
JUDCon
JBoss Users & Developers Conference
Boston : 2010

Using CDI and OSGi Services in JBoss AS

Aleš Justin, JBoss by Red Hat
Kabir Khan, JBoss by Red Hat

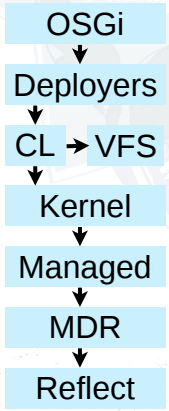
JUDCon:2010 JBoss Users & Developers Conference : Boston

Agenda

- JBoss Microcontainer (MC) overview
- Component mixture introduction
- Weld (CDI)/MC integration
- OSGi/MC integration
- Demo
- Q&A

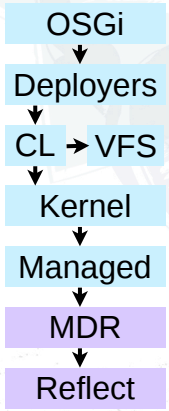
Microcontainer Overview

Microcontainer overview



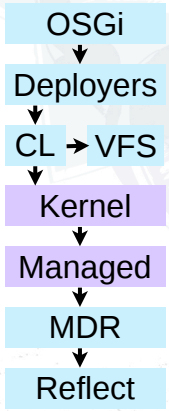
- Replaces the old JMX microkernel in JBoss AS >= 5
- Dependency Injection Framework
- Understands service lifecycle at its core
- Services/Beans as POJOs
  - JMX still supported
- Microcontainer is “umbrella project”

Microcontainer overview



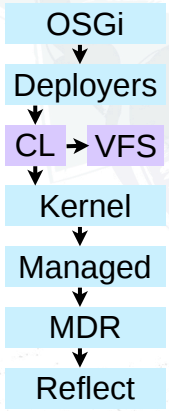
- JBoss Reflect
  - Classes/members abstraction similar to java.lang.reflect
    - MethodInfo[] m = ClassInfo.getDeclaredMethods()
  - Optimized
- MetaData Repository
  - Scoped metadata
  - Bean instance annotations via XML
  - Global metadata, e.g. @EJBPool per server

Microcontainer overview



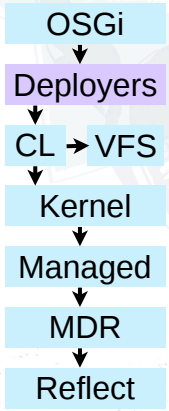
- JBoss Managed
  - @ManagedObject/-Property/-Operation/-Parameter
  - Clean client/server separation
  - Admin console
- JBoss Kernel
  - The “brain”
  - Core dependency injection framework + state machine
  - Extensible component model
    - POJO, JMX, Spring, Guice, Weld/CDI, OSGi

Microcontainer overview



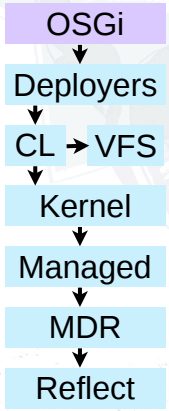
- JBoss Virtual Filesystem
  - Unified resources lookup
    - file/dirs accessed same as zip
    - extensible protocols
- JBoss ClassLoading
  - jboss-classloading.xml
  - Mimics UnifiedClassLoaders (default)
  - Configurable policies
    - import/export/version - OSGi...

Microcontainer overview



- Virtual Deployer Framework
  - Aspectized deployers in a chain
    - Deployment stages
    - DeploymentUnit attachments
      - Relevant deployers
      - Ordering -> inputs/outputs
  - Reuse existing deployers
    - e.g. xml parsing, classloader setup etc.

Microcontainer overview



- OSGi
  - Embedded - existing framework runs as service
  - Runtime - MC as set of services
  - Native - implement framework on top of MC *

Component Mixture Introduction

What is a component model mixture?

- What do we consider a component model?
- First of all, what do we consider a component?

One abstract way to express this would be that “components are reusable software programs that you can develop and assemble easily to create sophisticated applications.” To consider a bunch of components as an actual model, we also need to declare what kind of interactions we allow.

