Vortrag im Rahmen der Vorlesung MSI an der Hochschule Mannheim

06.04.2006

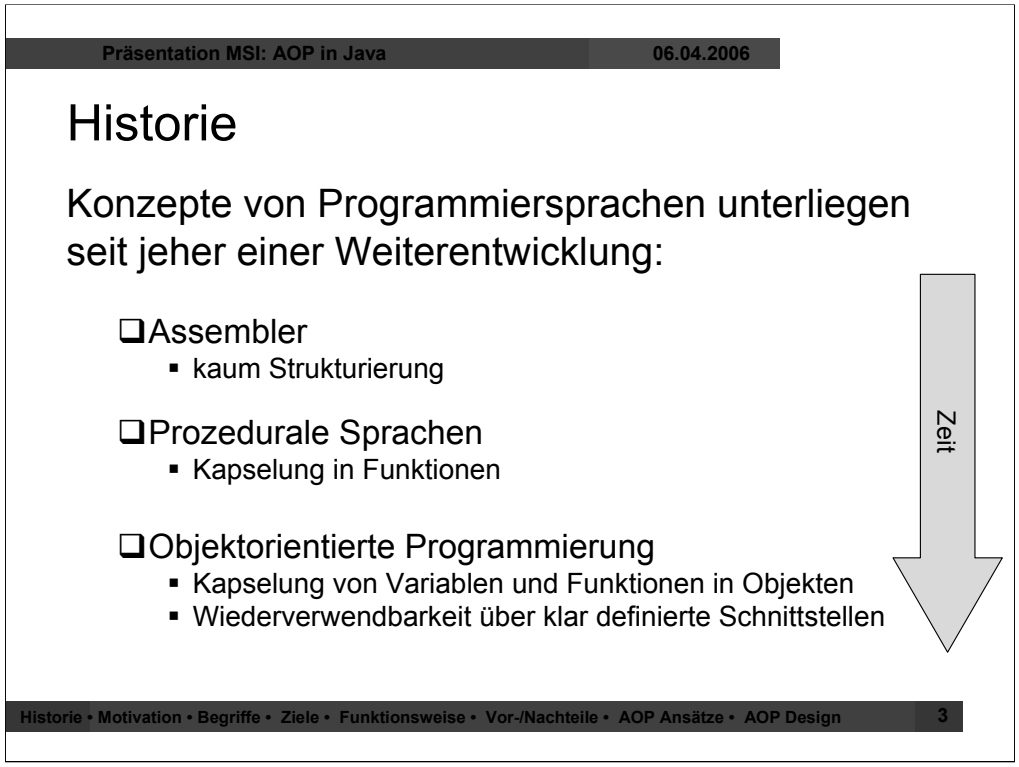
Aspektorientiertes Programmieren in Java

Vortragende: Anton Wagenblaß (anton.wagenblass@gmx.de)

Daniel Pech (daniel-p@gmx.de)

Übersicht

1. **Historie**
2. **Motivation**
3. **Begriffe**
4. **Ziele**
5. **Funktionsweise**
6. **Vor- und Nachteile**
7. **AOP Ansätze für Java**
8. **Aspekt-Orientiertes Design**



Der Assembler kapselt hardwareabhängige Binärbefehle in kurze generalisierten Zeichenketten.

Prozedurale Sprachen kapseln einfache Funktionen innerhalb von einzelnen Prozeduren.

Objektorientierte Anwendungsentwicklung und Programmierung (OOP) kann zu Recht als Stand der Technik angesehen werden. Grund für die weite Verbreitung ist die Möglichkeit einer sauberen Strukturierung der zu realisierenden Software mittels Objekte, welche zusammengehörige Variablen und Funktionen kapseln.

Motivation (1)

Software hat zwei verschiedene Arten an Anforderungen:

- funktionale Anforderungen (Core-Level-Concerns)

  → abgedeckt durch OOP

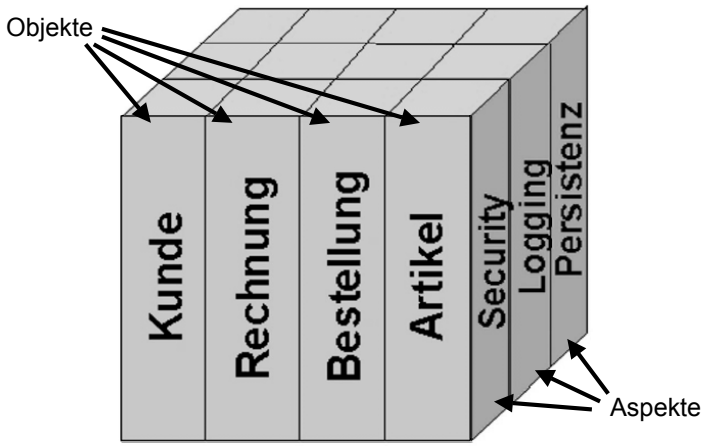
- technische Anforderungen (System-Level-Concerns)

