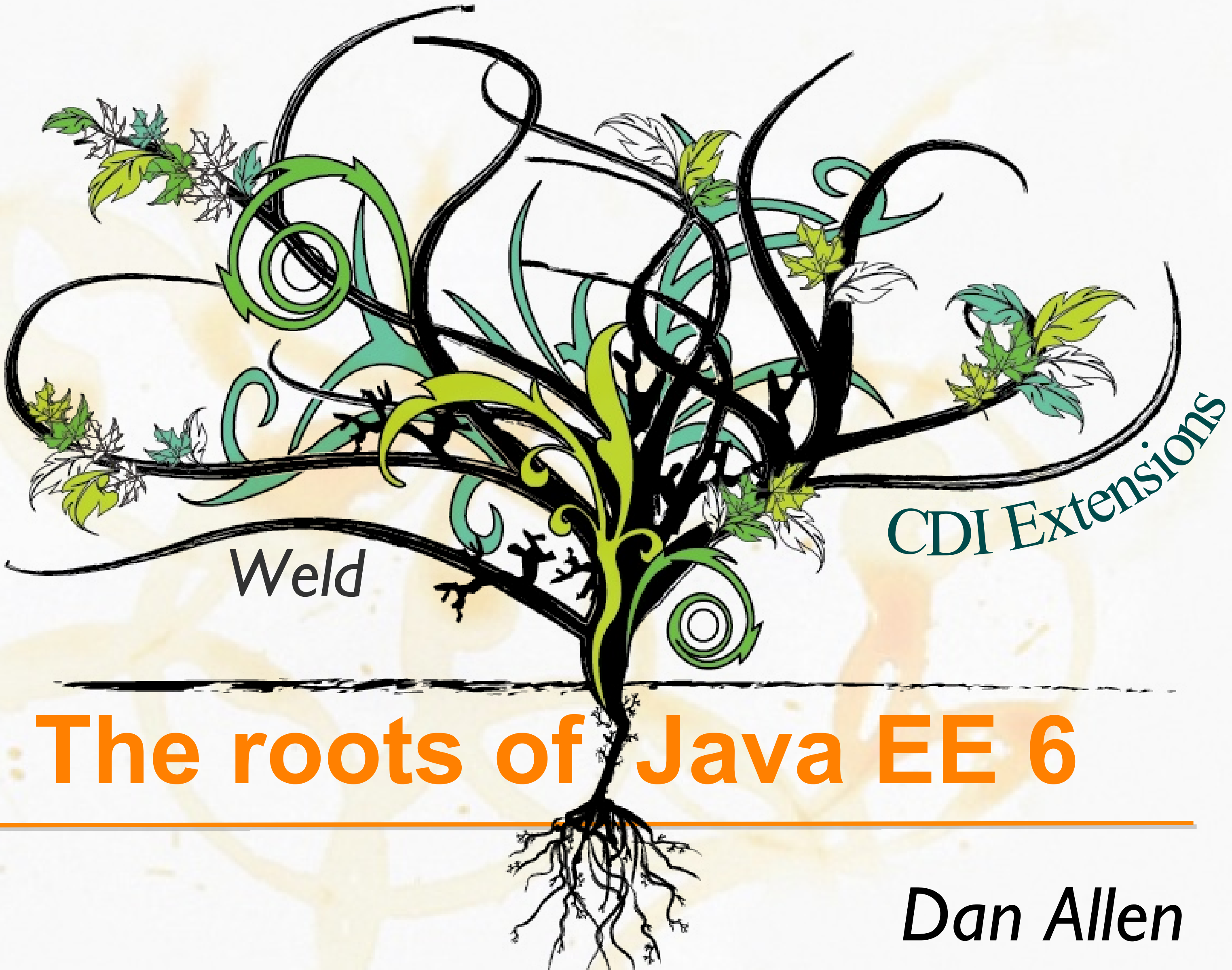
