What are the key issues for commercial AOP use

- how does AspectWerkz address them?

Jonas Bonér
Senior Software Engineer
BEA Systems

Outline

- **I AspectWerkz overview**
  - what is AspectWerkz?
  - how does it compare with AspectJ?
- **II Design goals and decisions**
  - what are the key issues for commercial AOP?
  - how does AspectWerkz solve them?
- **III Conclusion**
  - future, links, questions

Outline

- **I AspectWerkz overview**
  - what is AspectWerkz?
  - how does it compare with AspectJ?
- **II Design goals and decisions**
  - what are the key issues for commercial AOP?
  - how does AspectWerkz solve them?
- **III Conclusion**
  - future, links, questions

What is AspectWerkz?

- Dynamic AOP framework for Java
- Open Source, founded Q4 2002
- Sponsored by 

- Tailored for dynamic AOP in real world applications

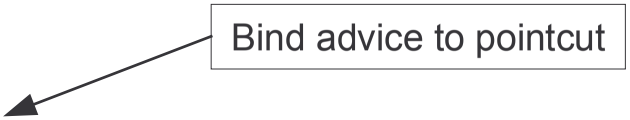
- JLS compatible
- Definition syntax in XML and/or Attributes
- Load time, runtime and static weaving
- Allows redefinition of aspects at runtime

Example using AspectJ

```
aspect AsynchAspect {
    private ThreadPool m_threadPool = ...


    Object around(): execution(void foo.bar.Baz.*(..)){
        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;
    }
}
```

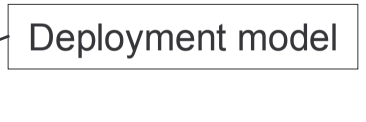
The same example using AspectWerkz

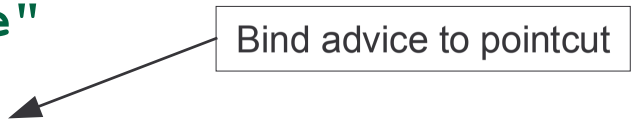
```
class AsynchAspect extends Aspect {
    private ThreadPool m_threadPool = ...

    /** @Around execution(void foo.bar.Baz.*(..)) */
    Object execute(JoinPoint joinPoint) throws Throwable {
        m_threadPool.execute(new Runnable() {
            public void run() {
                try {
                    // proceed the execution in a new thread
                    joinPoint.proceed();
                } catch (Throwable e) {
                    throw new WrappedRuntimeException(e);
                }
            });
        return null;
    }
}
```



