Annotation-Driven Aspect-Oriented Programming

Jonas Bonér
Senior Software Engineer
Alexandre Vasseur
Software Engineer

What will you learn?

Why and how
Java 5 Annotations and AOP
can complement and
benefit from each other

Who are we?

Jonas Bonér

- Senior Software Engineer at Java Runtime Products Group, BEA Systems
- Founder of the AspectWerkz AOP framework
- Speaker at JavaPolis 2003, eWorld 2004, AOSD 2004, BEA User Group 2004, JavaOne 2004

Who are we?

Alexandre Vasseur

- Software Engineer at Java Runtime Products Group, BEA Systems
- Co-founder of the AspectWerkz AOP framework
- Speaker at eWorld 2004, AOSD 2004, BEA User Group 2004, JavaOne 2004

Agenda

AOP crash course

Annotation defined AOP

Matching on annotations

Demo

Agenda

AOP crash course

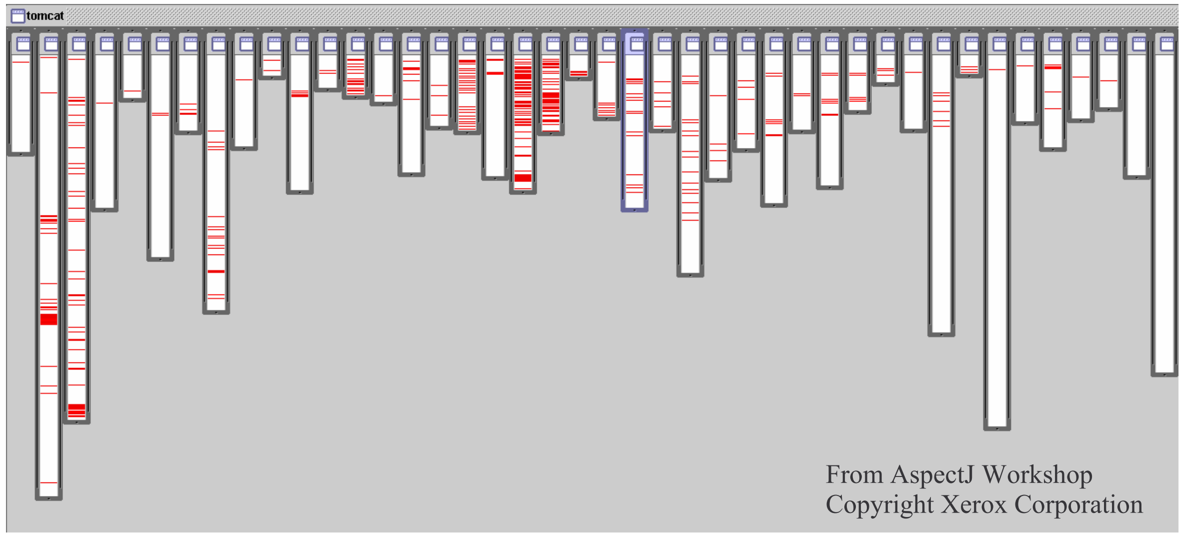
Annotation defined AOP

Matching on annotations

Demo

AOP Crash Course

- OOP fails to address "cross-cutting concerns"
  - Introduces "code tangling" and "code scattering"
  - Makes software harder to write, understand, reuse and maintain

