JavaOne℠

Sun's 2004 Worldwide Java Developer Conference℠

Aspect-Oriented Programming in Java™ 2 Platform, Enterprise Edition (J2EE™) Environments

java.sun.com/javaone/sf

Jonas Bonér
Senior Software Engineer
Alexandre Vasseur
Software Engineer

bea™

Java™

Sun microsystems

Session Goal

What you will learn

Learn how to apply AOP in J2EE™ environments

Speaker's Qualifications

Jonas Bonér

- Senior Software Engineer at JRockit™ Group, BEA Systems
- Founder of the AspectWerkz AOP framework
- Speaker at JavaPolis 2003, eWorld 2004, AOSD 2004, BEA User Group 2004

Speaker's Qualifications

Alexandre Vasseur

- Software Engineer at JRockit™ Group, BEA Systems
- Co-founder of the AspectWerkz AOP framework
- Speaker at eWorld 2004, AOSD 2004, BEA User Group 2004

Agenda

- AOP crash course
- Add-on aspects
- Architectural aspects
- AOP container integration in J2EE™ architecture

Agenda

AOP crash course

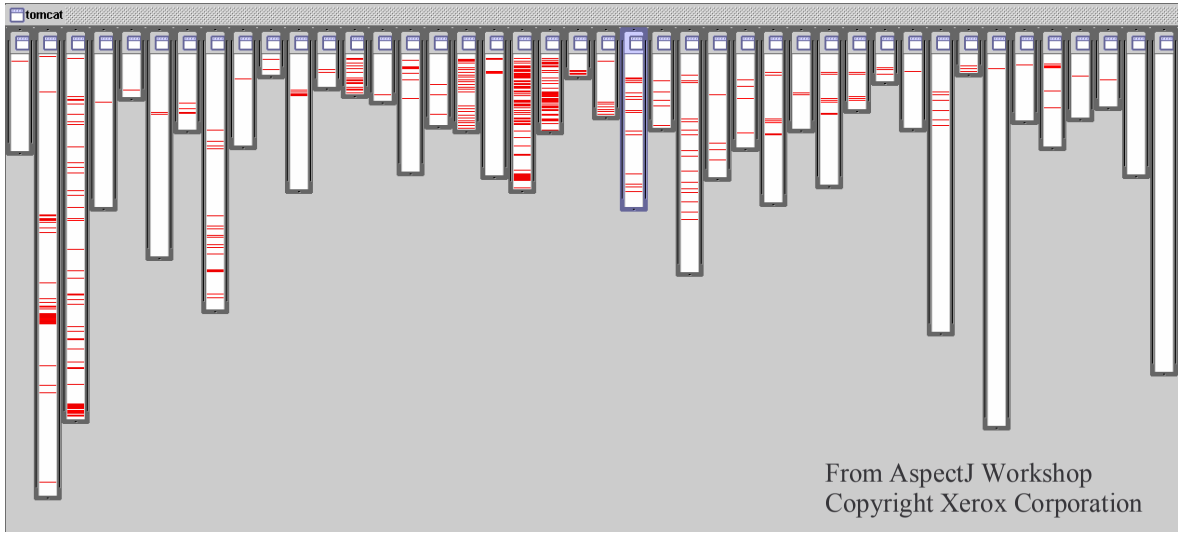
Add-on aspects

Architectural aspects

AOP container integration in J2EE™ architecture

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

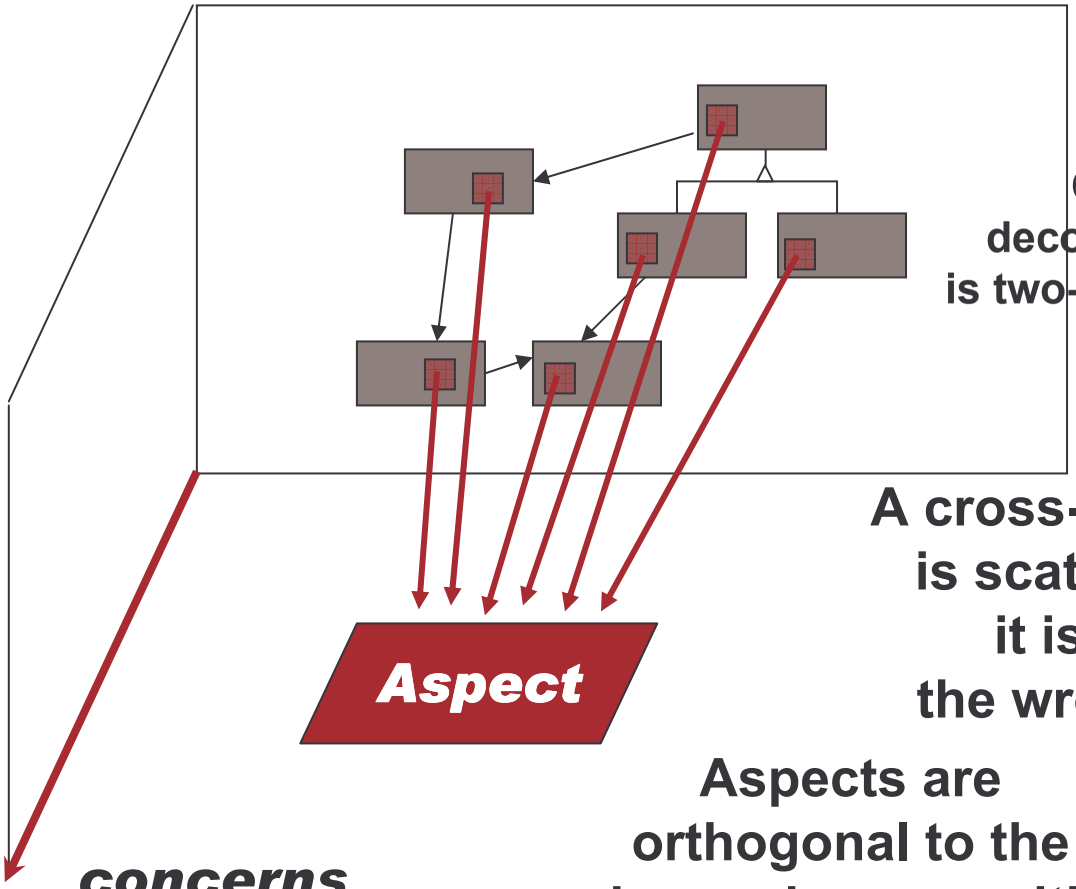


Logging in `org.apache.tomcat` is not modularized

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

Adds a new dimension to software dev.

