AspectWerkz

for Dynamic Aspect-Oriented Programming

Jonas Bonér
Senior Software Engineer - BEA Systems

Alexandre Vasseur
Senior Consultant - BEA Systems

Agenda

- What will you learn?
- AOP overview
- AOP constructs in AspectWerkz
- Aspect development and deployment
- [Break]
- Weaving and integration scenarios
- Dynamic AOP
- Enterprise application samples

Agenda

- **What will you learn?**
- AOP overview
- AOP constructs in AspectWerkz
- Aspect development and deployment
- [Break]
- Weaving and integration scenarios
- Dynamic AOP
- Enterprise application samples

What will you learn?

- You will learn how:
  - *AspectWerkz* addresses AOP
  - to write Aspects with *AspectWerkz*
  - to package and deploy Aspects
  - to use the different weaving and integration schemes
  - to use the dynamic features in *AspectWerkz*
  - to build real world enterprise applications with AOP using *AspectWerkz*

- What will be *AspectWerkz* in 2005?

What is AspectWerkz?

- Dynamic AOP framework for Java / XML
- Open source, founded Q4 2002
- Sponsored by bea™
- Tailored for dynamic AOP and real world integration

- JLS compatible (pure Java)
- Definition syntax in XML and/or attributes
- Load time, runtime and static weaving
- Allows rearrangement of Aspects at runtime

What can I use it for?

Good candidates for AOP in J2EE environments:

- role based security
- declarative transaction demarcation
- transparent persistence
- lazy loading
- eager loading (loading policies)
- asynchronous calls
- synchronization
- virtual mock objects for unit testing
- performance optimization
- design patterns
- business rules
- pure mixin based implementations

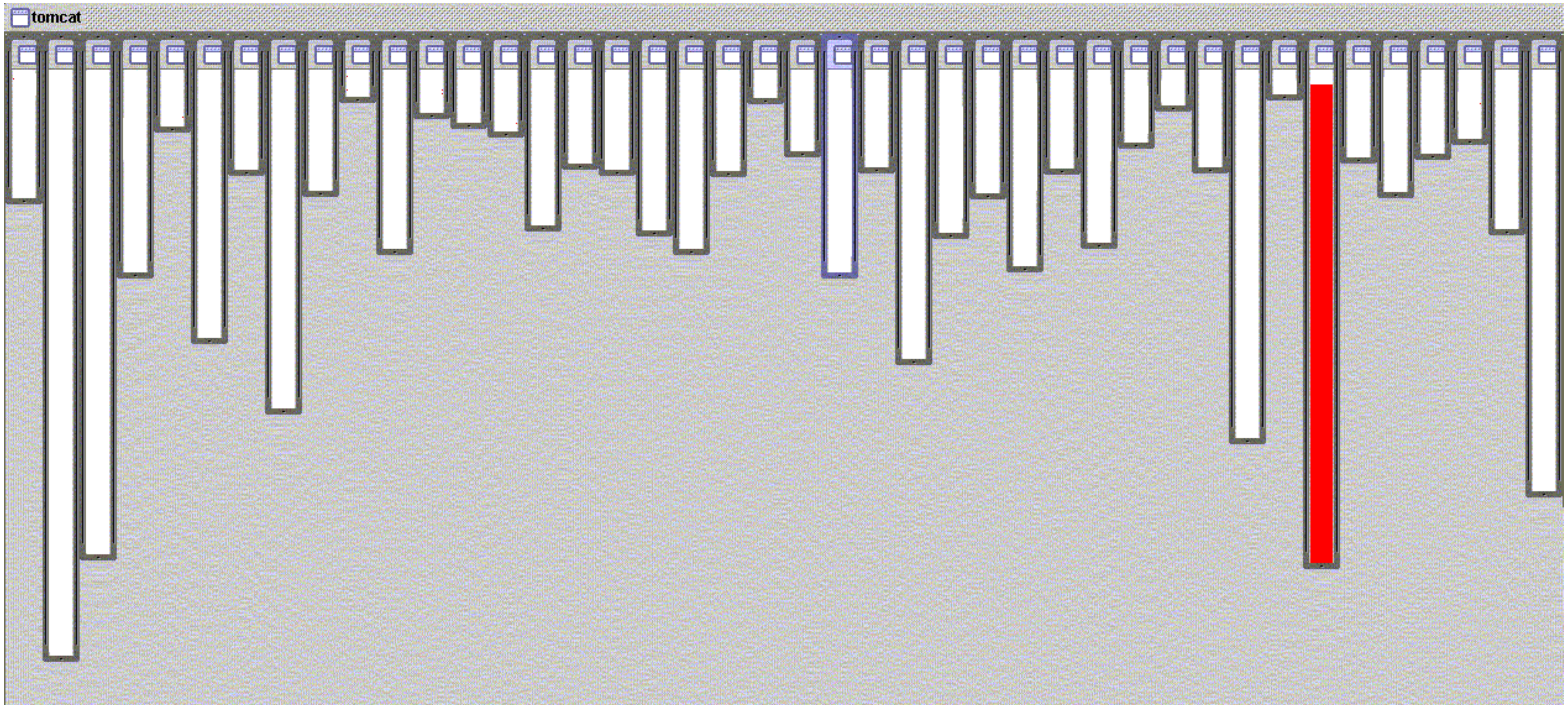
Agenda

- What will you learn?
- **AOP overview**
- AOP constructs in AspectWerkz
- Aspect development and deployment
- [Break]
- Weaving and integration scenarios
- Dynamic AOP
- Enterprise application samples

Good modularity

From AspectJ Workshop
Copyright Xerox Corporation

XML parsing

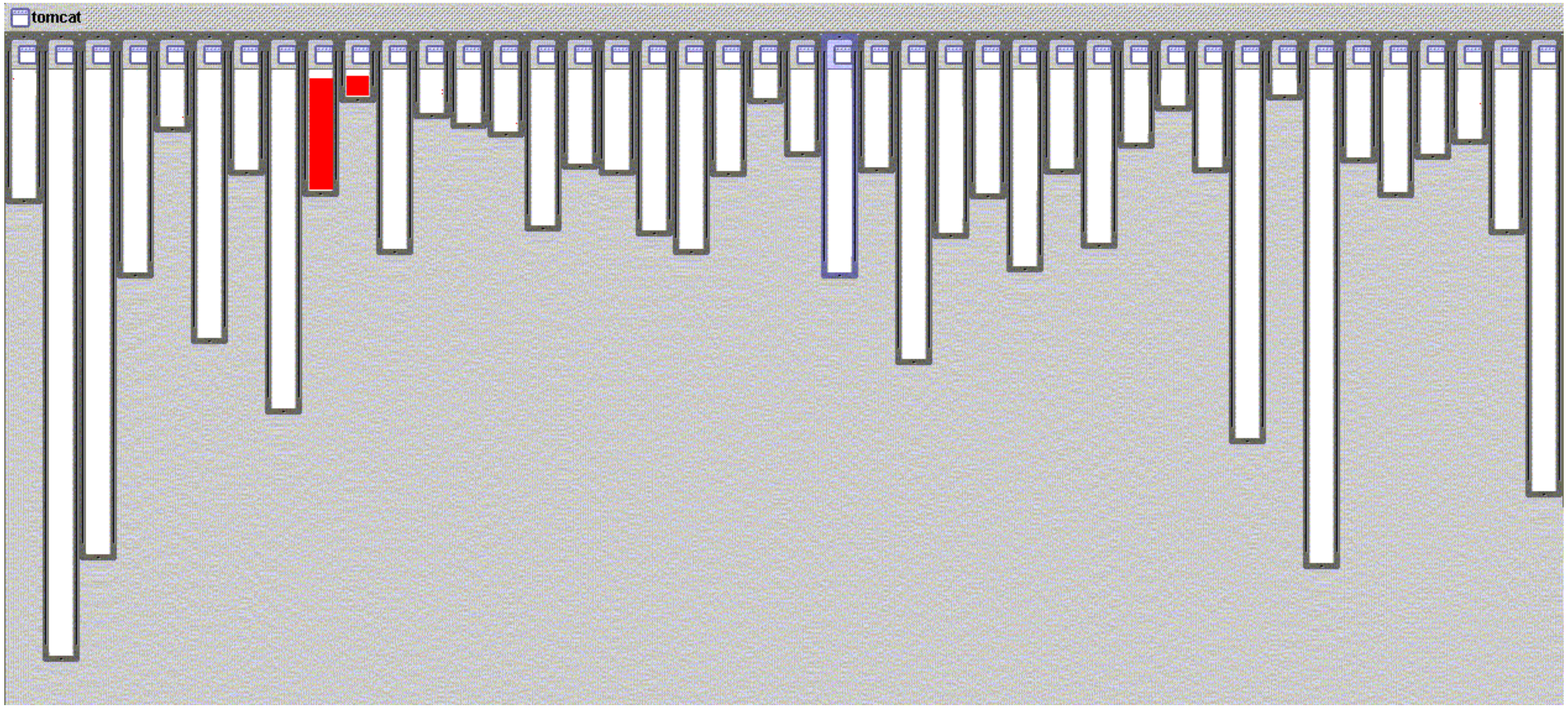


- XML parsing in org.apache.tomcat
  - red shows relevant lines of code
  - nicely fits in one box

Good modularity

From AspectJ Workshop
Copyright Xerox Corporation

URL pattern matching

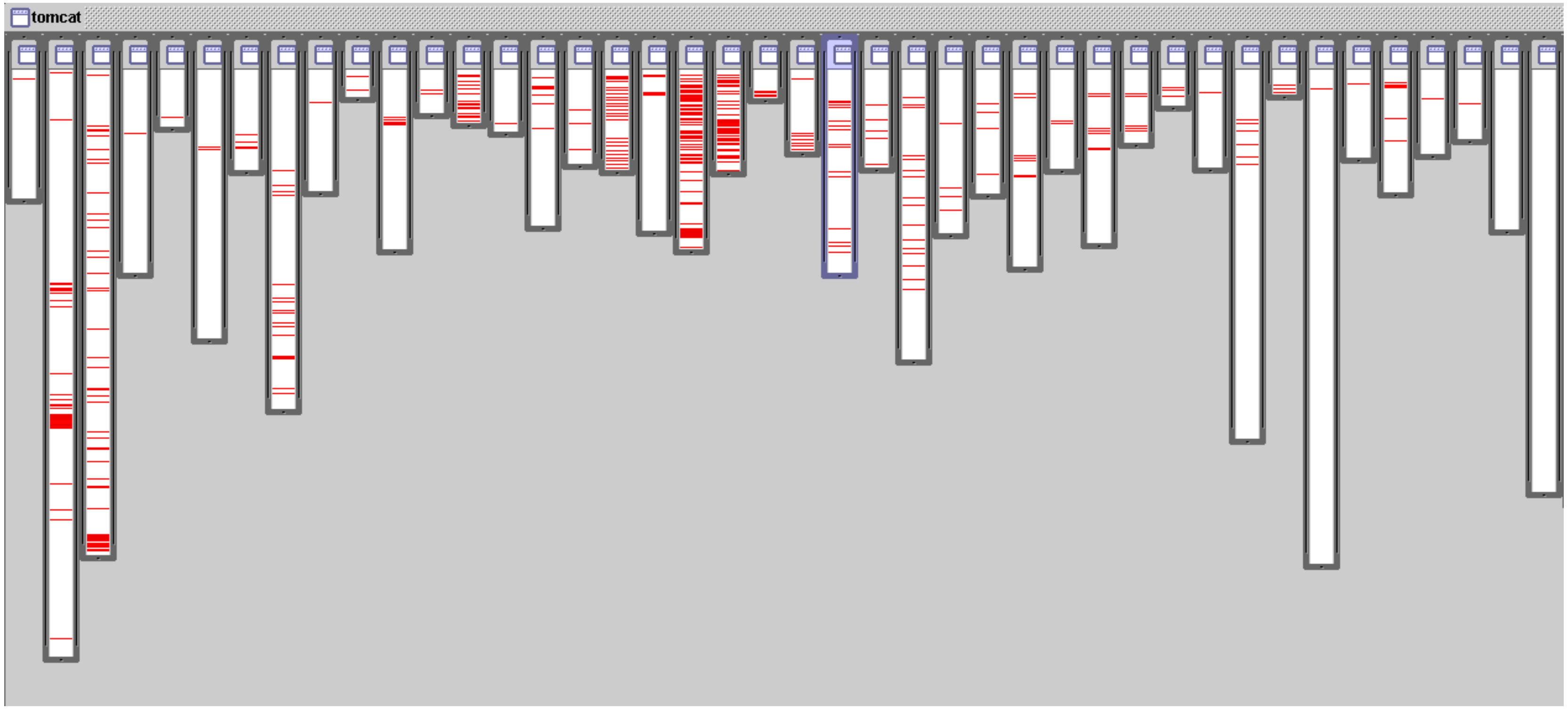


- URL pattern matching in org.apache.tomcat
  - red shows relevant lines of code
  - nicely fits in two boxes (using inheritance)

Problems like...