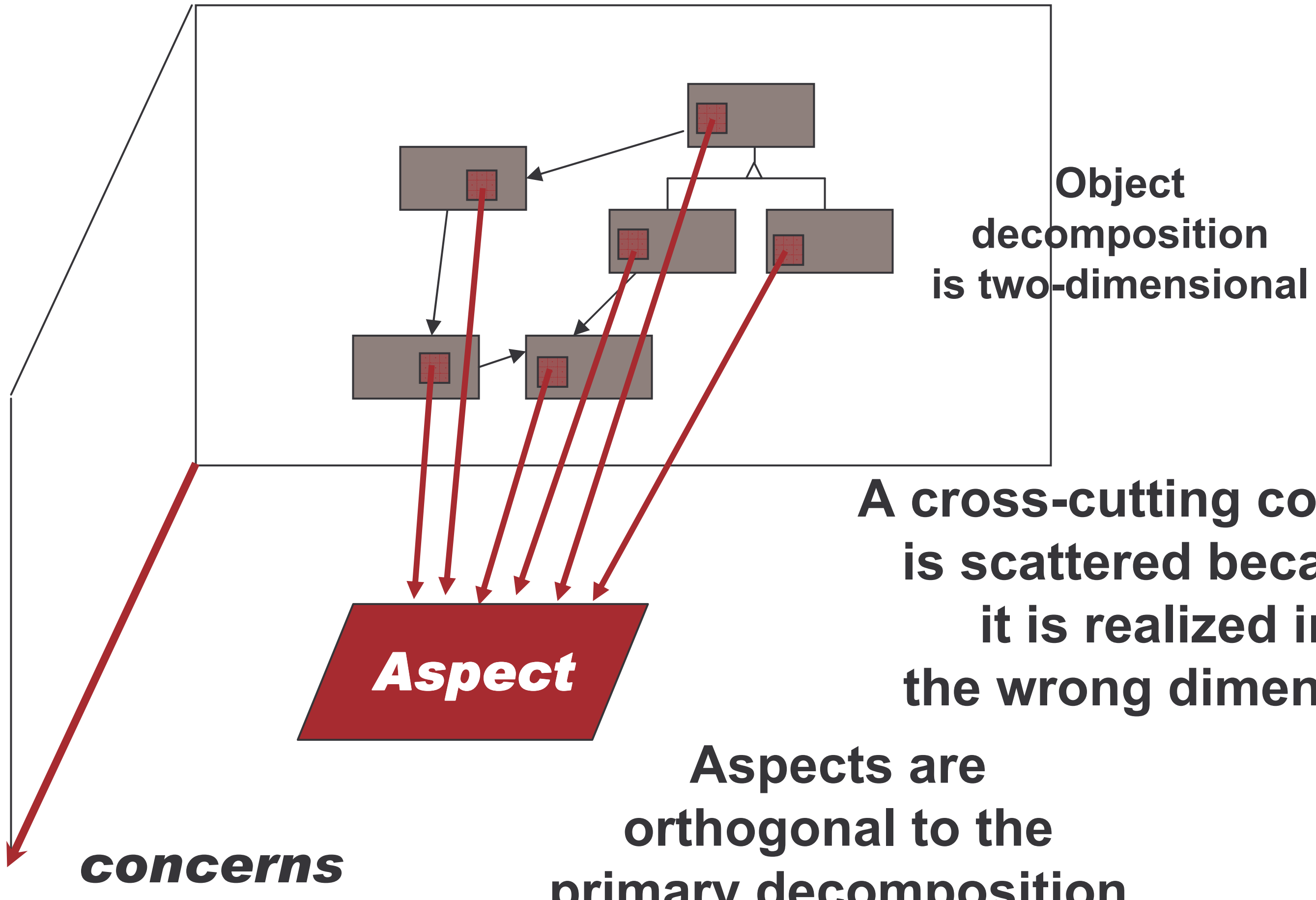


Logging in **`org.apache.tomcat`** is not modularized

- AOP enables ”Separation of Concerns”
- The Aspect modularizes a crosscutting concern

Adds a new dimension to software dev.

From presentation by Frank Sauer
Copyright Technical Resource Connection Inc.

