

# Building Frameworks in AspectJ

*Bart Vanhaute*

*Bart De Win*

*Bart De Decker*

*Report CW 318, June 2001*



Katholieke Universiteit Leuven

Department of Computer Science

Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

# Building Frameworks in AspectJ

*Bart Vanhaute*

*Bart De Win*

*Bart De Decker*

*Report CW 318, June 2001*

Department of Computer Science, K.U.Leuven

## **Abstract**

The combination of aspect-oriented programming and framework technology boosts software reuse and brings separation of concerns to a new, more generic level. On the one hand, AOP enables the separate implementation of crosscutting concerns. Frameworks, on the other hand, allow us to reuse and customize a particular implementation in different applications. By means of a security example, we show the ease and power of the AOP language AspectJ to analyze and design a solution in terms of a framework. We also suggest a number of improvements.

**Keywords :** challenge problem, AspectJ, framework, security

# Building Frameworks in AspectJ

Bart Vanhaute      Bart De Win      Bart De Decker  
Department of Computer Science, K.U.Leuven  
Celestijnenlaan 200A  
B-3001 Leuven, Belgium  
{Bart.Vanhaute, Bart.DeWin, Bart.DeDecker}@cs.kuleuven.ac.be

17th April 2001

## Abstract

The combination of aspect-oriented programming and framework technology boosts software reuse and brings separation of concerns to a new, more generic level. On the one hand, AOP enables the separate implementation of crosscutting concerns. Frameworks, on the other hand, allow us to reuse and customize a particular implementation in different applications. By means of a security example, we show the ease and power of the AOP language AspectJ to analyze and design a solution in terms of a framework. We also suggest a number of improvements.

## Keywords

challenge problem, AspectJ, framework, security

## 1 Introduction

The focus of the aspect-oriented community has shifted over the years from designing a specialized language and weaver for each aspect, towards a generally applicable aspect language based on a few clearly defined types of crosscuts. A notable example of such a language is AspectJ. On the up side, the effort of designing a language and weaver now has to be done only once. The down side is that implementing aspects becomes harder because it requires more programming. As a result, reuse of aspect code becomes more an issue.

The way aspects are implemented can vary according to the characteristics of the crosscuts between application and aspect code. First, it can be used to invasively add code to an application in an orthogonal way. In this case, the crosscutting nature of the added code is completely described by the specification of the pointcuts. The application is independent of the aspect, so it remains functional even without the aspect. There are very few concerns that can be described in this way. The best-known example is a debugging aspect. A generic exception handling mechanism is another candidate [8].

At the other extreme, aspect code can be deep crosscutting, when application state, structure or logic influences the aspect code in such a way that the aspect is only applicable in one specific application context. In this case, the aspect forms an integral part of the application. Moreover, an application can be composed of a collection of aspects

together with a core structure on which they are based. A number of large systems have been built this way [7, 10].

Between these two extremes lies a range of aspects that are not orthogonal, but not totally dependent either. These aspects typically address (non-functional) concerns that need some cooperation from the application in order to work. To improve reuse in this wide area, a common approach is to extract the generic part of the aspect from the application specific, and present it as a library of functions and classes. The use of framework technology [6] can take this a step further. The main advantage of frameworks over libraries is that the former can encode and enforce (to some degree) how the code should be used. Especially in the case of non-functional concerns, this enforcement can simplify usage of often complex implementations. The real challenge for aspect programmers is to design the solution in such a way that it is able to combine a general specification with a specialized application.

In this paper, we start from two specific security aspects for which we then try to build a more generally applicable framework, using AspectJ<sup>1</sup>. The paper shows in some detail how to implement concerns as a framework in AspectJ. Some steps are even more general and, with adaptation they should be applicable to other aspect-oriented technologies. To conclude, we point out the weak and strong points of the AspectJ language to handle frameworks.

## 2 Security Aspects

The problem of adding security to an application is very well suited to investigate aspect-based frameworks. On the one hand, it can be solved in a generic manner with reusable mechanisms. For example, access control has the same requirements for most of the applications : allow or deny access to certain resources in the system. Also, the mechanisms must be used in a correct manner. To give a simple example, authentication must have happened before access control can be performed. On the other hand, the specific policies (i.e. who should get access and to what resources) are clearly application dependent.

To summarize, security lies in between the two extremes and is an ideal target for our investigation. To keep a good focus, we will only discuss access control and confidentiality.

