Apache MyFaces CODI

Mark Struberg,
INSO TU-Vienna

About Myself

- struberg@yahoo.de
- struberg@apache.org
- http://github.com/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, ...
- CDI EG member

Agenda

- CODI Reasons and History
- JSR-299 – writing a portable Extension
- CODI Core highlights
- CODI Scopes
- JSF-Support-Features
- Typesafe Navigation
- Database Utils
- Compatibility with Seam3 and others

What is CODI

- MyFaces CODI is an ASF project
- http://myfaces.apache.org/extensions/cdi/index.html
- Wiki: https://cwiki.apache.org/confluence/display/EXTCDI
- use our mailing lists
  mailto://users@myfaces.apache.org
  mailto://dev@myfaces.apache.org
- download via maven or as distribution
- Bug Tracker: `https://issues.apache.org/jira/browse/EXTCDI`
- Source:

```
svn co https://svn.apache.org/repos/asf/myfaces/extensions/cdi/trunk
```

CODI History

- CODI started as collection of CDI utils
- @ViewScoped
- Shortcomings of CDI @ConversationScoped
- Contexts based on ideas of
  Apache MyFaces Orchestra got added
- typesafe navigation for JSF
- Validation and Message modules
- Focus on TypeSafety!

CODI Modules

- CODI Core
- CODI JSF-Module
- CODI Message-Module
- CODI JPA-Module
- CODI Bean-Validation
- CODI Scripting-Module
- Integration and Unit Test Module

CODI Core

- Covers all CODI tools which only needs Java SE (no Java EE dependencies!)

Writing a JSR-299 Extension

- CDI Extensions run on every EE6 container
- Easy to write!
- Are activated by simply dropping them into the classpath
- CDI Extensions are based on java.util.ServiceLoader Mechanism
- Communicate with the Container via System events (spec section 11.5. Container lifecycle events)

CODI ProjectStage

- determines the server status our project runs
- possible values: UnitTest, Development, SystemTest, IntegrationTest, Staging, Production
- Write your own one if needed!
- A Producer Method exists
- Usage:

```
private @Inject ProjectStage ps;
if (ps == ProjectStage.Production) {..
```

@ProjectStageActivated

- Sample Code from a CODI Extension
- Allows to enable Beans depending on ProjectStages
- Defining the annotation is easy:

```
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.TYPE})
public @interface ProjectStageActivated {
    Class<? extends ProjectStage>[] value();
}
```

@ProjectStageActivated

- As is the Extension implementation

```
public class ProjectStageActivationExtension
  implements Extension {

 protected void vetoAlternativeTypes(
     @Observes ProcessAnnotatedType pat) {
  if(hasProjectStageActivatedAnnotation(pat)){
    if (isInProjectStage(activatedIn)) {
      return;
    }
  // otherwise alternative shall not get used
  pat.veto();
 }
```

@ProjectStageActivated

- Extensions need to get registered in the ServiceLoader
- Create a file
  `META-INF/services/javax.enterprise.inject.spi.Extension`
- Write your Extensions classname into it
  `org.apache...ProjectStageActivationExtension`

@ProjectStageActivated

- Usage:

```
@ApplicationScoped
@Alternative
@ProjectStageActivated(ProjectStage.UnitTest.class)
public class MockSecurityService
    implements SecurityService {
...

@ApplicationScoped
@Alternative
@ProjectStageActivated(
 {UnitTest.class, Development.class})
public class MockMailService
    implements MailService {
```