From presentation by Frank Sauer



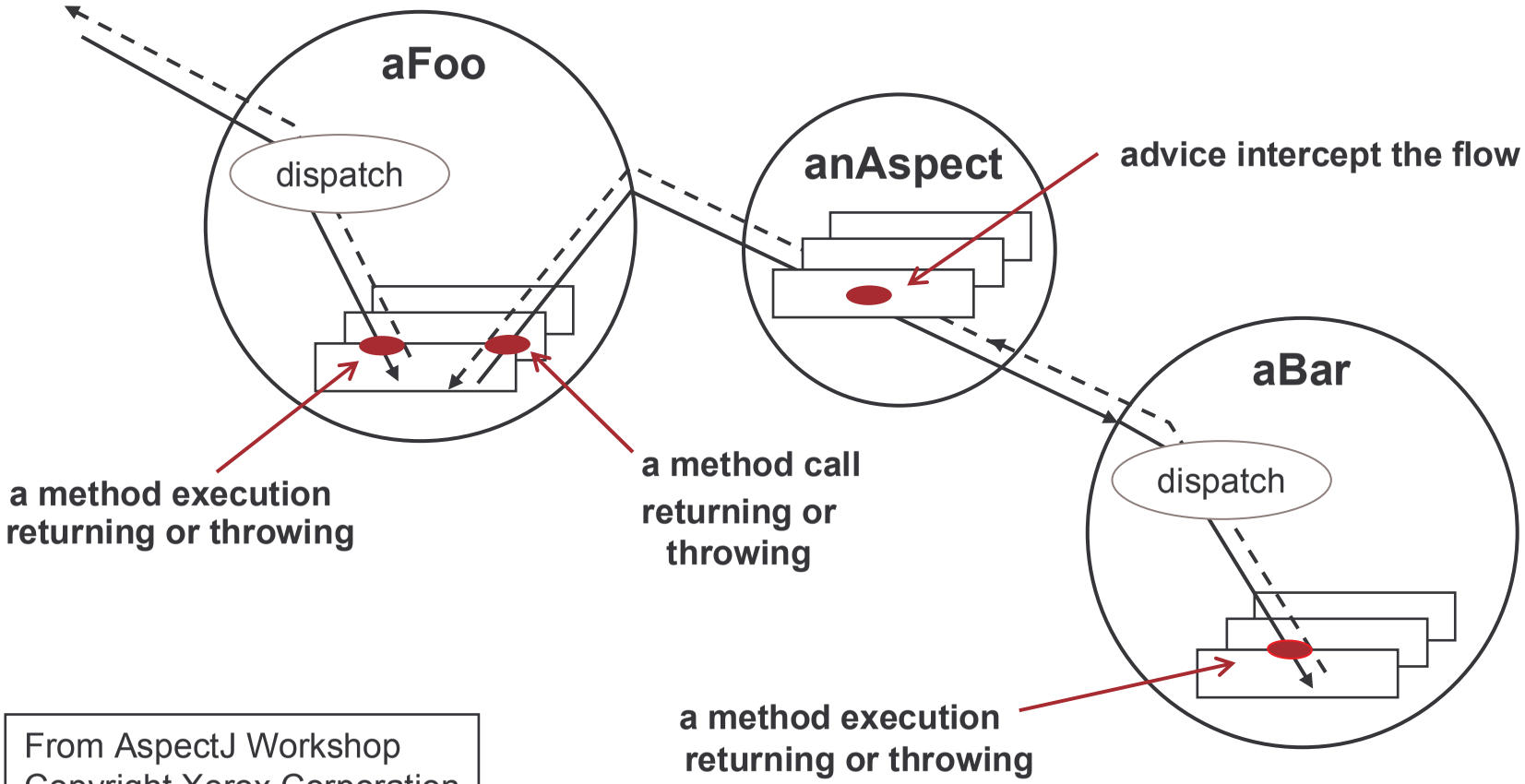
Object decomposition is two-dimensional

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



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, introductions
4. Weave everything together in a functional system
   - Weaver

AOP Crash Course (With AspectWerkz)

AspectWerkz AOP code sample

```
public class CacheAspect {

    //...utility methods etc.

    @Expression("execution(Integer Computation.exec(Integer))")
    Pointcut toCache;

    @Around("toCache")
    public Object cache(JoinPoint jp) {
        MethodRtti rtti = (MethodRtti)jp.getRtti();
        Integer parameter = (Integer)rtti.getParameterValues()[0];
        // if (inCache) return fromCache
        // else
        Object result = jp.proceed();
        // put result in Cache
        return result;
    }
}
```

Aspect is a Java class

Pointcuts are fields

Advice binding

Advice are methods

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

AOP Crash Course (With AspectWerkz)

Aspect XML deployment descriptor

```
public class CacheAspect {

    @Expression("execution(Integer Computation.exec(Integer))")
    Pointcut toCache;

    @Around("toCache")
    public Object cache(JoinPoint jp) {
        // ...
    }
}
```

```
<aspectwerkz>
  <system id="computation">
    <aspect class="aspect.CacheAspect">
      <pointcut name="toCache" expression="..."/>
      <advice type="around" name="cache"
              bind-to="toCache"/>
    </aspect>
  </system>
</aspectwerkz>
```

Aspect class name

Advice method name

Aspect defined in XML

AOP Crash Course (With AspectWerkz)

