

Aspect Oriented Programming met AspectJ

Ruud Steeghs

De Reehorst, Ede
12 oktober 2004

- Object Oriented Programming

- Concept Aspect Oriented Programming

- AspectJ: AOP Implementatie in Java

- Voorbeelden

OOP

- modellering van software / functionaliteit in kleine, herbruikbare objecten
- Gelijkenis met entiteiten in de wereld om ons heen

maar:

sommige onderwerpen zijn niet te modelleren in objecten

want:

ze hebben betrekking op meerder objecten, een hele module of zelfs het hele systeem

(System Wide Concern)

Voorbeelden:

- logging
- autorisatie
- authenticatie
- transactie management
- afdwingen ontwikkelstandaards / afspraken
- ondersteunen testomgeving

OOP:

Schiet te kort in het modelleren van System

Wide Concerns

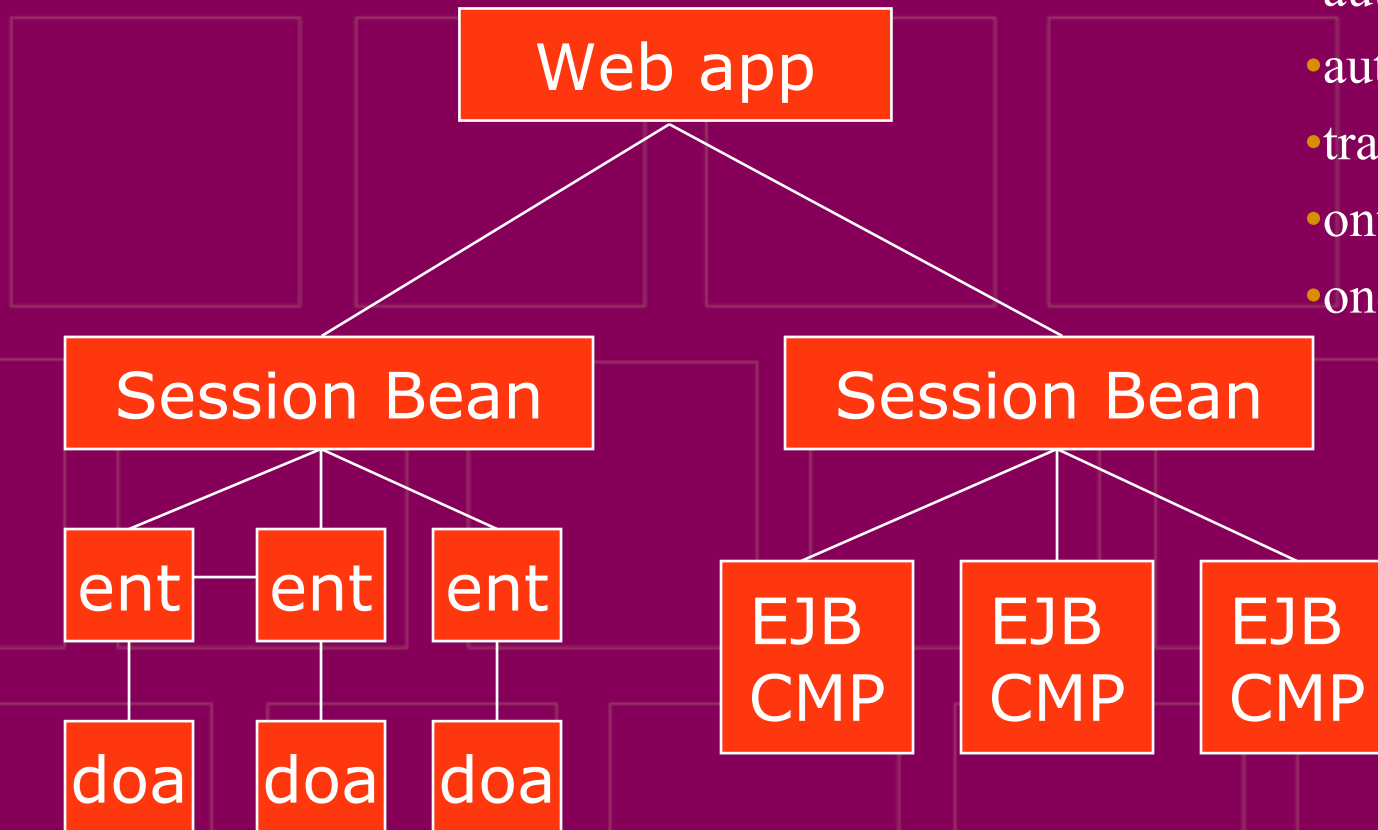
AOP:

Modellering van systeembrede functionaliteit in herbruikbare stukjes software: **Aspect**

