and the maintenance of the code base gets lot easier than it would be with plain object-oriented programming.

Regardless of the powerful characteristics of the aspect-oriented programming, it is still quite transparent and non-intrusive programming technique. With aspects one can easily add new or modify existing functionality in the base program, even without having access to the source code. In conclusion we can say that the aspect-oriented programming teams well with the object-oriented programming.

## References

- AlB04 Roger Alexander, James Bieman, Aspect-Oriented Technology and Software Quality. *Software Quality Journal*, Springer Netherlands, June 2004. Volume 12, Number 2, pages 93-97.
- Avg07 P. Avgustinov et al., Semantics of Static Pointcuts in AspectJ. *Proc. of the 34<sup>th</sup> annual ACM SIGPLAN-SIGACT symp. of Principles of prog. Languages*, Nice, France, 2007. Volume 42, Issue 1.
- Joh05 Rod Johnson, J2EE development frameworks. *Computer*, January 2005. Volume 38, Issue 1, pages 107-110.
- JuB07 Ke Ju, Jiang Bo, Applying IoC and AOP to the Architecture of Reflective Middleware. *Network and Parallel Computing Workshops, NPC Workshops, IFIP International Conference*, Septempter 18-21, 2007.
- Kic97 G. Kiczales et al., Aspect-Oriented Programming. *ECOOP'97*, 220-242. *Springer Lecture Notes in CS*, 1997.
- KiM05 G. Kiczales, M. Mezini, Aspect-Oriented Programming and Modular Reasoning. *Proceedings of international conference of Software engineering*, St. Louis, MO, USA, 2005, pages 49-58.
- LFM06 Otávio Augusto Lazzarini Lemos, Fabiano Cutigi Ferrari, Paulo Cesar Masiero, Cristina Videira Lopes, Testing aspect-oriented programming Pointcut Descriptors. *International Symposium on Software Testing and*

*Analysis, Proc. Of the 2<sup>nd</sup> workshop on Testing aspect-oriented programs*, Portland, Maine, USA, 2006, pages 33-38.

- RaS05 Hridesh Rajan, Kevin J. Sullivan, Classpects: Unifying aspect- and object-oriented language design. *Proc. of the 27<sup>th</sup> international conference on Software engineering*, St. Louis, MO, USA, 2005, pages 59-68.
- WKD04 M. Wand, G. Kiczales, C. Dutchyn, A semantics for advice and dynamic join points in aspect-oriented programming. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, September 2004. Volume 26, Issue 5.