What is a component model mixture?

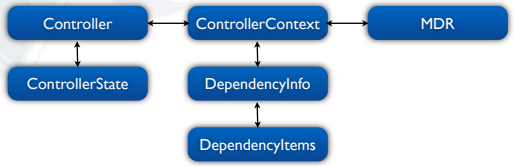
- Cross dependencies
- Transparent interaction
- Easy to add new behavior
- Easy to add new component model(s)

How do we support this “mixture”?

- Complete rewrite of the JBoss Kernel
- JMX MicroKernel ➠ MicroContainer (MC)
- Designed from ground up to support this
- Constant strive to support this vision
- Generalization of custom ideas

Mixture's design

- MDR - metadata repository
- Dependency - MC's component abstraction



Where are we now?

- POJOs - default, Weld *, Guice, Spring
- MBeans / JMX - legacy
- Aliases
- Deployments
- OSGi services *

Weld/MC Integration

MC/Weld Integration Overview

- Allow to inject MC beans <-> Weld/CDI beans
- Weld and MC are separate entities
- MC -> Weld
  - InjectionServices
  - Push MC beans to BeanManager
    - @WeldEnabled
- Weld -> MC
  - @Inject @Weld

InjectionServices

- MC -> Weld
- Weld SPI
- Pull beans not found in BeanManager from external source
  - Fails in validation step :-(

```
public class WeldBean{
   @Inject ExternalBean bean;
}

public class McLookupInjectionServices implements InjectionServices{

  public<T>  void aroundInject(InjectionContext<T>  ctx){
     ctx.proceed();
     //lookup non injected members in ctx.getInjectionTarget() from MC
  }
}
```

Push to Weld

```
@Default
public class WeldBean{
   @Inject
   WeldUser weldUser;

   @Inject
   McBean mcBean;
}

@Default
public class WeldUser{
}
```

```
@WeldEnabled
@Default
public class McBean{

  @Inject
  OtherMcBean otherMc;

  @Weld @Inject
   WeldUser weldUser;
}

public class OtherMcBean{
}
```

MC lifecycle

- Bean’s lifecycle goes through a number of states
  - NOT_INSTALLED
  - PRE_INSTALL
  - DESCRIBED
  - INSTANTIATED
  - CONFIGURED
  - CREATE
  - START
  - INSTALLED
- Dependencies checked before progression to each state

Example lifecycle action

- Handler for each state

```
public class DescribeAction extends AnnotationsAction
{
   protected void installActionInternal(KernelControllerContext ctx)
      throws Throwable
   {
      //Add dependencies coming from AOP
      ...
      super.applyAnnotations(ctx); //Handles @Inject etc.
   }
}
```

- KernelControllerContext -wrapper around bean
  - BeanMetaData - name, type, properties, dependencies...
  - Instance
  - State actions

@WeldEnabled

- DescribeAction.applyAnnotations()
- Contexts are put into WeldFromMcRegistry
- CDI triggers BeforeBeanDiscovery event