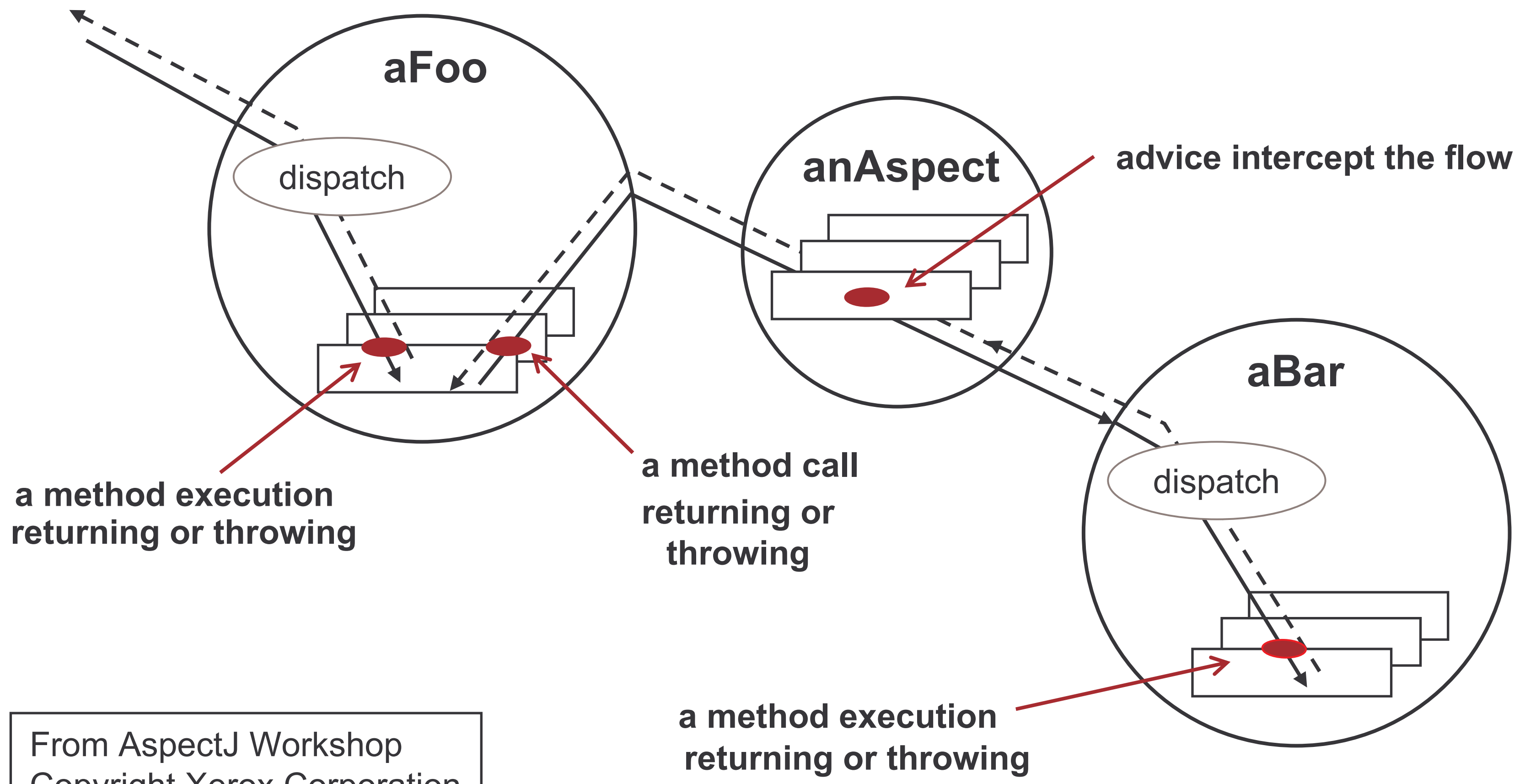


A cross-cutting concern
is scattered because
it is realized in
the wrong dimension

Aspects are
orthogonal to the
primary decomposition

Join points

- Well-defined points in the program flow



aFoo
dispatch
anAspect
advice intercept the flow
aBar
dispatch
a method execution returning or throwing
a method call returning or throwing
a method execution returning or throwing

From AspectJ Workshop
Copyright Xerox Corporation

AOP Crash Course

Core concepts

1. Define well-defined points in the program flow
   – Join points
2. Pick out these points
   – Pointcuts
3. Influence the behaviour at these points
   – Advice
4. Weave everything together in a functional system
   – Weaver

Example aspect

AspectJ sample

```
public aspect AsyncAspect {

    private Executor m_threadPool = ...

    pointcut async(): execution(void Math.async*(..));

    Object around(): async() {
        m_threadPool.execute(new Runnable() {
            public void run() {
                try {
                    // proceed the execution in a new thread
                    proceed();
                } catch (Throwable e) {
                    throw new WrappedRuntimeException(e);
                }
            });
        return null;
    }
}
```

aspect is a Java keyword

pointcut is a Java keyword

Advice binding

around defines and binds an around advice

proceed() invokes the next advice or the target join point (method, field …), – only for Around advice

