eclipse CON 2005

What's new in AspectJ 5 ?

Andrew Clement

IBM Java Technology

AspectJ Committer

Alexandre Vasseur

Software Engineer

BEA

© 2005 by IBM and BEA; made available under the EPL v1.0 | 3rd March 2005

About the presenters: Andrew Clement

- Software Engineer based at IBM Hursley Park in the UK
- Co-founder of the AJDT project
- Currently a committer on AspectJ and AJDT
- Frequent speaker on AOP
  - EclipseCon, AOSD, Java One, OOPSLA
- Co-author of ‘Eclipse AspectJ’

© 2005 by IBM and BEA; made available under the EPL v1.0

About the presenters: Alexandre Vasseur

- Software Engineer at Java Runtime Products Group, BEA Systems
- Co-founder of the AspectWerkz AOP framework
- Committer on AspectJ 5
- Frequent speaker on AOP
  - AOSD, BEA eWorld, JavaOne, JAOO

© 2005 by IBM and BEA; made available under the EPL v1.0

eclipse CON 2005

Agenda

- In the headlines: AspectJ and AspectWerkz
- Java 5 support in AspectJ
- Plain Java AOP with @AspectJ aspects
- Enhanced load-time weaving
- User experience with AspectJ 5 and AJDT

© 2005 by IBM and BEA; made available under the EPL v1.0

Agenda

- **In the headlines: AspectJ and AspectWerkz**
- Java 5 support in AspectJ
- Plain Java AOP with @AspectJ aspects
- Enhanced load-time weaving
- User experience with AspectJ 5 and AJDT

© 2005 by IBM and BEA; made available under the EPL v1.0

AspectJ 5: AspectJ and AspectWerkz join forces

- Announced January 2005
- Complementary skills and technology
- Growing AOP is more important than competing
  - Tools, Java 5, weaving, aspect libraries
- AspectJ 5 v1.5.0
  - Initial release 2Q05
  - Roadmap to bring more of the AW features into AJ5
- Backed by IBM and BEA, hosted on Eclipse

© 2005 by IBM and BEA; made available under the EPL v1.0

Agenda

- In the headlines: AspectJ and AspectWerkz
- **Java 5 support in AspectJ**
- Plain Java AOP with @AspectJ aspects
- Enhanced load time weaving
- User experience with AspectJ 5 and AJDT

© 2005 by IBM and BEA; made available under the EPL v1.0

Java 5

- Annotations
  - Metadata that can be attached to many of the Java constructs
- Autoboxing
  - Automatic conversion between primitive types and their OO equivalents (e.g. int and Integer)
- Varargs
  - Support for methods that take variable numbers of arguments, remember printf() in C?
- Covariance
  - When overriding methods, you can choose to narrow the return type
- Generics
  - Improves type checking, most useful for Collections
- Enums
  - Allows for a fixed set of values to be defined for a type

© 2005 by IBM and BEA; made available under the EPL v1.0

Annotations: matching

- How to match on annotations
- Should matching on values be supported ?

```
set(@SensitiveData * *)
get((@SensitiveData *) org.xyz..*.*)
execution(@Oneway * *.*(..))
within(@Secure *)
handler(!@Catastrophic *)
staticinitialization(@Persistent *)
call(* *.*(@Immutable *,..))
```


© 2005 by IBM and BEA; made available under the EPL v1.0

Annotations: runtime type, context exposure

- Runtime type matching
- Variations on `this, target, args`

```
@this(@Foo)
@target(@Foo)
@args(@Foo,*,@Goo)
```

- How to expose annotations as context

```
@this, @target, @args, @within
@withincode, @annotation
```

```
pointcut withinCriticalMethod(Critical c) :
         @withincode(c);
```

© 2005 by IBM and BEA; made available under the EPL v1.0

Annotations: matching on values ?

```
execution(@Transaction(TxPolicy.REQUIRED) * *.*(..))
```

- Not in the first release of AspectJ 5
  - needs some thought on the best design and scope of support

- Use of `if` pointcut in AspectJ 5

```
pointcut txReqMethod(Transaction tx) :
    call(* *.*(..))
    && @annotation(tx)
    && if(tx.value() == TxPolicy.REQUIRED);
```


© 2005 by IBM and BEA; made available under the EPL v1.0

Autoboxing: matching

`void doSomething(Integer i)`, a method execution…

`execution(* *(int))`

- Does it match ?
- In AspectJ 5 the answer is **NO**
  - signature patterns match on declared signatures
  - int -> int, Integer -> Integer

`args(int)`

- Does it match ?
- In AspectJ 5 the answer is **YES**
  - an int argument will be autoboxed to Integer
  - and the other way around (unboxing)