- Regular expression based pattern language
- Defined as Annotations

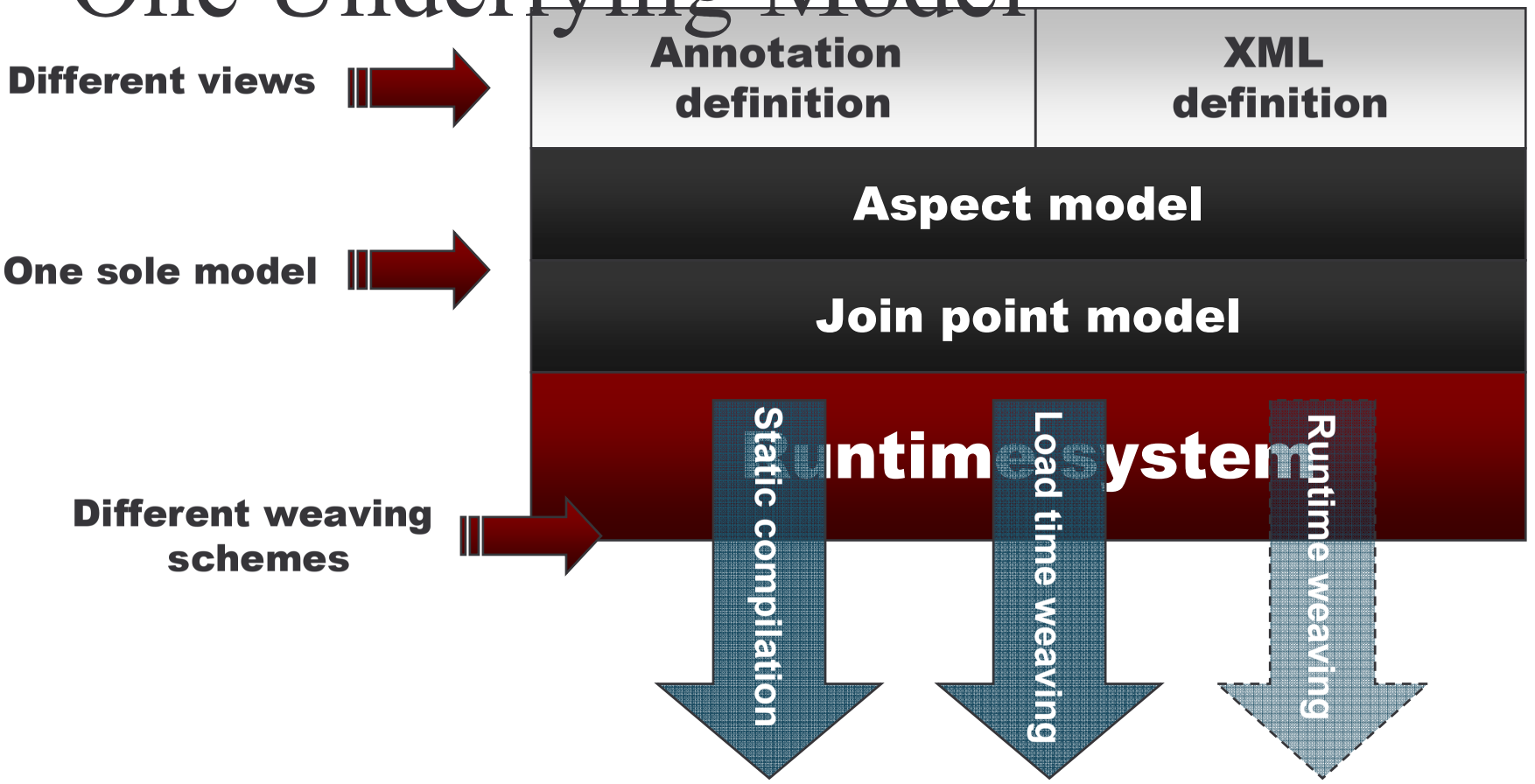
```
@Expression
    execution(@TxRequired *.*(..))
    call(* Interface+.*(..))
    get(Baz foo.Bar.field)
    etc.
```