  → nicht elegant lösbar in OOP

concern (engl.) – Anliegen, Belang

Motivation (2)

- Beispiel: Bestellsystem



Motivation (3)

- Probleme bei OOP
  - Code-Tangling
    - Gleichzeitige Präsenz mehrerer Aspekte in einem Modul, die eigentliche Funktionalität geht dabei unter
      → schlecht lesbar, schwer verständlich
  - Code-Scattering
    - Streuung eines Aspekts über mehrere Module
      → kaum wartbar

tangle (engl.) – Durcheinander, Verwirrung

scattering (engl.) – Streuung

Begriffe (1)

- Aspekt
  - orthogonal zur Objektfunktionalität
  - nicht getrennt kapselbar
  - mehrfaches Auftreten in verschiedenen Objekten
  - behindert Wiederverwendung der Objekte
  - senkt Verständlichkeit

Begriffe (2)

- Concern
  - Anforderung an ein Programm
- Join Point
  - Synchronisationspunkt im Programmfluss
    - bei Initialisierung eines Objektes
    - vor / nach Zugriff auf Attribut einer Klasse
    - vor / nach Ausführung einer Methode
    - bei Auftreten von Exceptions

to join (engl.) – vereinigen, verbinden

Begriffe (3)

- Code Weaver
  - spezieller Precompiler
  - verknüpft Aspektcode und Funktionscode
  - 3 verschiedene Typen
- Point Cut
  - Prädikat über einen oder mehrere Join Points
  - bei Zutreffen Ausführung des zugeordneten Codes
  - Point Cut kann auch ein Join Point darstellen

weaver (engl.) – Weber, Weberin

Begriffe (4)

- Advice
  - Code des Aspekts
  - wird ausgeführt bei erreichen des Point Cuts
- Introduction
  - fügen bestehenden Klassen Attribute und Methoden zu
  - Möglichkeit zur Beeinflussung des Typs und der Vererbungshierarchie einzelner Klassen

Begriffe (5)

- Separation of Concerns
  - Auftrennung des Programms in einzelne Module, jedes Modul realisiert einen Concern
- Interception
  - Unterbrechung des Programmflusses durch AOP Framework
    - Vorher-Interception
    - Nacher-Interception
    - Anstatt-Interception

Ziele

- Ergänzung zu OOP
- Auslagerung von aspektbezogenem Code aus der Objektfunktionalität in einzelnen Modulen:
  - Saubere Modularisierung
  - Wartbarkeit
  - Wiederverwendbarkeit
  - Rückverfolgbarkeit
  - Übersichtlichkeit

Funktionsweise (1)

Objektorientiert:

```
Class History-Buffer extends B-Buffer {
    boolean GOT = false ;
     public synchronized Object get () {
        while ( EMPTY ) {
            try ( wait() ) catch
             (InterruptedException e) {}
             }
        count -- ;
        GOT = true ;
        FULL = false ;
        if (count == 0 ) EMPTY = true ;
        return data (count+1) ;
        }
```

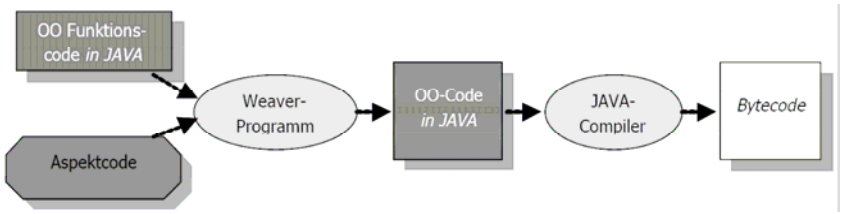
Aspektorientiert:

```
Class B-Buffer {
  int static MAX ;
  int count = 0;
  private Vector data ;

  public Object get () {
    count -- ;
    return data (count+1) ;
  }
```

```
condition EMPTY = true
  FULL = false

guard put requires !FULL
  on exit { EMPTY = false
            if(count == MAX)
              FULL = true }
```

Funktionsweise (2) – statisches Weaving

- Source-to-Source Transformation
  - ❑Verwebung der Aspect-Definitionen und dem Quelltext zu einem verwobenen Quelltext