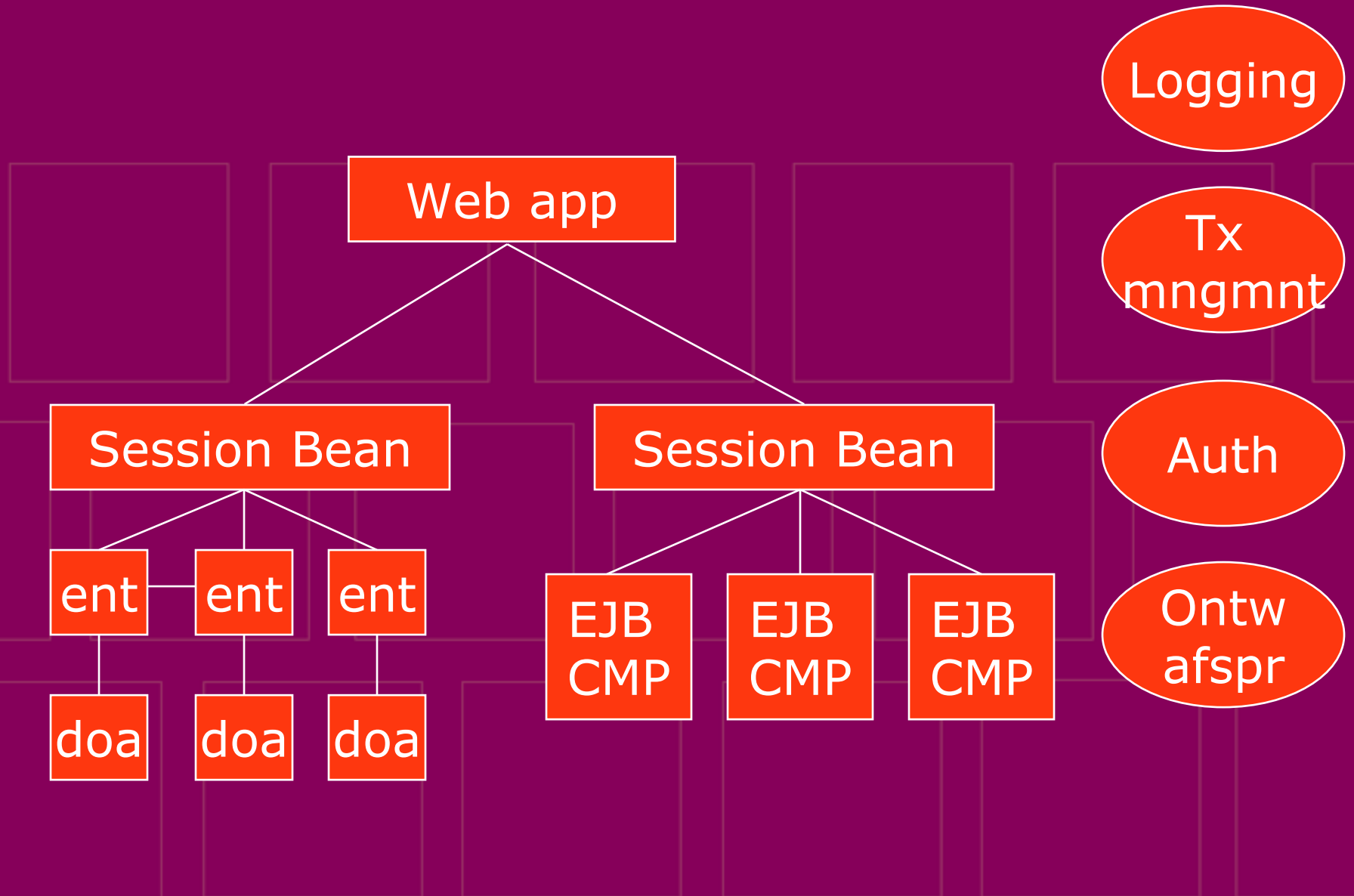
als aanvulling op bestaande software
ontwikkelingsmethodiek (OOP)

System wide concerns

- logging
- authorisatie
- authenticatie
- transactie management
- ontwikkelafspraken
- onderst. local deployment

