AspectJ 5 & Spring

Wim Vanderperren

wvdperre@vub.ac.be

Contents

- **What's new in AspectJ 5?**
  - Meta-data (aka Annotations)
  - Annotation-based development style
- AspectJ-Spring integration
  - Short Spring intro
  - Dependency injection of AspectJ aspects
  - Domain specific aspects & library aspects

AspectJ 5

- AspectJ release compatible with Java 5:
  - Generic types in pointcuts
  - Generic aspects
  - Variable arguments in pointcuts
  - Auto-boxing/unboxing in dynamic pointcuts
  - **Annotations in pointcuts**
  - **Annotation-based development style**

Annotations

- Good pointcut (observer protocol):
  - set(* Point.*)
- Fragile pointcut (caching):
  - call(String X.a())||call(Object Y.b*(..))||call(* C.*(*)) ..
- Solution: annotations
  - @Cachable

Caching

- Annotations to tag methods that could be cached
- @Cachable(timeToLive=500)

```
pointcut enableCaching(Cachable c) : call(* *(..)) &&
        @annotation(c) && if(c.timeToLive()>treshold);
```

Annotations& Pointcuts

- Pointcuts can reason on the presence of annotations
- Examples:
  - **within**(@Secure *)
  - **call**(@Cachable *.*(..))
  - **handler**(!@Catastrophic *)

Annotations& Pointcuts

- Allows new style of aspect application based on semantic tags
- Solves possibly fragile pointcuts
  - e.g. execution(* a()) || execution (* b()) .....
  - becomes call(@Cachable *.*(..))
- Base code is no longer oblivious
- BUT base code is no longer oblivious!

Annotation-based development style

- AspectJ aspects in pure Java
- Uses annotations to specify non-Java constructs
- Weaving can happen at load-time or at linking time
- Advantages:
  - A java compiler suffices
  - Integrates better with existing toolchain/IDE
  - There is no official new language

Example

- The following AspectJ pointcut:
  - pointcut anyCall() : call(* *.*(..));
- Translates to:
  - @Pointcut("call(* *.*(..))") void anyCall() { }

Pointcuts

```
//Pointcut arguments:
@Pointcut("call(* *.*(int)) && args(i) && target(callee)")
void someCall(int i, Foo callee) {}

//If pointcut:
@Pointcut("call(* *.*(int)) && args(i) && if()")
public static boolean someCallWithIfTest(int i) {
    return i > 0;
}

//Abstract pointcut
@Pointcut("")
protected abstract void toOverrideInSubclass();
```

Advice

```
//Simple Advice
@Before("execution(* *.*(..))")
void anyExec() {
    logger.info("Something happened");
}

//Advice with arguments
@After("execution(* *.*(..)) && target(myTarget)")
public void anyExec(Object myTarget) {
    logger.info("Something happend on "+ myTarget);
}

//Advice with joinpoint reflection
@After("execution(* *.*(..)) && target(myTarget)")
public void anyExec(JoinPoint thisJoinPoint, Object myTarget) {
    logger.info(thisJoinPoint.getSignature()+" happened on "+
        myTarget);
}
```

Advice (2)

```
//Around Advice
@Around(“execution(@Idempotent void *.*(..))”)
public Object skipMethod(ProceedingJoinPoint thisJoinPoint) {
     Object result = null;
     if(!hasAlreadyExecuted) {
       hasAlreadyExecuted=true;
       result = thisJoinPoint.proceed();
     }
     return result;
}
```

Advice (3)

```
//Pointcut-Advice Binding
@Pointcut("call(* *.*(..)) && @annotation(info) && if()")
protected static boolean cachableMethods(Cachable info) {
     return info.timeToLive()>treshold;
}

@Around("cachableMethods(info)")
public Object cache(ProceedingJoinPoint thisJoinPoint, Cachable
info) {
     Object result = getFromCache(thisJoinPoint);
     if(result==null) {
         result= thisJoinPoint.proceed();
         storeInCache(thisJoinPoint,result,info.timeToLive());
     }
     return result;
}
```

