Aspects and Software Components

A case study of the FRACTAL Component Model

Houssam Fakih [1]

GIP Department
Ecole des Mines de Douai
Douai, France

LIFL Laboratory, UMR CNRS 8022
Lille University of Science and Technology
Lille, France

Noury Bouraqadi [2]

GIP Department
Ecole des Mines de Douai
Douai, France

Laurence Duchien [3]

LIFL Laboratory, UMR CNRS 8022
Lille University of Science and Technology
Lille, France

Abstract

Component-Based Software Development (CBSD) swears software reuse but it suffers from code scattering and tangling. Aspect Oriented Programming (AOP) deals with these problems. We present in this paper, *Fractal-AOP*, an add-on to the Fractal component model that combines AOP and CBSD into an overall model. Combining is achieved by applying AOP principles on the Fractal component model. Indeed, we define new control interfaces on functional components that expose join points. Aspects are defined using plain Fractal generic components. Weaving relies only on classical operations on components, namely: Configuration and assembly. The configuration allows the definition of pointcuts. The assembly connects the components defining aspects with the rest of application components.

Key words: Aspect, Software Component, AOP, CBSD, Integration of aspects and components, FRACTAL component model

1 Introduction

Recently, several works that aim to integrate aspects [8,6] and software components [21], had emerged. We highlighted in [7] three facets allowing to achieve this goal:

(i) The first facet componentises aspect [10,14], *i.e.*, enhances aspects using properties that make software components attractive (e.g. reuse, deployment). Base code could be expressed by either object-based languages or procedural ones. The challenge is to find the equivalent of component's characteristics for aspects.

(ii) The second facet applies Aop on software components [20,3], *i.e.*, defines aspects that must be able to intercept normal component execution in order to perform advices. The aim is to define join points on software components and the way aspects and components must be woven together in order to produce the final enhanced application.

(iii) The third facet unifies aspects and software components [19,12,17]. This facet consists of defining a component model general enough to encompass not only 'traditional' software component concepts, but also Aop ones.

It worth noting that there is a variety of component models. Thus, the integration of AOP concepts into components depends surely on properties of the chosen component model. Indeed, solutions vary from a flat component model (*i.e.*, a model without the composite concept) to a hierarchical model (*i.e.*, a model with the composite concept).

We address in this paper the unification of aspects and components issue. We have chosen the Fractal component model [1] as a case study. This choice is related, on the one hand, to the Fractal model characteristics as the modularity, the extensibility, the support of the composite concept, and on the other hand, to its programming model and its architecture description language (Adl) [13] both simpler than those of actual industrial components as the Corba Component Model (Ccm) [15] and the Entreprise Java Beans (Ejb) [5].

The remainder of this paper is organised as follows: In section 2, we present an overview on the Fractal Component model then we define a diary application by means of Fractal components. Next, in section 3, we describe Fractal-Aop our add-on to the Fractal component model that makes for the unification of aspects and components. Then, in section 4, we show the related works. Finally, we conclude, in section 5, with a short summary and

[1] Email: `fakih@ensm-douai.fr`
[2] Email: `bouraqadi@ensm-douai.fr`
[3] Email: `duchien@lifl.fr`

future works.

2 The Fractal Component Model

2.1 An overview of Fractal

Fractal [1,2] is a tightly-typed hierarchical component model. It defines composite components to provide a uniform view of applications for various levels of abstractions. Besides, one of the Fractal specific characteristics is the sub-component sharing with many composites that aims to model resource sharing. Fractal provides various reflective capabilities for monitoring, deploying and dynamically reconfiguring a running system. Applications built using Fractal components support dynamic structural reconfiguration.

A Fractal component can only be handled through a set of *interfaces*. Interfaces refer to what is called "port" in other component models [21]. The connection of a client interface to a server one is named *binding*. Since Fractal is strongly typed, the server interface type must be a sub-type of the client interface type. Types are checked when we attempt to bind two interfaces.

Components use interfaces when they communicate with each other. A component that requires a service sends a message through one of its *client interfaces* to a *server interface* of a component providing the desired service. Each server interface gives access to a set of operations. And, each client interface defines a set of operations that the component may invoke.