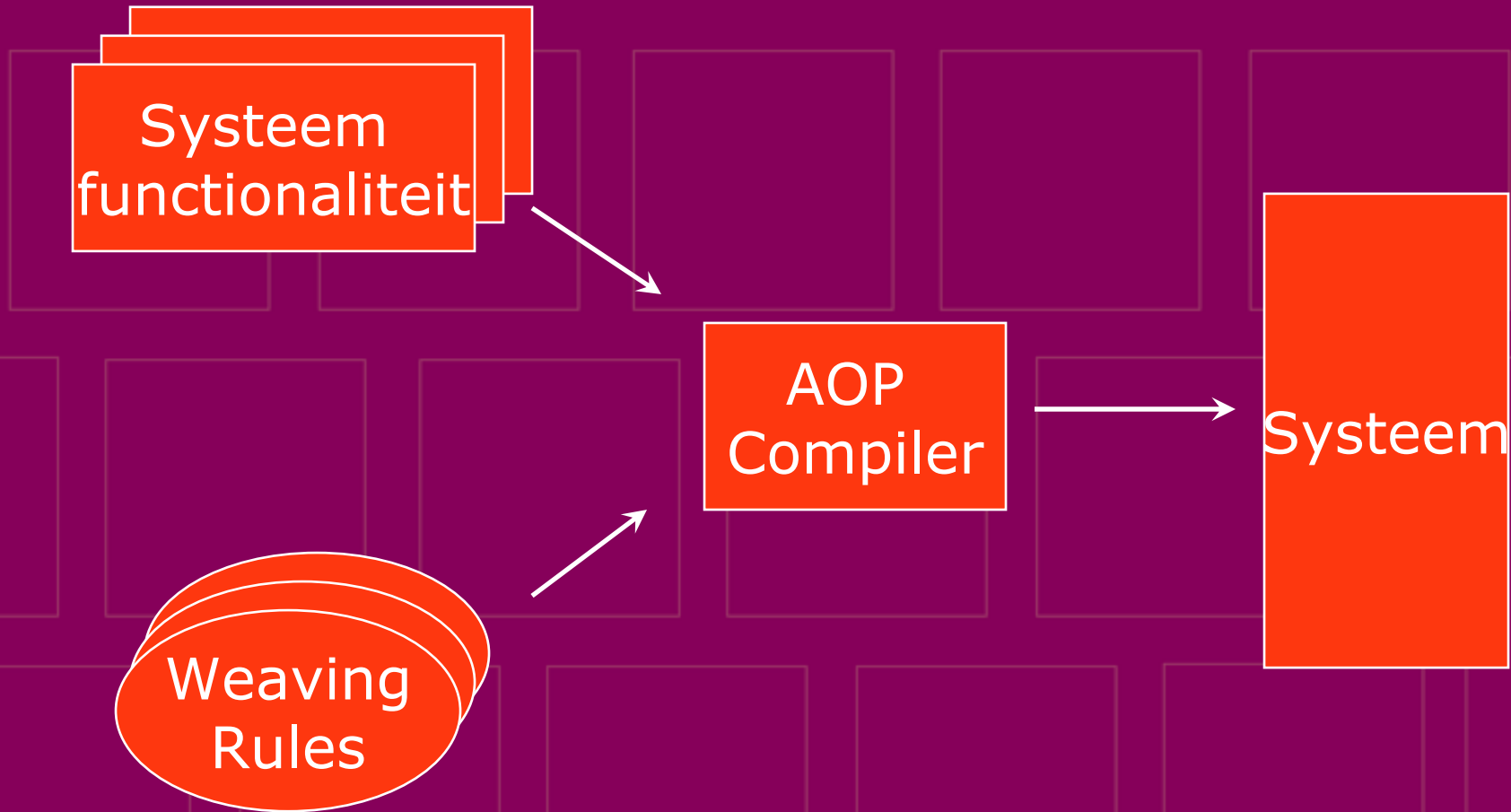


System wide concerns in aspects



Aspect Weaving

proces waarin Aspects door bestaande code worden geweven



- Aspect:
 - eenheid van modularisatie
- Joinpoint
 - identificeerbaar punt in programmaexecutie
- Pointcut
 - verzamelt joinpoints en hun context
- Advice
 - code die uitgevoerd wordt voor en/of na een joinpoint

public aspect TracingAspect

```
{  
    // aspect inhoud...  
}
```

Aspect:

- kan data members hebben en methodes
- access specificatie
- kan abstract zijn
- kan extenden van classes en abstract aspects
- kan interfaces implementeren
- kan niet extenden van niet-abstract aspect
- In eigen file, maar ook emmbedded in class of interface

Identificeerbaar punt in programma executie

- Methode aanroep / uitvoer
- Constructor aanroep / uitvoer
- Field access (get en set)
- Exception handling
- Class initialisatie
- Advice uitvoer

access specifier

pointcut naam

signature

```
public pointcut tracePoints() : execution (* *.* (*..))
```

keyword

pointcut type

- set / get
- call
- execution

```
public pointcut tracePoints() : execution (* SomeClass.methodeA (..))
```

- Advice kan voor, na of om een joinpoint uitgevoerd worden

```
before() : tracePoints() {  
    System.out.println("Before: " + thisJoinPoint);  
}
```

```
after() : tracePoints() {  
    System.out.println("After: " + thisJoinPoint);  
}
```

