ASPECT ORIENTED PROGRAMMING (AOP)

Veerle Ongenae, Wijnand Schepens

Overzicht

- Inleiding AOP
- Crosscutting concerns zonder AOP
- AOP
- Implementaties
- Voor- en nadelen AOP
- Informatiebronnen

Complexe software-systemen

- Software-systemen worden complexer
  - Business functionality
  - Toegang tot data
  - Presentatielogica
  - Beveiliging
  - Logging
  - Performantie monitoren
  - Transacties
  - Caching
  - Resource pooling

Core concerns

Crosscutting concerns

Complexe software-systemen

- Oplossing?
  - Modularisering
  - Opdelen in stukken volgens “belang” (concern)
    - Separation of concerns (scheiding van belangen)
- Core concerns (kerntaken)
  - OOP
- Crosscutting concerns (verweven taken)
  - Lopen door verschillende modules heen
  - AOP

AOP

- Modularisatie
  - Crosscutting concerns
- Aspect
  - Bevat één verweven taak
- Aspect weaver
  - Verweeft
    - Aspecten (crosscutting concerns)
    - Kerntaken (core concerns)

Overzicht

- Inleiding AOP
- Crosscutting concerns zonder AOP
- AOP
- Implementaties
- Voor- en nadelen AOP
- Informatiebronnen

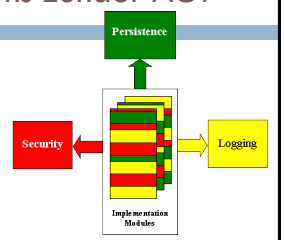
Crosscutting concerns - voorbeelden

7

- Elke opdracht van de “AccountManager” interface moet uitgevoerd worden in een transactie
- De resultaten van “dure” methodes moeten een bepaalde tijd bewaard (gecached) worden
- Als een veld van een domeinobject wordt veranderd, dan moet het gemarkeerd worden.
- Firma doet gedurende één maand een actie en wil het effect op de gebruikersactiviteit analyseren.

Crosscutting concerns zonder AOP

8



- Module
  - Kerntaken
  - Ondersteunende taken
- Bron: http://www.javaworld.com/javaworld/jw-01-2002/jw-0118-aspect.html?page=2

Voorbeeld

Beveiliging
Concurrency
Transactie
Logging

9

```
public class SomeBusinessClass extends OtherBusinessClass {
    // Core data members
    // Other data members: Log stream, data-consistency flag //
    // Override methods in the base class

    // More operations similar to above
    public void save (PersitanceStorage ps) {
        // ...
    }
    public void load (PersitanceStorage ps) {
         // ...
    }
}
```


Voorbeeld

Beveiliging
Concurrency
Transactie
Logging

10

```
public void performSomeOperation ( OperationInformation info) {
        // Ensure authentication
        // Lock the object to ensure data-consistency in case other
        // Start transaction
        // Log the start of operation
        // === Perform the core operation ===
        // Log the completion of operation
        // Commit or rollback transaction
        // Unlock the object
    }
```


Tweede voorbeeld (a)

11

```
public class AccountManagerImpl implements AccountManager {
   private MailSender mailSender;
   private SimpleMailMessage bericht;
   private AccountDao accountDao;

   public void setMailSender (…) { … }
   public void setMessage (…) { … }
   public void setAccountDao (…) { … }

   private void sendMail (Exception ex) { … }

   public Account getAccount (… ) … {…}
   public void createAccount (… ) … {…}
```

Tweede voorbeeld (b)

Beveiliging
Concurrency
Transactie
Logging

12

```
public Account getAccount (String id)
        throws AccountNotFoundException, DataAccessException
{
   try {
        return accountDao.findAccount(id);
   } catch (AccountNotFoundException ex) {
        sendMail(ex);
        throw ex;
   } catch (DataAccessException ex) {
        sendMail(ex);
        throw ex;
   }
}
```

Tweede voorbeeld (c)

Beveiliging
Concurrency
Transactie
Logging

13

```
public void createAccount (Account account)
   throws DataAccessException, IOException, InvalidAccountException
{
   try {
        if (isInvalid(account))
            throw new InvalidAccountException(account);
        accountDao.saveAccount(account);
   } catch (IOException ex) {
        sendMail(ex);
        throw ex;
   } catch (DataAccessException ex) {
        sendMail(ex);
        throw ex;
   }
```


Crosscutting concerns zonder AOP

14

- OOP: geen goeie oplossing
- Code duplicatie
  - "boilerplate code"
  - Nieuwe methode → code copiëren
  - Vergeten?
- Aanpassen procedure → nachtmerrie
- Voorbeeld 2
  - Wat indien er geen mail meer gestuurd moet worden bij fout, maar iets anders?
  - Andere logging toevoegen?