Limitations

- Types have to be fully referenced in pointcuts
  - NOT: @~~Pointcut("call(* List.*(..))")~~
  - BUT: @Pointcut("call(* java.util.List.*(..))")
- Limited intertype declarations
- No privileged aspects

Contents

- What's new in AspectJ 5?
  - Meta-data (aka Annotations)
  - Annotation-based development style
- **AspectJ-Spring integration**
  - Short Spring intro
  - Dependency injection of AspectJ aspects
  - Domain specific aspects & library aspects

Spring: very short intro

- Very popular J2EE application framework
- Plain POJO beans instead of heavy-weight EJB
- Dependency injection instead of lookup
- Abstraction layers for external APIs (e.g. transaction management)
- Integrates well with third-party frameworks (e.g. Struts)
- Compatible with a large range of application servers
- Excellent documentation

Spring: very short intro

```
//Plain POJO Component
public class MyService {
  private IMailService mailService;
  public void register() {
      ....
      mailService.sendMail(address,"registration successful");
  }
}

//Spring configuration:
<beans>
  <bean id="mailServiceC" class="org.vub.mytool.MyMail">
    <property name="host" value="smtp.vub.ac.be"/>
  </bean>
  <bean id="myService" class="org.vub.mytool.MyService">
    <property name="mailService" ref="mailServiceC"/>
  </bean>
</beans>
```

Spring and AOP

- Spring explicitly supports AspectJ AOP
- Aspects can be configured like normal Spring components (dependency injection)
- Supported syntax:
  - XML-based definition
  - AspectJ language
  - AspectJ annotation-based development style
  - Domain Specific Languages for e.g. Transaction Management
- Aspect library

Spring AOP Weavers

- AspectJ weaver or built-in Spring weaver
- Built-in Spring weaver:
  - No external tools
  - Weaving happens automagically
  - Proxy-based:
    - only weaving on configured beans
    - as such domain classes are typically excluded from weaving
  - Only supports execution pointcuts
    - No call, field set, field get etc...