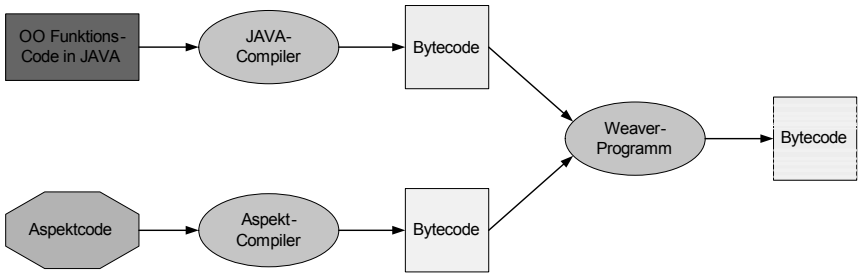


Source-to-Source Transformation:

Hierbei wird der Quelltext anhand der Aspekt-Definitionen zu einem neuen verwobenen Quelltext transformiert. Dieser transformierte Code kann nun wie normale Klassen von einem normalen Compiler zum ausführbaren Programm compiliert werden.

Funktionsweise (3) – statisches Weaving

- Weaving zur Compile-Zeit
  - AOP-Compiler erzeugt Bytecode der Aspekte, der anschließend in den Bytecode der Komponenten eingewoben wird



Weaving zur Compile-Zeit:

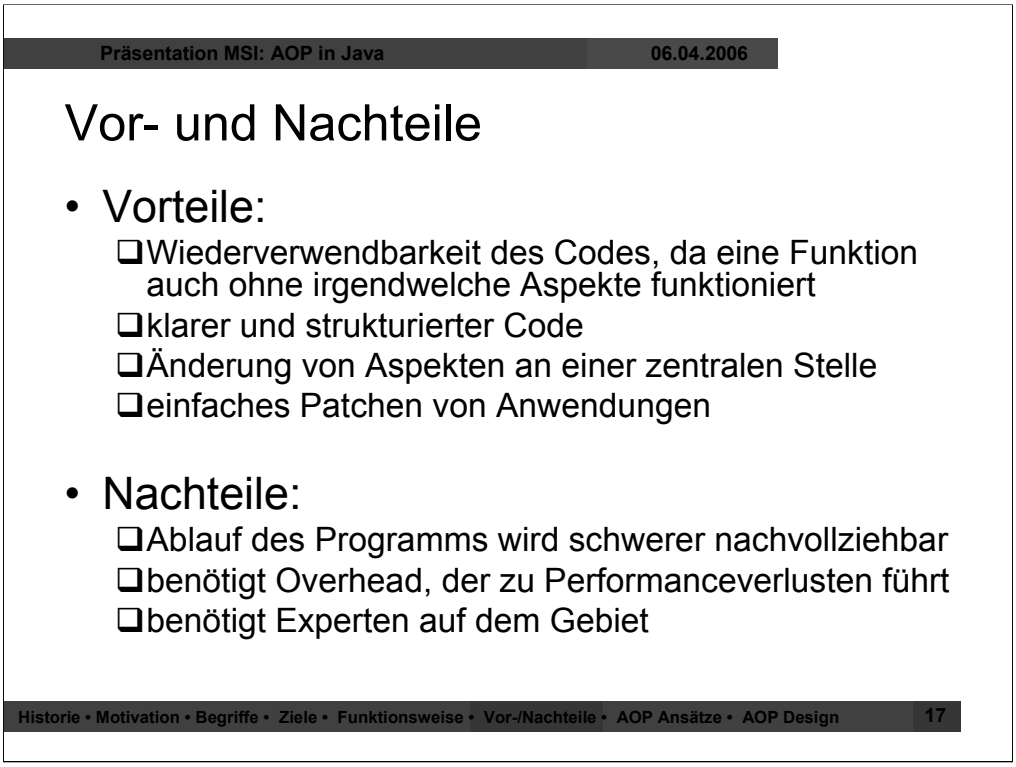
Hierfür wird ein spezieller AOP-Compiler benötigt. Hierbei werden die einzelnen Komponenten vorher von einem normalen Compiler in Bytecode übersetzt. Beim Weaving wird dann aus dem Bytecode ein neuer verwobener Bytecode erstellt.

Funktionsweise (4) – dynamisches Weaving

- Weaving zur Laufzeit
  - ❑spezielle Laufzeitumgebung nötig
    → Virtual Machine
  - ❑Vorteil: Saubere Trennung der Aspekte sogar im Bytecode
  - ❑Nachteil: spezielle Laufzeitumgebung benötigt zum Ausführen

Dynamisches Weaving:

Hierbei erfolgt das Weaving bei Ausführung des Programms. Aus diesem Grund ist eine spezielle Laufzeitumgebung erforderlich, die den Weaving Prozess durchführt.