From AspectJ Workshop
Copyright Xerox Corporation

logging is not modularized



- logging in org.apache.tomcat
  - red shows lines of code that handle logging
  - not in just one place
  - not even in a small number of places

Cross-cutting concerns

- Symptoms:
  - **Code tangling**: when a module or code section is managing several concerns simultaneously
  - **Code scattering**: when a concern is spread over many modules and is not well localized and modularized

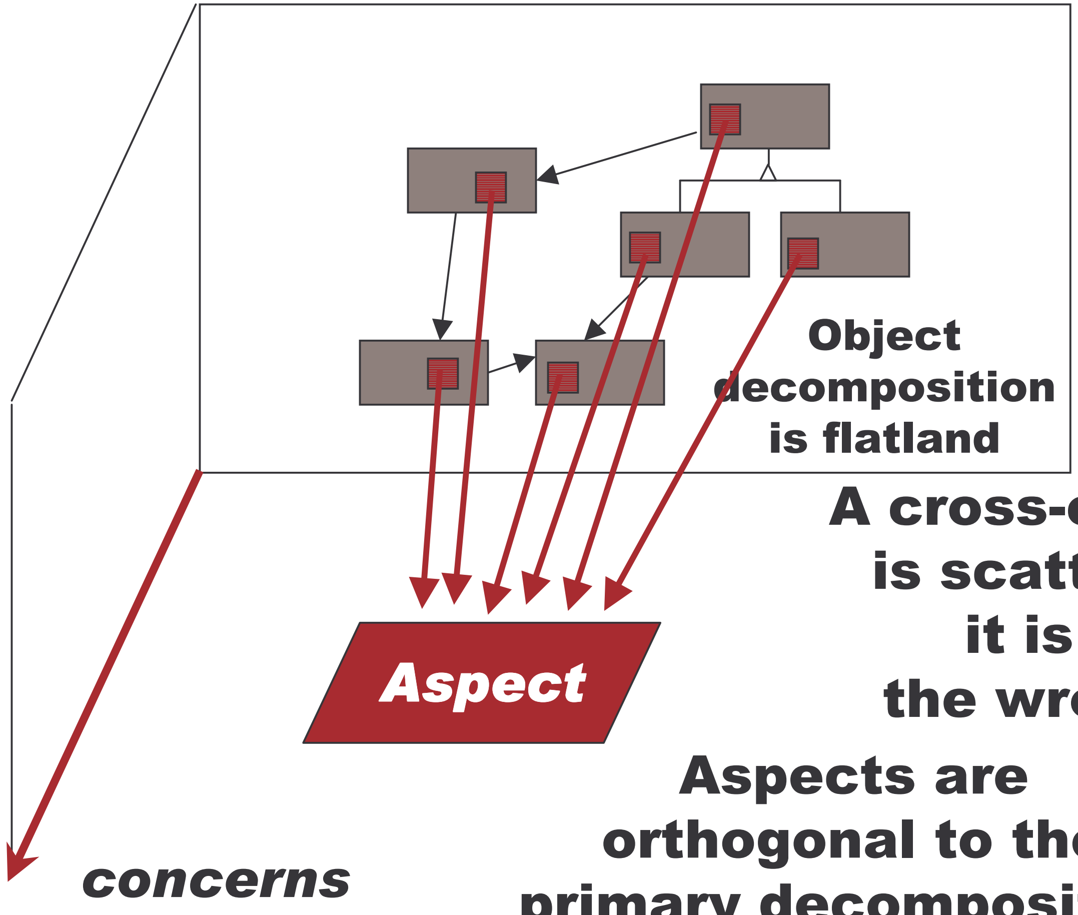
- Makes the software harder to:
  - Write
  - Understand
  - Reuse
  - Maintain

Enter Aspect-Oriented Programming

- AOP enables Separation Of Concerns
- Allows the concerns to be implemented in a modular and well-localized way
- Captures the concerns in a modular unit: the Aspect
- Should be seen as an addition to (and **not** a replacement for) OOP
- The 15% solution (according to Gregor Kiczales)

Adds a new dimension to software dev.

From presentation by Frank Sauer
Copyright Technical Resource Connection Inc.



Object decomposition is flatland

Aspect

concerns

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

Aspects are orthogonal to the primary decomposition

Core elements in AOP

- **Means to:**

1. Define well-defined points in the program flow
   - **Join points**

2. Pick out these points
   - **Pointcuts**

3. Influence the behavior at these points
   - **Advice (Introductions)**

4. Weave everything together into a functional system
   - **Weaver**

Section review

- Some concerns cannot be solved gracefully with OOP
- AOP enables separation of concerns by capturing them in Aspects
- AOP complements OOP
- AOP core vocabulary
  - Join points
  - Pointcuts
  - Advice and Introductions
  - Aspects
  - Weaver

Agenda

- What will you learn?
- AOP overview
- **AOP constructs in AspectWerkz**
- Aspect development and deployment
- [Break]
- Weaving and integration scenarios
- Dynamic AOP
- Enteprise application samples

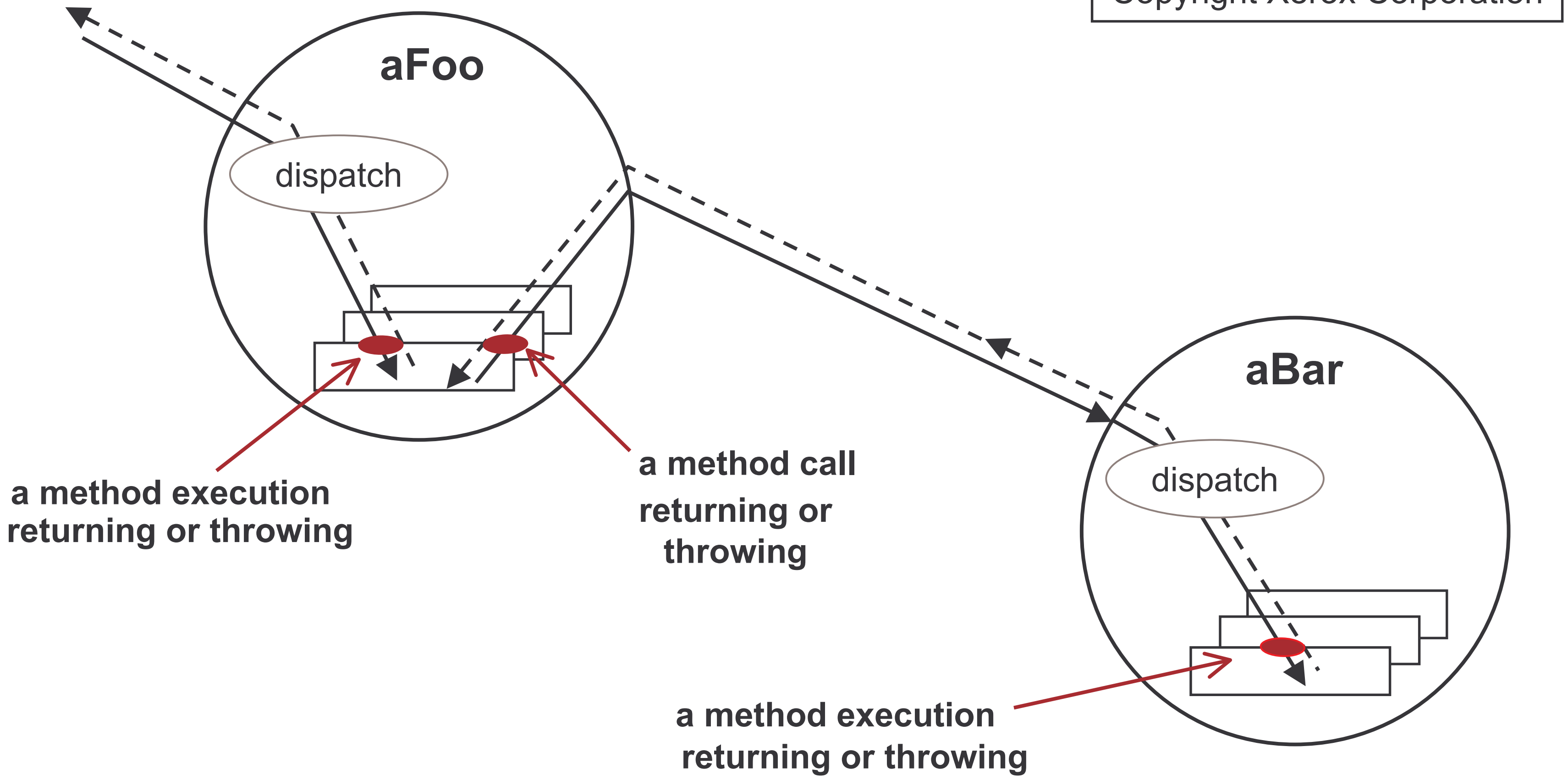
Section objectives

- You will learn
  - Pointcut types supported in *AspectWerkz*
  - How to define pointcuts using patterns
  - How to use pointcut composition to meet complex application requirements
  - How to write Before / After / Around advice
  - How advice interact at the join point
  - How to write introductions
- Write an Aspect
- Reuse Aspects

Join points

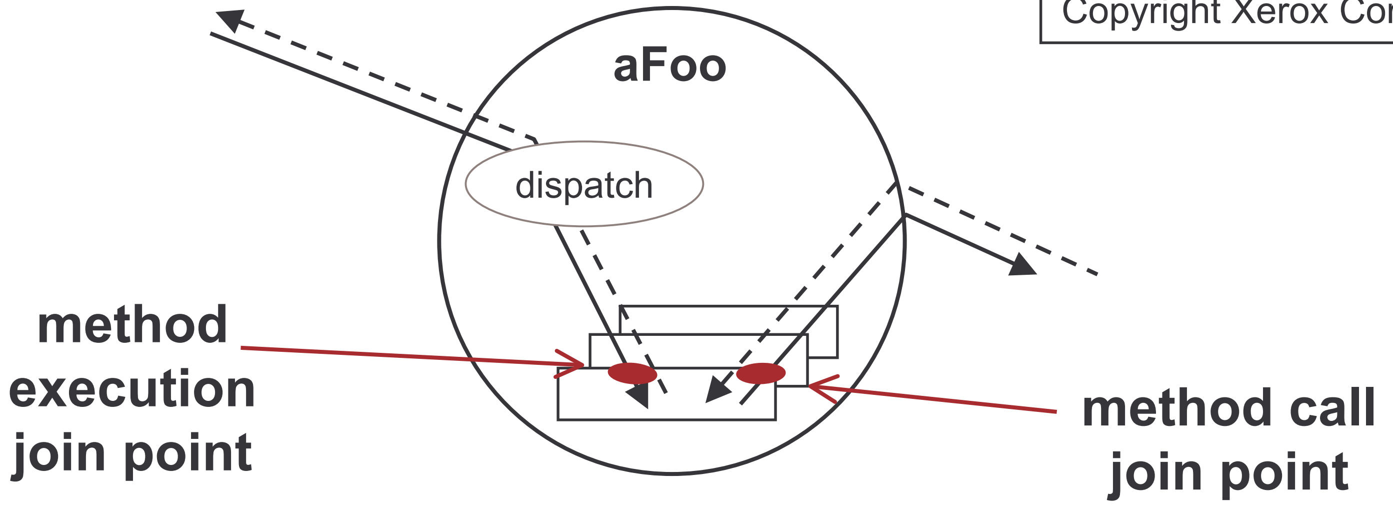
- Well-defined points in the program flow

From AspectJ Workshop
Copyright Xerox Corporation


aFoo
dispatch
aBar
dispatch
a method execution returning or throwing
a method call returning or throwing
a method execution returning or throwing

Join points

From AspectJ Workshop
Copyright Xerox Corporation



- Several kinds of join points
  - method & constructor call
  - method & constructor execution
  - field get & set
  - exception handler execution
  - control flow

Pointcuts

- Construct that picks out join points

```
execution(void Foo.addBar(Bar))
```

```
...
public void addBar(Bar bar) {
     // do stuff
}
...
```

Supported pointcut types

- Execution - picks out join points defining *method execution* (callee side)

```
execution(void Foo.addBar(Bar))
```

- Call - picks out join points defining *method call* (caller side)

```
call(Caller->void Foo.addBar(Bar))
```

- Set - picks out join points defining *field modification*
- Get - picks out join points defining *field access*

```
set(int Foo.barTotal)
```

- Cflow - picks out join points defining a *control flow*

```
cflow(int Foo.addBar(Bar))
```

- Handler - picks out join points defining where an exception is catched in a catch clause

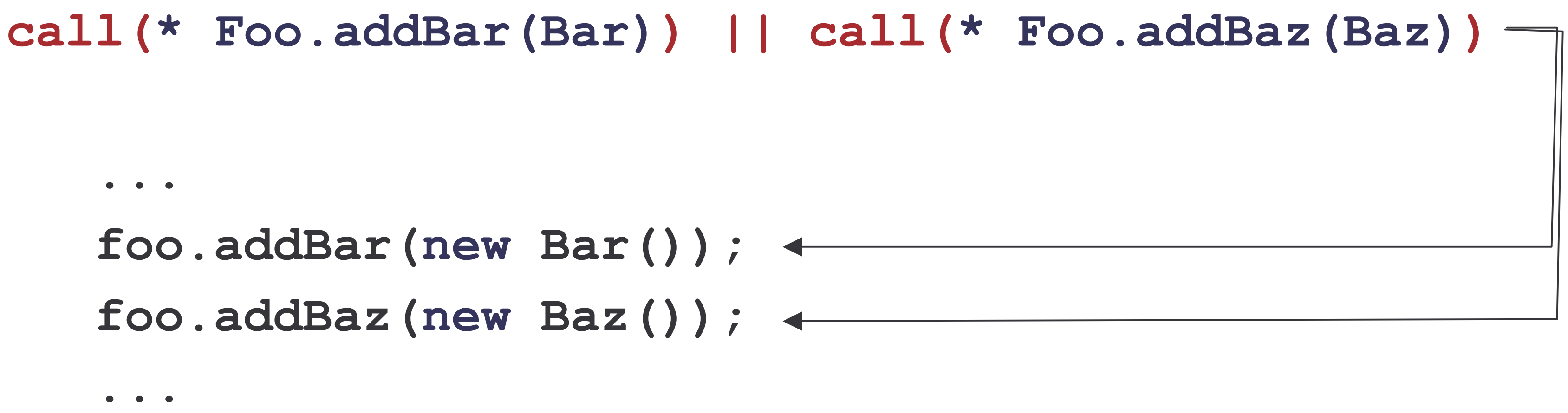
```
handler(java.lang.Exception+)
```

Wildcard matching

- Supports wildcards
  - `*` – matches exactly one type or package (1)
  - `..` – matches zero to many types or packages (0..N)

- Examples:
  - `* foo.baz.Bar.*(int, ..)`
  - `int foo..*.*(..)`
  - `String m_*`

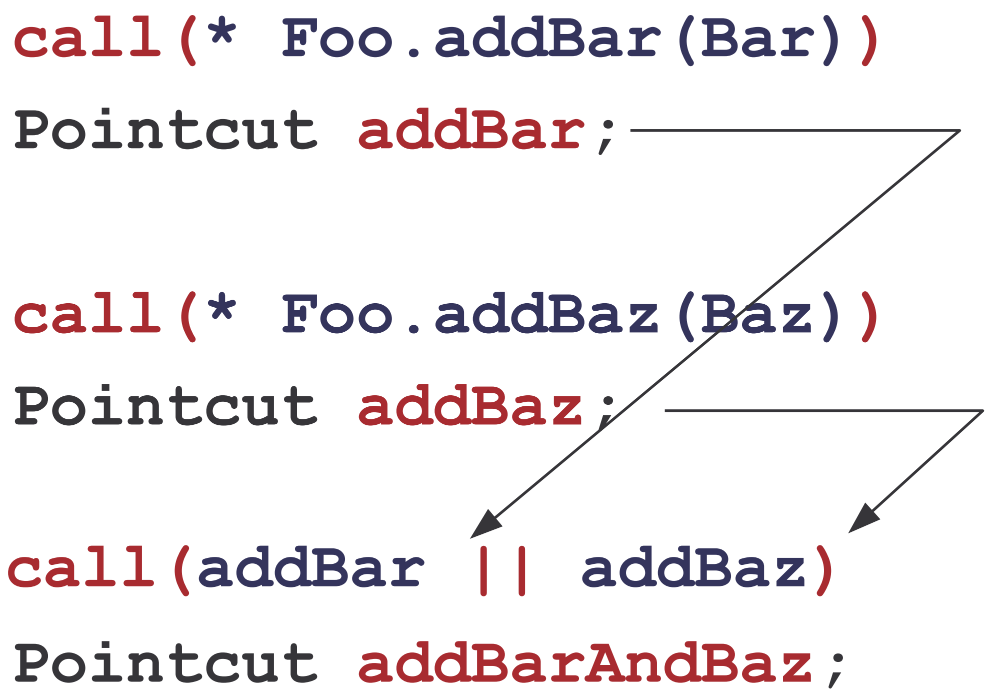
Pointcut composition



```
call(* Foo.addBar(Bar)) || call(* Foo.addBaz(Baz))

    ...
    foo.addBar(new Bar());
    foo.addBaz(new Baz());
    ...
```