- Defined as XML
- Pointcuts are composable

```
OR, AND, NOT, cflow(PCD)

call(* Interface+.*(..)) AND within(foo.Bar)
```

Two Definition Schemes—One Underlying Model



AOP Crash Course

Main points

- AOP brings a new theory to address cross-cutting concerns
- Different weaving schemes for different use-cases
- AspectWerkz is a pure Java solution
  - Aspects are Annotated Java™ classes
  - Activated and/or refined through a simple XML deployment descriptor

Agenda

AOP crash course

Add-on aspects

Architectural aspects

AOP container integration in J2EE architecture

Add-on Aspects

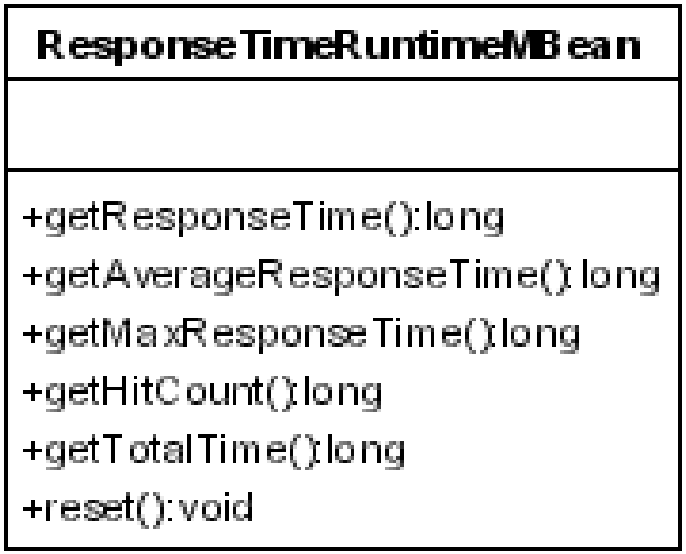
System level aspects

- Aspects deployed at application server level
  - Can be provided by application server vendor
  - Can be bundled as extensibility libraries
- Highly reusable
- Loosely coupled to functional requirements
  - General purpose tracing
  - Performance diagnostic
  - Arbitrary LRU cache

Add-on Aspects

Performance reporting aspect with JMX

- Aspect bundles with the application server
- Exposes runtime performance to Java™ Management Extensions (JMX)



Add-on Aspects

AspectWerkz AOP code sample

```
public class ResponseTimeAspect {

    MBeanServer m_mbeanServer;
    Map m_mbeans = new HashMap();

    public Object monitor(JoinPoint joinPoint) {
        long ts = System.currentTimeMillis();
        Object result = null;
        try {
            result = joinPoint.proceed();
        } finally {
            // fetch mbean from cache Map or register it
            ObjectInstance mbean = getOrRegisterMBean(joinPoint);
            m_mbeanServer.invoke(mbean.getObjectName(),
              "update",
               new Object[]{new Long(System.currentTimeMillis()-ts)},
               new String[]{long.class.getName()}
            );
        }
        return result;
    }
}
```

Definition is made in META-INF/aop.xml

Demo

JMX reporting

Agenda

- AOP crash course
- Add-on aspects
- Architectural aspects
- AOP container integration in J2EE architecture

Architectural Aspects

Aspect-oriented analysis and design

- AOP is not only for add-on and patch-like behaviour
- Should be a part of analysis and design
- Design core cross-cutting behaviour as aspects
- Cross-cutting concerns can be refactored out
- Deployed within the application and defined in `META-INF/aop.xml`

Architectural Aspects

Examples from the J2EE architecture world

- Role-based security
- Declarative transaction demarcation
- Persistence
- Synchronization
- Messaging
- Business rules
- Design patterns
- Much more…