Beispiel für Patchen von Anwendungen:

Wir haben eine Anwendung für ein Handy geschrieben, welche sich über GPRS updaten kann. Nun möchte man einen Fehler in der Hauptkomponente beseitigen. Man kann also das komplette Modul übertragen, damit wird allerdings auch ein großer Anteil an Daten übertragen, die sich nicht geändert haben.

Dies ist in einem Umfeld wie dem Handymarkt jedoch kein erstrebenswertes Verhalten. Der Benutzer muss länger warten und hat mehr Kosten. Es ist zu erwarten, dass dadurch die Zufriedenheit leiden wird. Daher könnte man die Anwendung patchen.

AOP Ansätze für Java

- AspectJ
- AspectWerkz
- JBoss AOP Framework
- Spring AOP Framework
- Hyper/J
- CME
- dynaop AOP Framework

•Mit der Zeit haben sich verschiedene Ansätze für das Aspekt Orientierte Programmieren entwickelt. Jedoch State-of-the-Art ist derzeit AspectJ.

AspectJ

- Ist eine Spracherweiterung zu Java
- Unterstützt „Weaving“
  - während der Kompilierung
  - nach der Kompilierung auf Bytecode-Ebene
  - während des Class-Loadings

AspectJ – Join Points

- Sind Ereignisse zur Programmlaufzeit
  - ❑ Der Zugriff auf ein Feld
  - ❑ Das Ausführen einer Methode
  - ❑ Die Rückkehr aus einer Methode
  - ❑ Das Auswerten eines Ausdrucks
  - ❑ Die Ausführung einer Ausnahmebehandlung

•Weitere Beispiele für Join Points wären das Ausführen oder Aufrufen einer Methode, die Rückkehr aus einer Methode oder das Auswerten eines Ausdruckes

Standard Java Programm

```
public class HelloWorld {

    public static void main(String[] args) {
        HelloWorld.sayHello();
    }

    public static void sayHello(){
        System.out.println("Hello World");
    }
}
```


•Das Hello World Program in Java wird benutzt um durch die folgenden Beispiele für die verschiedenen AOP Systeme zu führen.

AspectJ – Pointcuts

- Eine Art Filter für Join Points
- Drei Arten des Filterns:
  - Nach der Art des Join Points (Methoden-Aufruf/Ausführung)
  - Nach dem Umfeld (Liegt der aufrufende Kode in einem best. Packet)
  - Nach dem Kontext (Ist das derzeitig ausführende Objekt von einem best. Typ)
- Beispiele:

```
pointcut sayHello():
        execution(void sayHello());
pointcut inModelPack() :
        within(myModel);
pointcut exceptionHandler() :
        handler(IOException);
```

Name

Pointcut Ausdruck

Pointcut Designator

- Syntax:

[Sichtbarkeits-Modifizierer] ***pointcut*** *name(ParameterListe) :*
Pointcut Asudruck

•Pointcuts dienen als eine Art Filter für Join Points. Sie entscheiden ob ein Join Point ausgeführt wird oder nicht. Dafür werde Mechanismus aus Wildcards und Definitionen verwendet.

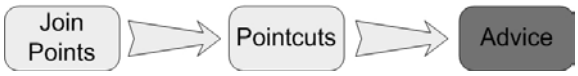
Zum Beispiel die Pointcut Definition „execution(* set*(..));“ besagt, dass zugehörige Join Points nur ausgeführt werden, wenn eine Methode einen beliebigen Rückgabewert besitzt, aber mit „set“ anfängt und eine beliebige Paramterliste hat.

Achtung! Der Wildcard „..“ besitzt eine unterschiedliche Semantik in einer Typen-Definition als in eine Parameter Listen-Definition

•Ein Pointcut Ausdruck kann wiederum zusammengesetzt sein aus weiteren Pointcut ausdrücken, die über logische Operatoren wie „&&“ , „||“ und „!“ verknüpft werden können. Ebenso kann in einem solchen Ausdruck wiederum auf andere Pointcuts, via deren Namen, verwiesen werden.

