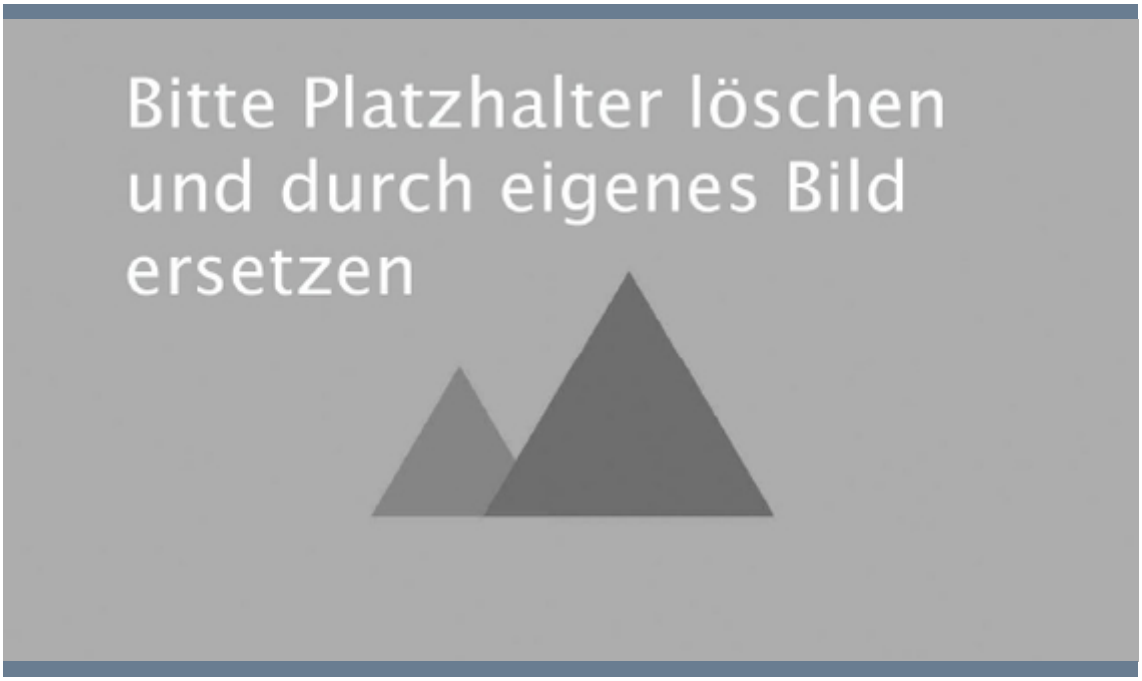

Aspektorientierte Programmierung (AOP) mit AspectJ

Bericht 1 - Infoseminar 2015

Studiengang:	Informatik
Autoren:	Emanuel Knecht, David Aeschlimann
Betreuer:	Prof. Dr. Jürgen Eckerle
Datum:	12. Oktober 2015

Inhaltsverzeichnis

1. Einleitung

Dieses Dokument ist der schriftliche Teil des Modules Informatikseminar an der Berner Fachhochschule. In den kommenden Kapiteln wird die Aspektorientierte Programmierung mit AspectJ vorgestellt und erklärt.

1.1. Auftrag

Der Auftrag ist es die folgenden Fragen mit diesem Bericht zu beantworten.

"Was versteht man unter dem Konzept der Aspektorientierten Programmierung?
Worin besteht der Vorteil gegenüber der OOP?
Erläutern Sie die wichtigsten Methoden und Ideen von AspectJ und stellen Sie heraus, in welcher Form OOP erweitert wird." [1]

Unsere Erkenntnisse werden in diesem Dokument festgehalten. Anschliessend an die Abgabe dieses Berichtes erfolgt eine Präsentation im Plenum mit Fragerunde und Diskussion.

In der ersten Besprechung mit dem betreuenden Dozenten wurde uns nahegelegt auf eine zu technische und detailreiche Ausarbeitung des Themas zu verzichten und stattdessen den Fokus auf die unterliegenden Konzepte und Vorteile der Aspektorientierten Programmierung zu legen insbesondere in der Präsentation.