- Compose with logical operators:
  - `&&` – logical AND
  - `||` – logical OR
  - `!` – logical NOT

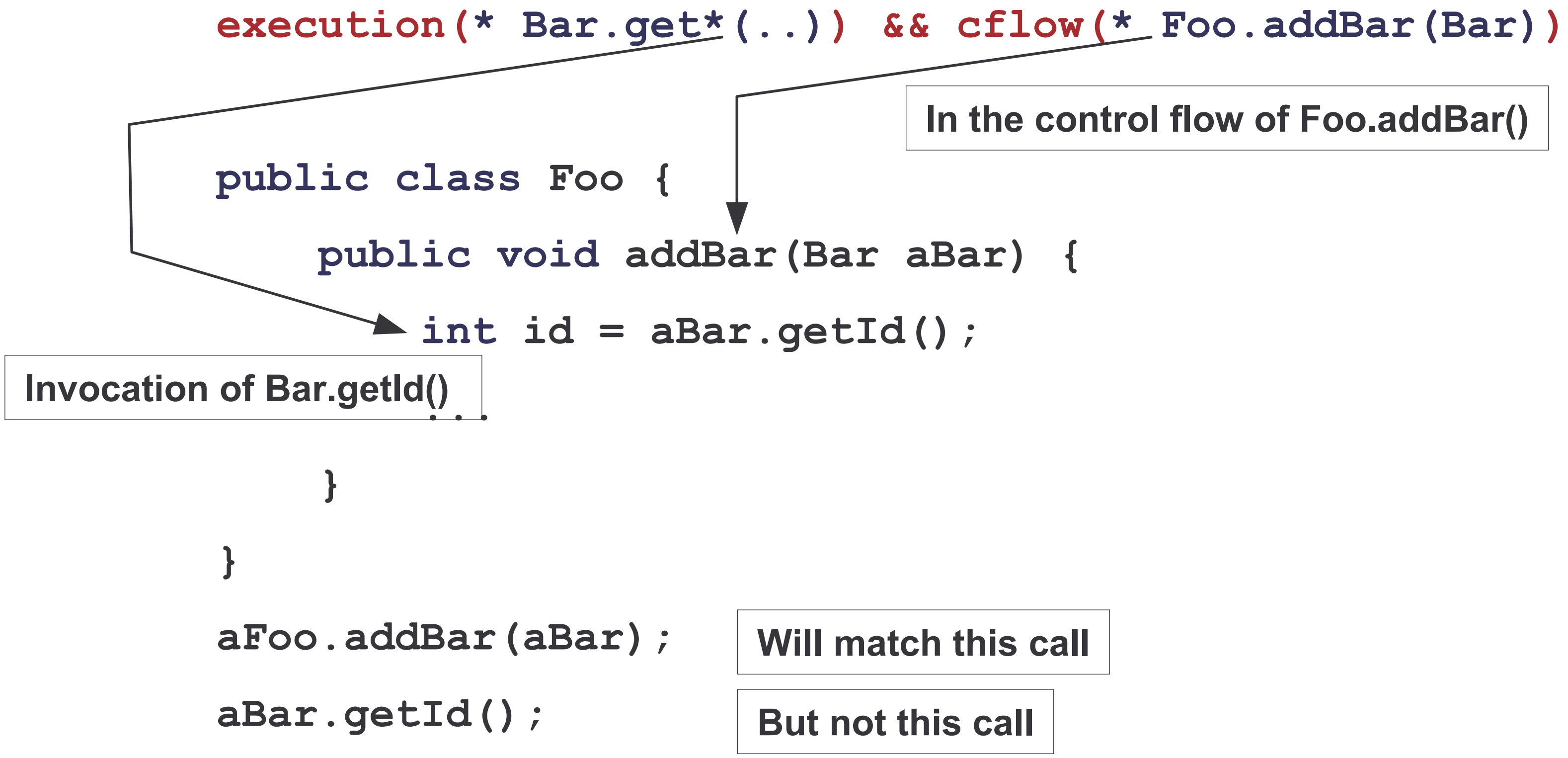
Named pointcuts



```
call(* Foo.addBar(Bar))
Pointcut addBar;

call(* Foo.addBaz(Baz))
Pointcut addBaz;

call(addBar || addBaz)
Pointcut addBarAndBaz;
```

Cflow syntax

- Cflow composition expresses the idea of the stack trace



```
execution(* Bar.get*(..)) && cflow(* Foo.addBar(Bar))

public class Foo {
    public void addBar(Bar aBar) {
        int id = aBar.getId();
        ...
    }
}
aFoo.addBar(aBar);
aBar.getId();
```

In the control flow of Foo.addBar()

Invocation of Bar.getId()

Will match this call

But not this call

Subtype patterns

- Can pick out subtype patterns using the ‘+’ operator
- Allows you to pick out all classes that either:
  - Implements a certain interface or
  - Extends a certain class
- Example:
  - `foo.bar.IntefaceBar+`
  - `foo.bar.SuperClassBaz+`

Advice

- Allows you to influence the behavior at the join points
- Defines what **to do** at the join points

- Three main types of advice:
  - Around: invoked *‘around’* the join point
  - Before: invoked *before* the join point is reached
  - After: invoked *after* the join point has been reached
- Implemented as regular method in Java

Before advice

- Is invoked *before* the join point is reached
- Takes a `JoinPoint` instance as its only parameter
- Example:

```
public void beforeAdvice(JoinPoint joinPoint)
    throws Throwable {
    // do stuff
}
```

After advice

- Is invoked *after* the join point has been reached
- Takes a **`JoinPoint`** instance as its only parameter
- Example:

```
public void afterAdvice(JoinPoint joinPoint)
    throws Throwable {
    // do stuff
}
```

Around advice : The proceed method

- The `JoinPoint` class has a `proceed()` method:

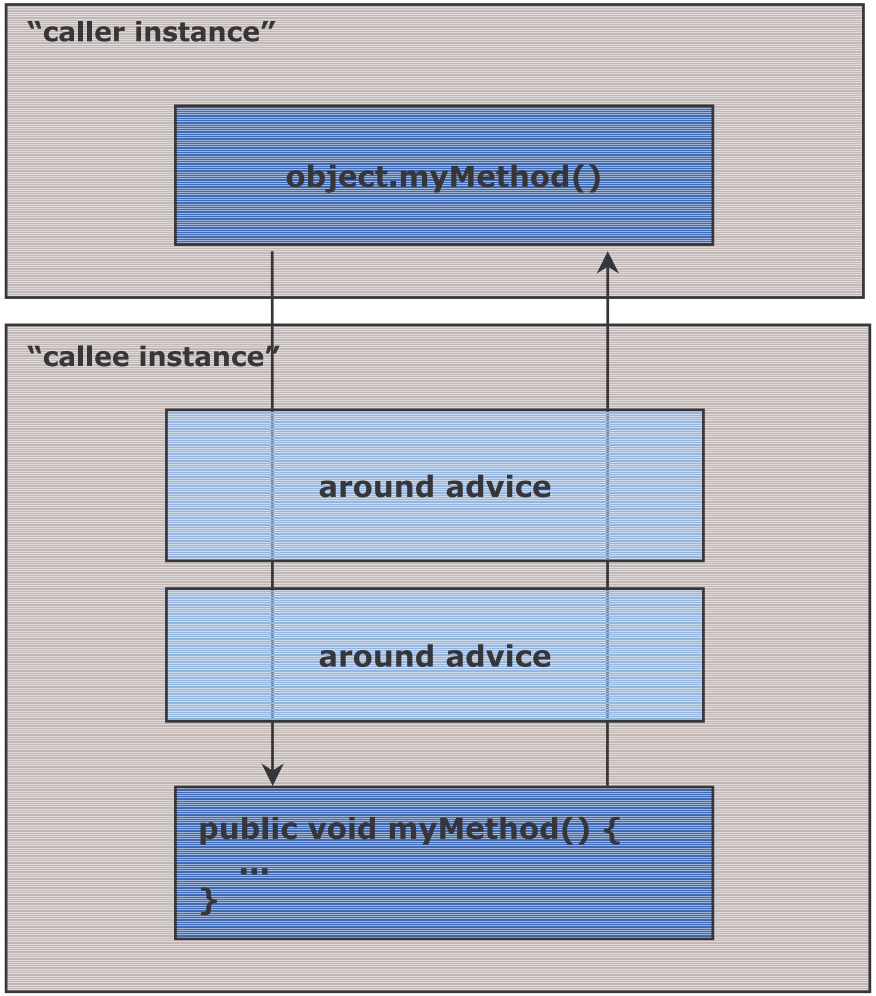
  ```
  Object result = joinPoint.proceed();
  ```

- Only works in *Around advice*
- It either invokes:
  - The next advice in the chain, or
  - The target join point (method, field, catch clause etc.)

- It returns the result from the join point invocation

Around advice

```
public Object aroundAdvice(JoinPoint joinPoint)
    throws Throwable {
    // do stuff
    Object result =
        joinPoint.proceed();
    // do more stuff
    return result;
}
```



JoinPoint instance

- Each advice is passed a `JoinPoint` instance
- Allows introspection possibilities
- RTTI (run-time type information) about a specific join point

- The RTTI is accessed and modified through one of the `Signature` interfaces

Signature interfaces

- The `JoinPoint` class has a `getSignature()` method
- This method returns the `Signature` for the join point that we are currently executing at
- This `Signature` can be casted to a more specific type:
  - `MethodSignature`
  - `FieldSignature`
  - `MemberSignature`
  - Etc.

Signature interfaces

- When executing at a method we can for example retrieve:
  - target instance and class
  - method instance
  - parameter values and types
  - return value and type

- Possible to modify parameters and return value at runtime

Deployment models

- Defines the ‘scope’ or life-cycle of the *AOP constructs*

- Supports four different deployment models:
  - **`perJVM`** - one instance per JVM (singleton)
  - **`perClass`** - one instance per target class
  - **`perInstance`** - one instance per target class instance
  - **`perThread`** - one instance per thread

What have we learned so far?

- Advices are regular Java methods
- The `JoinPoint` class allows to `proceed` in the advice chain or to the target join point
- There is a composition algebra and expression language for pointcuts
- Deployment models can be used to define the life-cycle of AOP constructs

How do we bring it all together?

- How do we specify which advice are bound to which pointcut?
- How do we define the deployment model?
- How do we tell the system which Aspects to use?

The Aspect brings it all together

- The Aspect is the unit of modularity in AOP
- Similar to the *Class* construct in OOP

- The Aspect
  - can have zero or more pointcuts
  - can have zero or more mixins bounded at defined pointcuts
  - can have zero or more advices bounded at defined pointcuts
  - supports abstraction and inheritance

- Implemented as regular class in Java

Example of an Aspect

define the deployment model

define the poincuts

bind the advice to the pointcuts

define the advice

```
/** @Aspect perInstance */
public class LoggingAspect extends Aspect {

    /** @Expression call(* foo.bar.*.*(..)) */
    Pointcut logMethodCall;

    /** @Expression execution(* foo.baz.*.*(..)) */
    Pointcut logMethodExecution;

   /** @Before logMethodCall */
   public void logEntry(JoinPoint joinPoint) { ... }

   /** @After logMethodCall */
   public void logExit(JoinPoint joinPoint) { ... }

   /** @Around logMethodExecution */
   public Object logExecution(JoinPoint joinPoint) { ... }
}
```

Deployment descriptor

- Needed to tell the systems which aspects to deploy
- Example:

```
<aspectwerkz>
    <system id="samples">
        <package name="examples">
            <aspect class="logging.LoggingAspect"/>
            <aspect class="caching.CachingAspect">
                <param name="timeout" value="10"/>
            </aspect>
        </package>
    </system>
</aspectwerkz>
```

Exercise: caching

- Naive implementation of fibonacci
- Many redundant calculations

```
public class Fibonacci {

    public static int fib(int n) {
        if ( n < 2) {
            System.err.println(n + ".");
            return 1;
        } else {
            System.err.print(n + ",");
            return fib(n-1) + fib(n-2);
        }
    }

    public static void main(String[] args) {
        int f = fib(10);
        System.err.println("Fib(10) = " + f);
    }
}
```

Exercise: caching

- Write a caching aspect that caches the return value based on the input parameter

```
public class Fibonacci {

    ... // old implementation

    public static class CacheAspect
        extends Aspect {
        private Map m_cache = new HashMap();

        // impl. your pointcut here...

        // impl. your advice here...
    }
}
```

Exercise: caching

- One possible solution

```
public static class CacheAspect extends Aspect {
    private Map m_cache = new HashMap();

    /** @Expression execution(int *..Fibonacci.fib(int)) */
    Pointcut fibs;

    /** @Around fibs */
    public Object cache(JoinPoint jp) {
        MethodSignature sig = (MethodSignature)jp.getSignature();
        Integer parameter = (Integer)sig.getParameterValues()[0];
        Integer cachedValue = (Integer)m_cache.get(parameter);
        if (cachedValue == null) {
            Object newValue = jp.proceed(); // calculate
            m_cache.put(parameter, newValue);
            return newValue;
        }
        else {
            return cachedValue; // return cached value
        }
    }
}
```

What’s behind the scene ?

```
public static class CacheAspect extends Aspect

    // ... utility methods etc.

    /** @Expression execution(int *..Fibonacci.fib(int)) */
    Pointcut fibs;

    /** @Around fibs */
    public Object cache(JoinPoint jp) {
        // ...
    }
}
```

Aspects are plain classes can be abstract, static, extended etc.