Agenda

AOP crash course

Annotation defined AOP

Matching on annotations

Demo

Annotation defined aspect

AspectWerkz sample

```
public class AsyncAspect {

    private Executor m_threadPool = ...

    @Expression("execution(void Math.async*(..))")
    Pointcut asyncMethods;

    @Around("asyncMethods")
    Object async(JoinPoint jp) {
        m_threadPool.execute(new Runnable() {
            public void run() {
                try {
                    // proceed the execution in a new thread
                    jp.proceed();
                } catch (Throwable e) {
                    throw new WrappedRuntimeException(e);
                }
            });
        return null;
    }
}
```

Aspect is a Java class

@Expression defines pointcuts

@Around annotated methods are around advices

Annotation defined AOP

- AOP constructs defined with Annotations

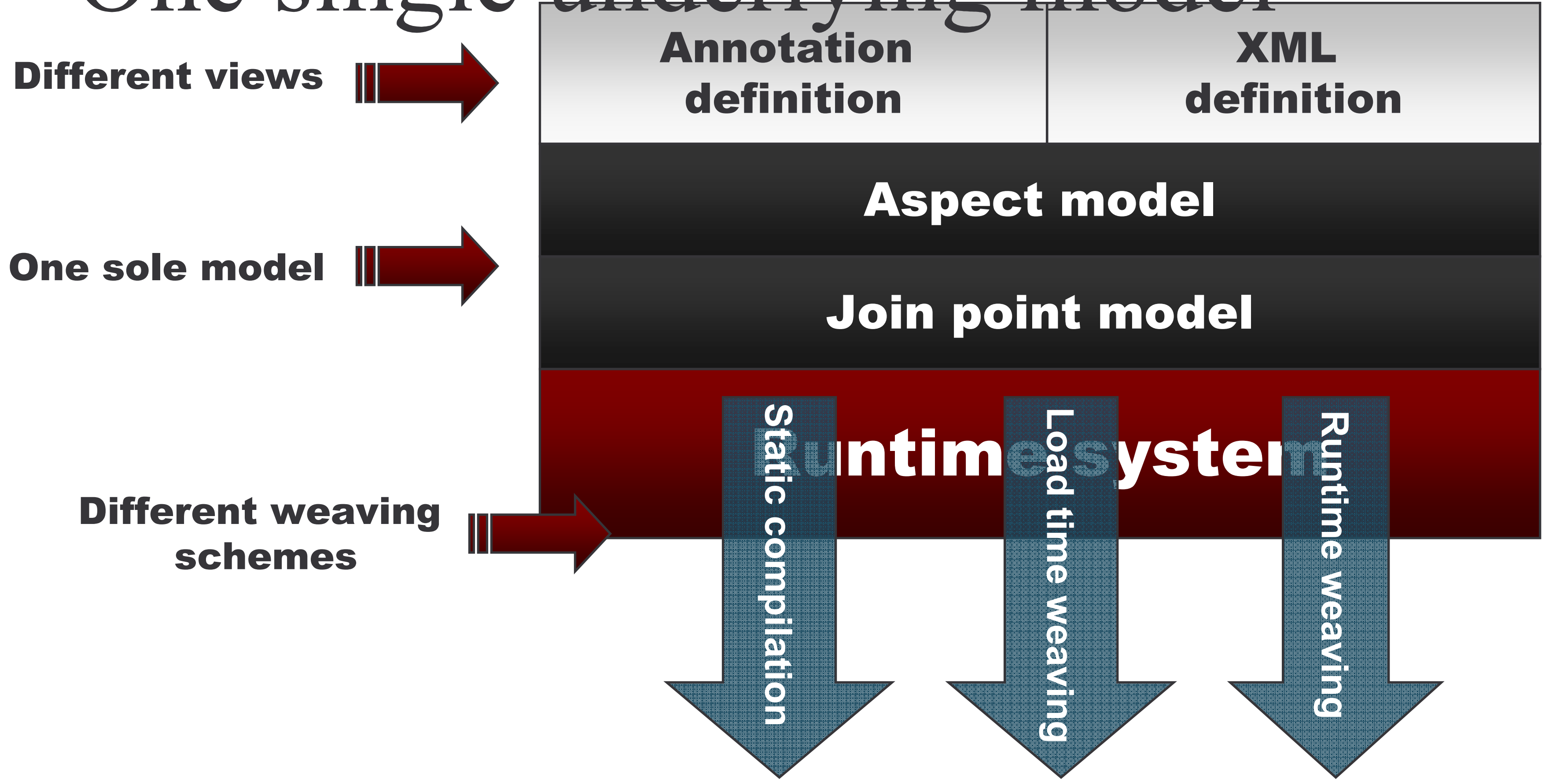
```
@Aspect
@Expression(...)
@Before(...), @Around(...), @After(...)
```