1.2. Vorgehen

In einem ersten Schritt musste das notwendige Wissen aufgebaut und gefestigt werden. Dazu wurden verschiedenste Informationsquellen konsultiert. Als eine wichtige Basis dieses Berichtes dient jedoch das Buch „AspectJ in Action"[2]. Nach gemeinsamer Ausarbeitung der Struktur unseres Berichtes teilten wir die Kapitel auf und arbeiteten individuell weiter.
Durch Gegenlesen der vom Partner verfassten Abschnitten gelang es uns Fehler zu erkennen und einige Themen verständlicher zu formulieren. Die Präsentation basiert auf dem Bericht, der Fokus liegt jedoch auf dem Kapitel Aspektorientierte Programmierung.

2. Aspektorientierte Programmierung

2.1. Geschichte

Das Konzept der Aspektorientierten Programmierung wurde im Xerox PARC (Palo Alto Research Center Incorporated) entwickelt und gewann ab 1995 an Wichtigkeit. Wie bei allen neuen Spezifikationen war der Umfang und die Bestandteile der Aspektorientierten Programmierung zuerst nicht klar abgegrenzt. Gregor Kiczales und sein Team waren massgeblich an der Entwicklung von AOP beteiligt.
Nach Entwicklung des theoretischen Grundlage folgte im Jahre 1998 eine erste Version von AspectJ, eine Implementation von AOP in Java. Die Version 1.0 von AspectJ wurde jedoch erst im Jahre 2002 nach weiterer Forschung veröffentlicht.[3]
Die Aspektorientierte Programmierung wurde durch die Publikation von AspectJ bekannt und es wurden seither Erweiterungen für die meisten populären Programmiersprachen entwickelt.

Die Entwicklung und der Betrieb von AspectJ wurde von XeroX Parc an Eclipse weitergereicht. Dort läuft AspectJ bis heute als Open-Source Projekt weiter. Mit der Version 1.8.7 wurde am 9. September 2015 die aktuellste Version veröffentlicht.

2.2. Motivation

Einer der Hauptgründe warum AOP entwickelt wurde ist die erweiterte Modularität die damit erreicht werden kann. Beim Design eines Systems werden in der Regel verschiedene Kategorien von Funktionalitäten aufgestellt und so die Software in verschiedene sogenannte Anliegen aufgeteilt. Dabei unterscheidet man zwischen den folgenden zwei Gruppen:

- Kernanliegen (core concerns)
  Hierbei handelt sich um die Kernfunktionalität der Applikation, die sogenannte Business Logic. Diese Gruppe beinhaltet zum Beispiel den Datenbankzugriff, die Interagierung mit dem Benutzer etc.
- System Übergreifende Anliegen (cross-cutting concerns)
  Andere Funktionalitäten wie das Logging, die Sicherheit, Concurrency sowie Transaktionen betreffen das gesamte System.

Diese Gruppen dienen als Grundlage zur Veranschaulichung, warum gerade bei der Modularisierung die OOP Schwachstellen aufweist.

2.2.1. Objektoriente Programmierung

Mit der Objektorientierten Programmierung wurden viele Probleme und Unschönheiten von Prozeduralen Sprachen gelöst. Die OOP besitzt riesige Vorteile, welche die Softwareentwicklung vereinfachen. Einige der Kernbestandteile von OOP sind:

- Encapsulation: Daten und Methoden zur Veränderung derjenigen werden in Objekten gekapselt
- Inheritance: Das Verhalten oder die Daten können von einer Klasse geerbt werden.
- Polymorphism: ....

Die OOP erlaubt es Code modular zu strukturieren und Daten zu kapseln. Mit steigender Komplexität jedoch wird es schwierig den Code klar zu trennen und Abhängigkeiten so klein wie möglich zu halten.

Die Kernanliegen der Applikation werden in Klassen der Business Logic abgebildet. Diese Klassen werden jedoch sehr schnell durch den Code der System übergreifenden Anliegen "verschmutzt", so dass eine Klasse nicht nur für ein Anliegen verantwortlich ist. Dadurch wird das Single Responsibility Principle verletzt. Die folgende Graphik zeigt eine solche Beispielklasse, welche das beschriebene Phänomen veranschaulichen soll. Nur ein kleiner Teil der Methode (rot markiert) beschäftigt sich mit der Ausführung des Kernanliegens dieser Klasse, der Rest mit den System übergreifenden Anliegen.

```
public class SomeClass {

    // Core data members
    // Logging object reference
    // Concurrency lock

    public void someOperation(Object param) {
        // check input parameter
        // Ensure authorization
        // Lock object (thread safety)
        // Start transaction
        // Perform operation
        // Log successful performance of operation
        // Commit transaction
        // Unlock object
    }

}
```

Abbildung 2.1.: Beispielsklasse - Motivation für AOP

Code Tangling Von Code Tangling (to tangle - sich verwickeln) spricht man wenn ein Modul verschiedene Anliegen bearbeitet. Als Beispiel kann man die Abbildung 2.1 nehmen. Während der Designphase werden die Anliegen seperat entworfen und in der Implementation werden alle Funktionen verwickelt.