Pointcuts are fields with attributes defining the pattern and type

Advice are methods with attributes which binds the advice to a pointcut

```
<aspectwerkz>
    <system id=“fibonnaci”>
        <aspect class=“Fibonacci$CacheAspect”/>
        ...
    </system>
</aspectwerkz>
```

The pointcut could have been inlined in the advice definition

XML deployment descriptor to use the Aspect during weaving

Exercise: aspect reuse

- Try to turn the previous Aspect into a reusable library
- Extract an abstract Aspect out of the caching aspect

```
public abstract class AbstractCacheAspect
    extends Aspect {
    // what goes here?
}
```

```
public static class CacheAspect
    extends AbstractCacheAspect {
    // what goes here?
}
```

Exercise: aspect reuse

- **Solution:** put the generic *advice* in the *abstract aspect* and the specific *pointcut* in the *concrete aspect*

```
public abstract class AbstractCacheAspect extends Aspect {
    /** @Around fibs */
    public Object cache(JoinPoint jp) {
        ...
    }
}
```

```
public static class CacheAspect extends AbstractCacheAspect {
    /** @Expression execution(int *..Fibonacci.fib(int) */
    Pointcut fibs;
}
```

Introductions

- Introductions allows you to add code to existing classes

- Implemented in *AspectWerkz* using mixins
- Mixins are:
  - a way of simulating multiple inheritance
  - common in dynamic languages like *Ruby, CLOS* and *Groovy*

Mixins

- Each mixin must consist of:
  - an interface (at least one)
  - an implementation of that / those interface(s) (at least one)
- The mixin implementation can be any regular Java class
- Implemented as an inner class in the Aspect class
- Other implementations can be provided and then chosen at runtime (swapped)

Example: mixin

- Mixin implementation is inner class of the Aspect

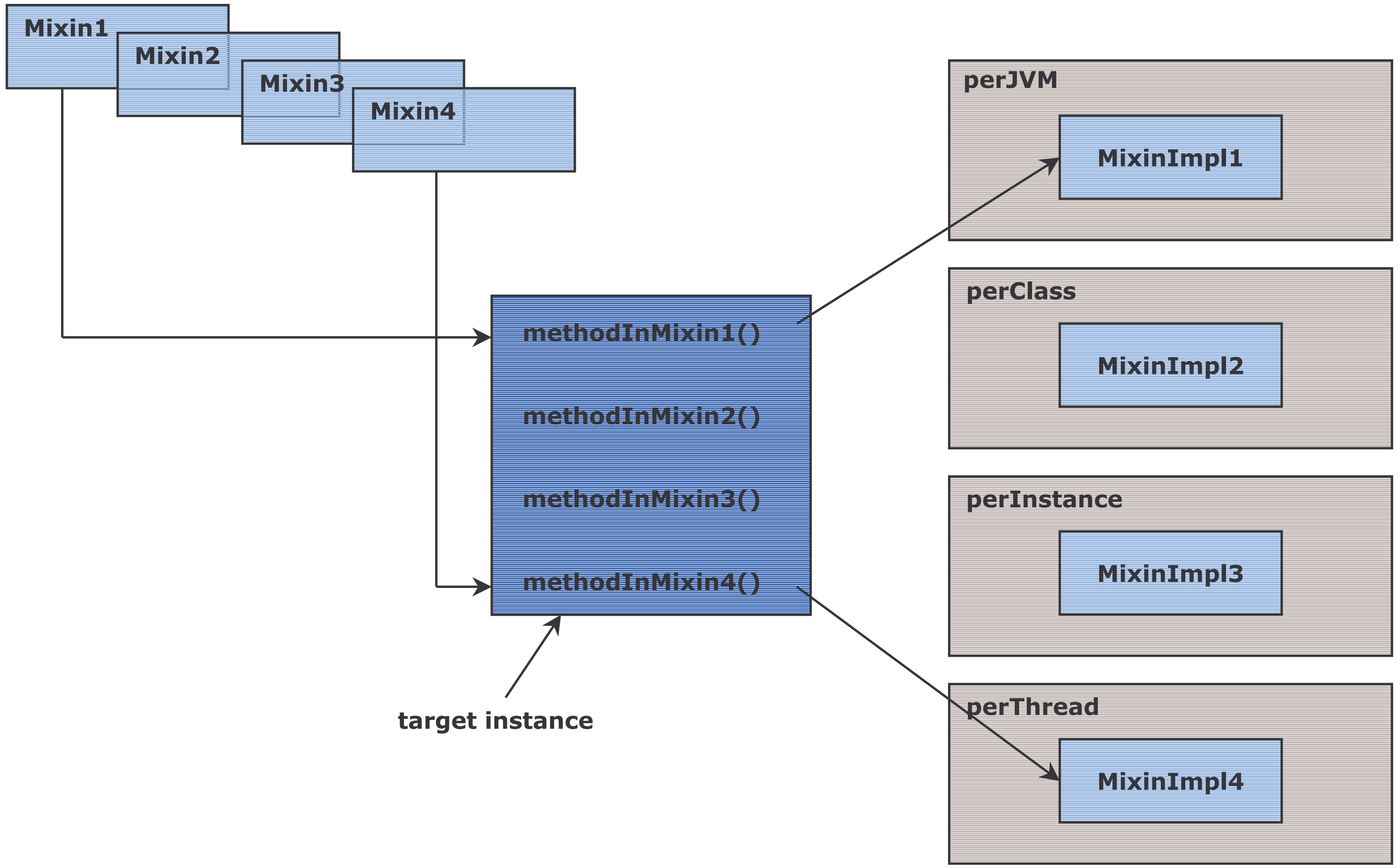
```
/**
 * @Aspect perInstance
 */
public class PersistenceAspect extends Aspect {
   ... // advices and pointcuts

  /**
   * @Introduce *..domain.*
   */
  public static class PersistableMixin extends MyBase
         implements Persistable {

         ... // implementation of the mixin
  }

      ... // more mixins
}
```

define the deployment model

define the binding
(anonymous pointcut in this example)

define the mixin as inner class.
The implementation
implements the introduced
interface(s)

How mixins work in AspectWerkz



Section review (1)

- Pointcuts are defined using patterns
- Pointcut composition algebra allows complex pointcuts and pointcut reuse

- Before / After / Around advices are regular Java methods

- The **`JoinPoint`** class contains RTTI about the join point.
- The **`proceed()`** method allows to continue the execution when applicable

Section review (2)

- How to put it all together, that an Aspect is a regular Java class with metadata
- Aspect reuse can be done through inheritance

- Mixins are implemented as inner classes of the Aspect

- But…
  - how do I package and deploy the Aspects?
  - what is this XML deployment descriptor?
  - how can I use it to make the design more loosely coupled than with abstraction?

Agenda

- What will you learn?
- AOP overview
- AOP constructs in AspectWerkz
- **Aspect development and deployment**
- [Break]
- Weaving and integration scenarios
- Dynamic AOP
- Enteprise application samples

Section objectives

- You will learn how to
  - write self-defined Aspects
  - package the self-defined Aspects with their XML deployment descriptor
  - write XML defined Aspects

- You will understand why
  - both Aspect views are equivalent
  - but might not be used to achieve the same things

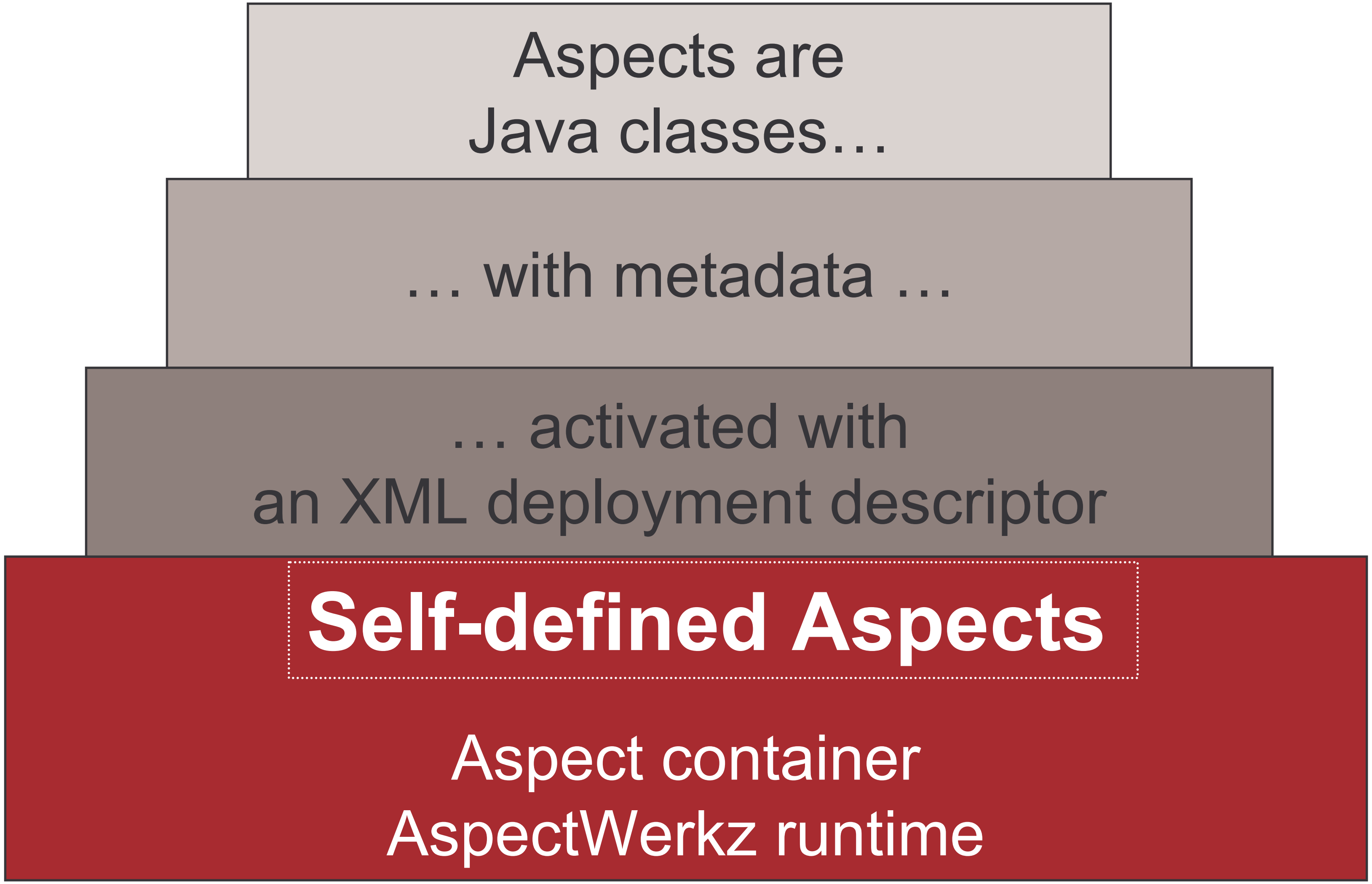
Aspect development and deployment

- *AspectWerkz* provides two ways of defining Aspects:
  - Java class with metadata (*Self-defined Aspects)*
  - Java class with bindings defined in XML

- To be deployed the Aspects need an XML deployment descriptor

- The XML descriptor allows
  - Definition of the aspect if no metadata used
  - Reuse and refinement of the model if metadata used

Self-defined Aspects



Aspects are
Java classes…

… with metadata …

… activated with
an XML deployment descriptor

Self-defined Aspects

Aspect container
AspectWerkz runtime

Self-defined Aspects

- The definition model we have used so far!

- Aspects are plain Java classes
- Pointcuts are fields
- Advices are methods
- Mixins are inner classes of the Aspect

- Metadata represented as attributes (or JSR-175)
- Custom doclet attributes are inserted in the compiled aspect *.class* file

Self-defined Aspects

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

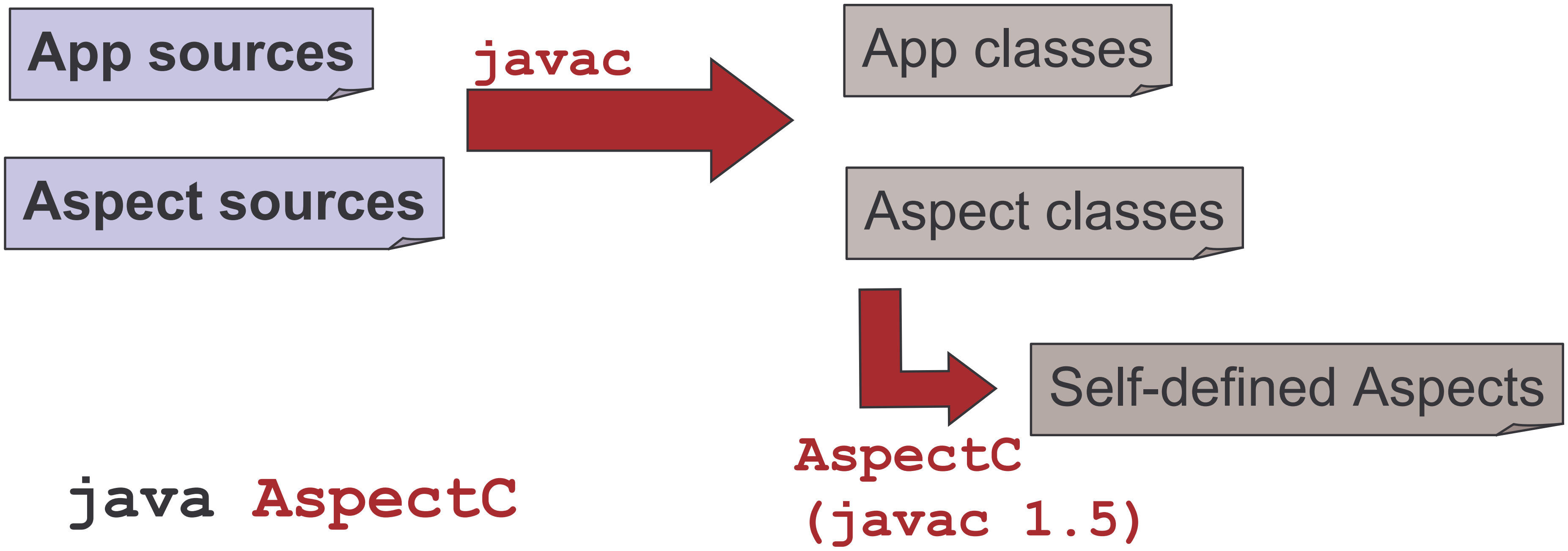
Self-defined Aspects

- Custom runtime attributes implementation:
  - *JavaDoc* tags (parsed using *QDox*)
  - Attributes inserted in bytecode of compiled class/method/field

- Not needed for Java 1.5 and above
- Ready for *JSR-175* (Metadata Facility for Java)

Self-defined Aspects compilation

- **AspectC** allows compilation of metadata into the Aspect's bytecode



```
java AspectC
    path/src/aspect/
    path/target/aspect-classes/
```

We have written a self-defined Aspect

```
public static class CacheAspect extends Aspect {

    // ... utility methods etc.

    /** @Expression execution(int *..Fibonacci.fib(int)) */
    Pointcut fibs;

    /** @Around fibs */
    public Object cache(JoinPoint jp) {
        // ...
    }
}
```

Aspects are plain classes can be abstract, static, extended etc.