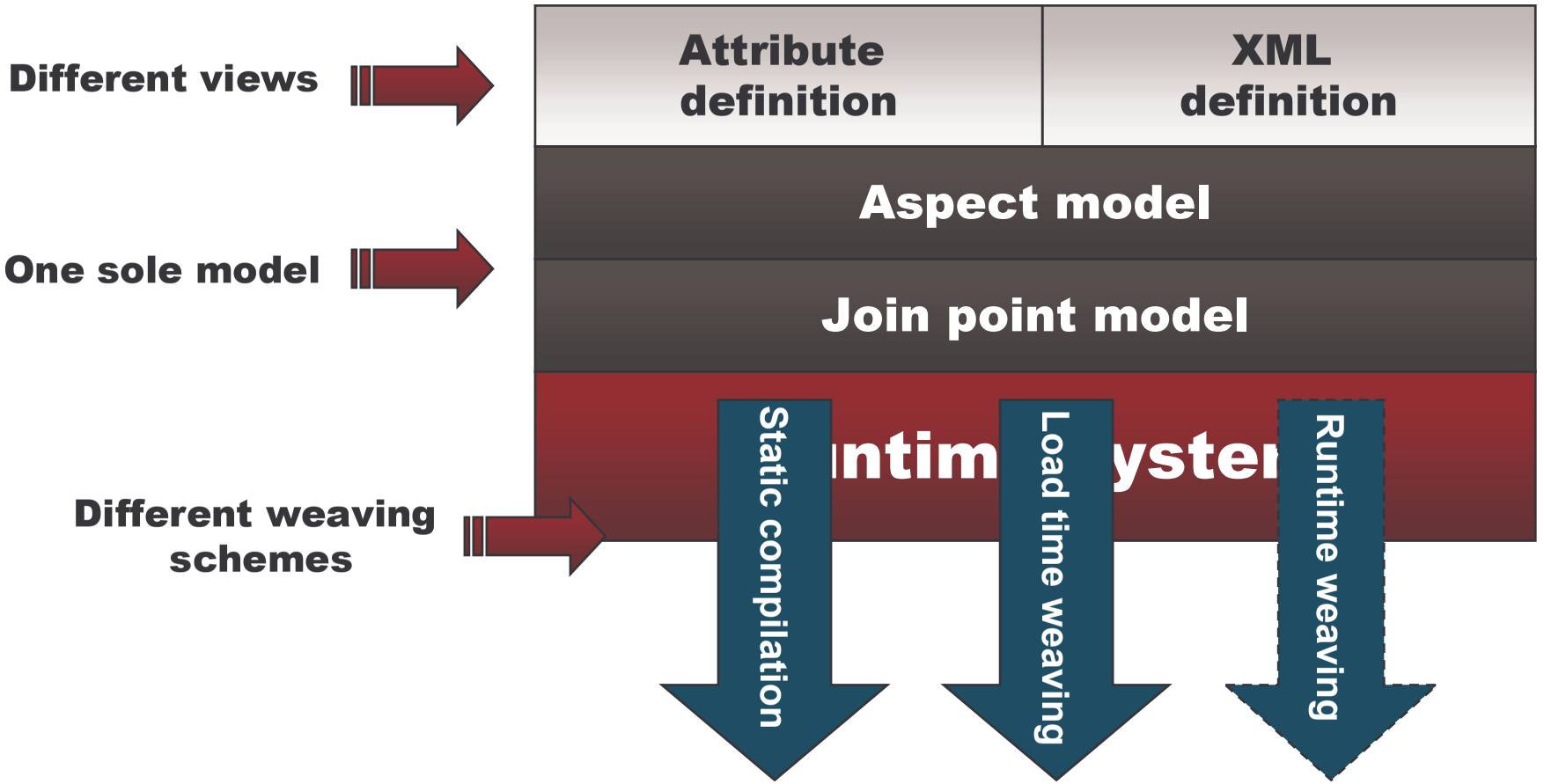
XML definition syntax

```
<aspect class="samples.AsynchAspect"
    deployment-model="perJVM">

    <advice name="execute"
        type="around"
        bind-to="execution(void foo.bar.Baz.*(..))"/>

</aspect>
```





One underlying model



Outline

- I AspectWerkz overview
  - what is AspectWerkz?
  - how does it compare with AspectJ?
- II Design goals and decisions
  - what are the key issues for commercial AOP?
  - how does AspectWerkz solve them?
- III Conclusion
  - future, links, questions, etc.

Design goals and decisions overview

Decisions / Goals	JLS compatibility	XML and Attribute definition	Load time and Runtime weaving	AspectJ's join point/ pointcut model	Dynamic runtime model	Runtime compiler (JIT)
Usability & Ease of adoption	X	X	X	X		
Integration	X	X	X			
Tool support	X	X				
Expressiveness & Orthogonality				X		
Performance vs. Dynamicity			X		X	X
Enterprise application support	X	X	X		X	X

Design goals and decisions overview

Decisions / Goals	JLS compatibility	XML and Attribute definition	Load time and Runtime weaving	AspectJ's join point/ pointcut model	Dynamic runtime model	Runtime compiler (JIT)
Usability & Ease of adoption	X	X	X	X		
Integration	X	X	X			
Tool support	X	X				
Expressiveness & Orthogonality				X		
Performance vs. Dynamicity			X		X	X
Enterprise application support	X	X	X		X	X

Design goals and decisions overview

Decisions / Goals	JLS compatibility	XML and Attribute definition	Load time and Runtime weaving	AspectJ's join point/ pointcut model	Dynamic runtime model	Runtime compiler (JIT)
Usability & Ease of adoption	X	X	X	X		
Integration	X	X	X			
Tool support	X	X				
Expressiveness & Orthogonality				X		
Performance vs. Dynamicity			X		X	X
Enterprise application support	X	X	X		X	X

Design goals and decisions overview

Decisions / Goals	JLS compatibility	XML and Attribute definition	Load time and Runtime weaving	AspectJ's join point/ pointcut model	Dynamic runtime model	Runtime compiler (JIT)
Usability & Ease of adoption	X	X	X	X		
Integration	X	X	X			
Tool support	X	X				
Expressiveness & Orthogonality				X		
Performance vs. Dynamicity			X		X	X
Enterprise application support	X	X	X		X	X

Design goals and decisions overview

Decisions / Goals	JLS compatibility	XML and Attribute definition	Load time and Runtime weaving	AspectJ's join point/ pointcut model	Dynamic runtime model	Runtime compiler (JIT)
Usability & Ease of adoption	X	X	X	X		
Integration	X	X	X			
Tool support	X	X				
Expressiveness & Orthogonality				X		
Performance vs. Dynamicity			X		X	X
Enterprise application support	X	X	X		X	X