Spring/AOP Syntax & Weavers

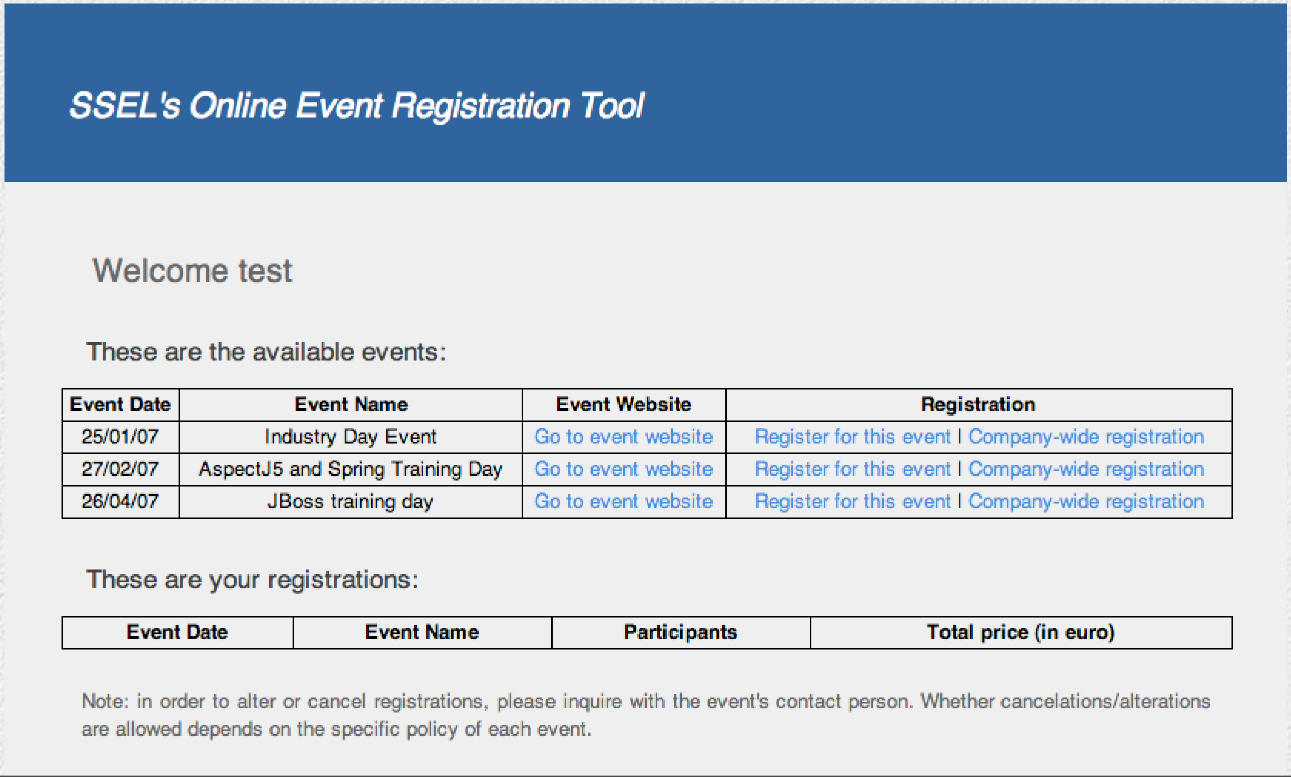
	AspectJ Language	AspectJ Annotation Style	XML Definition	DSL
Spring Weaver	No	Yes	Yes	Yes
AspectJ Weaver	Yes	Yes	No	No

Spring/AOP Syntax & Weavers

	AspectJ Language	AspectJ Annotation Style	XML Definition	DSL
Spring Weaver	No	Yes	Yes	Yes
AspectJ Weaver	Yes	Yes	No	No

Spring/AOP: case study

- Event Registration Tool
- Three aspects:
  - Transaction Management (DSL)
  - Discount Business Rule (Annotation-Style)
  - Dependency Injecting Domain Objects (Library Aspect)



SSEL's Online Event Registration Tool

Welcome test

These are the available events:

Event Date	Event Name	Event Website	Registration
25/01/07	Industry Day Event	Go to event website	Register for this event | Company-wide registration
27/02/07	AspectJ5 and Spring Training Day	Go to event website	Register for this event | Company-wide registration
26/04/07	JBoss training day	Go to event website	Register for this event | Company-wide registration

These are your registrations:

Event Date	Event Name	Participants	Total price (in euro)

Note: in order to alter or cancel registrations, please inquire with the event's contact person. Whether cancelations/alterations are allowed depends on the specific policy of each event.

Spring AOP: DSL Aspect

```
//Standard Spring XML Aspect definition
<aop:config>
    <aop:pointcut id="serviceOperation"
                expression="execution(*   regtool.service.*.*(..))"/>
    <aop:advisor advice-ref="txAdvice"
                pointcut-ref="serviceOperation"/>
</aop:config>


//Domain specific transaction management advice
<tx:advice id="txAdvice" transaction-manager="txManager">
   <tx:attributes>
    <tx:method name="get*" read-only="true"/>
    <tx:method name="*"/>
   </tx:attributes>
</tx:advice>
```

Discount Business Rule

- Registration for two events allows a discount
- Not anticipated
- Crosscuts the service, domain and presentation layers