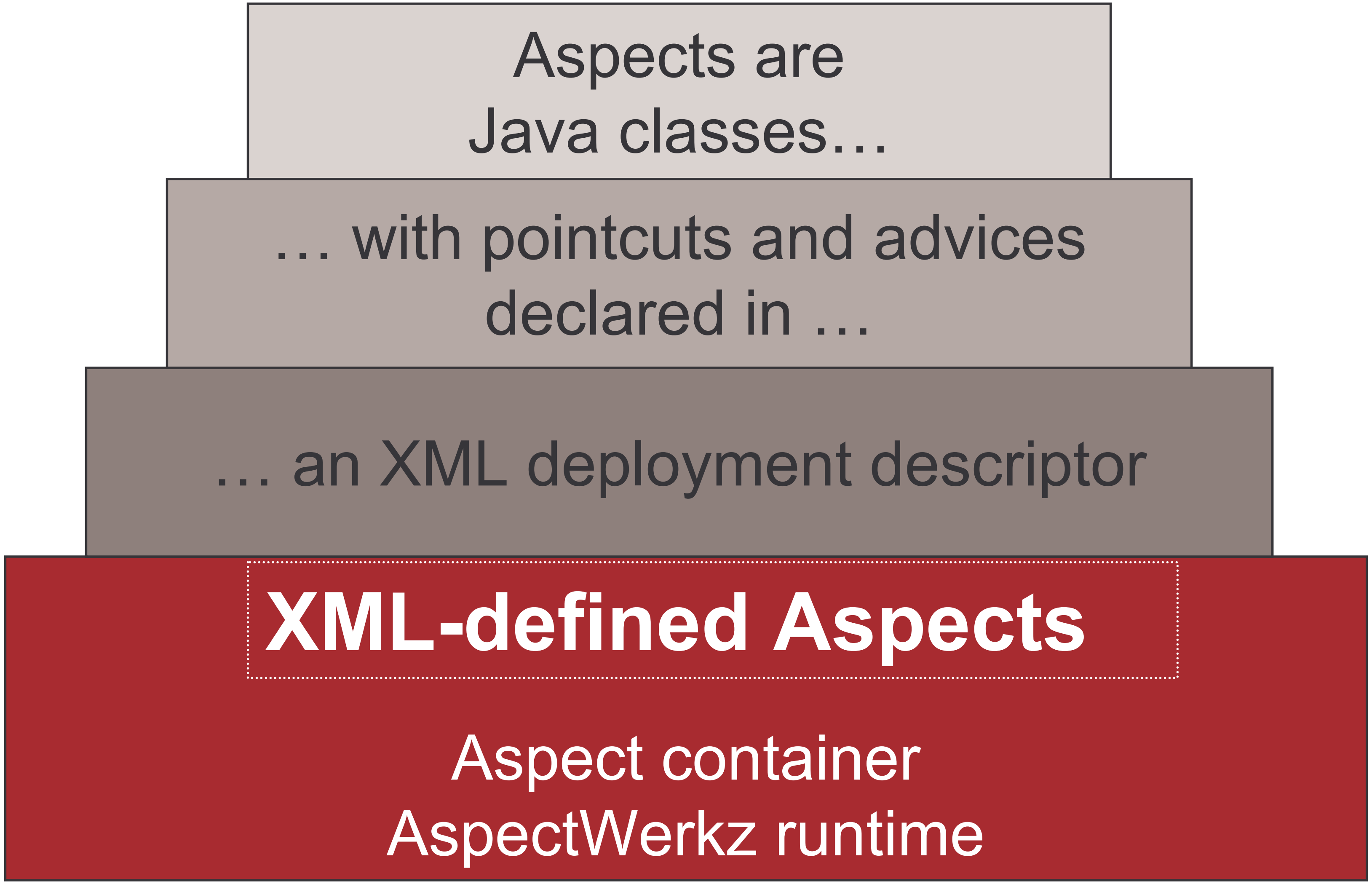
Pointcuts are fields with attributes defining the pattern and type

Advices are methods with attributes which binds the advice to a pointcut

```
<aspectwerkz>
    <system id="fibonnaci">
        <aspect class="Fibonacci$CacheAspect"/>
        ...
    </system>
</aspectwerkz>
```

XML deployment descriptor to use the Aspect during weaving

XML-defined Aspects



Aspects are
Java classes…

… with pointcuts and advices
declared in …

… an XML deployment descriptor

XML-defined Aspects

Aspect container
AspectWerkz runtime

XML defined Aspects

- Aspects are plain Java classes
- Advice are methods
- Mixins can be inner classes of the Aspect or external classes

- Pointcuts are defined in XML descriptor
- Binding is defined in XML descriptor

Example: XML defined Aspect

- This advice turns regular synchronous method invocations into asynchronous invocations

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

   public Object execute(JoinPoint joinPoint)
         throws Throwable {
         m_threadPool.execute(new Runnable() {
            public void run() {
               try {
                  // proceed in a new thread
                  joinPoint.proceed();
               } catch (Throwable e) {
                  throw new WrappedRuntimeException(e);
               }
            });
         return null;
     }
}
```

Extend Aspect

Declare a thread pool

Write the advice

Executes every new invocation (proceed) in a new thread

Example: XML definition syntax

- Example on how to define the **`AsynchAspect`** using the XML deployment descriptor

```
<aspectwerkz>
  <system id="examples">
    <aspect class="samples.AsynchAspect"
            deployment-model="perJVM">

      <pointcut name="asynchCalls"
          pattern="execution(void *..*.*(..))"/>

      <advice name="execute"
          type="around"
          bind-to="asynchCalls"/>

    </aspect>
  </system>
</aspectwerkz>
```

Deployment model

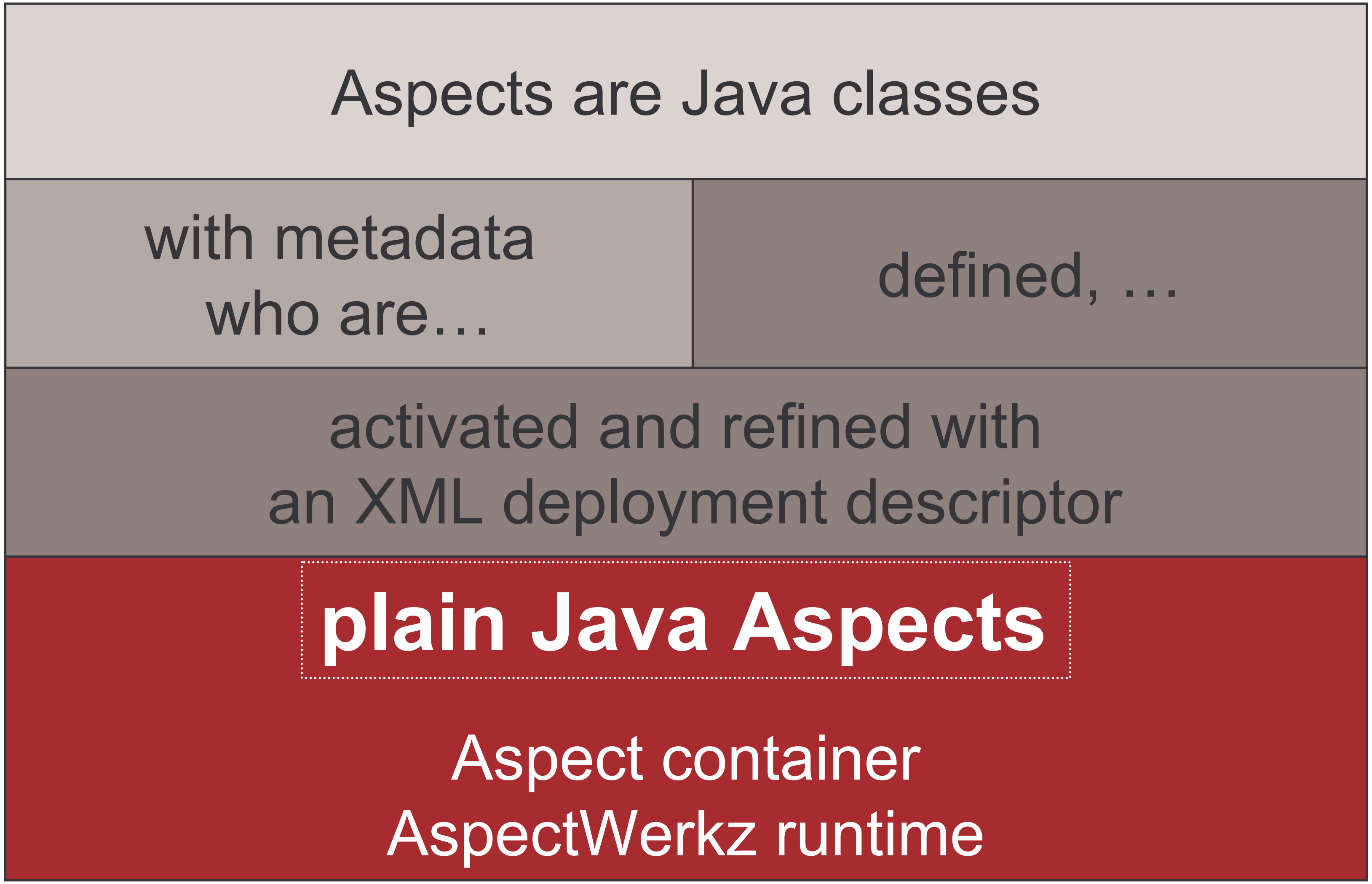
Define pointcut

Bind advice to pointcut

XML defined Aspects

- Advantages
  - No post compilation for metadata management
  - Great tool support (for editing, validation etc.)
  - Loosely coupled
- Drawbacks
  - Separates the implementation from the definition
  - Hard to read and to maintain
  - No refactoring support

Different view of the same model



Aspects are Java classes

with metadata
who are…

defined, …

activated and refined with
an XML deployment descriptor

plain Java Aspects

Aspect container
AspectWerkz runtime

Different views of the same model

- Both approaches are fully compatible
  - uses the same internal aspect container
  - implementation is the same
- The deployment descriptor can be used to override the metadata definition of a self-defined Aspect
- Reuse Aspects
  - Extends an Aspects and (re)define pointcut metadata
  - Refine pointcuts and/or bindings of Aspects in the XML definition

Aspect reuse (1)

- Reuse through inheritance and pointcut redefinition
- Let’s go back to the fibonnaci cache exercise:

```
public abstract class AbstractCacheAspect extends Aspect {
    /** @Around fibs */
    public Object cache(JoinPoint jp) {
        ...
    }
}
```

```
public static class CacheAspect
    extends AbstractCacheAspect {
    /** @Expression execution(int *..Fibonacci.fib(int)) */
    Pointcut fibs;
}
```

Aspect reuse (2)

- There is actually another way of making the **CacheAspect** reusable:
  1. Leave the concrete implementation but remove the **Pointcut** definition
  2. (Re)Define the pointcut in the XML definition:

Needed since the parent aspect is abstract