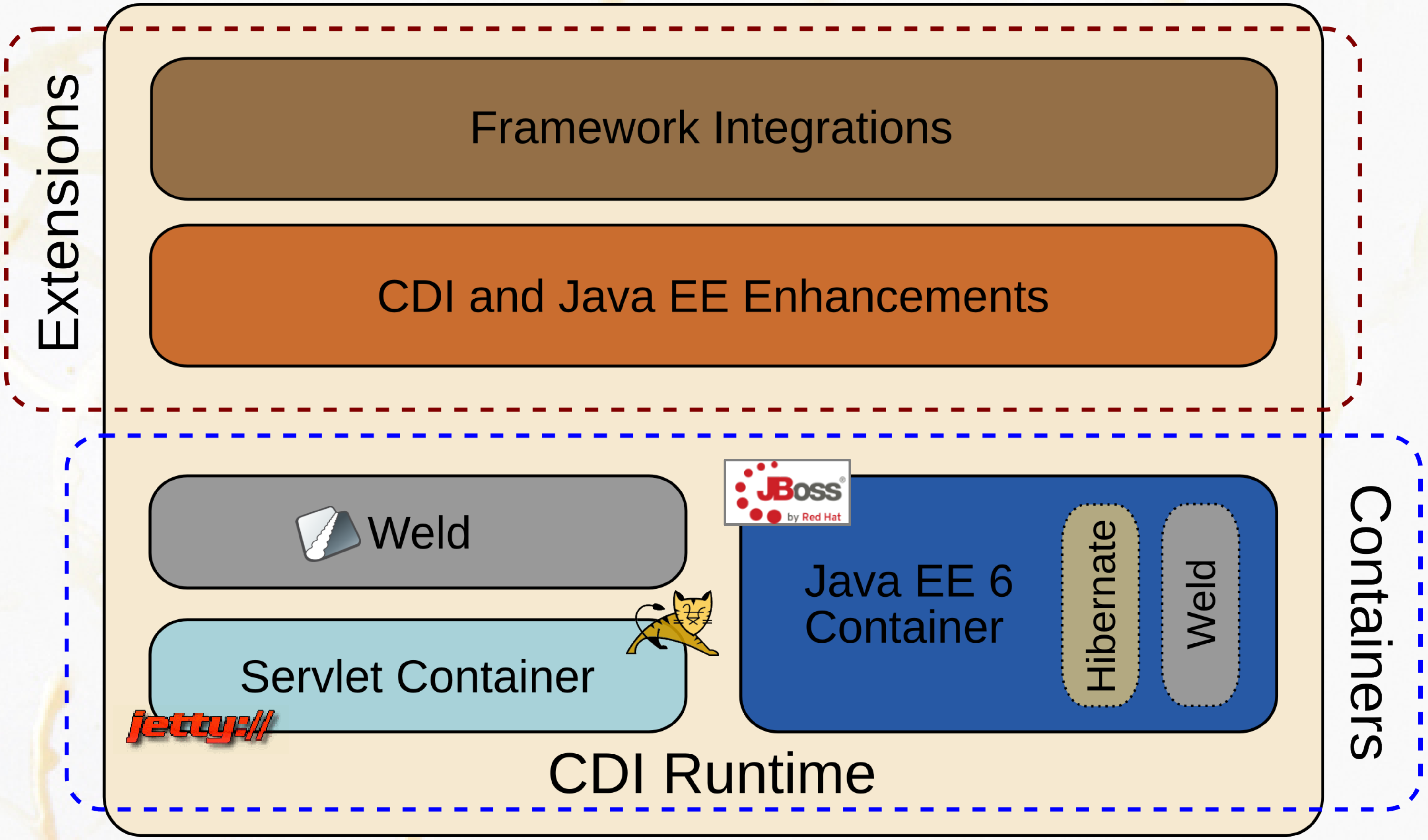