Architectural Aspects

Case study: Role-based security with AOP

- Security is a cross-cutting concern
- In a regular OO based implementation; the concern is scattered all over the code base:
  - Authentication code at all service methods
  - Authorization code at all critical business methods

Architectural Aspects

Enter Aspect-Oriented Programming

- AOP makes it possible to capture the concern in one single modular unit (the aspect)
- Security aspect that implements
  - Authentication
  - Authorization (on method or even field level)
  - Role-based using Java™ Authentication and Authorization Service (JAAS) (pluggable)
- Combination of annotation and XML definition can make the aspect reusable

Architectural Aspects

Authentication advice

```
@Around("authenticationPoints")
public Object authenticateUser(JoinPoint joinPoint)
    throws Throwable {
    if (m_subject == null) {
       // no subject => authentication required
       Context ctx = ... // principals and credentials
       m_subject = m_securityManager.authenticate(ctx);
    }
    Object result = Subject.doAsPrivileged(
       m_subject, new PrivilegedExceptionAction() {
          public Object run() throws Exception {
             return joinPoint.proceed();
          };
       }, null
    );
    return result;
  }
```

Architectural Aspects

Authorization advice

```
@Around("authorizationPoints")
public Object authorizeUser(JoinPoint joinPoint)
    throws Throwable {
    MethodRtti rtti = (MethodRtti)joinPoint.getRtti();
    if (m_securityManager.checkPermission(
        m_subject,
        rtti.getMethod())) {
        // user is authorized => proceed
        return joinPoint.proceed();
    }
    else {
        throw new SecurityException(...);
    }
}
```

Architectural Aspects

Integration in the web application

- Define the pointcuts:
  - `authenticationPoints`
  - `authorizationPoints`
- Authenticate on all methods with the
  - `@Authenticate` annotation
- Authorize on all methods with the
  - `@Authorize` annotation

Architectural Aspects

Define the pointcuts in the XML descriptor

```
<aspectwerkz>
  <system id="security">
    <aspect class="aspect.RoleBasedAccessProtocol">
      <pointcut
        name="authenticationPoints"
        expression="execution(@Authenticate * *.*(..))"/>
      <pointcut
        name="authorizationPoints"
        expression="execution(@Authorize * *.*(..))"/>
    </aspect>
  </system>
</aspectwerkz>
```

J2EE aspects

Take control over the configuration and instantiation of the aspects

- Pluggable container implementation
- In this demo we have a container based on the Spring framework
- Handles
  - Configuration (using rich metadata structures)
  - Instantiation (singleton or prototype pattern)
  - Initialization

…for the aspects

Demo

Role-based security
for web applications

Architectural Aspects

Reusable aspect libraries

- AWare (http://docs.codehaus.org/display/AWARE)
  - Aspect repository with reusable aspects for J2EE
  - AspectWerkz based
- JBoss (http://www.jboss.org/)
  - Library of system level aspects
  - Tied to JBoss application server
- aTrack (https://atrack.dev.java.net/)
  - Best practices on how to use AOP in J2EE application environments
  - Library with both add-on and architectural aspects
  - AspectJ based