### 2.1 Access control

Basically, access control can be described as follows : at a certain point, the application asks the user to authenticate himself, after which it can check access based on this identity. The key to turn this description into an aspect-oriented implementation is the identification of the important (domain) concepts, and how they relate to the application at hand. In this case, these concepts are threefold : the entity that has to be authenticated, the resources for which we want to control access, and the access point or the path from the authenticated entity to the resources. Algorithm 1 shows the skeleton of an implementation<sup>2</sup>, where the pointcuts have already been abstracted from

---

<sup>1</sup>We assume the reader is familiar enough with the language and its semantics. For further information, take a look at [1].

<sup>2</sup>A more elaborate description is available in [2].

the application. For a concrete application, one would fill in the abstract pointcuts to specify where the aspects should be activated<sup>3</sup>.

---

**Algorithm 1** Access Control (generalized)

---

```
abstract aspect Identification {
    Subject subject = null;
    public Subject doLogin() { <login> }
}

abstract aspect Authentication of eachcflowroot(Authentication.accessPoints()) {
    private Subject subject;
    abstract pointcut accessPoints();
    before(Object caller): accessPoints() && hasaspect(Identification){
        Identification id = Identification.aspectOf(caller);
        if(id.subject == null) {
            id.doLogin();
        }
        subject = id.subject;
    }
    public Subject getSubject() { return subject; }
}

abstract aspect Authorization {
    abstract pointcut checkedMethods();
    before(): checkedMethods() {
        Authentication au = Authentication.aspectOf();
        Subject subject = au.getSubject();
        <check access>
    }
}
```

---

## 2.2 Confidentiality

The target of confidentiality typically emerges in different forms. For example, objects containing sensitive information require encryption whenever they are written to a stream (e.g. to a file or to a connection). A particular communication link between different parts of an application might be unsafe. Here, it is useful to encrypt all the traffic over this connection. Perhaps less obvious, sensitive objects residing in memory might be swapped out to disk by the operating system and as such cause security leaks. To avoid this, it is necessary to obfuscate the sensitive internals of the particular object at all times. Due to space limitations, we will only address the first one. Other targets could be handled in a similar manner.

Analogously to the previous example, the following concepts can be identified: the object (or object-structure) that must remain confidential<sup>4</sup>, the owner of the object and the key material used for this purpose. The resulting code has the same structure as in 1. The implementation relies on Java object serialization to add encryption. A pointcut is declared for the point where a session needing confidentiality starts. At that point, the appropriate key is constructed according to some key agreement mechanism.

---

<sup>3</sup>Another important issue is the specific access control policy. Configuration files provide the necessary flexibility. In our implementation, we use JAAS [4].

<sup>4</sup>The location and mechanism chosen to perform the encryption are dependent on the particular target. For the object-centric approach, we rely on Java Cryptography [9] and the serialization mechanism.

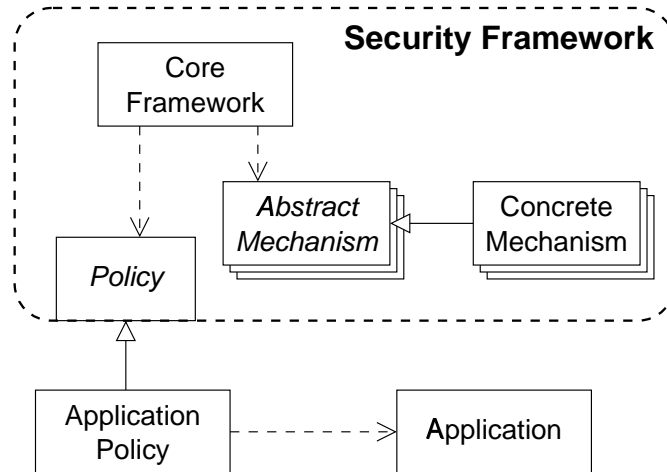


Figure 1: Security framework

### 2.3 Aspect frameworks

The two distinct security requirements of the previous sections can each be painlessly deployed for a concrete application. However, often a combination of different properties is required. Rather than mixing all aspects together and end up with chaos, embedding them in a generic structure leads to a better result. This is where framework technology really comes into play.

To generalize the aspects into a framework, we have to identify the points where aspect customization is desirable (the so called hot spots). Firstly, we want to support variation of the places where the aspects are to be applied. This can be achieved by specifying the pointcuts abstract, and then let an aspect deployer override them. For aspect code that adds additional behaviour or data to application objects, the deployer should also specify how instances of the aspect are attached to the application. These techniques were already applied in the aspect code of the previous examples. Secondly, when functional interchangeability is desired (e.g. the specific login mechanism for Authentication), the implementation of that functionality is extracted from the aspect code into an abstract mechanism, according to the Strategy design pattern [5].