The roots of Java EE 6

Dan Allen
Principal Software Engineer
JBoss, by Red Hat

Building on common ground



Weld

JSR-299 Reference Implementation & TCK with support for Servlet and Java SE

The axis of CDI

Implement
(RI)

Validate
(TCK)

Broaden
(Servlet, SE)

Document
(Tutorial)

What JSR-299 (CDI) provides

- *Services* for Java EE components
  - Lifecycle management of stateful beans bound to well-defined contexts
  - A type-safe approach to dependency injection
  - Interaction via an *event notification* facility
  - Reduced coupling between interceptors and beans
  - Decorators, which intercept specific bean instances
  - Unified EL integration (bean names)
- *An extension SPI*
  - Fosters an ecosystem of portable extension libraries for the Java EE platform

What JSR-299 (CDI) provides

Java EE architecture =

flexible + *portable* + *extensible*

♫ ♪ ♫ ♪ ♫ ♪

A type-safe programming model

`@Annotation` `Type`

Reaching deep into the Java type system

`<TypeParam>` `Method`

Loose coupling

- Decouple **server and client**
  - Contract based on well-defined types and "qualifiers"
- Decouple **lifecycle of collaborating components**
  - Stateful components interact like services
- Decouple **orthogonal concerns (AOP)**
  - Interceptors & decorators
- Decouple **message producer from consumer**
  - Events

Strong typing

- Type-based injection
  - Eliminate reliance on string-based names
  - Refactor friendly
- Compiler can detect typing errors
  - No special authoring tools required
  - Casting eliminated (or at least hidden)
- Resolution errors detected at application startup
- Strong typing == strong tooling
  - Preemptively detect errors
  - Navigate relationships

Bean ingredients

- Bean class
- Set of bean types
- Set of qualifiers

$\left.\begin{array}{l} \\ \end{array}\right\}$ DI contract

- Scope
- EL name (optional)
- Set of interceptor bindings
- Alternative classification
- Set of stereotypes
- Set of injection points

Welcome to CDI, managed bean!

```java
public class Welcome
{
   public String buildPhrase(String city)
   {
      return "Welcome to " + city + "!";
   }
}
```

Injection 101

```
public class Greeter
{
   @Inject
   private Welcome w;

   public void welcome()
   {
      System.out.println(w.buildPhrase("New York"));
   }
}
```

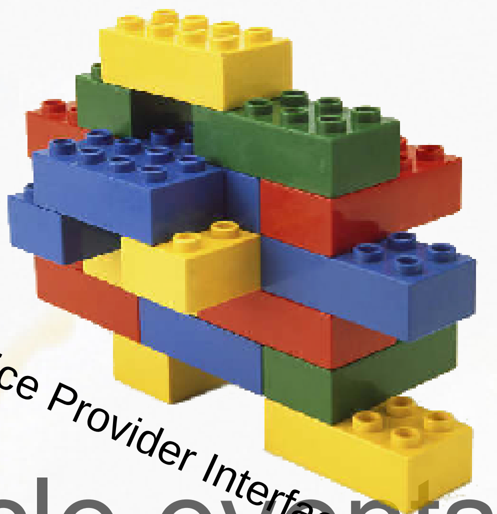
Qualifying an implementation

```
@Translating
public class TranslatingWelcome extends Welcome
{
   @Inject
   private GoogleTranslator translator;

   public String buildPhrase(String city)
   {
      return translator.translate("Welcome to " + city + "!");
   }
}
```

Using qualifier as “binding” type

```java
public class Greeter
{
   @Inject @Translating
   private Welcome w;

   public void welcomeVisitors()
   {
      System.out.println(w.buildPhrase("New York"));
   }
}
```