Let's do 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
```

CODI Logger

- Serializable Version of java.util.logging.Logger
- Beause jul.Logger is *not* Serializable and therefor cannot be injected into passivating scoped beans!

ClassDeactivator

- All Extensions are enabled by default
- Single Extensions can be 'disabled' via the ClassDeactivator mechanism
- Useful if single Extensions clash with Extensions from other libraries (e.g. Seam3)

CODI Scopes

- CODI provides new scopes for CDI

CDI Standard Scopes

- package javax.enterprise.context
- @RequestScoped
- @SessionScoped
- CDI @ConversationScoped
- @ApplicationScoped
- (@Dependent)

New CODI Scopes

- @WindowScoped
- CODI @ConversationScoped
- Grouped Conversations
- @ViewScoped
- @ViewAccessScoped
- JSF Scope Converter Extension

@WindowScoped

- Basically a Session for a window Tab
- Still no native browser support for window handling.
- multiple strategies for handling the 'windowId'
  - windowId action URL parameter
  - window.name on intermediate page
  - lazy context dropping

@WindowScoped sample

```
@WindowScoped
public class SearchListBean
  implements Serializable {
  //...
}
```

windowId drop script

- META-INF/resources/js/windowId.js

```
function assertWindowId() {
    var freshWindow = window.name.length < 1;
    if (freshWindow) {
        url = urlWithoutWindowId(window.location.href);
        window.name = "window";
        window.location = url;
    }
}
```

- layout facelet template:

```
<h:outputScript name="windowId.js" library="js"/>
<script type="text/javascript">
    assertWindowId()
</script>
```

CODI @ConversationScoped

- Shortcomings of the builtin CDI Conversation
  - starts lazily – transient by default
    -> cannot open conversation early enough
  - no window separation
  - no parallel access allowed
    -> problems with AJAX
  - only one Conversation at a time
  - ...

CODI @ConversationScoped

- roughly based on MyFaces Orchestra concepts
- bound to a single bean by default
- eagerly created
- can be immediately terminated manually
- can be restarted manually

CODI @ConversationScoped

- Example

```
@ConversationScoped
public class EditCarWizardBean
 implements Serializable {
  private @Inject Conversation conv;

  public String saveCar() {
    // savethecar...
     conv.close(); return null;
  }
  public String resetDialogue() {
    //
    conv.restart(); return null;
  }
}
```

Grouped Conversations

- Grouping beans which belong to the same conversation
- @ConversationGroup as marker Qualifier with a *binding* value!
- Terminating a whole group possible

Using ConversationGroups

- Defining the marker interface

```
interface CarEditConversationGroup {}
```

- Using it on various backing beans

```
@Named
@ConversationScoped
@ConversationGroup(CarEditConversationGroup.class)
public class ConversationDemoBean1
  implements Serializable {
```

Using 1 Bean for n Conversations

- Create ConversationGroups via Producers

```
@Produces
@ConversationScoped
@ConversationGroup(UseCase1.class)
public DemoBean createDemoBeanForUseCase1() {
  return new DemoBean("createDemoBeanForUseCase1");
}

@Produces
@ConversationScoped
@ConversationGroup(UseCase2.class)
public DemoBean createDemoBeanForUseCase2() {
  return new DemoBean("createDemoBeanForUseCase2");
}
```

@ViewScoped

- CDI Context making the JSF ViewMap available for @Inject
- uses the standard JSF annotation

```
@javax.faces.bean.ViewScoped
```

- Not an own storage but only a CDI access to the JSF ViewMap
- get's cleaned up whenver a customer leaves the view.

@ViewAccessScoped

- Has nothing to do with @ViewScoped
- kind of 'automatic Conversation'
- will be closed if the bean isn't accessed on a view anymore

Converting JSF Scopes to CDI

- A classical JSF managed bean

```
@Managed
@javax.faces.bean.SessionScoped
public class MyNiceLittleBunnyBackingBean {..
```

- MappedJsf2ScopeExtension active by default
- only JSF beans without @ManagedBeans
- very useful if IDEs import the wrong package
- Attention:
  - default scope JSF: @RequestScoped
  - default scope CDI: @Dependent

@JsfPhaseListener