Design goals and decisions overview

Decisions / Goals	JLS compatibility	XML and Attribute definition	Load time and Runtime weaving	AspectJ's join point/ pointcut model	Dynamic runtime model	Runtime compiler (JIT)
Usability & Ease of adoption	X	X	X	X		
Integration	X	X	X			
Tool support	X	X				
Expressiveness & Orthogonality				X		
Performance vs. Dynamicity			X		X	X
Enterprise application support	X	X	X		X	X

Design goals and decisions overview

Decisions / Goals	JLS compatibility	XML and Attribute definition	Load time and Runtime weaving	AspectJ's join point/ pointcut model	Dynamic runtime model	Runtime compiler (JIT)
Usability & Ease of adoption	X	X	X	X		
Integration	X	X	X			
Tool support	X	X				
Expressiveness & Orthogonality				X		
Performance vs. Dynamicity			X		X	X
Enterprise application support	**X**	**X**	**X**		**X**	**X**

Usability and Ease of adoption

Decisions / Goals	JLS compatibility	XML and Attribute definition	Load time and Runtime weaving	AspectJ's join point/ pointcut model	Dynamic runtime model	Runtime compiler (JIT)
Usability & Ease of adoption	X	X	X	X		
Integration	X	X	X			
Tool support	X	X				
Expressiveness & Orthogonality				X		
Performance vs. Dynamicity			X		X	X
Enterprise application support	X	X	X		X	X

Design decisions

- JLS compatibility
- Attribute definition (self-defined aspects)
  - Metadata represented as attributes
  - Currently custom *JavaDoc* tags are parsed and inserted in the compiled aspect *.class* file
  - Ready for JSR-175 (attributes in Java 1.5)
- XML definition
  - Definition of the aspect if no metadata used
  - Reuse and refinement of the model if metadata used

Advantages

- Pure Java and XML is intuitive to users of Java and J2EE

- Integrates well in **any** IDE source parse
- Works well with **any** testing framework and refactoring tool
- *AspectWerkz* stays “out of the way”

- Allows decoupling of pointcuts and advice
- Some decisions can be taken at deployment time

Disadvantages

- Less elegant and concise syntax compared to a language extension

Why two definition models?

- Why two definition models?
- Which to prefer?

- Complements each other
- Good in different contexts and situations

XML definition

- Advantages
  - No post compilation for metadata management
  - Loosely coupled
- Drawbacks
  - Separates the implementation from the definition
  - Can make the application harder to
    - maintain
    - refactor

Attribute definition

- Advantages
  - Aspects are self-defined and self-contained
  - Both implementation and definition in the same single class
  - Easy to build reusable aspect libraries
- Drawbacks
  - Requires an additional compilation step (not in Java 1.5 and above)
  - Stronger coupling

Conclusions

- Both *AspectWerkz's* approach and *AspectJ's* language extension introduces a new language
- Hard to keep the model orthogonal, extensible and expressive but yet simple and easy to understand
- Early versions with advice modeled after the command pattern did not scale at all
  - For example a single aspect could consist of five classes and a XML definition file

Integration

Decisions / Goals	JLS compatibility	XML and Attribute definition	Load time and Runtime weaving	AspectJ's join point/ pointcut model	Dynamic runtime model	Runtime compiler (JIT)
Usability & Ease of adoption	X	X	X	X		
Integration	X	X	X			
Tool support	X	X				
Expressiveness & Orthogonality				X		
Performance vs. Dynamicity			X		X	X
Enterprise application support	X	X	X		X	X

Target problem

- Using a custom class loader has problems (*JBoss*, *weblogic-aspects* and *cglib* etc.):
  - All classes needs to be loaded using this custom loader
  - Can not transform classes loaded by **any** other loader

- *AspectWerkz*'s solution:
  - Provides a **JVM wide** hook mechanism
  - Controls **all** class loading in the JVM
  - Platform and vendor independent
  - Tested and verified on: *WebLogic, JBoss, Tomcat, WebSphere, IBM JRE, BEA JRockit*, *Sun HotSpot,* Java 1.3, 1.4


Load time weaving

- *JDI* and *HotSwap*
  - Using classloader patching
  - Requires the **`-Xdebug`** flag (debug mode)
- *JRockit MAPI*'s class redefinition capabilities
  - No classloader patching or **`-Xdebug`** flag
- *JVMTI* (JSR-163) in Java 1.5
  - Will **standardize** class redefinition in Java
  - No classloader patching or **`-Xdebug`** flag

