JBoss World 2006 LAS VEGAS

From the JMX MicroKernel to the POJO MicroContainer

Dimitris Andreadis
AS Development Mgr

Kabir Khan
AOP Lead

JBoss Inc.

Agenda

- JBoss Kernel
  - ✓ Overview
  - ✓ Evolution
- Defining Services
  - ✓ JMX MicroKernel
  - ✓ EJB3 Extensions
  - ✓ POJO MicroContainer
- MicroContainer & AOP

JBoss Kernel

What is the JBoss Kernel?

- A lightweight component framework that wires together a set of *Services*
- *Services* are de-coupled, invocations are routed through an internal bus

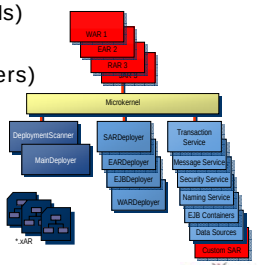
(a.k.a. The *SuperServer*)

Kernel Functionality

- Metadata Repository
- XML externalization
- Lifecycle/Configuration
- Classloading
- Naming/Registry
- Dependencies
- [Management]

Why the Kernel is important?

- Customize the server to your needs
  - ✓ Modular (use what you want)
  - ✓ Embeddable (ISVs/OEMs)
  - ✓ Manageable (JMX)
  - ✓ Extensible (by Developers)
    - Write your Service
    - Write your Deployer

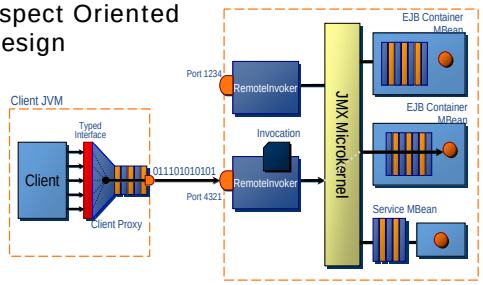


Typical User Services

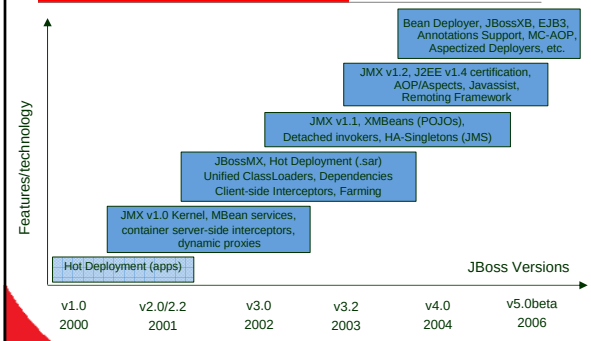
- Protocol listeners (http, socket, etc.)
- Remote object wrapping (RMI, CORBA)
- Dynamic configuration settings
- Lightweight event processing
- Lightweight adapters
- Deployer extensions
- Job Scheduling
- Startup Classes
- ...

Why the Kernel is important? (2)

- Basis for an Interceptor-based Aspect Oriented Design



From MicroKernel to MicroContainer

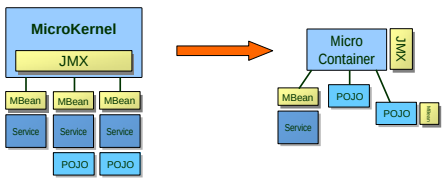


From MicroKernel to MicroContainer

- A standalone Project
  - ✓ Usable inside other containers
- Aspect-based deployers
  - ✓ Classloading, security, logging, etc.
  - ✓ Kernel handles instantiation/dependencies
  - ✓ VDF/VFS
- Improved configuration
  - ✓ Logical definition of services (e.g. JMS)
  - ✓ Versioned/Persistent changes
  - ✓ Classloading dependencies
  - ✓ Deployment modes
    - Auto, Manual, On-Demand, Disabled

From MicroKernel to MicroContainer

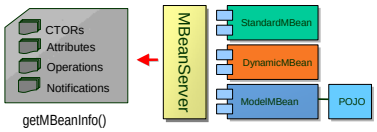
- JMX requirement becomes an option
  - ✓ JMX capability is a plug-in
  - ✓ POJOs can be JMX manageable
  - ✓ Existing MBean services still work