Code Scattering Bei Code Scattering (to scatter - streuen) ist die Perspektive eine andere. Die Funktionen eines Moduls werden in verschiedene anderen Modulen verwendetet und so im ganzen System gestreut. Folgende Abbildung soll dies anhand eines Sicherheitsmoduls veranschaulichen.

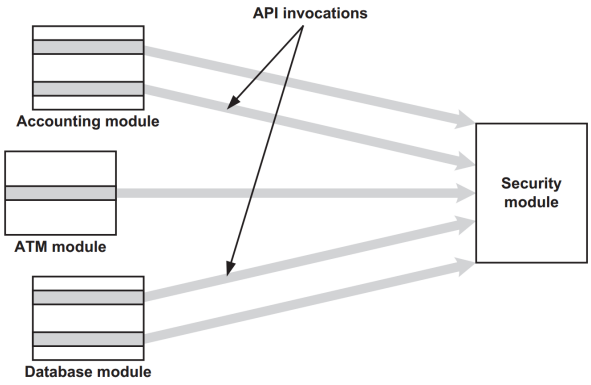

Abbildung 2.2.: Code Scattering (Source [2, p 54])

Bei Verwendung von OOP ist das Code Tangling und Code Scattering unumgänglich. Selbst bei einem perfekt designten System werden diese Phänomene vorhanden sein. Dies beeinflusst Software Design und Entwicklung auf vielerlei Arten: Schlechte Nachvollziehbarkeit, weniger Produktivität, weniger Wiederverwendbarkeit von Code, viele repetitive Arbeiten, schlechtere Qualität und Wartbarkeit von Applikationen. Aus diesen Gründen macht es Sinn nach Alternativen zu suchen, ohne jedoch auf die Vorzüge von OOP zu verzichten.

2.2.2. Modularisierung mit AOP

Bleiben wir beim Beispiel eines Security Moduls; Das Modul wird mit Klassen implementiert und mittels Interfaces gegen aussen sichtbar gemacht. In der OOP werden nun alle Codeteile welche Securityfunktionen verwenden möchten einen Aufruf dieses Moduls beinhalten. Bei einer Änderung in der API müssen unter Umständen tausende Aufrufe ebenfalls geändert werden. Mit AOP jedoch beinhalten die Client Module keine Aufrufe mehr. Aufrufe des Sicherheitsmoduls werden bei genau definierten Punkten im Code automatisch ausgelöst. Als Beispiel kann definiert werden, dass bei allen öffentlichen Methoden einer Klasse vor Ausführung des Bodys die Berechtigung des Benutzers geprüft wird. Bei dieser Deklaration der Einstiegspunkte im Code und des in diesen Fällen auszuführenden Codes spricht man von einem Aspect. Um diese automatische Ausführung möglich zu machen muss der Code mit dem des Aspects verwebt werden. Man spricht hierbei von Weaving.

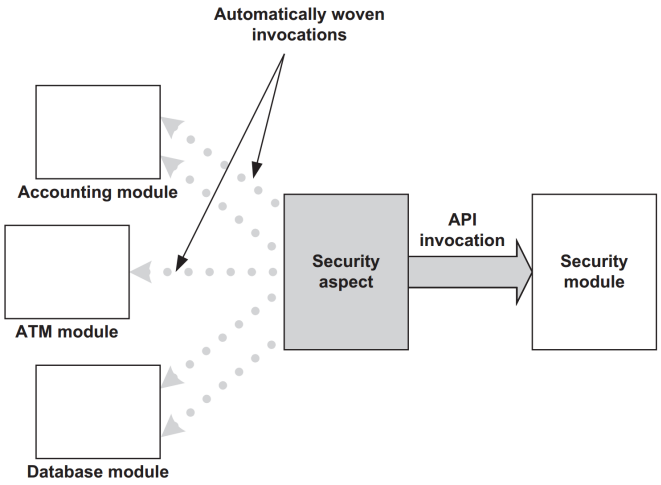

Abbildung 2.3.: Systemdesign mit AOP (Source[2, p 55])

2.3. Aspektorientierte Sprache

Die Aspektorientierte Programmierung ist eine Methode und muss um verwendet werden zu können spezifiziert und danach implementiert werden.