Crosscutting concerns zonder AOP

15

- Zondigt tegen OO-ontwerpprincipes
  - Single Responsibility Principle (Eén klasse, één verantwoordelijkheid)
  - Open/gesloten principe (Een klasse moeten open zijn voor uitbreiding, maar gesloten voor verandering)
- Een "crosscutting concern" is verspreid over verschillende modules en wordt er gedupliceerd
- Gevolgen
  - Weinig overzichtelijk, lagere productiviteit, minder hergebruik van code, slechte codekwaliteit, moeilijk aanpasbaar in de toekomst

Crosscutting concerns zonder AOP

16

- Bestaande oplossing
  - Frameworks zoals EJB (Entreprise Java Beans)
  - Nadelen
    - Voldoen aan eisen framework (↔ POJO's)
    - Moeilijk uitbreidbaar

Overzicht

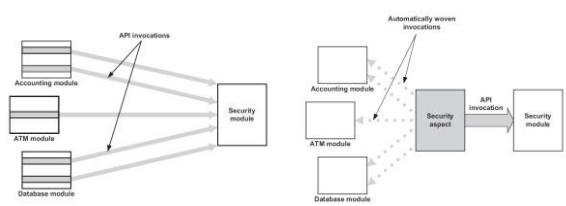
17

- Inleiding AOP
- Crosscutting concerns zonder AOP
- AOP
- Implementaties
- Voor- en nadelen AOP
- Informatiebronnen

Beveiliging met en zonder AOP

18

- Bron: http://www.manning.com/laddad2/Samplechapter1.pdf



AOP

19

- Specificatie
- Implementatie
- Fundamentele concepten

AOP - specificatie

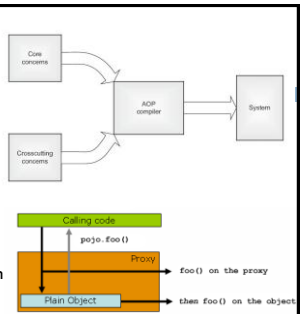
20

- Specificatie
  - Belangen implementeren
  - Weefregels
    - Hoe "crosscutting concern" verweven met de rest van de applicatie?
    - Algemeen ↔ specifiek

AOP implementatie

21

- Implementatie
  - Het eigenlijke verweven
  - Uitvoerbare code genereren
  - Mogelijk manieren
    - Broncode verweven
    - Gecompileerde code verweven
    - Automatisch gegenereerde proxy's



Bronnen:
http://www.manning.com/laddad2/Samplechapter1.pdf
http://static.springsource.org/spring/docs/2.5.x/reference/aop.html#aop-understanding-aop-proxies

AOP – Fundamentel concepten

22

- Join point model
- Advice
- Static Crosscuttings
- Aspect

AOP – Join point model

23

- Join points
  - Identificeerbare punten tijdens de uitvoering van het systeem
    - Oproepen van methode / constructor
    - Uitvoeren van methode / constructor
    - Aanmaken van objecten
    - Wijzigen van attributen
    - Gooien van excepties
    - …

AOP – Join point model

24

- Pointcut
  - Constructie om join points te selecteren
    - Selecteren verzameling join points
    - Context voor deze punten verzamelen (bv. argumenten methode)
  - Vergelijk met reguliere expressie
  - Vergelijk met CSS selector

AOP – Advice

25

- Constructie om het gedrag van het programma te veranderen
  - Extra gedrag toevoegen aan "join points" of gedrag veranderen
- Before advice
  - Extra gedrag voor het "join point"
- After advice
  - Extra gedrag na het "join point"
- Around advice
  - Extra gedrag rond "join point"
  - 0 of meer keer uitvoeren "join point"

AOP – Static crosscuttings

26

- Constructies om de statische structuur van het systeem te veranderen
- Inner-type declaration
  - Velden toevoegen aan bestaande klassen
  - Methodes toevoegen aan bestaande klassen
  - Bestaande klassen laten overerven/implementeren
- Weave-time declaration
  - Voorwaarden nakijken
    - Vb. bestaan van bepaalde "join points"