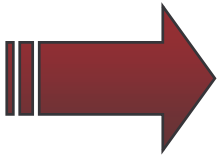
Runtime weaving

- **Problem**: some weaving operations requires schema changes
- Our current solution:
- "Two phase weaving"

  First phase: Classes are first prepared at load time (tiny)

  Second phase: Classes can then be woven at **any point**
- "In-process HotSwap"
  - No *JDWP* and startup JVM needed
- "Multi weaving"
  - Classes can be woven, rewoven or unwoven at **any point**

Runtime weaving



- **Zero overhead** at runtime (for non-woven or unwoven methods)
- **The original** bytecode is guaranteed to run (for non-woven or unwoven methods)

Conclusions

- The JVM wide hook mechanism makes it possible for the framework to work correct in complex class loader hierarchies (application servers etc.)

- We believe that load time and runtime bytecode weaving makes the integration more transparent than a static compilation process

Disadvantages

- Slows down startup time:
  - The startup process does not scale well in certain situations
- This problem will somehow be addressed with:
  - *JVMTI* => No **`-Xdebug`** flag needed
  - *JRockit MAPI* => No **`-Xdebug`** (Java 1.3 and 1.4)
  - Port to *ASM* bytecode library (>700% percent faster than *BCEL/Javassist*)
  - Optimizations (like parsing the constant pool to allow early filtering of classes)
  - Native JVM support

Tool support

Decisions / Goals	JLS compatibility	XML and Attribute definition	Load time and Runtime weaving	AspectJ's join point/ pointcut model	Dynamic runtime model	Runtime compiler (JIT)
Usability & Ease of adoption	X	X	X	X		
Integration	X	X	X			
Tool support	X	X				
Expressiveness & Orthogonality				X		
Performance vs. Dynamicity			X		X	X
Enterprise application support	X	X	X		X	X

Tool support

- *AspectWerkz* is currently lacking good tool support apart from:
  - Plugin for the Maven build system
  - Support for debugging aspects within an IDE
  - *JUnit* (unit testing framework) extension JUnit.org
- The standardization of attributes in JSR-175 will bring many tools for working with metadata
- We believe that good tool support will become a crucial factor for mass adoption

Expressiveness and Orthogonality

Decisions / Goals	JLS compatibility	XML and Attribute definition	Load time and Runtime weaving	AspectJ's join point/ pointcut model	Dynamic runtime model	Runtime compiler (JIT)
Usability & Ease of adoption	X	X	X	X		
Integration	X	X	X			
Tool support	X	X				
Expressiveness & Orthogonality				X		
Performance vs. Dynamicity			X		X	X
Enterprise application support	X	X	X		X	X

Join point and pointcut model

- Regular expression based pointcut language, largely inspired by *AspectJ*
- Join point model largely inspired by *AspectJ*
- Supported join points:
  - method (static and member) and constructor execution
  - method (static and member) and constructor call
  - field (static and member) modification and access
  - catch clauses
  - cflow
- Supports pointcut composition
- Supported advice: *around*, *before* and *after*

Orthogonality

- Orthogonality is:
  - crucial for the model to scale (with the needs of the application)
  - needed for the model to be intuitive and predictable
- In the latest version we have reached a high level of orthogonality:
  - All advice types works with all pointcut designators
  - Pointcut composition allows combinations of all types of pointcuts

Definition constructs

- Definition constructs in *AspectJ:*
  - join points
  - pointcuts
  - advice implementation
- *AspectWerkz* introduces a new construct
  - **the binding** from *pointcuts* to *advice implementation*
- Allows *AspectWerkz* to have named advice
  - Useful in dynamic AOP as a handle to the advice

Current limitations

- Does not support all the dynamic and static pointcut designators as *AspectJ*
- We currently do not support:
  - cflowbelow (in development)
  - within (in development)
  - args (planned)
  - target (planned)
  - static initialization (planned)
  - pre initialization
  - if

Performance vs. Dynamicity

Decisions / Goals	JLS compatibility	XML and Attribute definition	Load time and Runtime weaving	AspectJ's join point/ pointcut model	Dynamic runtime model	Runtime compiler (JIT)
Usability & Ease of adoption	X	X	X	X		
Integration	X	X	X			
Tool support	X	X				
Expressiveness & Orthogonality				X		
Performance vs. Dynamicity			X		X	X
Enterprise application support	X	X	X		X	X