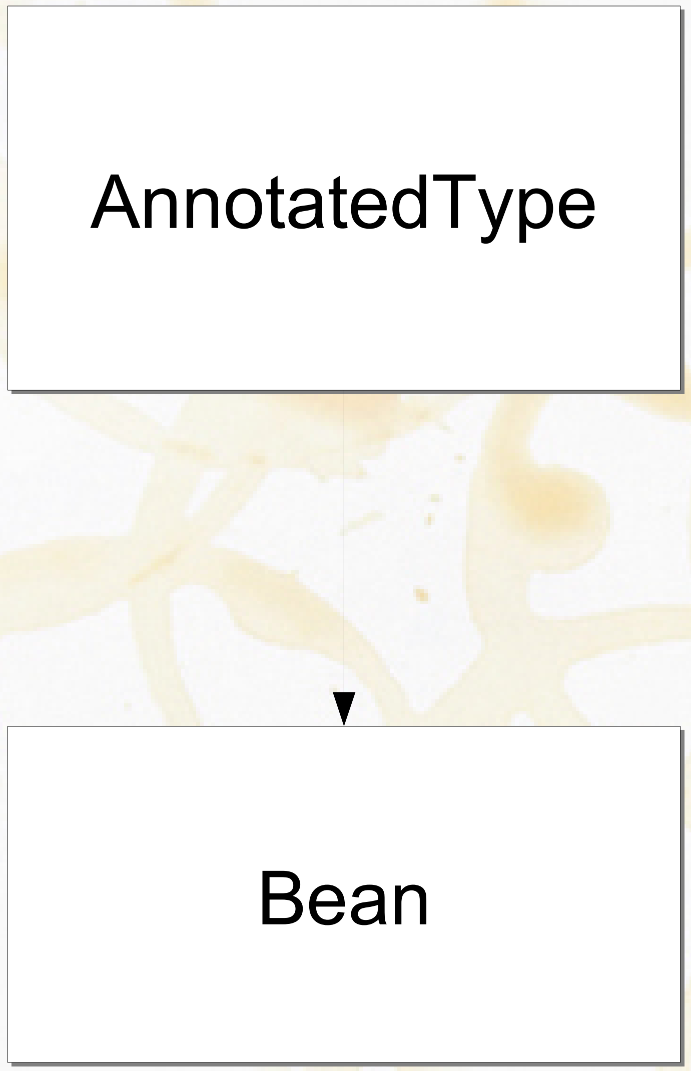
Portable extensions



- Implement Extension SPI
- Hook in by observing container lifecycle events
- Ways to integrate with container
  - Provide beans, interceptors or decorators
  - Satisfy injection points with built-in or wrapped types
  - Contribute a scope and context implementation
  - Augment or override annotation metadata

Deployment hooks

- BeforeBeanDiscovery
- ProcessAnnotatedType
- ProcessInjectionTarget
- ProcessBean*
- ProcessObserverMethod
- ProcessProducer*
- AfterBeanDiscovery
- AfterDeploymentValidation
- BeforeShutdown


AnnotatedType
Bean

Hacking Java EE

- Interceptor bindings
  - Declarative transaction boundaries
  - Declarative conversation boundaries
- Additional scopes
  - @ViewScoped
  - @TransactionScoped
- Event bridges
- Producers for implicit objects
- Type-narrowing producers
- Annotation aliasing
- Alter injection point
- Veto beans
- Built-in handlers for interfaces

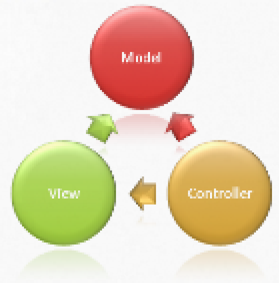
Java EE is your oyster

SeamSolder

a library of generally useful stuff for CDI programming and extension authoring

Something for everyone

- Developers
  - Enhancements to the CDI programming module
- Extension authors
  - Type metadata and bean utilities
- Framework authors
  - Configuration extensions



A swiss army knife for extensions

- AnnotatedType builder
- Annotation/meta-annotation inspector
- Annotation instance provider
- Reflection utilities
- Narrowing bean builder
- JavaBean property utilities
- Method injector
- Bean utilities

Bean veto

```
@Veto
public class WorkInProgress…
```

```
@Veto
@Entity
public class Employee…
```

Disabling vetoed beans

```java
public class VetoExtension implements Extension
{
   <X> void processAnnotatedType(@Observes ProcessAnnotatedType<X> event)
   {
      AnnotatedType<X> type = event.getAnnotatedType();
      if (type.isAnnotationPresent(Veto.class))
      {
         event.veto();
      }
   }
}
```

Bean requires class

```
@Requires("javax.persistence.EntityManager")
class EntityManagerProducer
{
   @Produces
   EntityManager createEntityManager()…
}
```

Enforcing required prerequisites

```
public class RequiresExtension implements Extension
{
   <X> void processAnnotatedType(@Observes ProcessAnnotatedType<X> event)
   {
      AnnotatedType<X> type = event.getAnnotatedType();
      if (type.isAnnotationPresent(Requires.class))
      {
         for (String required : type.getAnnotation(Requires.class).value())
         {
            try
            {
               Reflections.findClass(required,
                  type.getJavaClass().getClassLoader());
            }
            catch (Exception e)
            {
               event.veto();
            }
         }
      }
   }
}
```

When you're feeling choosy

```
public class PaymentProcessor
{
   @Inject @Exact(CreditCardPaymentService.class)
   private PaymentService paymentService;
   …
}
```

Giving them what they want

```
public class ExactExtension implements Extension
{
   <X> void processAnnotatedType(@Observes ProcessAnnotatedType<X> e)
   {
      AnnotatedType<X> type = e.getAnnotatedType();
      AnnotatedTypeBuilder<X> builder = null;
      for (AnnotatedField<? super X> f : type.getFields())
      {
         if (f.isAnnotationPresent(Exact.class))
         {
            Class<?> type = f.getAnnotation(Exact.class).value();
            if (builder == null)
               builder = new AnnotatedTypeBuilder<X>().readFromType(type);
            builder.overrideFieldType(f, type);
         }
      }

      if (builder != null)
         e.setAnnotatedType(builder.create());
   }
}
```