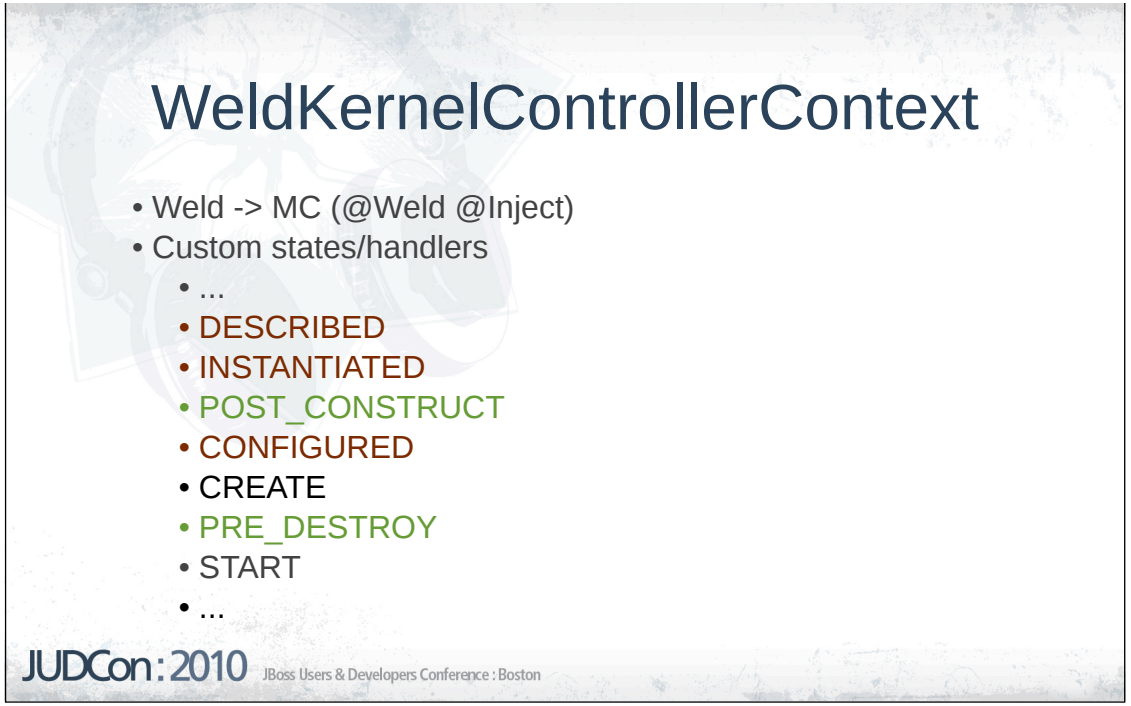
```
public class McBeanRegistryObserver implements Extension{
   ...
   public void addType(@Observes BeforeBeanDiscovery event,
          BeanManager beanManager){

      WeldFromMcRegistry.getInstance().initializeTypes(beanManager);
      Collection<AnnotatedType> types;
      for (AnnotatedTypeWrapper typeWrapper :
       WeldFromMcRegistry.getInstance().getTypes())
         event.addAnnotatedType(typeWrapper.getAnnotatedType());
   }
```

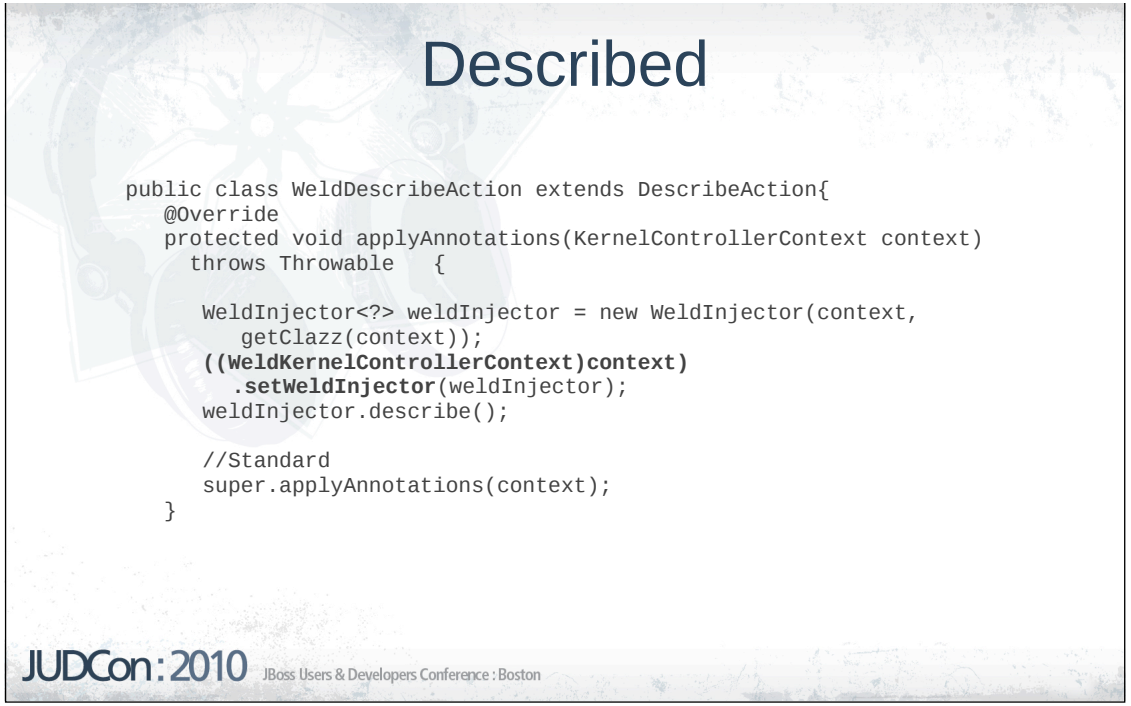
@WeldEnabled (2)