•Für weitere Informationen zu Pointcut Designatoren und Wildcards sei an dieser Stelle auf Colyer, A. et alii verwiesen

AspectJ – Advice



- Legen fest was getan wird, wenn ein Join Point aufgerufen wird
- Sind in der Lage zu spezifizieren ob die Aufgabe vor, nach oder um den Join Point herum ausgeführt wird
  - before(): Kann Kontextinformation nur lesen
  - after(): Kann Kontextinformation nur lesen
  - around(): Kann Kontextinformation auch ändern
- Beispiel:

  Pointcut Name

  before() : sayHello(){ System.out.println("Before Execution"); }

  before() : execution(void sayHello()){ System.out.println("Before Execution"); }

  Anonymer Pointcut
- Syntax:

  before (Parameter Liste) : PointcutAusdruck{…};

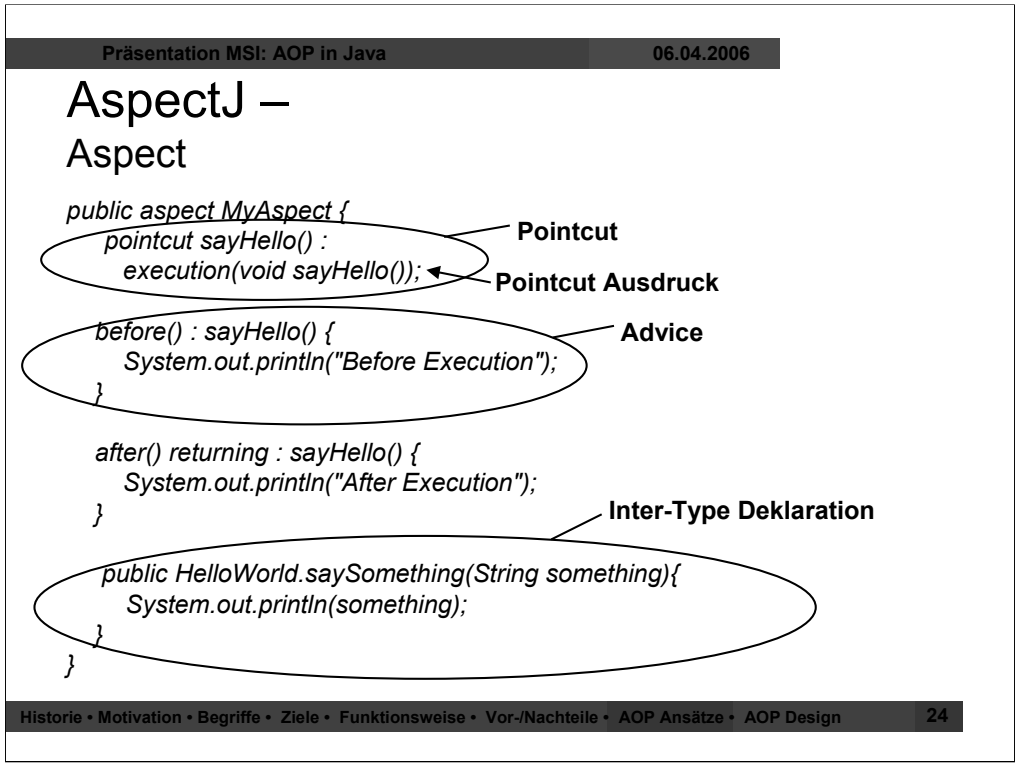
  after (Parameter Liste) **returning** : PointcutAusdruck{…};

  after (Parameter Liste) **throwing** : PointcutAusdruck{…};

  Typ **around** (Parameter Liste) : PointcutAusdruck{…};

•Kontextinformationen sind z.B. Argumente oder Rückgabewerte

•Pointcut Ausdrücke können auch anonym, d.h. anstelle von PointcutExpr kann ein konkreter Pointcut Ausdruck stehen oder auf ein bereits definierter Asudruck, auf den via seinen Namen Bezug genommen wird

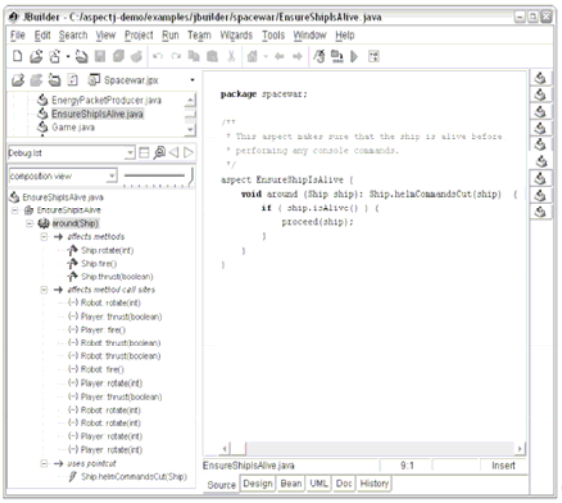


•Inter-Type Deklarationen ermöglichen es Klassen durch einen Aspekt, Felder und Methoden hinzuzufügen. Zum Beispiel könnte man damit den Code der für das Visitor-Pattern benötigt wird aus der eigentlichen Klasse „herausziehen“ und in einem Aspekt formulieren (siehe Demo)