Annotation aliasing

```
void processType(@Observes ProcessAnnotatedType evt)
{
   if (evt.getAnnotatedType()
      .isAnnotationPresent(ManagedBean.class)
   {
      AnnotatedType modified = new AnnotatedTypeBuilder()
         .readFromType(baseType, true)
         .addToClass(ModelLiteral.INSTANCE)
         .create();
      evt.setAnnotatedType(modified);
   }
}
```

How about XML instead?

- XML-based bean metadata
  - Define
  - Customize
  - Wire
- Originally proposed in JSR-299
- “Type-safe”
  - Uses fully-qualified types
  - “Import” types using XML namespaces
- Full capability of CDI, not watered down

Setting initial field values

```
<beans…
   xmlns:a="urn:java:org.jboss.seam.examples.auth">
   <a:Authenticator maxFailures="3"/>
   <a:PasswordManager algorithm="SHA-1" salt="devoxx"/>
</beans>
```

Repurposing a bean class

```
<beans …
   xmlns:d="urn:java:org.jboss.seam.examples.dnd">
   <d:Room><Qualifier/></d:Room>

   <d:GameRoom>
      <SessionScoped/>
      <d:Room value="start"/>
      <d:north><Inject/><d:Room value="empty1"/></d:north>
   </d:GameRoom>

   <d:GameRoom>
      <SessionScoped/>
      <d:Room value="empty1"/>
      <d:north><Inject/><d:Room value="empty3"/></d:north>
      <d:west><Inject/><d:Room value="dwarf"/></d:west>
      <d:east><Inject/><d:Room value="pit"/></d:east>
      <d:south><Inject/><d:Room value="start"/></d:south>
   </d:GameRoom>
</beans>
```

Define interceptor binding type

```
@InterceptorBinding
@Retention(RUNTIME)
@Target({TYPE, METHOD})
public @interface Transactional
{
   @Nonbinding
   TransactionAttributeType value()
      default TransactionAttributeType.REQUIRED;
}
```

Flag interceptor

```
@Transactional
@Interceptor
public class TransactionInterceptor
{
   @Inject
   private Instance<UserTransaction> transactionInstance;

   @AroundInvoke
   public Object aroundInvoke(InvocationContext ctx)
         throws Exception {
      return new TransactionWorker()
      {
         public Object doWork() throws Exception
         {
            return ctx.proceed();
         }
         ...
      }.workInTransaction(transactionInstance.get());
   }
}
```

Bind annotation to bean

```
@Transactional
public class AccountManager
{
   public boolean transfer(Account a, Account b)…
}
```

What about the native attribute?

```
@TransactionAttribute
public class AccountManager
{
   public boolean transfer(Account a, Account b)…
}
```

Swap annotations at startup

```java
public class TransactionExtension implements Extension
{
   <X> void processAnnotatedType(@Observes ProcessAnnotatedType<X> e)
   {
      AnnotatedType<X> type = e.getAnnotatedType();
      AnnotatedTypeBuilder<X> builder = null;
      boolean ejb = EjbApi.isEjb(type);
      if (type.isAnnotationPresent(TransactionAttribute.class) && !ejb)
      {
         builder = new AnnotatedTypeBuilder<X>().readFromType(type)
            .addToClass(TransactionalLiteral.INSTANCE);
      }
      for (AnnotatedMethod<? super X> m : type.getMethods())
      {
         if (m.isAnnotationPresent(TransactionAttribute.class) && !ejb)
         {
            if (builder == null)
               builder = new AnnotatedTypeBuilder<X>().readFromType(type);
            builder.addToMethod(m, TransactionalLiteral.INSTANCE);
         }
      }

      if (builder != null) e.setAnnotatedType(builder.create());
   }
}
```

Ultra EJB lite

```
@Stateless
public class AccountManager
{
   @PersistenceContext
   private EntityManager em;
   public boolean transfer(Account a, Account b)…
}
```