```
@ProjectStageActivated(Development.class)
@Advanced @JsfPhaseListener
public class DebugPhaseListener implements
PhaseListener {
  @Inject private Logger logger;
  public void beforePhase(PhaseEvent pe) {
    this.logger.info("before " + pe.getPhaseId());
  }
  public void afterPhase(PhaseEvent pe) {
    this.logger.info("after " + pe.getPhaseId());
  }
  public PhaseId getPhaseId() {
        return PhaseId.ANY_PHASE;
  }
}
```

@InvocationOrder

- Used to order various CODI helpers
- default order = 1000

```
@JsfPhaseListener
@InvocationOrder(10)
public class DebugPhaseListener
  implements PhaseListener {
  //...
}
```

JSF Lifecycle Observers

- Annotations
  - @BeforePhase
  - @AfterPhase
- Example

```
public void preInvokeApplication(
  @Observes @BeforePhase(ANY_PHASE)
  PhaseEvent event) {
  //...
}
```

TypeSafe JSF Navigation

- JSF-2 navigation is based on strings
  - hard to refactor
  - error prone
- CODI provides a type safe approach
  - Create your ViewConfig as class
  - Create your page structure as classes
  - hold your view definition in exactly 1 place
  - @Secured for pages, etc

A simple ViewConfig

- If no name given, the classname matches the JSF page (without .xhtml)

```
@Page
public final class Home implements ViewConfig {
}
```

- Or name your page explicitely and set modes

```
@Page(name="puh",
    navigation=NavigationMode.REDIRECT,
    viewParams=ViewParameterMode.INCLUDE)
public final class Home implements ViewConfig {
}
```

A simple Wizzard

- Convenient behaviour for corresponding pages:

```
@Page(navigation = REDIRECT)
public abstract class Wizard implements ViewConfig{
   @Page
   public final class Step1 extends Wizard {
   }
   @Page
   public final class Step2 extends Wizard {
   }
}
```

- Leads to `/wizzard/step1.xhtml` and `/wizzard/step2.xhtml`

Usage in the backing bean

- Use the ViewConfig class for your navigation case

```
public Class<? extends ViewConfig> showNext() {
  return Page2.class;
}
```

Persistence without EJB

- EJBs are transaction aware by default
- CDI beans don't have any default behaviour (besides the proxying)
- CODI provides a **@Transactional** annotation plus **TransactionalInterceptor**
- supports nested transactions
- even across multiple Databases

Create your EntityManager

- The ONLY working pattern is entitymanager-per-request
- All others (putting EntityManager in to the Session) is NOT working in clusters.

Produce your EntityManager

```
@RequestScoped
public class EntityManagerProducer {
  private @PersistenceContext(unitName="Course")
            EntityManager entityManager;
  @Produces @RequestScoped @QCourse
  public EntityManager createEntityManager() {
        return entityManager;
  }
  public void dispose(@Disposes @QCourse
                      EntityManager entityManager) {
    entityManager.close();
  }
}
```

@Transactional Service

```
@Transactional(qualifier=@QCourse)
public class MyUserService {
  private @Inject @QCourse EntityManager em;

  public User getUser(long id) {
    return em.find(id, User.class);
  }
}
```

CODI Messages

- Easily create internationalized Messages
- Support for creating FacesMessages

```
@Inject @Jsf
private MessageContext messageContext;

messageContext.message()
   .text("{msgUserRegistered}")
   .namedArgument("userName", user.getUserName())
   .create();
messageContext.message()
   .text("{msgLoginFailed}")
   .payload(ERROR)
   .add();
```

Documentation

- JSR-299 spec:
  http://jcp.org/en/jsr/detail?id=299
  http://jcp.org/en/jsr/detail?id=346
- OpenWebBeans:
  http://openwebbeans.apache.org
  svn co http://svn.apache.org/repos/asf/openwebbeans/trunk
- CODI
  https://cwiki.apache.org/confluence/display/EXTCDI
  svn co http://svn.apache.org/repos/asf/myfaces/extensions/cdi/trunk

Legal stuff

- Apache, OpenWebBeans, MyFaces, CODI, 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