Agenda

AOP crash course

Add-on aspects

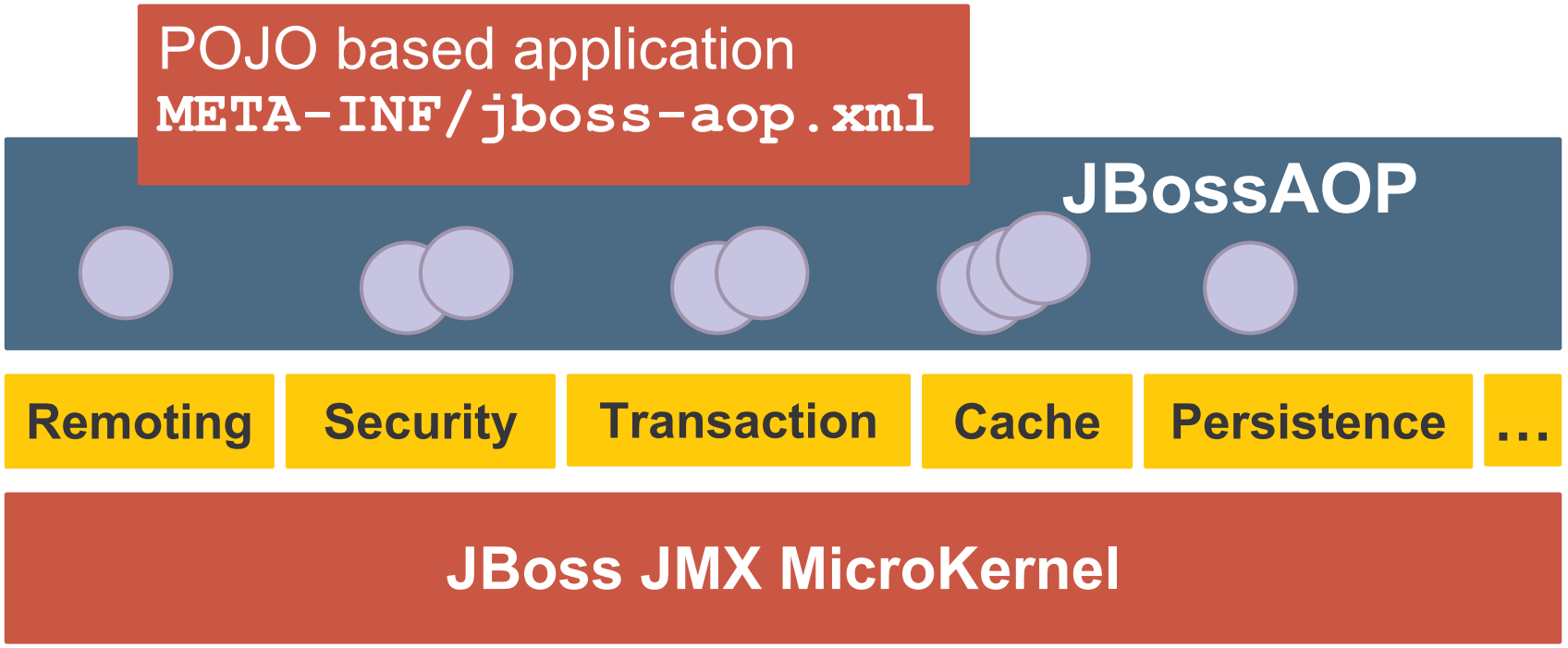
Architectural aspects

AOP container integration in J2EE architecture

Aspect Container Integration in J2EE Architecture

Second generation is integrated

- JBossAOP (JBoss 4)
  - Integrated in JBoss 4 class loading internals
  - Plain Old Java Object (POJO) middleware



