Business Technology Days

Mark Struberg / INSO, TU Vienna

CDI Extension Development

About Myself

- struberg@apache.org
- http://github.com/struberg
- http://people.apache.org/~struberg
- freelancer, programmer since 20 years
- elected Apache Software Foundation member
- Apache OpenWebBeans PMC + PMC and committer in other ASF projects: MyFaces, BVAL, OpenJPA, Maven, DeltaSpike, ...
- CDI EG member

How the ASF works

- Non profit Organization
- Contributors
- Committers
- PMC member
  (Project Management Committee)
- ASF member
- ASF board of directors

Agenda

- Short JSR-299 Overview
- The Container Lifecycle
- Behaviour Change wo Extensions: Alternatives, Specializes, Interceptors
- Extension basics
- Sample: Dynamic Interceptor
- Sample: @ViewScoped Context
- Apache DeltaSpike
- JSR-346 (CDI-1.1) preview

JSR-299 Overview

- Contexts and Dependency Injection for the Java EE platform (CDI)
- Component Model for Java (SE and EE)
- Originally designed for J2EE but also usable for standalone applications
- Based on JSR-330

CDI Features

- Typesafe Dependency Injection
- Interceptors
- Decorators
- Events
- SPI for implementing Portable Extensions
- Unified EL integration

Available Implementations

- JBoss Weld (RI)
- Apache OpenWebBeans
- Resin CanDI
- maybe Spring in the future?

A small Example

- Create a META-INF/beans.xml marker file
- Create a MailService implementation

```
@ApplicationScoped
public class MyMailService
implements MailService {
  public send(String from, String to,
              String body) {
    .. do something
  }
}
```

A small Example

- We also need a User bean

```
@SessionScoped
@Named
public class User {
  public String getName() {..}
  ..
}
```

A small Example

- Injecting the Bean

```
@RequestScoped
@Named(„mailForm“)
public class MailFormular {
  private @Inject MailService mailSvc;
  private @Inject User usr;
  public String sendMail() {
    mailSvc.send(usr.getEmail(),
                 „other“, „sometext“);
    return “successPage”;
  }
}
```

A small Example

- Use the beans via Expression Language

```
<h:form>
  <h:outputLabel value="username"
   for"username"/>
  <h:outputText id="username"
   value="#{user.name}"/>
  <h:commandButton value="Send Mail"
     action="#{mailForm.sendMail}"/>
</h:form>
```

Terms – „Managed Bean“

- The term Managed Bean means a Java Class and all it's rules to create contextual instances of that bean.
- Interface

  `Bean<T> extends Contextual<T>`
- 'Managed Beans' in JSR-299 and JSR-346 doesn't mean JavaBeans!

Terms – Contextual Instance

- The term 'Contextual Instance' means a Java instance created with the rules of the Managed Bean `Bean<T>`
- Contextual Instances usually don't get injected directly!

Terms – Contextual Reference

- The term 'Contextual Reference' means a proxy for a Contextual Instance.
- Proxies will automatically be created for injecting @NormalScope beans.

How Proxies work

- 'Interface Proxies' vs 'Class Proxies'
- Generated subclasses which overload all non-private, non-static methods.

```
public doSthg(parm) {
  getInstance().doSthg();
}
private T getInstance() {
  beanManager.getContext().get(...);
}
```

- Can contain Interceptor logic

Why use Proxies

- scope-differences - this happens e.g. if a @SessionScoped User gets injected into a @ApplicationScoped MailService.
- Passivation of non-serializable contextual instances (via it's contextual reference)
- Interceptors, Decorators

The big picture

- Creating the meta information at startup
  - Classpath Scanning and creating Managed Beans meta-information
- Contextual Instance creation at runtime
  - Based on the Managed Beans, the Context will maintain the instances
- Well defined contextual instance termination

homework: some hacking

- Create a new project from a maven archetype

  $> mvn archetype:generate -DarchetypeCatalog=
  http://people.apache.org/~jakobk/m2_archetypes_103_release

- Select 'myfaces-archetype-codi-jsf20'
- start the web application in jetty

  $> mvn clean install -PjettyConfig jetty:run-exploded

  $> tree src

Behaviour Change without CDI Extensions

Interceptors

- An Interceptor decouples technical concerns from business logic
- Applying cross cutting concerns to beans
- JSR-299 allows you to write your own interceptors
- Interceptors are treated as being @Dependent to the intercepted class