```
<aspect class="CacheAspect">

  <pointcut name="fibs"
    pattern=“execution(int *..Fibonnaci.fib(int))"/>



</aspect>
```

Aspect reuse (3)

- Benefits in defining the specific pointcuts in XML:
  - More loosely coupled design
  - Easier to configure/reconfigure
  - No need to compile a concrete aspect class implementing the pointcuts (Java 1.4 and below)
  - Define the aspect at deployment time and not at compile time

- Drawbacks:
  - Not pure Java
  - Might be harder to keep implementation and definition in synch

Section review

- Self-defined Aspects use metadata compiled in Aspect class' bytecode
- XML defined Aspects are described in the XML deployment descriptor
- Metadata and XML are different views of the same model
- The XML deployment descriptor allows reuse and refinement of Aspects (as well as activation)

Break

Agenda

- What will you learn?
- AOP overview
- AOP constructs in AspectWerkz
- Aspect development and deployment
- [Break]
- **Weaving and integration scenarios**
- Dynamic AOP
- Enteprise application samples

Section objectives

- Learn how to apply Aspect in target applications
- Learn what is the deployment unit of an AOP enabled application
- Offline weaving - when and why?
- Online weaving - when and why?
- [Optional] use *AspectWerkz* for any load time bytecode transformation
- Learn what will be the next generation of weaving solutions

Weaving

- Weaving
  - instrumentation of the classes
  - when the advice and introductions are added (weaved in) to the classes
- *AspectWerkz* supports two types of weaving:
  - Offline: classes are weaved in a compilation phase (post-processed)
  - Online: classes are weaved transparently

Online and Offline Weaving

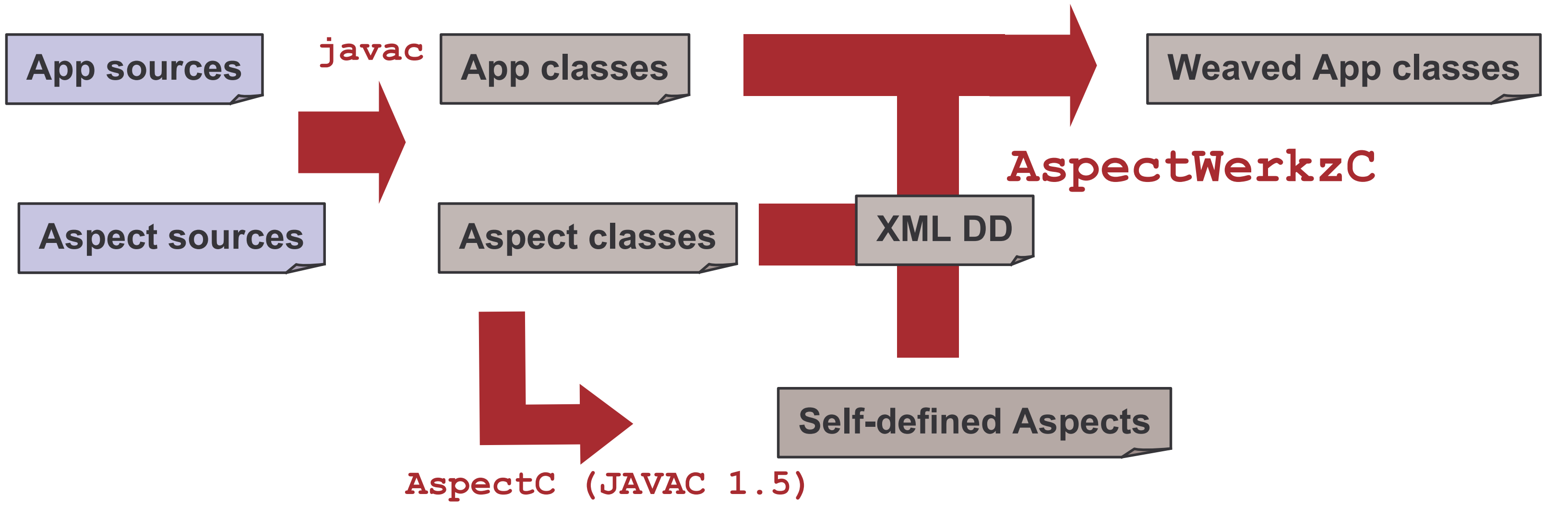
- Modifies the bytecode the same way

- Enables dynamic AOP
  - Add advice at runtime
  - Remove advice at runtime
  - Reorder advice at runtime
  - Swap mixin implementation at runtime

- Do not address the same use-cases
- Complements each other

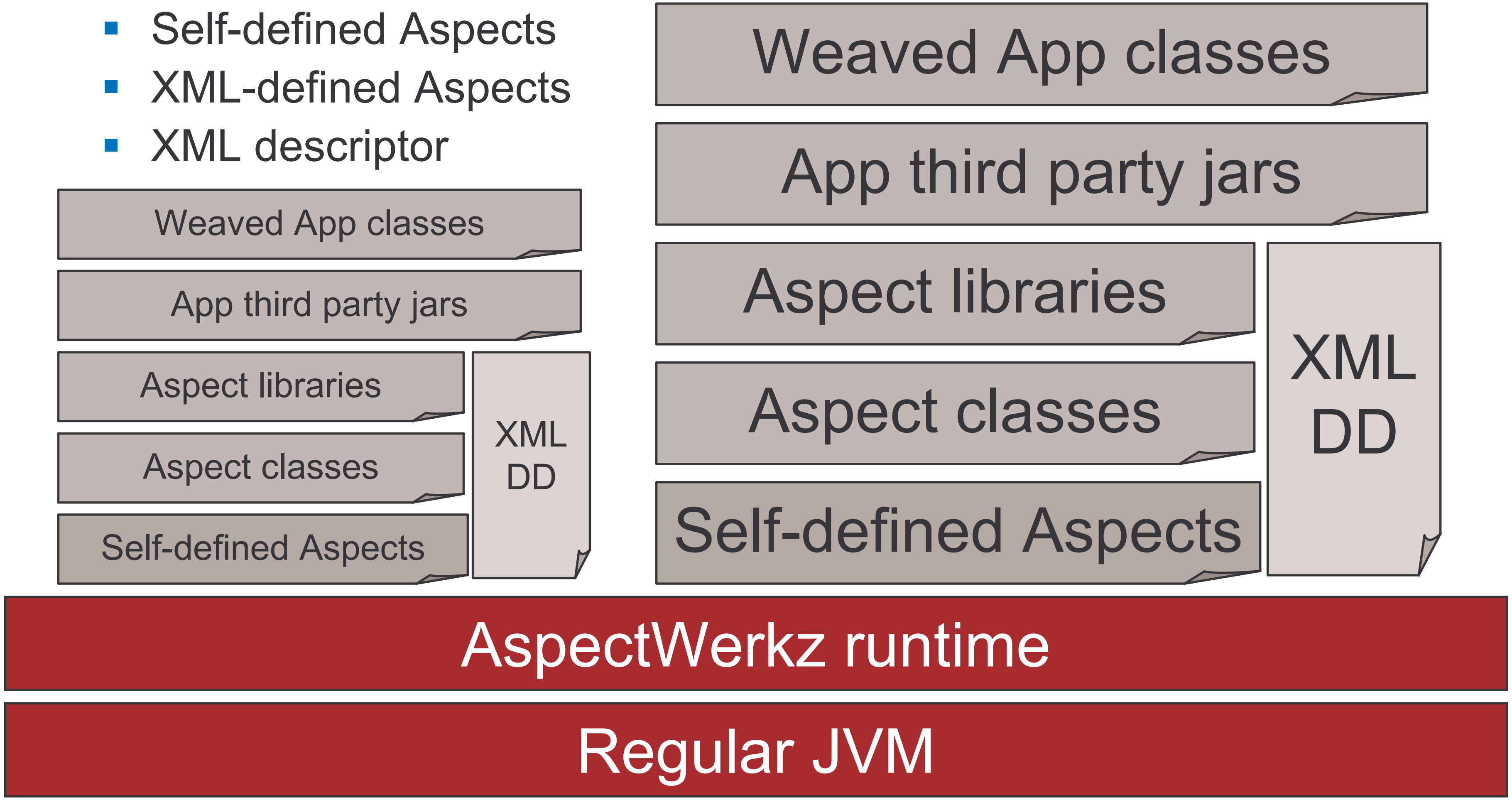
Offline weaving

- Offline weaving alters target classes based on pointcuts and introductions defined by self-defined Aspects and XML deployment descriptor
- Aspects can be in separated jar(s)
- All third party jars of the application should be available in the *offline weaving classpath*



Offline weaving

- The deployment unit is
  - Weaved application classes and third party jars
  - Reused aspect / aspect libraries
  - Self-defined Aspects
  - XML-defined Aspects
  - XML descriptor



Offline weaving

- Advantages
  - Non intrusive: Use when you don’t have full control over the system startup e.g. when deploying a web app in a shared application server
  - Performs a little bit better at load time (no weaving at class load time)
- Drawbacks
  - Adds a compilation step to the build process (**`AspectWerkzC`** can be scripted with *Ant* or *Maven*)
  - Requires a dedicated action to enable AOP. If you deploy your web app and the sys admin wants to have a performance measurement aspect on **all** *Servlets*, he has to tell you to change your offline weaving phase

Exercise: offline weaving

1. Check documentation of `AspectWerkzC`
   - `-verbose`
   - `-verify`
   - `-cp .. -cp ..`
   - `-Daspectwerkz.transform.verbose=true`

2. Integrate the offline weaving into an Ant target

3. [optional] use command line facility*

* *Maven* plugin developed by Vincent Massol

Exercise: offline weaving

- Ant sample for the `CacheAspect` sample

```
<target name="transform" depends="compile, aspectc">
     <java classname=
         "org.codehaus.aspectwerkz.compiler.AspectWerkzC"
         fork="true">
         <classpath ...>
         <jvmarg value="-Daspectwerkz.definition.file=
                       ${src.test.dir}/aspectwerkz.xml"/>
         <arg value="${build.test.dir}"/>
     </java>
</target>
```

Exercise: offline weaving

- Command line tool sample
  - Hide the *classpath* details
  - The command line tool allows quick start

```
bin/aspectwerkz.sh
    -offline
    src/aspectwerkz.xml
    build/classes
```

Online weaving

- A recurrent problem in Java AOP
- No real **standardized facilities** (until Java 1.5 *JSR-163*)

- Two problems to solve
  - Class load time weaving (that works everywhere no matter the class loading scheme e.g. J2EE)
  - Runtime weaving, AKA *HotSwap* weaving

Online weaving: why do we need it? (1)

- Class load time weaving
  - seamless weaving at JVM class loading time
  - based on AOP defined in the deployment unit
  - can also be based on the container configuration
  - allows transparent AOP middleware
- Runtime weaving
  - On demand weaving without class reloading
  - A new dimension in dynamic AOP
  - Redefine pointcuts at runtime

Online weaving: why do we need it? (2)

- Current solutions for class load time weaving
  - Custom classloader for specific usages:
    - *BEA*’s `ClassPreProcessor` in *WLS* 6+
    - *JBoss 4DR2*
    - *weblogic-aspect* for *AspectJ*
    - *etc.*
  - Not reliable / generic enough

Online weaving

- *AspectWerkz* online mode
  - Class load time weaving
  - Cross platform JVM wide weaver hook
  - Validated on *WebLogic, JBoss, Tomcat, WebSphere, IBM JRE, BEA JRockit*, Java 1.3, 1.4
- Runtime weaving support
  - Define new pointcuts at runtime
  - Remove old pointcuts at runtime
  - Without application redeploy

Online weaving - hooking

- *AspectWerkz* provides several way to enable class load time weaving by hooking in at **`java.lang.ClassLoader`** level
- **`-Xbootclasspath`** for Java 1.3 and 1.4
  - Done transparently (a JVM launches the JVM) or prepared manually
  - Needs Sun agreement

- HotSwaps the **`java.lang.ClassLoader`** in Java 1.4
  - Pioneered by *JMangler, AOSD 2003*
  - Requires **`–Xdebug`** mode (to allow *HotSwap*)

1. Done through another JVM (remotely, at startup or not)
2. Done in process (C native *JVMPI* module, at VM init time)

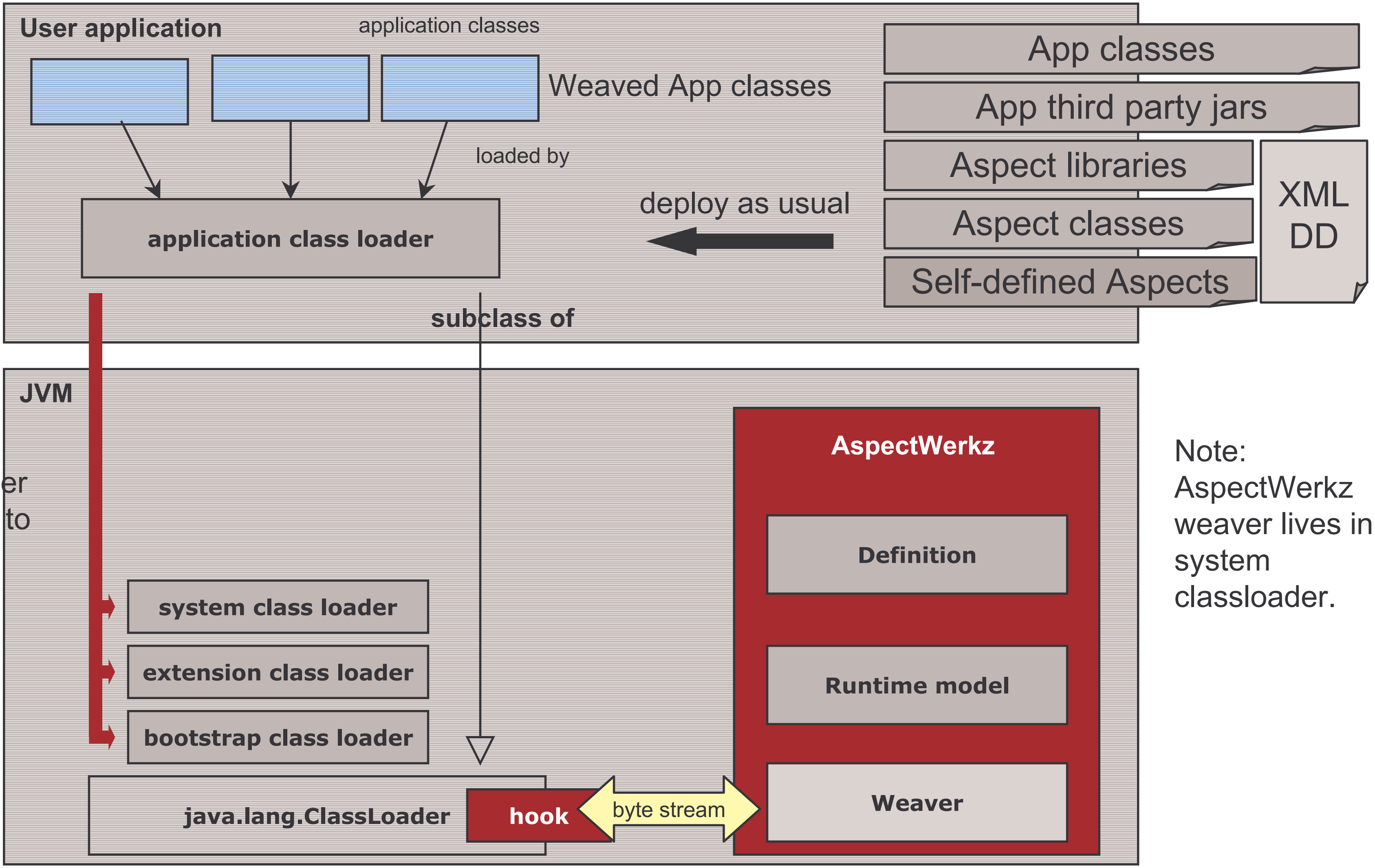
Online weaving - hooking

- *BEA JRockit* dedicated module for Java 1.3 and 1.4
  - The most seamless experience
  - **`ClassPreProcessor`** interception is part of *JRockit*
  - No **`–Xdebug`** mode

- *AspectWerkz* command line tool chooses the easiest for you (Java version auto detection, classpath…)

- Hooking standardized with Java 1.5 *JSR-163* through the **`java.lang.instrument.ClassFileTransformer`**

Load time weaving using HotSwap



ClassLoader
delegation to
parent first

Note:
AspectWerkz
weaver lives in
system
classloader.

Integration efforts

- So online weaving interacts at the **`java.lang.ClassLoader`** level

- How hard is it to integrate in my own application ?
  - Standalone application
  - Application server
  - What about IDE support for testing ?

- What about the Java 1.5 *JSR-163*?

Online weaving – integration efforts

- Command line tool
  - Minimal effort, java command line replacement
  - Poor optimization under Java 1.4 (stdout/err piped between two JVM)

```
aspectwerkz.sh <vm options>
    -Daspectwerkz.definition.file=...
    -cp <additional classpath>
    MainClass
```

Online weaving – integration efforts

- Change your application startup script
  - More effort (set classpath etc)
  - More control (force `-Xbootclasspath`, turn on/off options etc)
  - Force native in process module:

System dependent JVMPI module

```
java -Xdebug
     -Xrunaspectwerkz
     -Daspectwerkz.definition.file=...
     -cp <additional classpath>
     MainClass
```

Online weaving – integration efforts

- *BEA JRockit* enables seamless AOP
  - Without **-Xdebug**
  - Solution for Java 1.3 and Java 1.4
  - Full Java implementation

```
java -Xmanagement:class=
      ..aspectwerkz.JRockitPreProcessor
```

AspectWerkz JRockit extension

- Exercise: use online mode for enterprise application

Online weaving in Java 1.5

- Online weaving is standardized by *JSR-163*

- **`java.lang.instrument.ClassFileTransformer`**
  - Full Java API
  - Equivalent at C level if required
  - Supports multiple transformation
  - No **`–Xdebug`** mode required

```
java -Xjavaagent=..aspectwerkz.PreMain
```

AspectWerkz JSR-163 preMain agent to register the AspectWerkz **`ClassFileTransformer`**

Online weaving is generic

- Online weaving and hooking is generic
- Can be used to have online weaving for *AspectJ*, *JBoss AOP*, or your own solution
- Allows to write ones’ own bytecode transformation at load time
- Independent from bytecode manipulation libraries

Online weaving – writing a hook

- *The following are only required if ones wants to use online weaving architecture of AspectWerkz without using AspectWerkzAOP!*
- Step 1 [optional]
  - Write a **`ClassLoaderClassPreProcessor`** to alter the **`java.lang.ClassLoader`** as you want (*BCEL*, *Javassist*, *ASM* available already for a **`ClassPreProcessor`** mechanism)

```
/**
 * Instruments the java.lang.ClassLoader bytecode
 */
 public byte[] preProcess(byte[] b);
```

Online weaving – writing a hook

- Step 2
  - Write a `ClassPreProcessor` as the weaver entry-point