2.3.1. Spezifizierung

Programmiersprache Als Grundlage für die Entwicklung eines Aspektorientierten Programms dient eine Programmiersprache. Es werden in der Regel bestehende Programmiersprachen wie C, C++, CSharp und Java verwendet. Mit dieser Sprache werden die einzelnen Module unabhängig entwickelt ohne auf einander zuzugreifen.

Weaving Rules Spezifizierung Die Sprache muss eine Möglichkeit bieten um diese entwickelten Module miteinander verknüpfen zu können. Dazu müssen sogenannte Weaving Rules deklariert werden. Die Weaving Rules bestimmen wie der Code verknüpft wird. Hierfür können Standardelemente einer Sprache verwendet werden (Annotations) oder auch die bestehende Sprache um neue Elemente erweitert werden (Keywords).

2.3.2. Implementation

Eine Implementation einer AOP Sprache kann in zwei Schritte gegliedert werden. Zuerst werden die verschiedenen Anliegen mittels Weaving rules verknüpft und daraus anschliessend ausführbarer Code konvertiert. Ein sogenannter Weaver führt diese Aufgaben aus. Es gibt drei verschiedene Typen von Weavern:

- Source-to-Source
  Der Sourcecode der Core und Crosscutting concerns wird zuerst verwoben und dieser neu entstandene Source Code von einem regulären Compiler kompiliert.
- Byte Code
  Der Sourcecode der Core und Crosscutting concerns wird zuerst kompiliert und der daraus entstandene Byte Code wird vom Weaver verknüpft.
- Load time
  Vergleichbar mit dem Byte Code weaving, ausser dass der Verknüpfungsvorgang erst beim Aufrufen des Programms statt findet.

2.4. Konzepte

- Identifizierbare Punkte in der Ausführung des Systems
- Selektion der gewünschten Punkte
- Veränderung des regulären Programmablaufs
- Veränderung der statischen Struktur des Systems
- Ort zur Deklaration

2.5. Programmiersprachen

Java

.NET Framework

C++

3. AspectJ

3.1. Bestandteile von AspectJ

3.1.1. Common crosscutting

3.1.2. Dynamic crosscutting

3.1.3. Static crosscutting

Inner-type declaration

Weave-time declaration

3.2. Syntaxvarianten

3.3. Join-Point Model

3.4. Weaving

3.4.1. Source weaving

3.4.2. Binary weaving

3.4.3. Load-time weaving

3.5. Entwicklungstools

4. Schlussfolgerungen

4.1. Nutzen von AspectJ

4.2. Nachteile von AspectJ

4.3. Alternativen

4.4. Fazit

Selbständigkeitserklärung

Wir bestätigen, dass wir die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der im Literaturverzeichnis angegebenen Quellen und Hilfsmittel angefertigt habe/n. Sämtliche Textstellen, die nicht von uns stammen, sind als Zitate gekennzeichnet und mit dem genauen Hinweis auf ihre Herkunft versehen.

Ort, Datum:	Biel, 12. Oktober 2015	
Namen Vornamen:	Emanuel Knecht	Aeschlimann David
Unterschriften:	.....	.....

Literaturverzeichnis

[1] Dozenten, "Auftragsbeschreibungen informatikseminar." [Online]. Available: https://moodle.bfh.ch/mod/data/view.php?d=462

[2] R. Laddad, *AspectJ in Action 2nd Edition*. Manning Publications, 2009, vol. 2.

[3] C. V. Lopes, "Aspect-oriented programming:an historical perspective," 2002. [Online]. Available: http://isr.uci.edu/tech_reports/UCI-ISR-02-5.pdf

Abbildungsverzeichnis

Tabellenverzeichnis

A. Demoprogramm

Dieses Demoprogramm wurde mit Eclipse Mars und den AspectJ Developer Tools für Eclipse entwickelt und getestet. Es soll die grundlegenden Möglichkeiten und die Funktionsweise von AspectJ veranschaulichen.

A.1. Programmbeschreibung

A.2. Source Code