Design decisions

- Dynamic runtime model
  - Allows redefinition of the system at runtime
  - Introduces a layer of indirection, loose coupling
  - Makes use of delegation and reflection
- Runtime (Just-In-Time) Compiler
  - Similar to JIT compilers in modern JVMs
  - Detects advice chains that are often executed and compiles a custom class on the fly that invokes the advice chain and the target join point statically
  - Aware of redefinitions of the runtime model
- Makes heavy use of caching and lazy loading to improve runtime performance

Advantages

- The dynamic runtime model allows
  - addition of advice
  - removal of advice
  - reordering of advice
  - swapping of introduced implementations

  at runtime with almost zero overhead

- The dynamic compiler allows us to have the “best of both worlds”
  - Performance closer to a statically compiled approach
  - The advantage of a dynamic runtime model

Simple benchmark

- The overhead of five *around* advice applied to a *method call* join point
  - AspectJ 1.1.1 — 0.000097 ms/call
  - AspectWerkz 0.10 RC1 — 0.000163 ms/call
  - JBoss AOP 1.0Beta — 0.000263 ms/call

- Configuration:
  - Hardware: Intel Pentium 4 Mobile 1.6 MHz
  - JVM: HotSpot 1.4.2_01
  - OS: Windows XP

Limitations

- Hard to match the speed of statically compiled systems (*AspectJ* etc.) since:
  - We have to keep track of join point state:
    - Need to handle the state management
    - Need to know if a join point is redefined and has become “dirty” (and the JIT compiled class is “stale”)
    - Need a level of indirection
- Can not add new pointcuts to the system without:
  - Reloading of the classes
  - Inserting traps at **all** potential points in the system

Runtime weaving is needed

Enterprise application support

Decisions / Goals	JLS compatibility	XML and Attribute definition	Load time and Runtime weaving	AspectJ's join point/ pointcut model	Dynamic runtime model	Runtime compiler (JIT)
Usability & Ease of adoption	X	X	X	X		
Integration	X	X	X			
Tool support	X	X				
Expressiveness & Orthogonality				X		
Performance vs. Dynamicity			X		X	X
Enterprise application support	**X**	**X**	**X**		**X**	**X**

Requirements for Enterprise AOP

- **Security**
- **Isolation**
- **Visibility**
- **Runtime management**
- **Deployment modules**

- Needs to be handled both:
  - **Horizontally** – within one single class loader
  - **Vertically** – following a class loader hierarchy

Aspect Container

Aspect Container

- We have a problem when **multiple** dynamic aspect systems are deployed **in the same** JVM
- Need a way to allow them to co-exist and interact in a well-defined, organized, layered way
  - A class is transformed according to the definitions of all aspect systems that is visible from its defining class loader
  - Security, visibility and isolation needs to follow the same scheme
- To be implemented

Outline

- **I AspectWerkz overview**
  - what is AspectWerkz?
  - how does it compare with AspectJ?
- **II Design goals and decisions**
  - what are the key issues for commercial AOP?
  - how does AspectWerkz solve them?
- **III Conclusion**
  - future, links, questions

Future work

- Native JVM support for AOP
  - JRockit JVM 1.5
- Java 1.5 support for generics and attributes
- Weaving based on JVMTI
- Metadata driven AOP
- Tool support
- Aspect Container

Where can I find more information?

- Documentation
  - http://aspectwerkz.codehaus.org/
- Technical papers
  - http://codehaus.org/~jboner/papers/aosd2004_aspectwerkz.pdf
  - http://aspectwerkz.codehaus.org/downloads/papers/aosd2004-daw-aspectwerkz.pdf
- Articles
  - http://www.onjava.com/pub/a/onjava/2004/01/14/aop.html
- Weblog articles
  - http://blogs.codehaus.org/people/jboner/
  - http://blogs.codehaus.org/people/avasseur/
  - http://blogs.codehaus.org/projects/aspectwerkz/

Questions?

Thanks for listening

Example of use-cases

- Very useful in enterprise application environments
- Weave third party libraries without having post process all the jars
  - For example when advising on an interface, e.g.

```
java.sql.PreparedStatement+.execute()
```

- Sometimes the decision to apply an aspect can not be taken at build time but at:
  - Deployment time, perhaps much later and by another person
  - Runtime (runtime diagnostics, profiling, performance optimizations etc.)

Lessons learned

- Some extra effort needed to deal with
  - differences in application server implementations (class loading schemes etc.)
  - differences in JVM implementations (IBM JRE)
- We had problems with thread safety in the transformation process
- Issues with bugs in bytecode libraries

Loose coupling using delegation