```
/**
 * Invoked before a class is defined in the JVM
 */
public byte[] preProcess(
    String className, byte[] b, ClassLoader cl
);
```

- Step 3
  - Use it for online mode (will work in offline mode as well)
  - `-Daspectwerkz.transform.classloaderpreprocessor=...`
  - `-Daspectwerkz.transform.classpreprocessor=...`
  - Defaults to *AspectWerkz AOP* (*Javassist* based in 0.10)

Section review

- AspectWerkz has two weaving modes:
  - Offline
  - Online
- Offline mode post-compiles the application classes before deployment and does not required environment changes
- Online mode transforms the application classes at load time but requires to be integrated in the environment
- AspectWerkz provides several online mode options, and is ready for *JSR-163*
- Online mode can address new use-cases e.g. track down EJB CMP SQL calls without prior knowledge of the target JDBC driver

Agenda

- What will you learn?
- AOP overview
- AOP constructs in AspectWerkz
- Aspect development and deployment
- [Break]
- Weaving and integration scenarios
- **Dynamic AOP**
- Entreprise application samples

Section objectives

- Learn about *AspectWerkz*’ dynamic AOP capabilities

- Use the API to swap mixin implementations and change the advice bound to a specific pointcut

Dynamic AOP (1)

- *AspectWerkz* is join point centric: the transformed bytecode depends only on the pointcuts and the introductions

- Example of use-cases:
  - Enable/disable tracing or performance statistics on demand
  - Change the implementation of your AOP based cache at runtime
  - Compose aspects for fault tolerance mechanisms

Dynamic AOP (2)

- Dynamic AOP is achieved at existing pointcuts
  - Using the `cflow` pointcut
  - Swap mixin implementation to alter behavior
  - Add aspect and bind its advice on existing pointcuts
  - Reorder or remove advice bounded at existing pointcuts
- Pointcut addition and removal requires runtime weaving

Dynamic runtime model

- Allows you to redefine the system at runtime:
  - Swap mixin implementation at runtime

```
SystemLoader.getSystem(systemId).
    getMixin(oldMixinName).
     swapImplementation(newMixinClassName);
```

  - Add new aspects and advice at runtime

```
SystemLoader.getSystem(systemId).createAspect(
    aspectName,
    className,
    DeploymentModel.PER_INSTANCE,
    classLoader
);
```

  - Reorder advice at runtime (API is being reimplemented)
  - Remove advice at runtime (API is being reimplemented)

Agenda

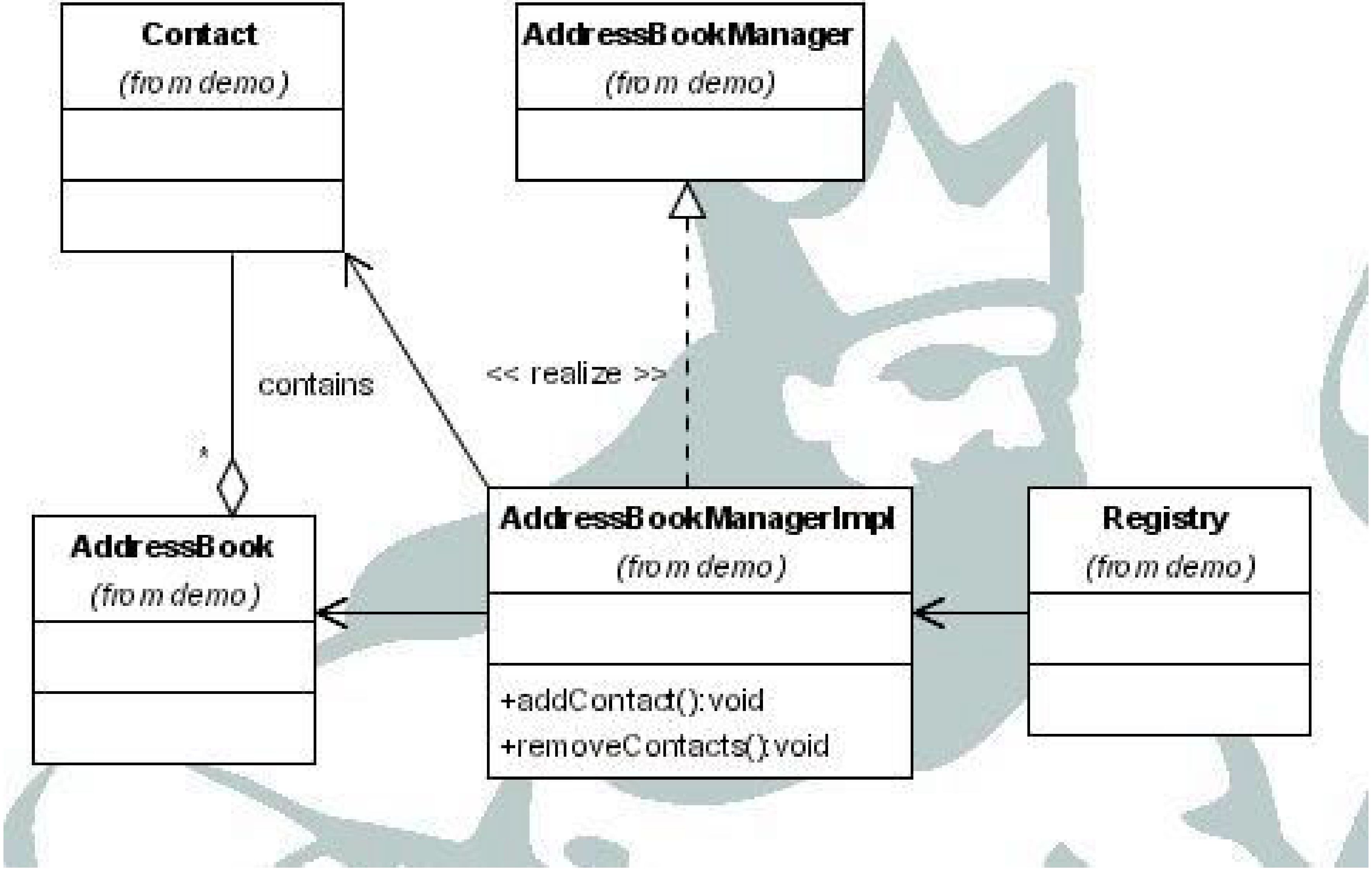
- What will you learn?
- AOP overview
- AOP constructs in AspectWerkz
- Aspect development and deployment
- [Break]
- Weaving and integration scenarios
- Dynamic AOP
- Enterprise application samples

Example: Enterprise Application

- Address book web application
  - Login / logout
  - List user’s contacts
  - Add a contact
  - Remove one or more contacts

- Services
  - Authentication
  - Authorization
  - Persistence of the address books
  - Transaction integrity

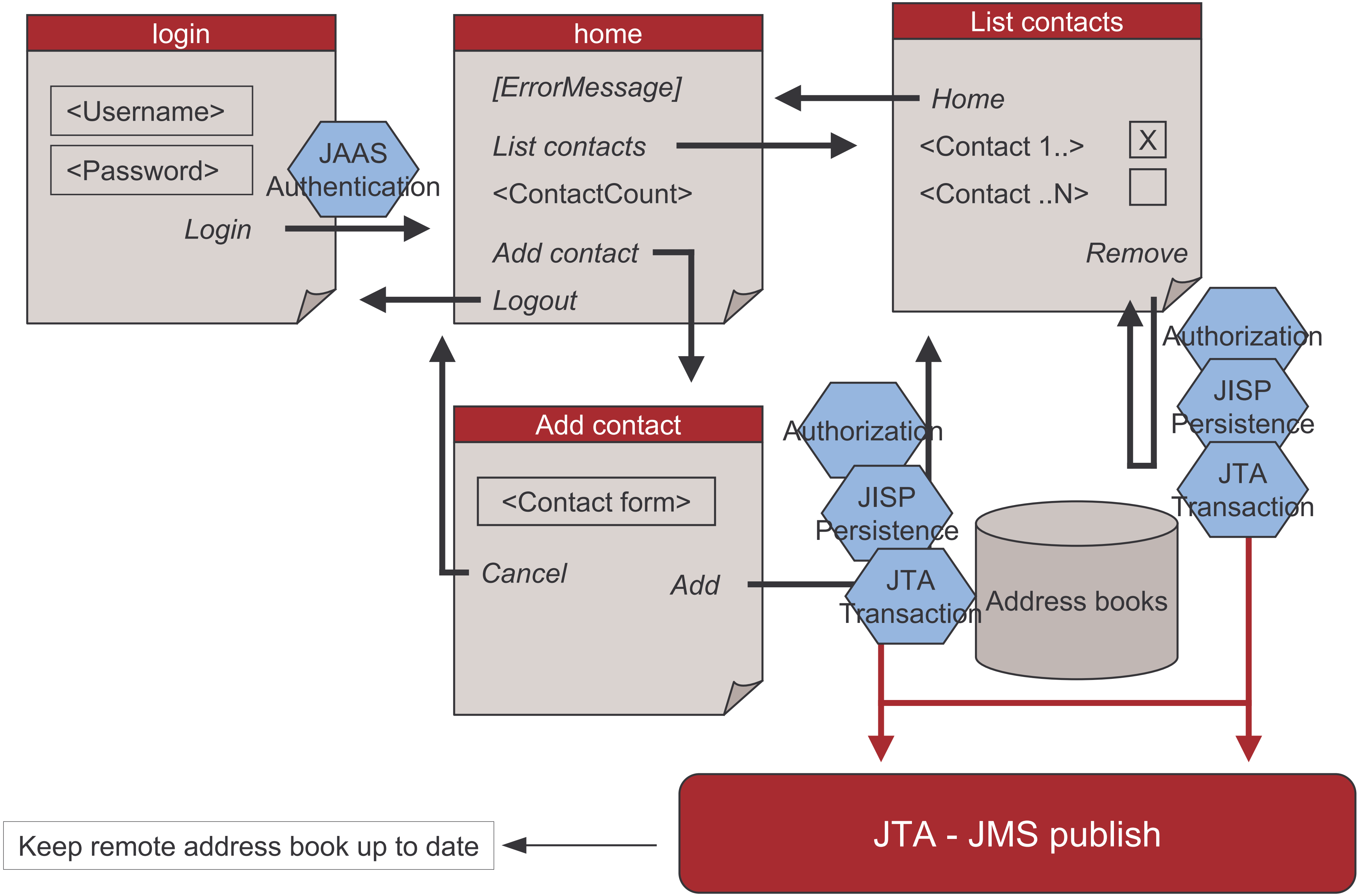
AddressBook application: UML diagram



Services to implement using AOP

- Role-based security (using JAAS)
- Transaction handling (using JTA)
- Transparent persistence (using JISP)

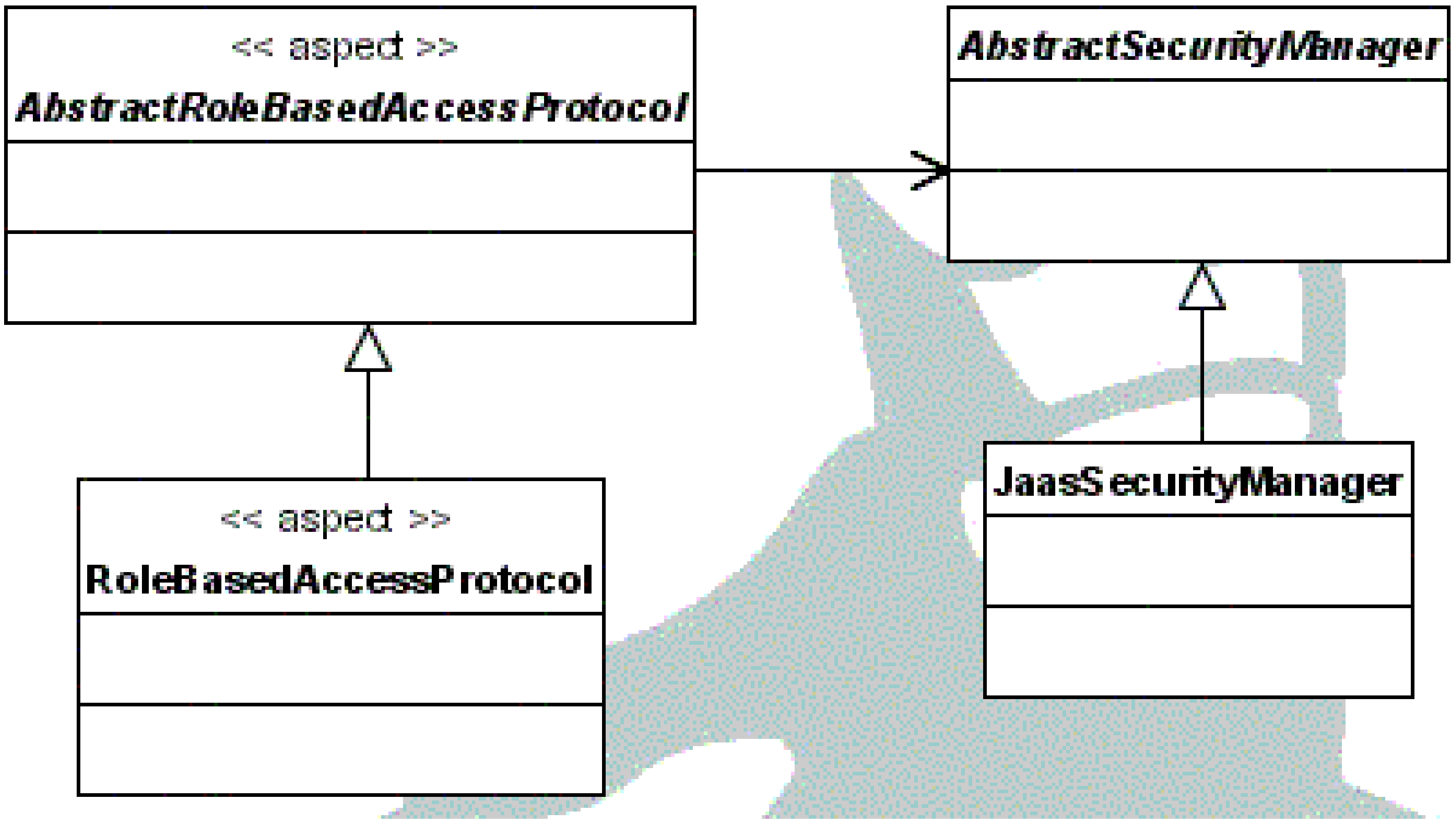
Where is the cross-cutting code?



Why use AOP?

- Role based security through AOP has lot of value
  - A **`ServletFilter`** could only implement authentication and URL based authorization, and would be *web specific*
  - Ease of reuse with Aspect abstraction
- **`UnitOfWork`** integrates in JTA so that it fits nicely when external entreprise components (JMS, EJB etc.) are called
- **`UnitOfWork`** integrates transparent persistence without coupling with the persistence layer

Role Based Security: UML diagram



Abstract base aspect

- Implements the advice
- Defines the “abstract” pointcuts

```
public abstract class AbstractRoleBasedAccessProtocol
   extends Aspect {

   protected Subject m_subject = null;

   protected final SecurityManager m_securityManager = ...