- CDI triggers ProcessInjectionTarget event
  - EIIT.produce() -> context.getTarget()
- All CDI interaction spec defined

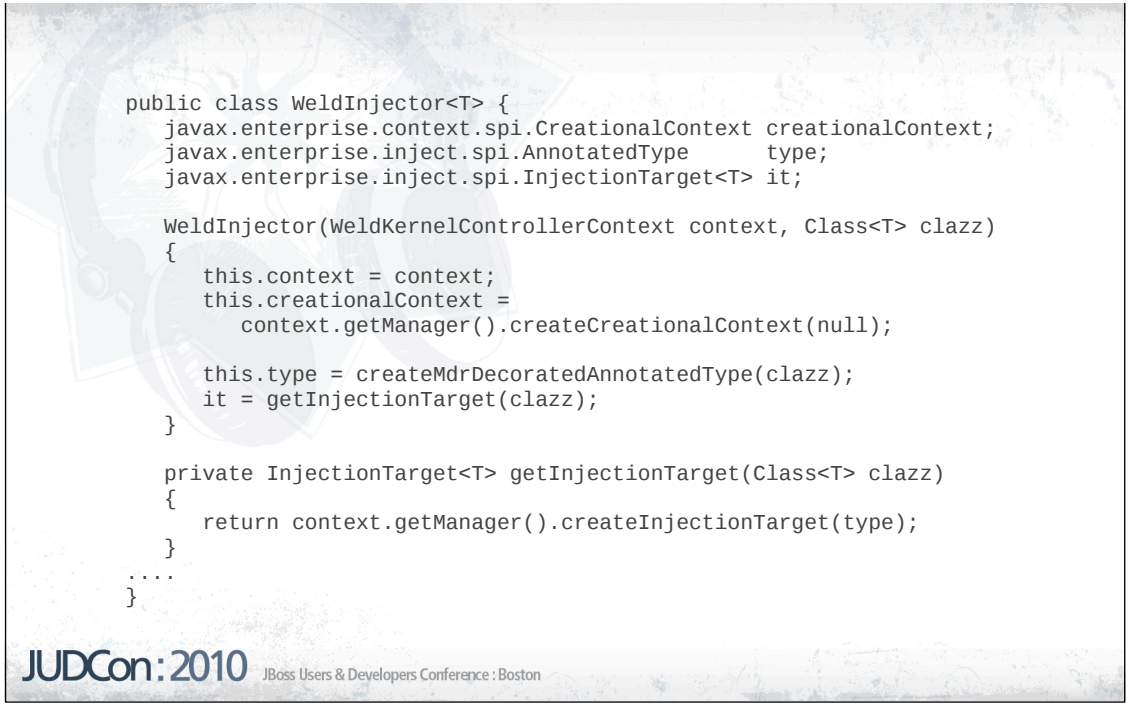
```
public class McBeanRegistryObserver implements Extension{
   ...
   public <X> void processInjectionTarget(@Observes
         ProcessInjectionTarget<X> event){
      AnnotatedType<?> type = event.getAnnotatedType();
      KernelControllerContext context =
          WeldFromMcRegistry.getInstance().getContext(type);
      if (context != null){
         InjectionTarget<X> target = event.getInjectionTarget();
         ExistingInstanceInjectionTarget<X> tgt =
            new ExistingInstanceInjectionTarget<X>(target, context);
         event.setInjectionTarget(tgt);
      }
   }
```



To have injection between Weld and the MC we need to deploy our beans as WeldKernelControllerContexts.
For the WeldKernelControllerContexts we have extended a few of the default state actions and added a few extra to handle @PostConstruct and @PreDestroy.