The Fractal component structure is breaking down into two parts:

- A *content* which consists of a finite set of *operations*, *attributes* and possibly *sub-components*, and

- A *membrane* which contains component's interfaces. This membrane monitors the component content by intercepting incoming and outgoing operation invocations. We use this feature for supporting Aop (see section 3).

A Fractal membrane contains two kinds of interfaces: The functional interfaces and the *control* ones. Functional interfaces are related to the component business domain. While control interfaces allow to introspect and to reconfigure components (e.g. life cycle, interfaces, attributes, bindings).

2.2 An example: A diary application using Fractal

We illustrate, in Figure 1, an example showing the core business of a diary application built by making use of Fractal. Each diary provides a Meeting Manager (MM) interface that permits to invoke two operations: *addMeeting* and *removeMeeting*. Each Meeting Organiser component helps organise meetings between diary's owners. This component provides a Meetings Organiser (MO) interface that permits to invoke one operation: *arrangeMeeting*. It re-

quires also a MM *collection*[4] client interface that could be bound with the MM server interfaces of *multiple* diaries. Upon receipt of arrangeMeeting call, Organiser manages to find a date to put meeting in appropriate time for all participants.

It should be noted that, by convention, functional server interfaces are represented at the left of the component and functional client ones at the right. We omit, in Figure 1, to show controller interfaces allowing for the introspection and the reconfiguration of components.
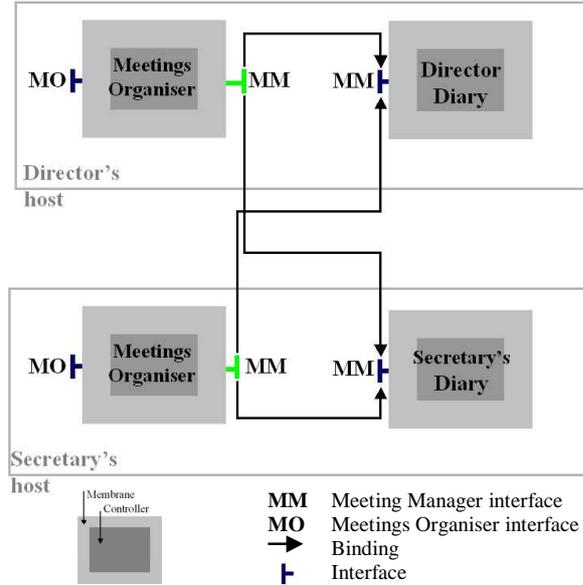

Fig. 1. A Diary application with FRACTAL

3 FRACTAL-AOP: An AOP Add-on to the FRACTAL model

The diary application presented in section 2.2 has to deal with crosscutting concerns, i.e., aspects. Examples of such aspects are concurrency of requests for meetings addition and removal, persistence of diaries and security which includes the authenticity of entities before allowing them to change meetings in a diary.

We describe, in this section, FRACTAL-AOP an add-on to the FRACTAL component model that supports AOP. We show that FRACTAL-AOP enables the definition of aspects building blocks by making use of components. In this context, the weaving consists of the assembly of those components with application functional components.

Please note that we make use of the diary example throughout this section to illustrate our approach.

---

[4] We mean by collection a property of a FRACTAL interface type that indicates how many interfaces of this type a given component may have.

3.1 The Fractal-Aop component model

To support Aop in Fractal, we add on the Fractal component model a new membrane that holds two extra control interfaces namely the *Execution Controller (*cEc*)* and the *Proceed Controller (*sPc*)* (see Figure 2).
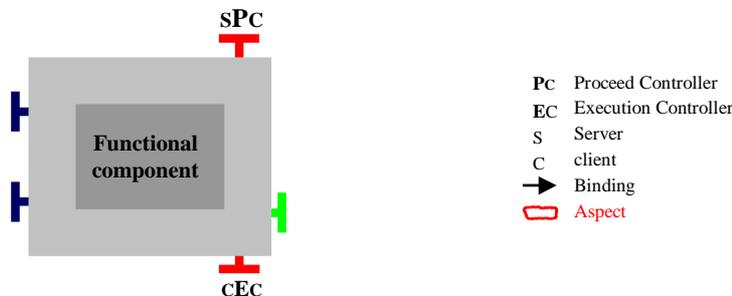

Fig. 2. A Fractal-Aop functional component

The cEc interface is a *client-control* interface. It exposes join points into the component execution flow. This exposition is totally transparent to the application functional code developper, since it relies only on the membrane interception capabilities.