Discount Business Rule

```
//Injecting discount info into Domain Objects
@DeclareParents(value="regtool.model.AbstractRegistration",
      defaultImpl=DefaultDiscountImpl.class)
private IDiscount discountInterface;

//Intercepting price computation to add discount
@Around(value="execution(* regtool.service.RegToolService.computeCost(..))
     && args(reg,user,event)")
public Object computePrice(ProceedingJoinPoint thisJoinPoint, AbstractRegistration
reg, User user, Event event) {
 ..... //compute discount and add discount to reg domain object
}

//Inserting the discount in the view (Spring MVC Controllers)
@AfterReturning(
    value="execution(*
         org.springframework.web.servlet.mvc.AbstractFormController+.referenceData
                    (javax.servlet.http.HttpServletRequest, java.lang.Object,..))
         && within(regtool.web.controller.*)
         && args(request,registration,..)",
    returning="referenceData"
)
public void insertDiscount(HttpServletRequest request,
         AbstractRegistration registration, Map referenceData) {
   IDiscount discount = (IDiscount) registration;
   if(discount!=null&&discount.hasDiscount())
      referenceData.put("discount", discount);
}
```

Discount Business Rule

```
//Spring Aspect Configuration (DI)
<bean id="discountAspect"
        class="regtool.service.br.BulkDiscountAspect"
        autowire="autodetect" factory-method="aspectOf">
     <property name="discountPercentage" value="0.40"/>
     <property name="viableEvents">
          <list>
          <value>1</value>
          <value>2</value>
          </list>
     </property>
     <property name="message" value="discount when
        registering both the AspectJ and JBoss Training Days"
      />
</bean>
```

Discount Business Rule

```
//enabling the Spring Weaver (listing aspects only needed in
//     mixed AspectJ weaver/Spring weaver config)
<aop:aspectj-autoproxy>
  <include name="discountAspect"/>
  <include name="myOtherAspect"/>
</aop:aspectj-autoproxy>


//disabling the Spring Weaver for Discount Aspect
<aop:aspectj-autoproxy>
  <include name="myOtherAspect"/>
</aop:aspectj-autoproxy>
```

Dependency Injection of Domain Objects

- Dependency injection only for Spring managed beans
- Domain objects are typically managed by the ORM framework
- Spring ships with "AnnotationBeanConfigurerAspect"
  - @Configurable Domain Objects have dependency injection
  - Aspect intercepts constructor of domain object

AnnotationBean-ConfigurerAspect

```
//creation of any object that we want to be configured by Spring
pointcut configuredObjectCreation(Object  newInstance,
                Configurable cAnnot)
      : initialization((@Configurable *).new(..)) &&this(newInstance) &&
        @this(cAnnot);

//ask Spring to configure the newly created instance
after(Object newInstance, Configurable cAnnot) returning
  : configuredObjectCreation(newInstance, cAnnot) {
    String beanName = getBeanName(newInstance, cAnnot);
    beanFactory.applyBeanPropertyValues(newInstance,beanName);
}


....
```

Dependency Injection of Domain Objects

```
//Domain object implementation
@Configurable("account")
public class Account {
   ...
}

//Spring configuration
<aop:spring-configured/>

<bean id="account" class="com.xyz.myapp.domain.Account"
     scope="prototype">
 <property name="fundsTransferService" ref="transferBean"/>
  ...
</bean>
```

Dependency Injection of Domain Objects

```
//Domain object implementation with auto wiring
@Configurable(autowire=Autowire.BY_TYPE,dependencyCheck=true))
public class Account {
   ...
}


//Spring configuration with auto detection
<aop:spring-configured/>
```

Common Practices

- Define reusable pointcuts
- e.g. Knowledge of system layers:

```
@Aspect
public class SystemArchitecture {

  @Pointcut("within(com.xyz.someapp.web..*)")
  public static void inWebLayer() {}

  @Pointcut("within(com.xyz.someapp.service..*)")
  public static void inServiceLayer() {}

  .....
}

//Transactional Advice
@Around("execution(* *.*(..)) && SystemArchitecture.inServiceLayer()")
...
```

Conclusion

- AspectJ and Spring, a perfect match?!
- Annotation-based aspect development style provides greatest flexibility
- Pointcuts referring to annotations instead of fragile code features
- Spring aspect library too limited