Declarative conversation controls

```
@ConversationScoped
public class BookingAgent implements Serializable
{
   @Inject private EntityManager em;
   private Booking booking;

   @Begin
   public void select(Hotel h)
   {
      booking = new Booking(em.find(Hotel.class, h.getId()));
   }

   @End
   public void confirm()
   {
      em.persist(booking);
   }
}
```

Feeding events to CDI

```
@WebListener
public class ServletLifecycleBridge implements ServletContextListener
{
   private static Annotation INITIALIZED =
         new AnnotationLiteral<Initialized>() {};
   private static Annotation DESTROYED =
         new AnnotationLiteral<Destroyed>() {};

   @Inject @Any
   private Event<ServletContext> bridgeEvent;

   @Override public void contextInitialized(ServletContextEvent e)
   {
      bridgeEvent.select(INITIALIZED).fire(e.getServletContext());
   }

   @Override public void contextDestroyed(ServletContextEvent e)
   {
      bridgeEvent.select(DESTROYED).fire(e.getServletContext());
   }
}
```

Observing event through CDI

```
public class ApplicationInitializer
{
   @Inject
   private ReferenceDataCache cache;

   public void setup(@Observes @Initialized ServletContext ctx)
   {
      cache.loadReferenceData();
   }
}
```

Implementation magic

- Service handler can automatically implements:
  - Interfaces
  - Abstract classes
- Call to abstract method invokes handler
  - Works like interceptor without call to proceed
- For instance...

Query service

```
@QueryService
public interface UserRepository
{
   @Query("select u from User u");
   public List<User> findAll();
}
```

```
public class QueryHandler
{
   @Inject EntityManager em;
   @AroundInvoke Object handle(InvocationContext ctx)
   {
      return em.createQuery(getQueryValue(ctx.getMethod())
         .getResultList();
   }
   …
}
```

```
List<User> users = userRepo.findAll();
```

A fresh perspective on logging

- Abstraction for logging frameworks, ***but...***
- Actually introduces *new* concepts:
  - Innovative, type-safe logger
  - Internationalization support (in development)
- Suits real-world scenarios
  - Developers work in Java
  - Translators work with message bundles
- ***Serializable*** loggers

Defining a typed logger

```
@MessageLogger
public interface CelebritySightingsLog
{
   @LogMessage @Message("Spotted %s at %s!")
   void celebritySpotted(String who, String location);

   @LogMessage @Message("Secret's out, %s and %s are BFFs!")
   void newBff(String who, String andwho);

   @LogMessage @Message("Uh oh, %s and %s are no longer BFFs!")
   void bffNoMore(String who, String andwho);

   @LogMessage @Message("%s stole %s's BFF!")
   void bffStolen(String who, String oldOwner);
}
```

Using a typed logger

```
@Inject @Category("gossip")
private CelebritySightingsLog log;

log.bffStolen("Victoria Beckham", "Jessica Simpson");
```

Typed exception messages

```
@MessageBundle
public interface AccountTransferMessages
{
   @Message("Insufficient funds. Overdrafted by %.02f")
   String insufficientFunds(BigDecimal overdraftAmount);
}
```

```
@Inject AccountTransferMessages msg;
```

```
throw new AccountTransferException(msg.insufficientFunds(amt));
```

Resource loading

- Built-in, extensible resource loader
- Can resolve as:
  - java.net.URL
  - java.io.InputStream
  - java.util.Properties
- Searches in:
  - Classpath
  - Servlet context (if available)
- Automatically manages input streams

Loading specific resources

```
@Inject
@Resource("WEB-INF/web.xml")
InputStream webXml;

@Inject
@Resource("META-INF/beans.xml")
Collection<URL> beansXmls;

@Inject
@Resource("database.properties")
Properties databaseProps;
```