AOP - Aspect

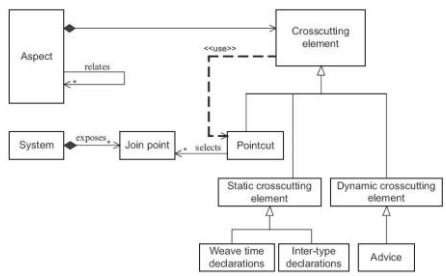
27

- Aspect = module met alle "crosscutting" constructies
  - Pointcuts
  - Advice
  - Static crosscuttings
- Aspect is een speciaal soort klasse
  - kan gewone velden en methodes bevatten
  - kan overerven en/of implementeren
  - kan abstract zijn
- Aspect kan verbonden zijn met andere aspecten

AOP - Model

28

- Bron: http://www.manning.com/laddad2/Samplechapter1.pdf



Overzicht

29

- Inleiding AOP
- Crosscutting concerns zonder AOP
- AOP
- Implementaties
- Voor- en nadelen AOP
- Informatiebronnen

Implementaties AOP

30

- AspectJ
- Spring AOP
- …

AspectJ

31

- Voorbeelden
  - Eigenlijke applicatie
    - Project AOPVoorbeeld
    - aopvoorbeeld.HelloWorld
    - aopvoorbeeld.Main
  - Aspecten
    - Project AOPVoorbeeldAspect
    - aopVoorbeeld.HelloWorldAspect.aj
  - Aspecten met annotaties
    - Project AOPVoorbeeldAspectAnnotatie
    - aopVoorbeeldAnnotatie.HelloWorldAspect.java

AOP - Aspect

32

- Module met alle "crosscutting" constructies
  - Pointcuts
  - Advice
  - Static crosscuttings
- Kunnen verbonden zijn met andere aspecten

AspectJ – Voorbeeld Aspect

33

```
public aspect MyAspect       // MyAspect.aj
{
    // pointcut:
    pointcut callSayMessage()
        : call(public static void HelloWorld.say(String));

    // advice:
    after() : callSayMessage() {
        System.out.println("Thank you!");
    }
}
```


AspectJ - Pointcuts

34

- Join points selecteren (omschrijving)
  - call, execution, handler, this, target, within, cflow
- Omschrijvingen combineren
  - &&, ||, !
- Jokertekens
  - * : return waarde, naam methode, naam klasse
  - .. : argumenten

AspectJ – Voorbeelden pointcuts

35

```
pointcut callSayMessage1()
   : call(public static void HelloWorld.say(String));

pointcut callSayMessage2()
   : call(public static void HelloWorld.sayToPerson(String,String));

pointcut callSayMessage3()
   : call(public static void HelloWorld.say(String))
  || call(public static void HelloWorld.sayToPerson(String,String));

pointcut callSayMessage4()
   : call(public static void HelloWorld.say*(..));
```


AspectJ – Pointcuts – Voorbeeld 2

36

```
class Point {
    private int x, y;
    Point(int x, int y) { this.x = x; this.y = y; }
    void setX(int x) { this.x = x; }
    void setY(int y) { this.y = y; }
    int getX() { return x; }
    int getY() { return y; }
}

pointcut setter():
   target(Point) && (call(void setX(int)) || call(void setY(int)));

pointcut ioHandler(): within(MyClass) && handler(IOException);
```

AspectJ – Pointcuts met parameters

37

- Zonder parameter:

```
pointcut setter():
   target(Point) && (call(void setX(int)) || call(void setY(int)));
```

- Met parameter:

```
pointcut setter(Point p):
   target(p) && (call(void setX(int)) || call(void setY(int)));
```

- Parameter kan gebruikt worden in advice

AOP – Advice

38

- Constructie om het gedrag van het programma te veranderen
- before:
  - Extra gedrag voor het "join point"
- after:
  - Extra gedrag na het "join point"
- around:
  - *Vervangt* oproepen join point
  - 0 of meer keer uitvoeren "join point" (proceed())
  - Extra gedrag rond "join point" (voor en/of na proceed())

AspectJ – Voorbeelden advice

39

```
before() : callSayMessage() {
   System.out.println("Good day!");
}

after() : callSayMessage() {
   System.out.println("Thank you!");
}
```