   /** @TO_BE_DEFINED */
   Pointcut authenticationPoints;

   /** @TO_BE_DEFINED */
   Pointcut authorizationPoints;

   ... // implementation of the advices
}
```

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;
  }
```

Authorization advice

```
/**
 * @Around authorizationPoints
 */
public Object authorizeUser(JoinPoint joinPoint)
   throws Throwable {
   MethodSignature sig =
        (MethodSignature)joinPoint.getSignature();
   if (m_securityManager.checkPermission(
      m_subject,
      joinPoint.getTargetClass(),
      sig.getMethod())) {
      // user is authorized => proceed
      return joinPoint.proceed();
   }
   else {
      throw new SecurityException(...);
   }
}
```

Integration in the AddressBook webapp

- Authenticate the user at the application level
  - Servlet’s methods
- Authorize on methods that modifies the **`AddressBook`**
  - `AddressBookManager+.addContact(..)`
  - `AddressBookManager+.removeContacts(..)`
- Extend **`AbstractRoleBasedAccessProtocol`** aspect and define the pointcuts:
  - `authenticationPoints`
  - `authorizationPoints`

Concrete aspect

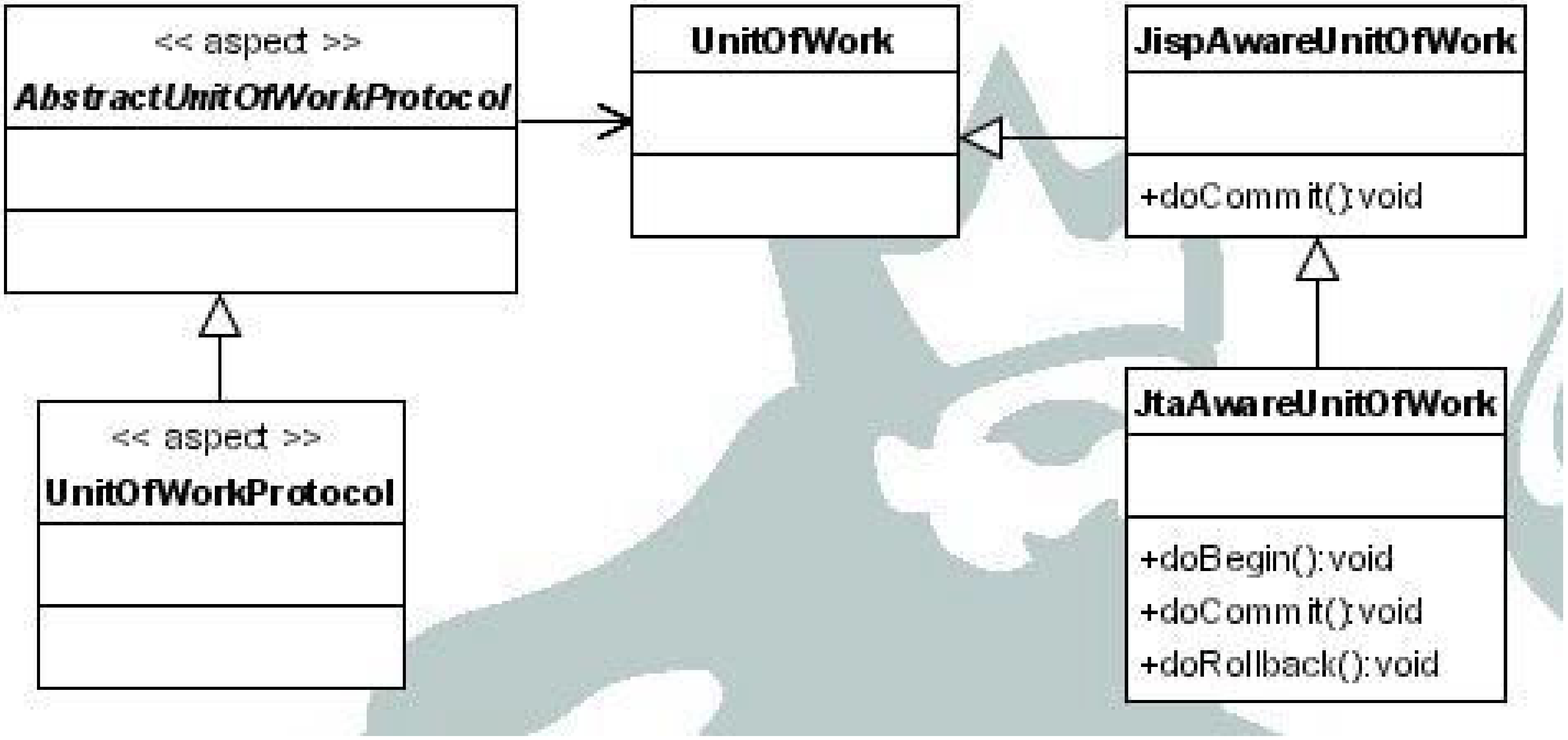
- Defines the poincuts and the deployment model

```
/**
 * @Aspect perThread
 */
public class RoleBasedAccessProtocol
    extends AbstractRoleBasedAccessProtocol {

    /**
     * @Expression execution(* web.HomeServlet.doGet(..))
     */
    Pointcut authenticationPoints;

    /**
     * @Expression execution(* AddressBookManager+.*(..))
     */
    Pointcut authorizationPoints;
}
```

Unit Of Work

- Unit Of Work
  - Common pattern in enterprise application architectures
  - Implements a transaction
  - Keeps track of new, removed and dirty objects
- Will be used to implement:
  - Transaction demarcation for Plain Old Java Objects (POJOs)
  - Persistence handling for POJOs

Unit Of Work: UML diagram



The Unit Of Work API

```
public class UnitOfWork {
    public static UnitOfWork begin() {...}
    public void commit() {...}
    public void rollback() {...}

    // registers the transactional objects
    public void registerNew(Object obj) {...}
    public void registerRemoved(Object obj) {...}
    public void registerDirty(Object obj) {...}

    // template methods
    public void doBegin() {...}
    public void doCommit() {...}
    public void doPreCommit() {...}
    public void doPostCommit() {...}
    public void doRollback() {...}
    public void doDispose() {...}
}
```

Template methods

- The **`UnitOfWork`** has some template methods:
  - **`public void doBegin() {...}`**
  - **`public void doCommit() {...}`**
  - etc.
- These allows subclasses to define what to do at specific points:
  - TX begin
  - TX commit
  - TX pre-commit
  - TX post-commit
  - TX rollback
  - TX dispose

Problems with non AOP solution (1)

- Is a cross-cutting concern
- Introduces code scattering
- Introduces code tangling

Problems with non AOP solution (2)

- For example, this code:

```
...
AddressBook book = new AddressBook(...);
book.addContact(contact);
...
```

- Would have to be replaced by:

```
UnitOrWork unitOfWork = UnitOfWork.begin();
try {
   AddressBook book = new AddressBook(...);
   unitOfWork.registerNew(book);
   book.addContact(contact);
   unitOfWork.registerDirty(book);
   unitOfWork.commit();
} catch(Exception e) {
   unitOfWork.rollback();
}
```

Enter Aspect-Oriented Programming

- Can make the **UnitOfWork** completely transparent
- Abstract base aspect

```
public abstract class AbstractUnitOfWorkProtocol
    extends Aspect {

    /** @TO_BE_DEFINED */
    Pointcut transactionalObjectCreationPoints;
    /** @TO_BE_DEFINED */
    Pointcut transactionalObjectModificationPoints;
    /** @TO_BE_DEFINED */
    Pointcut transactionalMethods;

    ... // advice and introductions
}
```

Advice: RegisterNew

- Registers the newly created instance

```
/**
 * @Around transactionalObjectCreationPoints
 */
public Object registerNew(JoinPoint joinPoint)
    throws Throwable {
    Object newInstance = joinPoint.proceed();
    if (UnitOfWork.isInUnitOfWork()) {
        UnitOfWork unitOfWork = UnitOfWork.getCurrent();
        unitOfWork.registerNew(newInstance);
    }
    return newInstance;
}
```

Advice: RegisterDirty

- Registers an object as dirty just before a field is modified

```
/**
 * @Before transactionalObjectModificationPoints
 */
public void registerDirty(JoinPoint joinPoint)
   throws Throwable {
   if (UnitOfWork.isInUnitOfWork()) {
      Signature sig = joinPoint.getSignature();
      UnitOfWork unitOfWork = UnitOfWork.getCurrent();
      unitOfWork.registerDirty(
         joinPoint.getTargetInstance(),
         sig.getName()
      );
   }
}
```

Advice: ProceedInTransaction

```
/** @Around transactionalMethods */
public Object proceedInTransaction(JoinPoint joinPoint) {
    if (UnitOfWork.isInUnitOfWork()) {
        return joinPoint.proceed();
    }
    UnitOfWork unitOfWork = UnitOfWork.begin();
    final Object result;
    try {
        result = joinPoint.proceed();
        if (unitOfWork.isRollbackOnly()) {
            unitOfWork.rollback();
        } else {
            unitOfWork.commit();
        }
    } catch (Throwable throwable) {
        throw handleException(throwable, unitOfWork);
    } finally {
        UnitOfWork.dispose();
    }
    return result;
}
```

Exception handling

- Uses the same approach as in EJB
  - Rollback on `RuntimeException`

```
private Throwable handleException(
    Throwable throwable,
    UnitOfWork unitOfWork) {
    if (throwable instanceof RuntimeException) {
        unitOfWork.rollback();
    }
    else {
        unitOfWork.commit();
    }
    return throwable;
}
```

Transactional mixin

- Mixin with life-cycle and utility methods
- Applied to all transactional objects
- Inner class in the abstract aspect

```
 /** @Introduce TO_BE_DEFINED */
public abstract class TransactionalImpl
   implements Transactional, Serializable {

   public void setRollbackOnly() {...}
   public UnitOfWork getUnitOfWork() {...}
   public TransactionContext getTransaction() {...}
   public void create() {...}
   public void remove() {...}
   public void markDirty() {...}
   public boolean exists() {...}
 }
```

Integration in the AddressBook webapp

- Implement a concrete `JispAwareUnitOfWork` for persistence
  - Implements persistence callback at `UnitOfWork.doCommit()`

    to persist only objects part of Unit Of Work and registered as dirty
- Extend it in a concrete `JtaAwareUnitOfWork` so that persistence commit can be part of a JTA transaction
  - Allow to commit the JTA only if the persistence was successful (and vice versa)
  - Looks like distributed transaction

JispAwareUnitOfWork

- Overrides the `doCommit()` template method

```
public class JispAwareUnitOfWork extends UnitOfWork {
    ... // declare the persistence manager

    public void doCommit() {
        for (Iterator it = m_dirtyObjects.values().
             iterator(); it.hasNext();) {
            ObjectBackup backup =
                (ObjectBackup)it.next();
            s_persistenceManager.store(
                backup.getReference()
            );
        }
    }
}
```

JtaAwareUnitOfWork

```
public class JtaAwareUnitOfWork extends JispAwareUnitOfWork {
    ... // declare the member TX manager and the TX
    public void doBegin() {
        m_transaction = s_txManager.getTransaction();
    }
    public void doRollback() {
        s_txManager.rollback(m_transaction);
    }
    public void doCommit() {
        // if the JTA transaction is set to rollback only;
        // rollback the transaction as well as the unit of work
        if (m_transaction.isExistingTransaction() &&
            m_transaction.isRollbackOnly()) {
            rollback();
            s_txManager.rollback(m_transaction);
        }
       else {
            // invoke the doCommit() method in the JispAwareUnitOfWork
            // that will handle the persistence
            // simplified needs to deal with exceptions in the PM
            super.doCommit();
            s_txManager.commit(m_transaction);
        }
    }
```

Integration in the AddressBook webapp

- Extend `AbstractUnitOfWorkProtocol` aspect and define the pointcuts for
  - `transactionalObjectCreationPoints`
  - `transactionalObjectModificationPoints`
  - `transactionalMethods`

Integration in the AddressBook webapp

- Register the creation of `Contact` instances in the `UnitOfWork`
  - `call(Contact.new(..))`

- Register `Contact` and `AddressBook` as dirty when their fields are modified
  - `set(* Contact.*)`
  - `set(* AddressBook.*)`

Integration in the AddressBook webapp

- Define service methods on `AddressBook` as transactional, part of a `JtaAwareUnitOfWork` (JISP + JTA transaction control)
- Meaning, we define all methods that should start and commit a new transaction

Demo

Conclusion (1)

- *AspectWerkz* supports a broad scope of AOP constructs

- The pointcuts are based on a pattern based expression algebra allowing pointcut composition

Conclusion (2)

- The Aspects constructs are pure Java
- Self-defined Aspects use metadata at class, method, field and inner class level for aspect, advice, pointcut and introduction constructs
- A small XML deployment descriptor allows
  - enabling of self-defined Aspects
  - definition of XML defined Aspects
  - reuse and refinement of Aspects

Conclusion (3)

- Offline mode allows to apply aspects through a post-compilation phase
- Online mode allows to integrate the weaving in the underlying environment at class load time and supports J2EE app servers
- Both modes provides dynamic AOP features

- Time for AOP in enterprise applications
- Will the Aspect Container be *The Next Big Thing*?

Future plans (1)

- Aspect Container
  - Support multiple Aspect systems (multiple XML deployment descriptors) within one JVM
  - Support for hierarchical scoping of Aspects, f.e:
    - «An aspect deployed at the server level should impact all deployed applications»
    - «The application cannot change Aspects defined at the server level (security policy)»
  - Responsibilities: security, isolation, visibility, deployment and runtime management

Future plans (2)

- Runtime weaving and pointcut redefinition
- Java 1.5 support for generics and attributes
- Metadata driven AOP
  - Metadata seen as join points (can be matched and introspected)
  - Metadata seen as a cross-cutting concern that can be attached to join points in a modular and reusable way
- Native JVM support
  - Deep AOP support in the JRockit JVM

AspectWerkz @ AOSD

- Tuesday: Dynamic Aspects Workshop
  - HotSwapped based Runtime weaving
- Wednesday 16:00: Industry Panel
- Friday 11:00: Invited Talk
  - What are the key issues for commercial AOP - how does AspectWerkz address it?

Links

- http://aspectwerkz.codehaus.org/
- http://wiki.codehaus.org/aspectwerkz

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

- http://www.aosd.net/

Questions?

Thanks for listening