© 2005 by IBM and BEA; made available under the EPL v1.0

Autoboxing: context binding

- Does context binding support autoboxing ?

```
pointcut foo(int i) : args(i);

before(Integer i) : foo(i)  {
    ...
}
```

- In AspectJ 5 the answer is **YES**

© 2005 by IBM and BEA; made available under the EPL v1.0

Varargs

- We allow a vararg in the last parameter position in a signature pattern

```
call(* *(.., String...))
```

- For runtime type matching and context exposure, must use the runtime type

`String[]` in this case

```
call(* *(..,String...)) && args(..,String[])
✓ before(String[] ss) : args(ss)
✗ before(String... ss) : args(ss)
```


© 2005 by IBM and BEA; made available under the EPL v1.0

Covariance

- How do covariant signatures affect join point matching ?
- The signatures of **`B.whoAmI()`** are:

  `B B.whoAmI()`

  `A A.whoAmI()`

`call(A whoAmI())`

- matches

`call(B A.whoAmI())`

- does NOT match

```
class A {
    A whoAmI() {
        return this;
    }
}

class B extends A {
    B whoAmI() {
        return this;
    }
}

B b = new B();
b.whoAmI();
```


© 2005 by IBM and BEA; made available under the EPL v1.0

Generics

- How to match generic signatures at join points
- Pattern wildcards vs generic wildcards (* == ?)
- How to expose generic types as context
- Generics and inter-type declarations
- Generic aspects ?

Lots of possibilities!

- Starting with support for common use cases
- Growing from there…

© 2005 by IBM and BEA; made available under the EPL v1.0

eclipse CON 2005

Matching generic signatures – our initial thoughts…

- `call`, `execution`, `get`, `set` match based on signature
- For each of these signatures, which pointcuts will match?

```
void foo(List<Number> ns) {...}
✓ execution(* foo(List<Number>))
✓ execution(* foo(List<*>))
✘ execution(* foo(List<?>))
✓ execution(* foo(List<Object+>))

void goo(List<? extends Number> ns) {...}
✘ call(* goo(List<?>))
✓ call(* goo(List<? extends Number>))
✘ call(* goo(List<Number+>))
```


© 2005 by IBM and BEA; made available under the EPL v1.0

Other

- Enums
  - Could have enum as the target for ITDs …
  - Not in first version of AspectJ 5

- Annotations
  - Declare annotations

```
declare @field: int x: @SimpleAnnotation
```

  - Definetly in first version of AspectJ 5

© 2005 by IBM and BEA; made available under the EPL v1.0

Agenda

- In the headlines: AspectJ and AspectWerkz
- Java 5 support in AspectJ
- **Plain Java AOP with @AspectJ aspects**
- Enhanced load-time weaving
- User experience with AspectJ 5 and AJDT

© 2005 by IBM and BEA; made available under the EPL v1.0

The @AspectJ aspects

- Java 5 annotations enable plain Java aspects (including ITDs)
  - A style supported by AspectWerkz

```
@Aspect public class MyAspect { }

org.aspectj.lang.annotation.*
    @Aspect
    @Pointcut
    @Before, @Around, @After, ...
    @DeclareParents, ...
```

- Design goals
  - Support compilation of the largest subset of AspectJ applications possible using a standard Java 5 compiler
  - Be able to mix styles in the same application

© 2005 by IBM and BEA; made available under the EPL v1.0

The @AspectJ aspects

- AspectJ has
  - **ONE** language
  - **ONE** semantics
  - **ONE** weaver
- With two different development styles
  - Code Style

```
public aspect MyAspect { }
```

  - Annotation Style

```
@Aspect public class MyAspect { }
```


© 2005 by IBM and BEA; made available under the EPL v1.0

An @AspectJ aspect

```
@Aspect // defaults to singleton
public class NoOpAspect {

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


    @Around("asyncMethods()")
    public Object noop(ProceedingJoinPoint jp)
    throws Throwable {
        // proceed() is a method of "JoinPoint"
        return jp.proceed();
    }

}
```

Aspect is ***@Aspect*** class

@Pointcut defines pointcuts

@Around annotated methods are around advice

JoinPoint.proceed() respects Java type checking

© 2005 by IBM and BEA; made available under the EPL v1.0

eclipse CON 2005

Parameter binding

```
package pack;
class Foo { ... }

@Before("call(* dup(int)) && this(foo) && args(i)")
public void callFromFoo(Foo foo, int i) {
    println("call from Foo: " + foo);
    println("arg = " + i);
}
```