DEMO

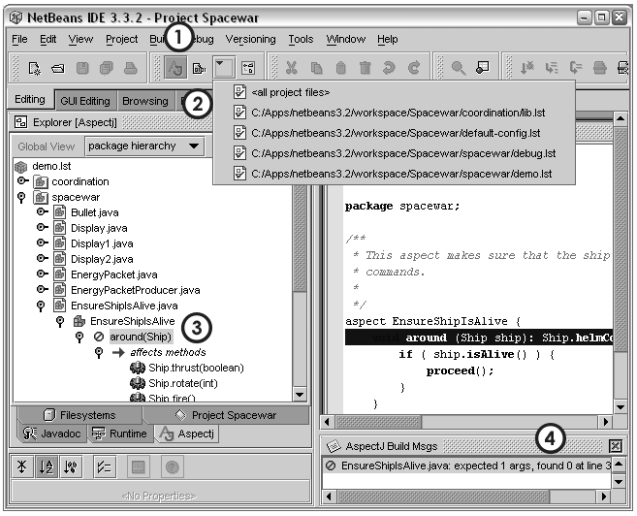
AspectJ Development Tools (AJDT)

Toolunterstützung – JBuilder



Quelle: Kersten, M.

Toolunterstützung – NetBeans (AJDE)



Quelle: AspectJ4Netbeans

AspectWerkz

- Implementiert in „Pure Java“
- Unterstützt „Weaving“ wie AspectJ
- Steht unter der LGPL-style Lizenz (Freie Nutzung in Kommerzieller SW und Open Source SW)
- Steht in Kooperation mit AspectJ ➔ AspectJ 5

•AspectJ erweitert die JLS (Java Language Specification), wohingegen AspectWorkz auf Annotationen und XML basiert. Mit AspectJ5 sollen sich diese Möglichkeiten vereinen. Desweiteren legte AspectJ den Schwerpunkt für das Weaving während der Kompilierung, AspectWerkz hingegen arbeitet hauptsächlich mit Weaving während dem Laden von Klassen.

•AspectJ5 kombiniert die bisherigen AspectJ Eigenschaften mit denen von AspektWerz, wie z.B. die Formulierung von Aspekten mittels Java Annotationen

AspektWerkz –
Aspekte, Pointcuts & Advice formulieren

```
Package MyPack;

import org.codehaus.aspectwerkz.joinpoint.JoinPoint;

public class MyAspect {

    /**
     * @Before execution(* MyPack.HelloWorld.sayHello(..))
     */
    public void beforeSayHello(JoinPoint joinPoint) {
        System.out.println("before saying hello...");
    }

    /**
     * @After execution(* MyPack.HelloWorld.sayHello(..))
     */
    public void afterSayHello (JoinPoint joinPoint) {
        System.out.println("after saying hello...");
    }
}
```

```
<aspectwerkz>
  <system id="AspectWerkzExample">
    <package name="MyPack">
      <aspect class="MyPack.MyAspect">
        <pointcut
          name="sayHelloMethod"
          expression=
            "execution(
                * MyPack.HelloWorld.sayHello(..)
              )"/>
        <advice
          name="beforeSayHello"
          type="before"
          bind-to=" sayHelloMethod "/>
        <advice
          name="afterSayHello"
          type="after"
          bind-to=" sayHelloMethod "/>
      </aspect>
    </package>
  </system>
</aspectwerkz>
```


•Um mit AspektWerkz einen Aspekt zu erstellen wird eine Standard Java Klasse erstellt, dessen Methoden Advice entspricht. Um Advice an Pointcuts zu binden kann man entweder Java Annotationen oder eine XML Datei benutzen. Ein XML Stub, der eine Klasse als Aspekt identifiziert, wird jedoch in jedem Fall benötigt.