The powerful techniques of abstract pointcut specification and inheritance of AspectJ have enabled us to design a generic framework with a clear structure, as shown in figure 1. The core of the framework contains the bare aspects. The policy part consists of the abstract pointcuts, and thus provides the way to specify the deployment in an application. For each abstract mechanism, several concrete implementations can be available within the framework. In order to instantiate this framework, a security engineer will have to define a concrete policy according to the needs of the application. Furthermore, he or she can implement additional mechanisms.

Combining the security concerns in the framework has additional benefits. Some security requirements are not totally independent of others (e.g. non-repudiation requires authentication). Those dependencies can be inserted and enforced within the framework. Consequently, the security engineer does not need to worry about them any longer. Moreover, by choosing a good representation for the various policies, more

generic policies can be built (e.g. secure session instead of authentication, confidentiality and integrity). The end result is comparable to a specific aspect language for security, but without the cost of a special purpose weaver.

### 3 Discussion

The two examples in the previous section demonstrated that it is possible to design a framework of aspects using the AspectJ language. Constructs such as abstract pointcuts and aspect-inheritance are very helpful for generalization. However, there are still a number of difficulties, especially if the framework becomes more complex. Some of these could be or already have been easily amended by small changes in the language and weaver. A number of difficulties are more fundamental, because they reveal limitations of the current model of AspectJ.

Within the implementation of an advice, we often need access to runtime information in the pointcut. Normally, this is available through parameterization. However, when the pointcut definition is part of the framework, it can often not be foreseen what type of parameters will be required. For this purpose, an introspection API is accessible within advice implementations. Unfortunately, this approach tends to look more like runtime meta-programming. The complexity of writing meta-programs is one of the reasons the aspect-oriented approach was developed. A more open mechanism than parameter passing could be the answer here.

A more profound problem is the use of pointcut definitions. As these definitions are static, they can only be referenced and extended in a static way. This restriction for instance makes it impossible to isolate an abstract pointcut definition into a separate construct, and rely on delegation to select one of many concrete versions, depending on where the original pointcut is used. The source of this problem is the fact that the expressive power in defining pointcuts is limited to a combination of pointcut designators as defined by the AspectJ implementers. We suggest a more open weaving process, such that pointcut definitions can be the result of a function evaluation over a representation of the application structure.

Aspect technologies with a framework approach [3] instead of based on a language can handle the second point much better, as they already provide a representation of the application structure. However, we feel that programming in these frameworks is more difficult, partly because the flexibility is often too high. A mix of a language based approach with a framework based translator could perhaps yield the best result.

It is also interesting to look at the development process of aspect frameworks. A critical point is the definition of the main concepts and their interplay. The best approach is to start from a concrete solution and then try to abstract away the crosscutting points. However, this generalization is far from easy, mainly because reasoning in terms of aspects is still new and not so well understood. Support for aspect-oriented development would be highly beneficial.

### References

- [1] The aspectj web site. <http://www.aspectj.org/>.
- [2] B. De Decker B. De Win, B. Vanhaute. Towards true separation of concerns. Technical report, Departement Computerwetenschappen, 2001.

- [3] T. Elrad M. Fayad P. Netinant C. Constantinides, A. Bader. Designing an aspect-oriented framework in an object-oriented environment. *ACM Computing Surveys Symposium on Application Frameworks*, 32(1), 2000.
- [4] K. Koved A. Nadalin R. Schemers C. Lai, L. Gong. User authentication and authorization in the java platform. In *Proceedings of the 15th annual Computer Security Applications Conference*, December 1999.
- [5] R. Johnson J. Vlissides E. Gamma, R. Helm. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994.
- [6] D. Schmidt M. Fayad. Object-oriented application frameworks. *Communications of the ACM*, 40(10):32–38, 1997.
- [7] G. Murphy M. Kersten. Atlas: a case study in building a web-based learning environment using aspect-oriented programming. In *Proceedings of the 1999 ACM conference on Object-Oriented Programming, Languages and Applications*, October 1994.
- [8] C. Lopes M. Lippert. A study on exception detection and handling using aspect-oriented programming. In *Proceedings of the 22nd International Conference on Software Engineering*, Limerick, Ireland, June 2000.
- [9] Sun Microsystems. Java cryptography extension. <http://java.sun.com/products/jce/index.html>.
- [10] D. Silva U. Kulesza. Reengineering of the jaws web server design using aspect-oriented programming. In *Workshop on Aspects and Dimensions of Concern ECOOP*, Sophia Antipolis and Cannes, France, June 2000.