The sPc interface is a *server-control* interface. It allows to proceed any captured join point.

Both cEc and sPc interfaces have the same signature (see Figure 3). Fractal-Aop defines as join points the send and the reception of operations invocations. Join points set includes also the attribute read / write access, the modification of the component's life cycle state, the connection / disconnection of two components, the addition / removal of a sub-component to / from a composite.

Both join points related to the connection / disconnection of two components and the addition / removal of a sub-component to / from a composite are specific to the component model. These join points can be useful in the case of

```
package org.objectweb.fractal.api.aop;
interface ExecutionController {
    receiveMessage(string aMessage, any[] args,
     Interface itfReceiver, Component receiver,
     Interface itfSender, Component sender);
    sendMessage(string aMessage, any[] args,
     Interface itfSender, Component sender,
     Interface itfReceiver, Component receiver);
    getAttribute(string attributeName, any newValue, Component cmp);
    setAttribute(string attributeName, any newValue, Component cmp);
    connect(Interface itf1, Component cmp1, Interface itf2, Component cmp2);
    disconnect(Interface itf1, Component cmp1, Interface itf2, Component cmp2);
    addSubComponent(Component subComponent, Component composite);
    removeSubComponent(Component subComponent, Component composite);
    changeState(string newState, Component cmp);
}
```

Fig. 3. Execution Controller Api written in the Fractal Idl

3.2   An aspect in Fractal-Aop

An aspect in Fractal-Aop is composed of a set of generic (and hence reusable) components and specific ones. These components are of two kinds: *Advice* components and *Weaving* components. The weaving is enabled while these components are connected to application functional components as shown in Figure 4.
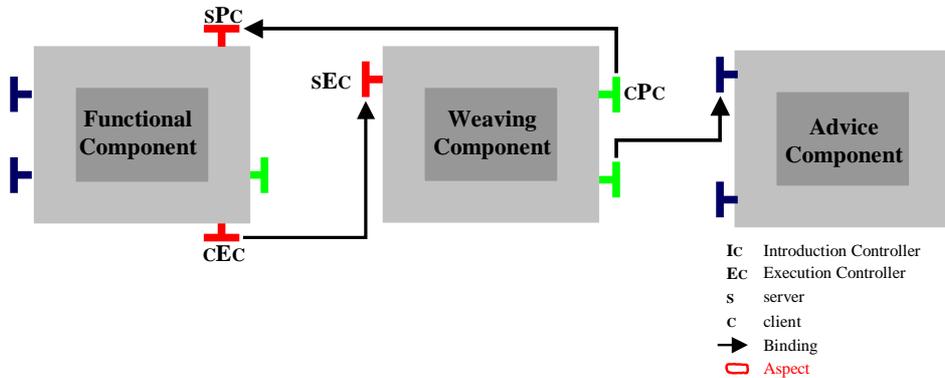

Fig. 4. Fractal-Aop: An approach applying Aop on Fractal

3.2.1   Advice components

Advice component is a fractal component that defines aspect's computations belonging to only one crosscutting concern or property. Advice components are defined in a generic way, *i.e.*, independently of any context. Operations in the interfaces of a given Advice component are related to the core business of an aspect.

3.2.2   Weaving components

Weaving components act as a "meta-components" [5] . It controls the behaviour of some functional component since it decides what to do for each join point in the application execution flow. When a join-point matches one of the aspect's point-cuts, the weaving component triggers the right operations in the advice component. Attributes of the weaving component define point-cuts, and which operations to trigger before and/or after join-points.

The Weaving component makes use of the sPc interface to proceed functional code at each join-point. So, once the weaving component has triggered the "before processing", it allows the join-point execution to proceed, and then it triggers the "after processing" if any. Note that the weaving component could interfere on the execution of the join-point. For example, it can avoid the execution of the join-point, or make it proceed with new parameter values.

---

[5] We made here an analogy with meta-object found in works related to reflection in OO languages [11].