Interceptor usage

- Interceptor annotations can be applied on method or class level

```
@ApplicationScoped
public class UserService {
  @Transactional
  public storeUser(User u) {
    ...
  }
}
```

InterceptorBinding Type

- InterceptorBindings are used to identify which Interceptors should be applied to a bean

```
@InterceptorBinding
@Retention(RetentionPolicy.RUNTIME)
@Target( { ElementType.TYPE,
           ElementType.METHOD })
public @interface Transactional {
}
```

The Interceptor Impl

```
@Interceptor @Transactional
public class TransactionalInterceptor {
  private @Inject EntityManager em;
  @AroundInvoke
  public Object invoke(InvocationContext context)
  throws Exception {
    EntityTransaction t = em.getTransaction();
    try {
      if(!t.isActive()) t.begin();
      return context.proceed();
    } catch(Exception e) {
      .. rollback and stuff
    } finally {
      if(t != null && t.isActive())
        t.commit();
```

Enabling Interceptors

- Interceptors need to be **enabled manually** in beans.xml
- You can define the **order** in which multiple Interceptors stack

```
<beans>
  <interceptors>
    <class>org.mycomp.Secured</class>
    <class>org.mycomp.Transactional</class>
  </interceptors>
</beans>
```

@Specializes

- 'Replaces' the Managed Bean of a superclass
- Bean must extends the superclass
- possible to specialize Producer Methods
- or even EJBs:

```
@Stateless
public class UserSvcFacade implements UserSvc
 { ... }
@Stateless @Mock @Specializes
public class MockUserSvcFacade extends
UserSvcFacade
 { ... }
```

@Alternative

- Swap Implementations for different installations
- Kind of an optional override of a bean

```
@Alternative
public class MockMailService
implements MailService { ...
```

- Disabled by default

```
<beans>
  <alternatives>
    <class>org.mycomp.MockMailService</class>
  ...
```

Pitfalls with Alternatives and Interceptors

- Per CDI-1.0 only active for the BDA (Bean Definition Archive)
- That will officially be changed in CDI-1.1 see https://issues.jboss.org/browse/CDI-18
- OpenWebBeans always used global enablement
- Weld **now** globally enables them if defined in WEB-INF/beans.xml

Typesafe Configuration

- Config without properties or XML files
- Just create an Interface or default config class

```
public interface MyConfig{
   String getJdbcUrl();
}
```

- Later just implement, @Specializes, @Alternative, @ProjectStageActivated, @ExpressionActivated your config impl.

Events

- Observer/Observable pattern
- Inject a Event Source

```
private @Inject Event<UserLoggedIn> loginEvent;
private @Inject User usr
loginEvent.fire( new UserLoggedIn(user) );
```

- Event Consumer

```
void onLogin(@Observes UserLoggedIn ule) {
  ule.getUser().getName; ...
}
```

- Attention: don't use private observer methods!

@Typed

- Defines the Java Type which should be used for that bean
- Possible to 'disable' a bean with @Typed() to prevent AmbiguousResolutionExceptions

```
@Typed() // basically disables this Bean
public class MenuItem implements Serializable {
```

- handy for subclasses which should NOT conflict with their parent class types

Explicitly @Typed

```
@ApplicationScoped
public class TrackReadOnlyService {
  public Track readTrack(int id) {..}
}

@Typed(TrackReadWriteService.class)
@ApplicationScoped
public class TrackReadWriteService
extends TrackReadOnlyService {
  public void writeTrack(Track t) {..}
}
```

Portable Extensions

Writing own Extensions

- CDI Extensions are **portable**
- and **easy to write**!
- Are **activated by** simply **dropping** them **into** the **classpath**
- CDI Extensions are based on java.util.ServiceLoader Mechanism

Extension Basics

- Extensions will get loaded by the BeanManager
- Each BeanManager gets it's own Extension instance
- During the boot, the BeanManager sends Lifecycle Events to the Extensions
- **Attention**:
  strictly distinguish between boot and runtime!