AspectJ – Advice – Voorbeeld 2

40

```
pointcut setter(Point p1, int newval):
  target(p1) && args(newval)
  && (call(void setX(int) || call(void setY(int)));

before(Point p1, int newval): setter(p1, newval) {
  System.out.println("About to set something in " + p1 +
          " to the new value " + newval);
}
```


AspectJ – Advice – Voorbeeld 3

41

```
public class Test {
    public static void main(String[] args) { foo(); }
    static void foo() { goo(); }
    static void goo() { System.out.println("hi"); }
}
aspect A  {
    pointcut fooPC(): execution(void Test.foo());
    pointcut gooPC(): execution(void Test.goo());
    pointcut printPC(): call(void java.io.PrintStream.println(String));

    before(): cflow(fooPC()) && cflow(gooPC()) && printPC() && !within(A) {
        System.out.println("should occur");
    }
}
```


Compileren en uitvoeren

42

- HelloWorld
  - HelloWorldCompileren.bat
  - HelloWorldTesten.bat
- HelloWorldAnnotation
  - HelloWorldCompileren.bat
  - HelloWorldTesten.bat

Compileren en uitvoeren

43

- Compileren
  - ajc
    - .java en .aj → class
  - javac en ajc
    - javac: .java → jar
    - ajc: jar's verweven
  - Exclipse IDE
- Uitvoeren
  - java

Alternatieve notatie met annotaties

44

- "pure" Java (*.java)
- import org.aspectj.lang.annotation.*;

```
@Aspect
public class HelloWorldAspect {
  @Pointcut("call(public static void HelloWorld.say (String))")
  void callSayMessage() {}

  @After("callSayMessage()")
  public void thankYou() {
        System.out.println("Thank you!");
  }
}
```


Overzicht

45

- Inleiding AOP
- Crosscutting concerns zonder AOP
- AOP
- Implementaties
- Voor- en nadelen AOP
- Informatiebronnen

Voor- en nadelen AOP

46

- Nadelen
  - Meer opgeleide werknemers
  - Complexere programma-flow
    - Hoger abstractie niveau
    - Code beschrijft niet volledige programma-uitvoering
- Voordelen
  - Eenvoudiger ontwerp
  - Properder implementatie
  - Beter hergebruik van code

Overzicht

47

- Inleiding AOP
- Crosscutting concerns zonder AOP
- AOP
- Implementaties
- Voor- en nadelen AOP
- Informatiebronnen

Infobronnen

48

- "AspectJ in Action, Second Edition", Ramnivas Laddad, Manning, 2009
  - Chapter 1. Discovering AOP (http://www.manning.com/laddad2/Samplechapter1.pdf)
- The AspectJTM Programming Guide (http://www.eclipse.org/aspectj/doc/released/progguide/index.html)
  - Chapter 2. The AspectJ Language (http://www.eclipse.org/aspectj/doc/released/progguide/language.html)
- The AspectJTM 5 Development Kit Developer's Notebook (http://www.eclipse.org/aspectj/doc/released/adk15notebook/index.html)
  - Chapter 9. An Annotation Based Development Style (http://www.eclipse.org/aspectj/doc/released/adk15notebook/ataspectj.html)

Infobronnen

- "I want my AOP!, Part 1: Separate software concerns with aspect-oriented programming", Ramnivas Laddad, JavaWorld.com, 01/18/02 (http://www.javaworld.com/javaworld/jw-01-2002/jw-0118-aspect.html)
- I want my AOP!, Part 2: Learn AspectJ to better understand aspect-oriented programming, By Ramnivas Laddad, JavaWorld.com, 03/01/02 (http://www.javaworld.com/javaworld/jw-03-2002/jw-0301-aspect2.html)

Infobronnen – AOP Spring

- "Professional Java Development with the Spring Framework", Rod Johnson, Juergen Hoeller, Alef Arendsen, Thomas Risberg, Colin Samaleanu, Wrox, Wiley Publishing, Inc., 2005
- The Spring Framework - Reference Documentation (http://static.springsource.org/spring/docs/2.5.x/reference/)
  - Chapter 6. Aspect Oriented Programming with Spring
- "An introduction to AOP", Sing Li (http://www.ibm.com/developerworks/java/tutorials/j-aopintro/section3.html)