The Weaving component addresses also conflicts resolution known as the feature interaction problem [18], *i.e.*, the case of several aspects that must act upon the same join point.

3.3   Weaving mechanism in Fractal-Aop

Aspects are defined as an assembly of components. Weaving aspects with functional components is broken down into two steps, *i.e.*, the assembly mechanism and the definition of weaving rules:

- The first step consists of the definition of point-cuts and weaving rules at these point-cuts on functional components. That includes *where* and *when* this additional behaviour will be included. This step is represented in Figure 4 by the Weaving component, and

- The second step consists of the assembly of a functional component with the Weaving one, *i.e.*, the Weaving component is aware of the component reference. This step is represented, in Figure 4, by links that bind, on the one hand, both cEc and sEc interfaces of the functional component, and, on the other hand, both sPc and cPc.

The binding/unbinding of these interfaces can be made at the runtime. Then the weaving and unweaving of an aspect and a component can be performed at runtime by stopping the functional component execution without restarting them.

3.4   The authentication aspect of the diary application

Authentication is one aspect of the diary application presented in section 2.2. We illustrate in Figure 5 how the authentication aspect can be supported by making use of Fractal-Aop.

We employ the upfront login authentication approach [9] that consist of asking the caller for the user name and the password when he requests access to the diary.

The authentication Advice component provides an *Authentication Manager (AM)* interface that contains two operations: *login* and *isAuthenticated*. The former operation takes two parameters: The user name and the password, and checks the authenticity of a user. The latter operation checks if a user is already authenticated or not.

The authentication aspect includes two Weaving components: The one installed at the caller side (the Weaving component 1) and the other installed at the callee side (the Weaving component 2).

We show, in Figure 6, how the aspect takes control, at the callee side, of the execution flow upon the sending of a message from the MM interface of the Meeting Organiser component.

Firstly, he knows that a join point (the send of the message addMeeting from the interface MM of the Meetings Organiser) is occurred. It compares
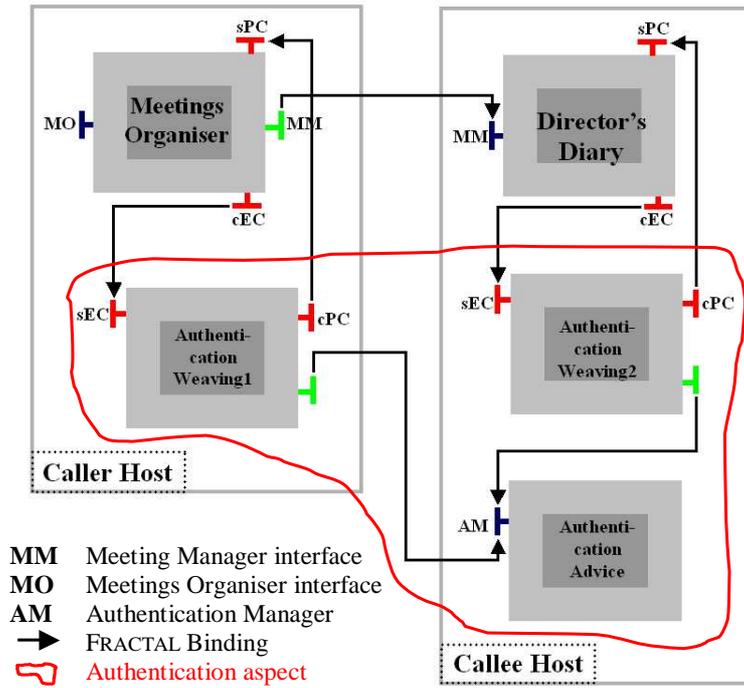
Fig. 5. the definition of an authentication aspect on the director's diary

the join point with point-cuts defined in the Weaving component 1. So, it finds that the join point matches the point-cut. It performs the before advice that consist of a logging operation to be authenticated. Then it proceeds the original message.