Extension Lifecycle Events

- Extensions picked up and instantiated at Container startup
- Container sends 'system events' which all Extensions can @Observes:
  - BeforeBeanDiscovery
  - ProcessAnnotatedType
  - ProcessInjectionTarget
  - AfterBeanDiscovery
  - AfterDeploymentValidation
  - BeforeShutdown

New CDI-1.1 Lifecycle Events

- ProcessModule
  - allows modifying Alternatives, Interceptors and Decorators
- ProcessSyntheticAnnotatedType
- ProcessInjectionPoint
- Currently under discussion: ProcessAnnotatedType with @HasAnnotation(Annotation[])

Register the Extension

- Extensions need to get registered for the ServiceLoader
- Create a file

  META-INF/services/javax.enterprise.inject.spi.Extension
- Simply write the Extensions classname into it

  com.mycomp.MyFirstCdiExtension

The AnnotatedType

- Provides access to all important reflection based Information
- Can be modified via CDI Extensions in ProcessAnnotatedType
- Each scanned Class is one AnnotatedType
- @since CDI-1.1: BeforeBeanDiscovery#addAnnotatedType
- Typically changed in ProcessAnnotatedType

Sample: Dynamic Interceptor

- https://github.com/struberg/InterDyn

```
public class InterDynExtension
implements Extension {
public void processAnnotatedType(
    @Observes ProcessAnnotatedType pat) {
 for (InterceptorRule rule : interceptorRules)
  if (beanClassName.matches(rule.getRule())){
    AnnotatedType at = pat.getAnnotatedType();
    at= addInterceptor(at, rule);
     ...
    pat.setAnnotatedType(at);
 }  ...
```

Popular Extensions

- Apache MyFaces CODI
  http://myfaces.apache.org/extensions/cdi
- JBoss Seam3
  http://seamframework.org/Seam3
- CDISource
  https://github.com/cdisource
- Apache DeltaSpike
  https://cwiki.apache.org/confluence/display/DeltaSpike/Index

Vetoing Beans

- ProcessAnnotatedType#veto()
- Can be used to dynamically disable a Bean
- DeltaSpike#Exclude

DeltaSpike @Exclude

- Used to disable bean depending on a certain situation
- @Exclude()
- @Exclude(ifProjectStage=ProjectStage.Development)
- @Exclude(exceptIfProjectStage=ProjectStage.Development)
- @Exclude(onExpression=”myproperty=somevalue”)

BeanManagerProvider

- Access to the BeanManager in places where it cannot get injected
  - JPA Entity Listners
  - Tomcat Valves
  - JMX Beans
  - ServletListener or ServletFilter if not running in EE Server

BeanProvider

- #injectFields Used to fill all injection points of a existing instance

Implementing an own Scope

- via Scope annotation:

```
@Target( { TYPE, METHOD, FIELD})
@Retention(RUNTIME)
@NormalScope(passivating=true)
@Inherited
public @interface SpecialScoped
{}
```

Programmatically add Scope

- add random annotation as Scope via System Event

```
public void addViewScoped(
    @Observes BeforeBeanDiscovery bbd){
  // Scope class, normal, passivating
  bbd.addScope(ViewScoped.class, true, true);
}
```

Register the Context

- You need to tell the CDI container which Context should be serve this Scope

```
public void registerViewContext(@Observes
  AfterBeanDiscovery abd) {
  abd.addContext(new ViewScopedContext());
}
```

Implementing the Context

- A Context stores the 'Contextual Instances'
- Implements `javax.enterprise.context.spi.Context`

The needed Interface

- getScope() → defines the Scope the Context serves
- isActive() → ContextNotActiveException
- BeenManager#createCreationalContext
- get() with and wo CreationalContext
- Contextual#create(CreationalContext)
- Contextual#destroy(instance, cc)

The CreationalContext

- Stores all Explicit @Dependent Objects:
  - injected @Dependent Objects
  - Instance<T> for @Dependent objects since CDI-1.1
- Also stores all Implicit @Dependents:
  - Interceptors
  - Decorators
- needed to destroy a Contextual Instance

The Bean lookup

- See BeanProvider#getContextualReference
- BeanManager#getBeans(type, qualifiers) gets all Beans which can
  - serve the type of the Injection Point and
  - have the given qualifiers
  - are not disabled via @Alternatives or @Specializes
- BeanManager#resolve()

'BeanBags'

- Contextual Instances will be looked up via it's Bean extends Contextual
- For each Contextual Instance stored in the Context you need:
  - The actually Contextual Instance
  - The Bean
  - The CreationalContext

Lifecycle Ending

- uses requestDestroyed, sessionDestroyed, JSF destroy events etc
- properly destroy all Contextual instances if the container shuts down
- Possible via BeforeShutdown

Legal stuff

- Apache, OpenWebBeans, MyFaces, OpenEJB and OpenJPA are trademarks of the Apache Software Foundation
- Seam3 and Weld are trademarks of JBoss Inc
- CanDI is a trademark of Caucho Inc