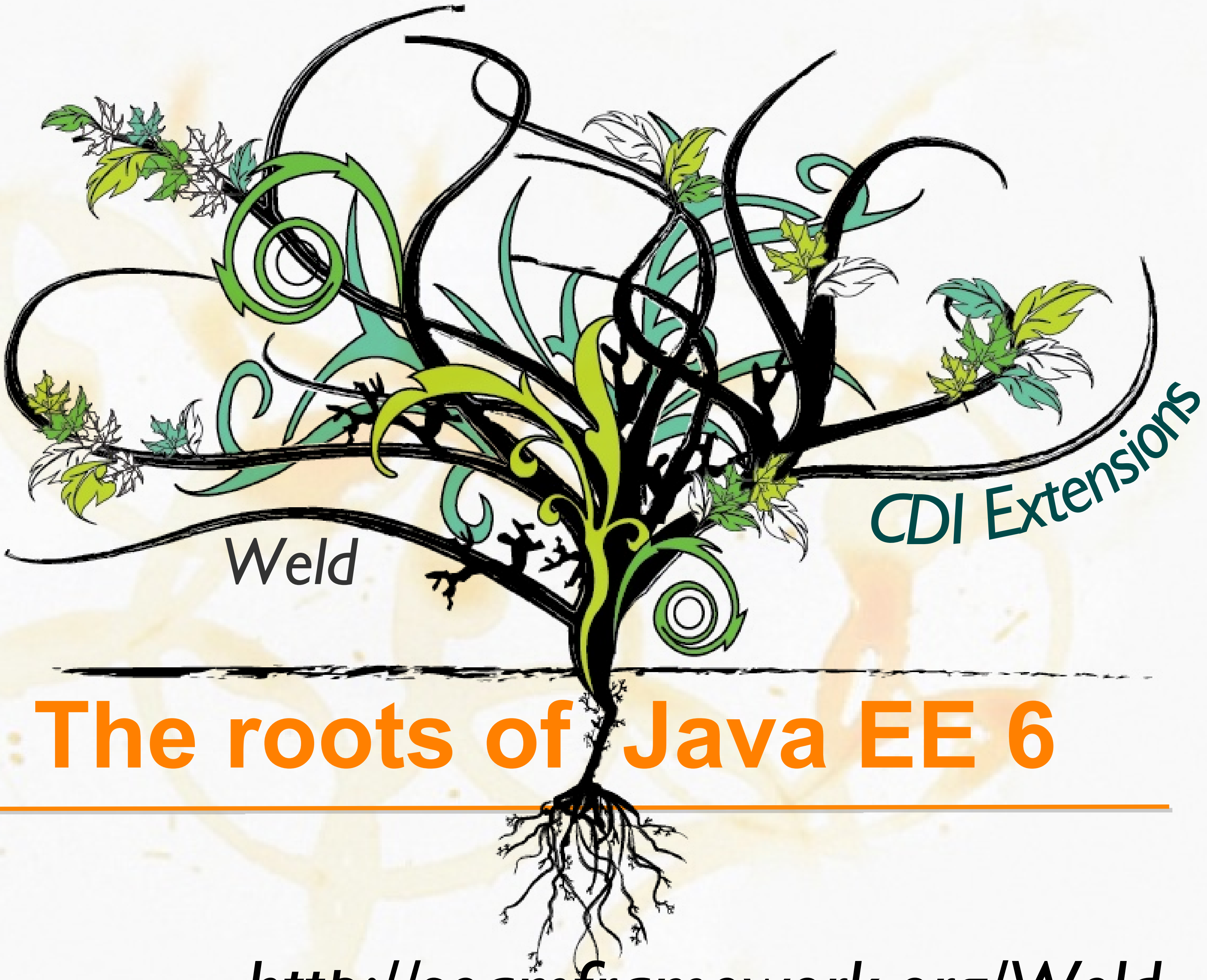
How do I test this stuff?

Integration testing fit for CDI

```
@RunWith(Arquillian.class)
public class MyExtensionTestCase
{
   @Deployment
   public static Archive<?> createDeployment()
   {
      return ShrinkWrap.create(JavaArchive.class)
         .addClasses(Sample.class, MyExtension.class)
         .addServiceProvider(Extension.class, MyExtension.class)
         .addManifestResource(EmptyAsset.INSTANCE, "beans.xml");
   }

   @Inject Sample sample;

   @Test
   public void functionality_should_work() throws Exception
   {
      assertTrue(sample.functionalityWorks());
   }
}
```

Summary

- Java EE 6 is flexible, portable and extensible
- CDI provides a set of services for Java EE
  - Offers loose coupling with strong typing
  - Provides a type-based event bus
  - Decoupled AOP
  - Provides Extension SPI for writing add-ons
- Weld: JSR-299 Reference Implementation
- Seam Solder
  - Swiss army knife for extension writers
- Extensions are growing every day!



The roots of Java EE 6

http://seamframework.org/Weld
http://seamframework.org/Seam3
http://groups.diigo.com/group/cdi-extensions