Een voorbeeld...

```
public aspect LoggingAspect {
```

```
    pointcut tracePoints() : execution(* *.* (..)) && !within(LoggingAspect);
```

```
    before() : tracePoints() {
```

```
        System.out.println("Before: " + thisJoinPoint);
```

```
    }
```

```
    after() : tracePoints() {
```

```
        System.out.println("After: " + thisJoinPoint);
```

```
    }
```

```
}
```

- Statische crosscutting instructie om compile-time warnings of errors toe te voegen bij gebruik bepaalde gebruikspatronen

```
declare warning : get(* System.out) :
```

```
“System.out: Overweeg om Log4J te gebruiken.”;
```

- 3 soorten after() advice

- after()

- Altijd uitgevoerd

- after() returning

- Nadat een methode succesvol is uitgevoerd

- after() throwing

- Nadat een Exception is gegooid

- around() advice

- Omsluit het gegeven joinpoint

- before() en after()

- Heeft de mogelijkheid om het joinpoint wel of niet uit te laten voeren

- proceed()

- Met dezelfde of andere argumenten

- Control flow based pointcuts

- `cflow(call (* Class.method(..))`

- Alle join points in de control flow van de aangeroepen methode, methode inclusief

- `cflowbelow(call (* Class.method(..))`

- Alle joinpoints in de control flow van de aangeroepen methode, de methode zelf niet meegerekend

- Lexical-structure based pointcuts

- `within(Class)`

- Alle join points binnen de lexical scope van de genoemde class (dus ook inner classes)

- `withincode (* Class.method(..))`

- Alle join points binnen de lexical scope van de genoemde methode (dus ook inner classes)

- Execution Object pointcuts

- this(Class)

- Alle join points waar **this** een instance is van de genoemde class (ook subclasses)

- target (Class)

- Alle join points waar het object waarvan een methode is aangeroepen een **instance of Class** is, of een subclass

- Statische crosscutting instructie die veranderingen aan class, interface of aspect aan brengt.

- Hiermee is het mogelijk een variabele of methode toe te voegen

- declare parents: Account implements Serializable

- Concept AOP
- Korte introductie in AspectJ, er is nog meer...
- Mogelijkheden zijn legio
- Nieuw paradigma heeft tijd nodig

- Home page

<http://eclipse.org/aspectj>

- Plug-ins voor Eclipse / WSAD

<http://eclipse.org/ajdt/>

- boek: AspectJ In Action

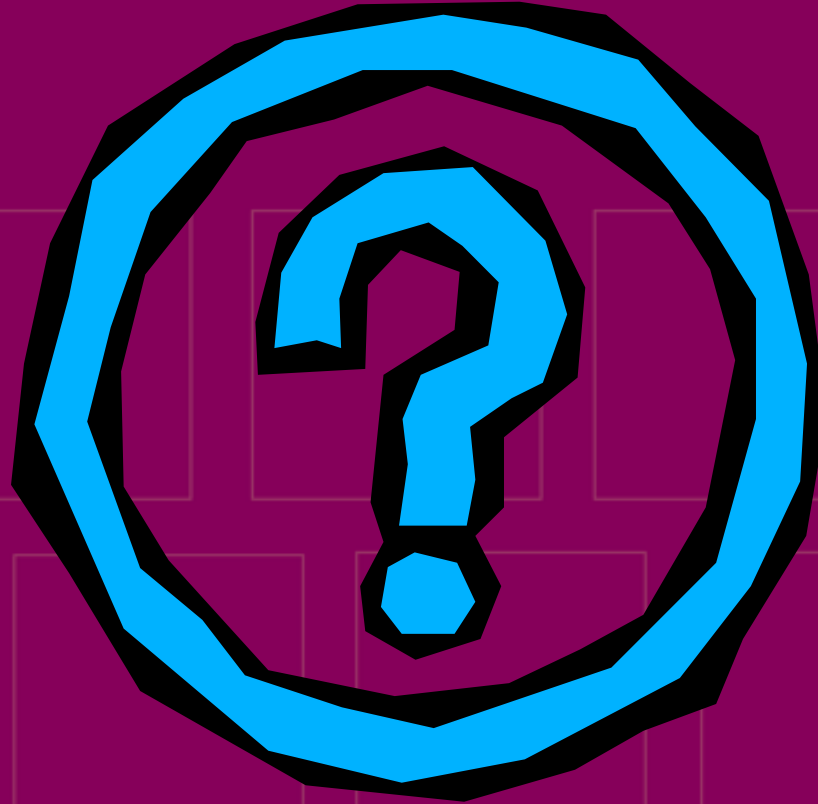
Ramnivas Laddad

ISBN: 1-930110-93-6

- artikel: Aspect Oriented Programming met AspectJ

http://www.nl-jug.org/pages/articles/members/steeghs_aop/

Vragen...



ruud.steeghs@sogeti.nl