Aspect Container Integration in J2EE Architecture

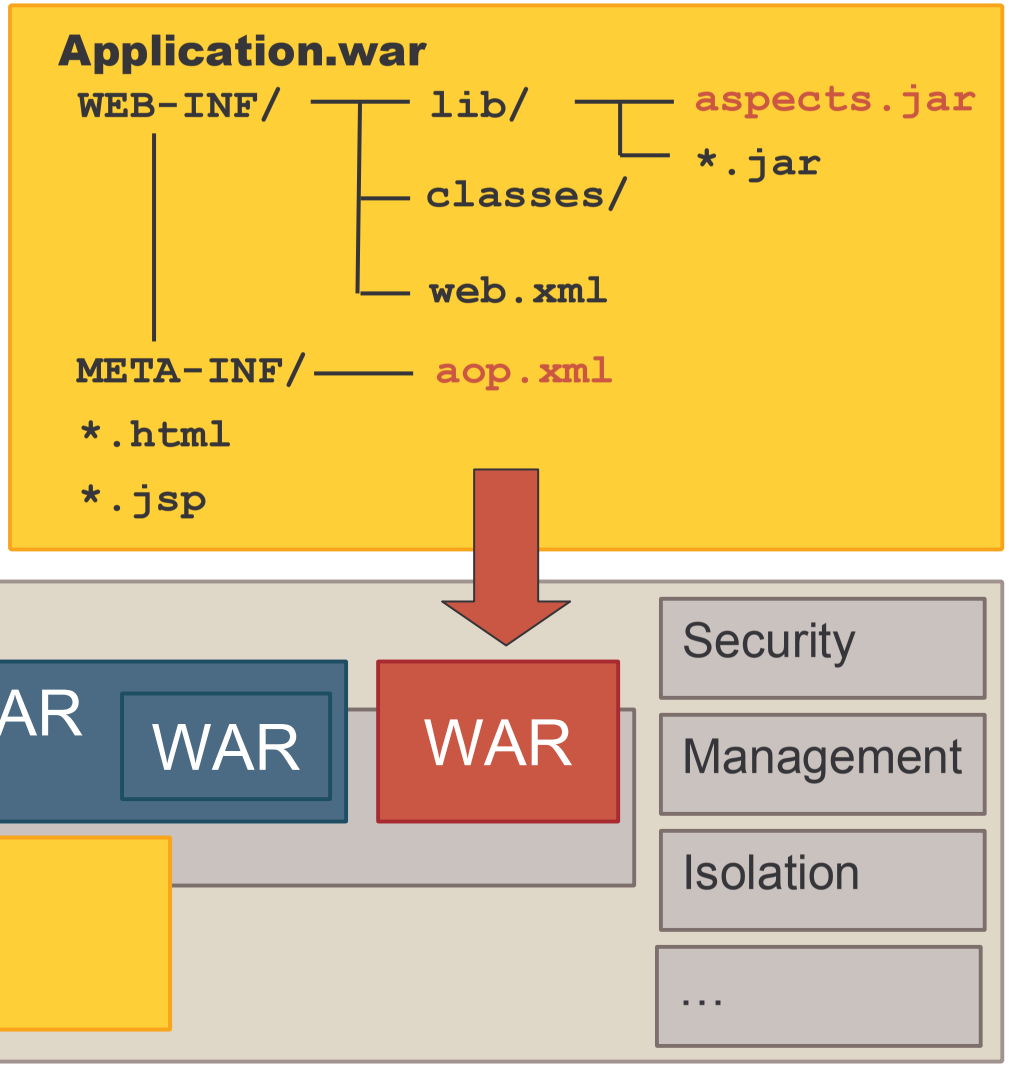
Second generation is integrated

- AspectWerkz
  - Cross server integration, Java 1.3+
  - Runtime weaving support through BEA JRockit
  - Aspects are plain Java classes with annotations (Tiger)
  - Provides cross-platform AOP container


Aspect Container Integration in J2EE™ Architecture

AspectWerkz's Aspect Container

- Define the `META-INF/aop.xml`
- Package aspects
- Deploy as usual



Aspect Container Integration in J2EE™ Architecture

AspectWerkz's Aspect Container

- `META-INF/aop.xml`
  - Defines the scope of the deployed aspects
- An aspect deployed at system level will affect all your deployed applications
- An aspect deployed within an application will affect only this application

Summary

- Java compatible AOP frameworks can be used in any J2EE™ application server
  - Flattens out the learning curve
  - Seamless integration, added value
- Add-on aspects and architectural aspects capture different cross-cutting concerns at different J2EE™ levels
  - System aspects at application server level
  - Aspect deployed alongside applications with XML definition
- J2EE™ integration of AOP is **`META-INF/aop.xml`**
  - Made seamless and cross-platform with AspectWerkz
  - AOP is used in JBoss EJB 3 implementation

For More Information

- http://aspectwerkz.codehaus.org

- Attend AOSD.05 in Chicago
- http://aosd.net

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

Q&A

Aspect-Oriented Programming in Java™ 2 Platform, Enterprise Edition (J2EE™) Environments

Jonas Bonér
jboner@bea.com

Alexandre Vasseur
avasseur@bea.com

java.sun.com/javaone/sf