Besides, at the caller side, the director's diary receives an addMeeting message. Upon the receipt of this message, the aspect takes control and the Weaving component 2 acts as we had explained above for the weaving component 1 to check if the join point matches a point cut. The difference is that the Weaving component 2 checks if it is about an authenticated user or not. The proceed call is only performed in case of authenticated users. We don't show a diagram for the callee side because it is quite similar to the diagram of Figure 6.
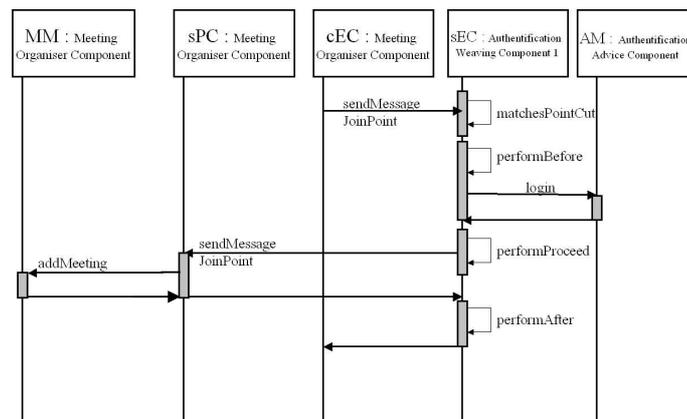

Fig. 6. A sequence diagram among objects in the caller side

4 Related Works

Related works can be organised in two groups: Works related to the integration of aspects and software components and works that propose increments to Fractal.

4.1 Works related to the integration of aspects and components

JAsCo [20] applies Aop on the Java Bean component model. JAsCo introduces two entities: *An aspect bean* and *a connector*. An aspect bean describes crosscutting behaviours by making use of *hooks*. Hooks specify *when* (kind of abstract point-cuts) the normal execution of component methods should be intercepted and *what* (advices) extra behaviour should be executed. A connector is used for deploying one or more hooks on a specific context. It specifies *where* the crosscutting behaviour should be deployed (concrete point-cuts) and it addresses also the feature interaction problem. Connectors can be loaded and unloaded at runtime. A special registry called *connector registry* detects whether connectors are removed or added to the system which can take then appropriate actions. Fractal-Aop modularizes better the aspects building blocks. Advices are defined as traditional components. The Fractal-Aop Weaving component defines both the "when" and the "what" of the aspect. The configuration of this component defines the aspect's "where". So, all interactions among functional components and Advice ones are defined in a Weaving component or composite. To modify these interactions, we should just modify this Weaving component. In this case, functional and Advice components remain unchanged.

JBoss-Aop [3] has been designed to be used on top of JBoss, which is a J2ee-based application server. However it can be used as a standalone Aop framework. JBoss-Aop tries to solve the limitation of flat component-based platforms based on load-time transformations of Java classes. JBoss-Aop uses interceptors that work as the advice construct in AspectJ. They can be used to intercept method invocations, constructor invocations and field access. Pointcuts and Introductions are defined by making use of Xml descriptors. JBoss-Aop applies Aop on java classes and not on component concept itself.

4.2 Works related to increments to Fractal

David and al. [4] defines an add-on to the Fractal component membrane that aims to reify all messages it receives. Instead of being sent to its original target the reified message is sent to a sub-component which implements a meta-level message invocation interface. The component manages then the message in a generic way and could perform a pre- or post-processing or even replaces the original behaviour. The meta sub-component resembles to the union of the weaving and the advice components in our approach but the use of Aop concepts is not explicit as in the ours.

Pessemier and al. [16] apply Aop on Fractal components. This approach defines a new controller interface on functional components that captures only the messages invocation. This interface is limited comparing to our execution controller interface. The weaving is released by means of a new binding type called *direct crosscut binding*. Besides, Fractal-Aop makes use of the traditional types of binding defined in the Fractal specification to achieve weaving.

5 Conclusions and perspectives

In this paper, we have presented Fractal-Aop our approach that makes for the unification of aspects and components. An aspect, in Fractal-Aop, consists of two parts: The specific one and the generic one.

- the specific part is composed of weaving components that define pointcuts, weaving rules and conflict resolution.
- the generic part is composed of advice components that define the core business of an aspect.

Fractal-Aop has been implemented on the top of FracTalk [6], the Smalltalk implementation of the Fractal Component Model.

One perspective of our work is to study how to deal with composite components. We must address especially how an aspect would be weaved with sub-components of a composite.