- But…

```
@Before("call(* dup(int)) && this(pack.Foo) && args(i)")
public void callFromFooWithRTTypeCheck(int i) { ... }
```

- While code style would use `this(Foo) + import pack.Foo;`

© 2005 by IBM and BEA; made available under the EPL v1.0

thisJoinPoint

```
import org.aspectj.lang.JoinPoint;

@Before("call(* dup(int)) && this(foo)")
public void callFromFoo(JoinPoint thisJoinPoint, Foo foo) {
    println("call from Foo: " + foo);
    println("at " + thisJoinPoint);
}
```

- While with code style **thisJoinPoint** is implicitly available

```
before(Foo foo) : call(* dup(int)) && this(foo) {
    println("call from Foo: " + foo);
    println("at " + thisJoinPoint);
}
```


© 2005 by IBM and BEA; made available under the EPL v1.0

Around advice and proceed()

- `proceed()` is not possible in an @AspectJ advice body

```
@Around("call(* dup(..))")
public Object doNothing() {
  return proceed();  // this line won't compile
}
```

- To address this, we introduce `org.aspectj.lang.ProceedingJoinPoint`

```
@Around("call(* dup(..))")
public Object doNothing(ProceedingJoinPoint jp) {
  return jp.proceed();
}
```


© 2005 by IBM and BEA; made available under the EPL v1.0

Inter-type declaration

- `declare parents ... implements` follows a **mixin** strategy

```
@Aspect public class MoodIndicator {

    public static interface Moody {
        Mood getMood();
    }

    @DeclareParents("org.xyz..*")
    static class MoodyImpl implements Moody {
        private Mood m_mood;
        public Mood getMood() { return m_mood; }
    }

    ...
}
```


© 2005 by IBM and BEA; made available under the EPL v1.0

@AspectJ limitations

- `declare parents ... extends`
- ITDs on classes
- `declare soft`
- These features don't make sense with a standard Java 5 compiler

© 2005 by IBM and BEA; made available under the EPL v1.0

Agenda

- In the headlines: AspectJ and AspectWerkz
- Java 5 support in AspectJ
- Plain Java AOP with @AspectJ aspects
- **Enhanced load-time weaving**
- User experience with AspectJ 5 and AJDT

© 2005 by IBM and BEA; made available under the EPL v1.0

Load-time weaving in AspectJ 5

- We introduce a deployment descriptor

  `META-INF/aop.xml`

  `META-INF/aop.properties` (J2ME …)

- Weaving is `ClassLoader` aware
  - Eligible classes are advised by aspects they are visible to
  - One or more deployment descriptor(s)
- Enabled through
  - Java 5 agents (JVMTI), JRockit agents (Java 1.3)
  - Command line script
  - Specific integration
- Similar to AspectWerkz schemes

© 2005 by IBM and BEA; made available under the EPL v1.0

Load-time weaving

- Controls
  - Aspects to use
  - Weaver configuration
  - Elligible classes

```
<aspectj>
  <aspects>
    <!-- <aspect name="com.ltw.MyDebugAspect"/> -->
    <aspect name="com.ltw.Aspect"/>
  </aspect>
  <weaver options="-XlazyTjp">
    <include within="com.webapp..*"/>
  </weaver>
</aspectj>
```