JBoss

- AOP Framework
- Implementiert in „Pure Java“
- Besitzt vordefinierte Aspekte wie, Caching, asynchrone Kommunikation, Transaktionen, …

•„Pure Java“ bedeutet, dass keine zusätzlichen Tools benötigt werden. Alles ist mittels des Standard Java Compilers und der Standard JVM lauffähig

JBoss – Aspekte, Advice & Pointcuts formulieren

```
public class MyInterceptor implements Interceptor {

  public String getName() { return "MyInterceptor"; }

     @InterceptorDef
     @Bind (pointcut = "execution(* HelloWorld->sayHello(..))")
     public InvocationResponse invoke(Invocation pInvocation)
          throws Throwable {

          System.out.println(" before saying hello...");

          InvocationResponse retVal =
               pInvocation.invokeNext();

          System.out.println("after saying hello...");

          return retVal;
     }
}
```

```
<aop>
...
  <bind pointcut=
     "execution(* HelloWorld-
      >sayHello(..))">

       <interceptor class="MyInterceptor"/>

  </bind>
...
</aop>
```


•Der größte Unterschied zwischen der Verwendung von Annotationen und einer XML Datei zum binden von Advice an Pointcuts ist, dass man keine extra jboss-aop.xml Datei benötogt. Dafür muss man aber dem aopc Compiler explizit mitteilen wo er die annotierten Dateien findet.

•Beim „Matchen“ eines Execution-Pointcuts wird die Methode invoke aufgerufen, deren Parameter den eigentlichen Methoden-Aufruf enthält, sowie einige Umgebungsinformationen. Dadurch kann vor der Ausführung der eigentlichen Methode eine Ausgabe oder sonstiges stattfinden. Mit pInvocation.invokeNext() wird der ursprüngliche Programmfluss vorgesetzt. Danach kann optional wieder eine Ausgabe oder sonstiges stattfinden.

Spring Framework

- Spring ist mehr als nur ein AOP Framework
- Implementiert in „Pure Java“, Proxy-Basierter Ansatz
- Spezialisiert auf J2EE Applikationen
- Unterstütz kein „Field Interception“
- Weaving nur zur Laufzeit

•Spring ist nicht nur ein AOP Framework, wobei jedoch einer der Schlüsselpunkte das AOP Konzept ist

•„Field Interception“ bedeutet Zugriffe auf Felder mittels Pointcuts abzufragen. Es wir von Spring nicht unterstützt, weil es deren Meinung nach das OO Kapselungskonzept verletzt

•Spring benutzt z.B. selbst AOP um Deklaratives Transaktionsmanagement ohne EJBs zu ermöglichen

Spring Framework – Beispiel (1)

```
…
import org.springframework.aop.MethodBeforeAdvice;

public class MyAspect implements MethodBeforeAdvice  {

   public MyAspect () {
   }

   public void before(Method method, Object[] args, Object target) throws
         Throwable {

      if (method.getName().equalsIgnoreCase(“sayHello")) {
            System.println.out(„Before saying Hello“);
      }
   }
}
```


•Das Spring AOP Framework definiert Schnittstellen, die verwendet werden um Aspekte zu definieren, wie z.B. der „MethodBeforeAdvice“

Spring Framework – Beispiel (2)

```
public class HelloWorldFactory {

  public IHelloWorld getHelloWorld() {
    IHelloWorld hw = new HelloWorld();
    ProxyFactory pf = new ProxyFactory();
    pf.setTarget(hw);
    pf.setInterfaces(new Class[]{IHelloWorld.class});
    pf.addAdvice(new MyAdvice());
    return (IHelloWorld)pf.getProxy();
  }

}
```


•Um den Aspekt mittels Spring AOP Framework einzuweben muss man so genannte Proxys erstellen, welche das ursprüngliche Object mit dem dem Aspekt verweben

Spring Framework – Beispiel (3)

```
public interface IHelloWorld  {
  public void sayHello();
}

public class HelloWorld implements IHelloWorld{

    public static void main(String[] args) {

        IHelloWorld hw = HelloWorldFactory.getHelloWorld()
        hw.sayHello();
     }

   public void sayHello(){
       System.out.println(„Hello World“);
   }
}
```


•Um Proxys einsetzen zu können muss eine Klasse ein Interface bereitstellen

Benchmarks

AWBench (ns/invocation)	AspectWerkz	AspectJ	JBoss	Spring
before, args() target()	10 (1x)	10 (1x)	220 (22x)	355 (35.5x)
around x 2, args() target()	80 (1x)	50 (0.6x)	290 (3.6x)	436 (5.4x)

Quelle: http://docs.codehaus.org/display/AW/AOP+Benchmark

•Die Tabelle vergleicht die hier vorgestellten AOP Ansätze bezüglich des Laufzeitverhaltens. Hier aufgezeigt sind jedoch nur die Ausführung eines „before“ Advice zuzüglich der Abfrage des Targets und der Argumente und die Ausführung eines „around“ Advice zuzüglich der Abfrage des Targets und der Argumente. Die Zeiten wurden in Nano Sekunden gemessen. Die Werte in den Klammern geben relative Werte, bezüglich der Zeiten gemessen für AspectWerkz, an.