Another perspective is to study issues related to the *Introduction* concept. The Introduction in components means that we can modify the component's behaviour/structure by allowing the addition/removal of operation(s), interface(s) and attributes to/from a component. In this case, we must address problems related to the modification of a component type.

References

[1] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quema, and Jean-Bernard Stefani. An Open Component Model and Its Support in Java. In *Proceedings of the International Symposium on Component-based Software Engineering*, Edinburgh, Scotland, May 2004.

[2] Eric Bruneton, Thierry Coupaye, and Jean-Bernard Stefani. *The Fractal Component model specification*. ObjectWeb Consortium, France Telecom and INRIA, February5 2004. http://fractal.objectweb.org.

[3] Bill Burke, Austin Chau, Marc Fleury, Adrian Brock, Andy Godwin, and Harald Gliebe. JBoss aspect oriented programming. http://www.jboss.org/, February 2004.

---

[6] `http://csl.ensm-douai.fr/FracTalk`

[4] Pierre-Charles David and Thomas Ledoux. Towards a framework for self-adaptive component-based applications. In *Proceedings of DAIS'03*, Lecture Notes in Computer Science, Paris, November 2003. Federated Conferences, Springer-Verlag.

[5] Linda Demichiel, Ümit Yalçinalp, and Sanjeev Krishnan. *Enterprise JavaBeans Specification, Version 2.0*. Sun Microsystems Inc., version 2.0 edition, August 2001.

[6] Tzilla Elrad, Robert E. Filman, and Atef Bader. Aspect-oriented programming. *Comm. ACM*, 44(10):29–32, October 2001.

[7] Houssam Fakih, Noury Bouraqadi, and Laurence Duchien. Towards integrating aspects and components. In Yvonne Coady and David Lorenz, editors, *Proceedings of the Third AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software*, Lancaster University, UK, March 2004.

[8] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-Oriented Programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[9] Ramnivas Laddad. *AspectJ in Action*. Manning Publications Co., Grennwich, Conn., 2003.

[10] Karl Lieberherr, David H. Lorenz, and Mira Mezini. Programming with aspectual components. Technical Report NU-CCS-99-01, College of Computer Science, Northeastern University, Boston, MA 02115, March 1999.

[11] Pattie Maes. Concepts and experiments in computational reflection. pages 147–155, Orlando, FL, USA, October 1987. ACM Press.

[12] Sean McDirmid and Wilson C. Hsieh. Aspect-oriented programming with jiazzi. In *Proceedings of the 2nd international conference on Aspect-oriented software development*, pages 70–79. ACM Press, 2003.

[13] Nenad Medvidovic and Richard N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, January 2000.

[14] Mira Mezini and Klaus Ostermann. Conquering aspects with caesar. In *Proceedings of the 2nd international conference on Aspect-oriented software development*, pages 90–99. ACM Press, 2003.

[15] Object Management Group. *Corba Component Model*, 1999.

[16] Nicolas Pessemier, Lionel Seinturier, Laurence Duchien, and Olivier Barais. Partage de composants fractal pour l'aop. In *First French Workshop on Aspect-Oriented Software Development (JFDLPA 2004)*, Paris, France, September14 2004. In French.

[17] Monica Pinto, Lidia Fuentes, and Jose Maria Troya. DAOP-ADL: an architecture description language for dynamic component and aspect-based development. In *Proceedings of the second international conference on Generative programming and component engineering*, pages 118–137. Springer-Verlag New York, Inc., 2003.

[18] E. Pulvermüller, A. Speck, J.O. Coplien, M. D'Hondt, and W. DeMeuter, editors. *Proceedings of the Workshop on Feature Interaction in Composed Systems; In Association with the 15th European Conference on Object-Oriented Programming (ECOOP) 2001*. Universitaet Karlsruhe, June 2001.

[19] Davy Suvée. Fusej: Achieving a symbiosis between aspects and components. In *5th GPCE Young Researchers Workshop 2003*, Erfurt, Germany, September 2003.

[20] Davy Suvée, Wim Vanderperren, and Viviane Jonckers. JAsCo: an aspect-oriented approach tailored for component based software development. In *Proceedings of the 2nd international conference on Aspect-oriented software development*, pages 21–29. ACM Press, 2003.

[21] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. ACM Press and Addison-Wesley, New York, NY, 1998.