The custom describe action does all the normal work, as well as initialising the WeldInjector and adding it to the context.



The WeldInjector is our interface into CDI
It is initialised with a CDI creational context, a CDI annotated type representing the bean class which allows for annotations to also come from MC metadata, and a CDI injection target which is used to perform injection.
All the interaction is via the CDI spec defined API

Instantiated

```
public class WeldInstantiateAction extends InstantiateAction{
   @Override
   protected void installActionInternal(
     KernelControllerContext context) throws Throwable{

      WeldInjector<?> injector =
        ((WeldKernelControllerContext)context).getWeldInjector();
      if (injector.createInWeld()){
         //Constructor had @Weld @Inject
         context.setTarget(injector.instantiate());
      }
      else
         super.installActionInternal(context);
   }
}
public class WeldInjector<T> {
   ...
   T instantiate(){
      T t = it.produce(creationalContext);
      return t;
   }
}
```


During the instantiate stage we check if CDI should be in charge of instantiating the bean by checking if the constructor had the @Weld @Inject annotations.
If CDI should instantiate the bean we delegate to the WeldInjector which calls InjectionTarget.produce() to obtain the instance and inject the constructor parameters via Weld.
Otherwise, we just delegate to the normal instantiate action which uses the Microcontainer to do the instantiaton and constructor parameter injection.

Configured

```
public class WeldConfigureAction extends ConfigureAction{
   @Override
   protected void installActionInternal(
     KernelControllerContext context) throws Throwable{
      //Normal MC property configuration & injection
      super.installActionInternal(context);
      //@Weld @Inject annotated members
      WeldInjector<?> injector =
       ((WeldKernelControllerContext)context).getWeldInjector();
      injector.inject();
   }
}

public class WeldInjector<T> {
   ...
   void inject(){
      it.inject((T)context.getTarget(), creationalContext);
   }
}
```


In the configure state we delegate to the standard implementation for normal Microcontainer injection.
Then we delegate to the Weld injector to inject into @Wed @Inject annotated members
The Weld injector delegates to the CDI injection target to do the injection

PostConstruct/PreDestroy

```
public class WeldPostConstructAction extends InstallsAwareAction{
   @Override
   public void installActionInternal(KernelControllerContext context){
      WeldInjector<?> injector =
       ((WeldKernelControllerContext)context).getWeldInjector();
      injector.postConstruct(context.getTarget());
   }
}

public class WeldInjector<T> {
   ...
   void postConstruct(Object instance) {
      it.postConstruct((T)instance);
   }

   void preDestroy(Object instance) {
      it.preDestroy((T)instance);
      creationalContext.release();
   }
}
```


@PostConstruct annotated methods are invoked in the PostConstruct install phase
@PreDestroy annotated methods are invoked in the PreDestroy uninstall phase
Again both of these scenarios are handled by the WeldInjector delegating to the InjectionTarget.

OSGi Integration

How is this related to OSGi?

- Reusing existing MC concepts
  - Bundles == Deployments
  - Services == “beans”
- Same level of mixture
- Step further - other components “pretending” to be OSGi-like

✦ New MC OSGi facade = Core Framework

Features and issues

- Services / beans interaction
- Everything can look like OSGi
- Customizing POJO view

Features and issues

- Component tracking
- Different undeploy CL behavior
- New dependency resolution rules
- Lazy CL callback
- OBR integration

Demo

In subversion:
http://anonsvn.jboss.org/repos/jbossas/projects/demos/microcontainer/trunk/

Summary

- JBoss MC supports several component models
  - Native: POJO, JMX, OSGi
  - External: Weld, Guice
  - Easy to extend for other models
- Project page and contact
  - http://jboss.org/jbossmc
  - ales.justin@jboss.org
  - kabir.khan@jboss.com
- Q&A