© 2005 by IBM and BEA; made available under the EPL v1.0

Deployment-time pointcut definition

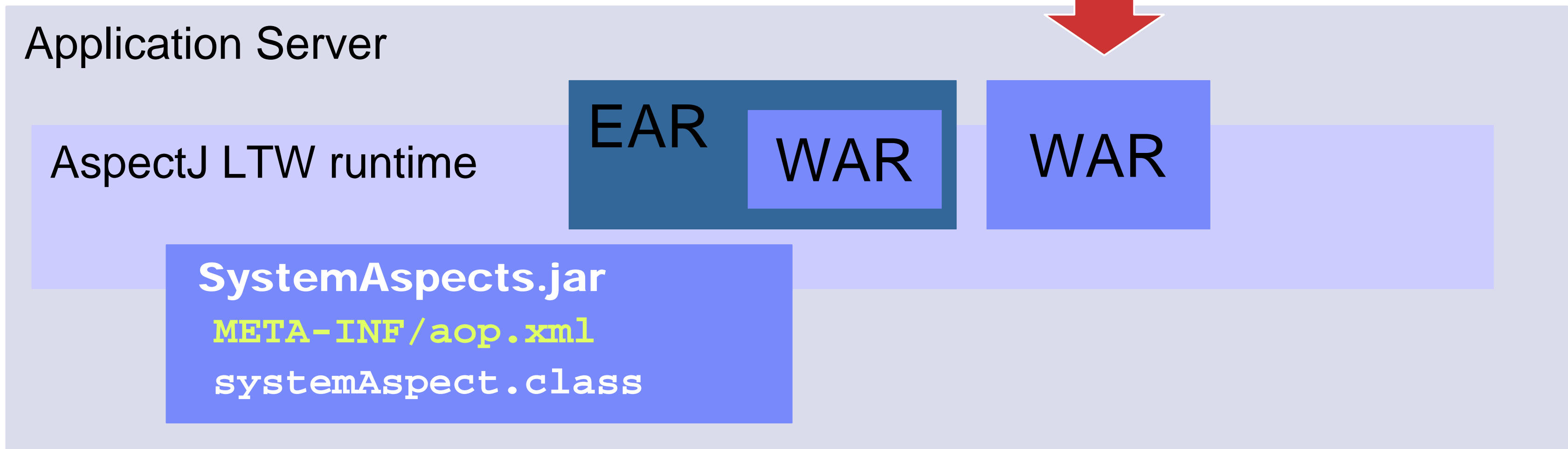
```
abstract aspect com.generic.AbstractLogging {
    abstract pointcut tracingScope();
    ...
}

<aspectj>
  <aspects>
    <concrete-aspect
        name="com.ltw.DeploymentTimeAspect"
        extends="com.generic.AbstractLogging">
      <pointcut name="tracingScope"
        expression="within(com.biz.*)"/>
    </concrete-aspect>
  </aspect>
  <weaver options="-XlazyTjp"/>
</aspectj>
```


© 2005 by IBM and BEA; made available under the EPL v1.0

Load-time weaving use case

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



© 2005 by IBM and BEA; made available under the EPL v1.0

Agenda

- In the headlines: AspectJ and AspectWerkz
- Java 5 support in AspectJ
- Plain Java AOP with @AspectJ aspects
- Enhanced load-time weaving
- **User experience with AspectJ 5 and AJDT**

© 2005 by IBM and BEA; made available under the EPL v1.0

AJDT

- Simply understands either style (code or annotation)
- Integrates the enhanced LTW support

- Plus other benefits unrelated to AspectJ 1.5.0
  - Visualizer enhancements
    - deow, general markers
  - Incremental compilation & structure model
  - Eager parsing & model update
  - Cross-reference view

© 2005 by IBM and BEA; made available under the EPL v1.0

Demo

© 2005 by IBM and BEA; made available under the EPL v1.0

Summary

- Support for new Java 5 features
  - Some decisions to be taken
- Improved performance
- Annotation style development
  - Brought in by the AspectWerkz team
- Enhanced Load Time Weaving support
  - Much more flexible deployment options
  - Brought in by the AspectWerkz team
- AJDT will offer a consistent experience for either style of development

© 2005 by IBM and BEA; made available under the EPL v1.0

The question is WHEN?

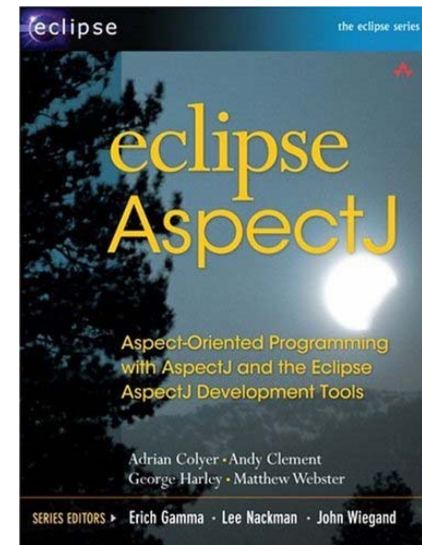
- `1.5.0M1` released December 10th
  - Included binary weaving of Java 5 compiled code
- Current dev stream
  - Compilation of Java 5 features and full support for annotations, autoboxing, varargs, covariance
    - For release as `1.5.0M2`
  - Work on enhanced LTW and annotation style going on in a branch
  - Generics work to be done, for release as `1.5.0M3`
- Possibly a `1.5.0M4` then release candidates and a final release
  - 2Q05

- AJDT support available for the new features shortly after each release

© 2005 by IBM and BEA; made available under the EPL v1.0

Useful resources

- More info
  - http://eclipse.org/aspectj
  - http://aspectwerkz.codehaus.org
  - For new language features, see the **AspectJ developers notebook** linked from the AspectJ homepage
  - Buy the book ☺



Andy Clement
`clemas@uk.ibm.com`

Alexandre Vasseur
`avasseur@bea.com`

© 2005 by IBM and BEA; made available under the EPL v1.0