- Pointcuts can expose contextual information

```
@Args, @This, @Target

@Before("call(double Math.divide(int, int))")
void advice(JoinPoint jp,
            @Args int value,
            @Args int divideBy
            @This Object callee) {
   if (divideBy == 0) {
       throw new IllegalArgumentException(
           "division with 0 by " +
           callee.toString());
   }
}
```

Annotations and XML – One single underlying model



Annotation defined AOP

Annotations improve Java support for AOP

- AOP constructs are regular Java elements
- Does not break Java parser, IDEs
- Integrates well into existing tools (refactoring, testing, etc.)

But might not always be the best option

- XML deployment descriptor allows for late definition and binding
- JavaDoc annotations are needed for Java 1.3/1.4

Agenda

AOP crash course

Annotation defined AOP

Matching on annotations

Demo

Annotation driven AOP

Matching on annotations

```
@Service
public class Math {

    @Async(timeout=10)
    public void asyncAdd(int a, int b) {
        ...
    }
}

public class AsyncAspect {
    @Around
    @Execution(Async.class)
    @Within(Service.class)
    @Around("execution(void Math.asyncAdd(..))")
    @Around("@execution(@Async) && @within(@Service)")
    Object async(JoinPoint jp) {
        ...
    }
}
```

Strongly typed

Annotation driven AOP

Matching on annotations

```
@Service
public class Math {

    @Async(timeout=10)
    public void asyncAdd(int a, int b) {
        ...
    }
}

public class AsyncAspect {
    @Around
    @Execution(Async.class)
    @Within(Service.class)
    Object async(JoinPoint jp) {
        ...
    }
}
```

Strongly typed

Matching on annotations

Advantages

- Strongly typed matching
- More predictable matching
- Supports refactoring
- Higher abstraction level
  - The "contract“ is based on meta-data instead of implementation details (names and types)
- Possibility of using standardized annotations (JSR-250, EJB3, JSR-181 etc.)

Demo

Annotation driven AOP

- Annotation defined AOP
- Matching on annotations

Matching on annotations

Drawbacks

- Annotations introduce "*code scattering*“
- Requires developer awareness
- Obliviousness of AOP is reduced

- Might introduce stronger coupling between aspects and target application
- Defines an explicit contract that both needs to be aware of

Wrap up

- Annotation defined AOP
  - Framework-defined annotations
  - AOP constructs are regular Java elements
  - Does not break existing Java parsers, IDEs, tools etc.

- Matching on annotations
  - User-defined annotations can be used to define system behaviour
  - AOP is a way to implement this behaviour
  - Standardization of annotations is starting
  - More predictable strongly typed matching

If you only remember one thing…

AOP adds behaviour to
annotations static metadata

For More Information

- http://aspectwerkz.codehaus.org
- http://www.bea.com/products/weblogic/jrockit/

- http://blogs.codehaus.org/people/jboner
- http://blogs.codehaus.org/people/avasseur

Q&A

Annotation-Driven Aspect-Oriented Programming

Jonas Bonér
jboner@bea.com

Alexandre Vasseur
avasseur@bea.com