Objekt-Orientiertes Design –
Finden von OO-Elementen

- Nomen geben Hinweise auf Klassen
- Verben geben Hinweise auf Operationen
- Bsp:

  Einem Kunden muss es möglich sein, auf sein Konto Geld einzuzahlen, vom Konto Geld abzuheben und den Kontostand einzusehen. Bevor ein Zugriff auf ein Konto stattfindet, muss eine Authentifizierung statt finden.

Aspekt-Orientiertes Design –
Finden von AO-Elementen in der Spezifikation

- Für gewöhnlich Erfahrungswerte
- Adverbien & Adjektive geben Hinweise auf Aspekte
- Zeitliche Angaben geben Hinweise auf Advice
- Bsp:

  Einem Kunden muss es möglich sein, auf sein Konto Geld einzuzahlen, vom Konto Geld abzuheben und den Kontostand einzusehen. Bevor ein Zugriff auf ein Konto stattfindet, muss eine Authentifizierung statt finden. Greift der Benutzer über das Internetportal zu, muss dafür eine gesicherte Verbindung zum Online-Banking-Server aufgebaut werden.

•Wörter wie „bevor, im Voraus, zu erst“ geben z.B. einen Hinweis auf ein „before“-advice

•Wörter wie „ danach, später, als nächstes, folglich“ geben einen Hinweis auf einen „after“-advice

Aspekt-Orientiertes Design –
Finden von AO-Elementen im Quellkode

- Produziert man dieselben Kodezeilen (auch wenn nur eine) immer wieder an verschiedenen Stellen? (Scattering)
- Hat man Logik an verschiedenen Stellen im Quellkode die dennoch zusammen arbeitet um ein gemeinsames Ziel zu erreichen?

•Unter „verschiedenen Stellen“ kann man Stellen verteilt über das gesamte System aber auch verteilt in einer einzelnen Klasse verstehen. Ein Beispiel dafür ist das Observer Muster, bei dem für jede Änderung der Modeldaten ein notifyListeners() aufgerufen wird.

Aspekt-Orientiertes Design – Der Versuchung widerstehen

- Es ist nicht immer sinnvoll gleich alles in Aspekten zu formulieren!
- Was macht einen guten Aspekt aus?
  - Würde ein Aspekt:
    - die Kopplung zwischen Typen im Design reduzieren?
    - den Wartungsaufwand veringern?
    - in manchen Versionen gar nicht gebraucht werden?
    - das Verständnis des Programms erleichtern oder eher erschweren?

Referenzen (1)

- Colyer, A. et alii:
  Eclipse AspectJ – Aspect-Oriented Programming with AspectJ and the Eclipse AspectJ Development Tools, Addison-Wesley, Maryland 2004
- Buss, M.:
  Aspektorientierte Programmierung – „Konzepte und Werkzeuge für die Softwareentwicklung mit Java“, Universität Leipzig 2005
- AspectJ4Netbeans:
  http://aspectj4netbean.sourceforge.net/doc/ajdeForNetbeans.html
- Kersten, M.:
  AO Tools: State of the (AspectJ™) Art and Open Problems, 2002 Palo Alto

Referenzen (2)

- Scharf, T.: Aspektorientierte Programmierung mit AspectJ, JavaSPEKTRUM 4/2003
- Uni Paderborn: Abschlussbericht Projektgruppe "Aspektorientierte Entwicklung nebenläufiger Systeme„, 2000, http://wwwcs.uni-paderborn.de/cs/ag-engels/Papers/1999/AbschlussberichtPGAOP99.pdf
- Prilmeier, F.: Diplomarbeit „AOP und Evolution von Software-Systemen“, 2004, TU München, http://home.in.tum.de/~prilmeie/da/da-aop.pdf
- Wikipedia (Hrsg.): Aspektorientierte Programmierung, http://de.wikipedia.org/wiki/Aspektorientierte_Programmierung (5.4.2006)

Referenzen (3)

- AOP Alliance
  http://aopalliance.sourceforge.net/
- AspectWerkz :
  http://aspectwerkz.codehaus.org
- Hyper/J :
  http://www.research.ibm.com/hyperspace/
- CME :
  http://www.eclipse.org/cme
- JBoss :
  http://labs.jboss.com/portal/jbossaop/index.html
- Spring :
  http://www.springframework.org
- dynaop :
  http://dynaop.dev.java.net/