

Bitte Platzhalter löschen
und durch eigenes Bild
ersetzen



Aspektorientierte Programmierung (AOP) mit AspectJ

Bericht 1 - Infoseminar 2015

Studiengang: Informatik
Autoren: Emanuel Knecht, David Aeschlimann
Betreuer: Prof. Dr. Jürgen Eckerle
Datum: 10. Oktober 2015

Inhaltsverzeichnis

1. Einleitung	2
1.1. Auftrag	2
1.2. Vorgehen	2
2. Aspektorientierte Programmierung	3
2.1. Geschichte	3
2.2. Motivation	3
2.3. Aspektorientierte Sprache	4
2.4. Konzepte	4
2.5. Programmiersprachen	4
3. AspectJ	6
3.1. Bestandteile von AspectJ	6
3.2. Syntaxvarianten	6
3.3. Join-Point Model	6
3.4. Weaving	6
3.5. Entwicklungstools	6
4. Schlussfolgerungen	7
4.1. Nutzen von AspectJ	7
4.2. Nachteile von AspectJ	7
4.3. Alternativen	7
4.4. Fazit	7
Selbständigkeitserklärung	8
Literaturverzeichnis	9
Abbildungsverzeichnis	9
Tabellenverzeichnis	10
A. Demoprogramm	12
A.1. Programmbeschreibung	12
A.2. Source Code	12

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 der 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.[?]

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. Objektorientierte 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 Beispielsklasse, 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

Code Scattering

2.2.2. Modularisierung mit AOP

2.3. Aspektorientierte Sprache

2.3.1. Spezifizierung

2.3.2. Implementation

2.4. Konzepte

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, 10. Oktober 2015

Namen Vornamen: Emanuel Knecht Aeschlimann David

Unterschriften:

Literaturverzeichnis

- [1] Dozenten, "Auftragsbeschreibungen infoseminar." [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.

Abbildungsverzeichnis

2.1. Beispielsklasse - Motivation für AOP	4
---	---

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