MicroKernel JMX Services

JMX MicroKernel Services

- Services are implemented as
  - ✓ Standard MBeans
  - ✓ Dynamic MBeans
  - ✓ POJOs wrapped by Model MBeans (XMBeans)



Simple Standard MBean

```
public interface VersionMBean
{
  void setVersion(String version);
  String getVersion();
  String printVersionAsHtml();
}
```

```
public class Version implements VersionMBean
{
  String version;

  public void setVersion(String version) {
    this.version = version;
  }

  public String getVersion() {
    return version;
  }

  public String printVersionAsHtml() {
    return "<B>" + version + "</B>";
  }
}
```


Configuring an MBean

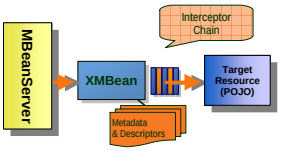
- Classes packaged in a .sar along with META-INF/jboss-service.xml

jboss-service.xml

```
<server>
  <mbean code="Version" name="jboss:service=Version">
    <attribute name="Version">4.0.4</attribute>
  </mbean>
</server>
```


POJO Services

- JBoss' XML-based implementation of ModelMBean
- Provides for
  - ✓ Rich metadata
  - ✓ Attribute caching
  - ✓ Attribute persistence
  - ✓ Attribute change notifications
  - ✓ Ability to plug-in custom interceptors



Simple POJO Service

```
// No MBean interface!

public class Version
{
  ...
}
```

xmbean.xml

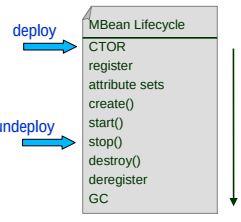
```
<mbean>
  <description>Simple POJO</description>
  <class>Version</class>
  <attribute access="read-write"
    getMethod="getVersion" setMethod="setVersion">
    <name>Version</name>
    <type>java.lang.String</type>
  </attribute>
  <operation>
    <description>Prints the Version</description>
    <name>printVersionAsHtml</name>
    <return-type>java.lang.String</return-type>
  </operation>
</mbean>
```

jboss-service.xml

```
<server>
  <mbean code="Version"
name="jboss:service=Version"
    xmbean-dd="xmbean.xml">
    <attribute name="Version">4.0.4</attribute>
  </mbean>
</server>
```

Service Lifecycle

- Services may optionally expose lifecycle operations
  - ✓ create()
    start()
    stop()
    destroy()
- No need to extend JBoss classes



MBean with Dependency Injection

```
public interface VersionedServiceMBean {
  void setVersion(VersionMBean vhm);
  void start() throws Exception;
  ...
}

public class VersionedService implements VersionedServiceMBean {
   VersionMBean vhm;

   public void setVersion(VersionMBean vhm) { this.vhm = vhm}

   public void start() {
     System.out.println("Started " + vhm.getVersion());
   }
   ...
}
```

Lifecycle callback.
Dependencies start first

my-service.xml

```
<server>
  <mbean code="VersionedService" name="jboss:service=VersionedService">
    <depends optional-attribute-name="Version" proxy-type="attribute">
          jboss:service=Version
    </depends>
  </mbean>
</server>
```


MBean with Attribute Injection

```
<server>
 <mbean code="VersionedService"
name="jboss:service=VersionedService">
   <attribute name="Version"
         attributeClass="Version"
         serialDataType="javaBean">
    <property name="Version">4.0.4</property>
   </attribute>
 </mbean>
</server>
```


EJB3 Service Extensions

EJB 3 @Service Beans

- A stateful managed bean that can expose a local/remote interface!
  - ✓ Deployed as an EJB 3 bean
- No XML, Annotation based!
  - ✓ @Remote/@Local
  - ✓ @Management
  - ✓ @Depends
- Supports standard EJB3 annotations
  - ✓ @Interceptors
  - ✓ @EJB, @Resource etc.

@Service Example

```
@Management
public interface Managed
{
public int getRequestCount();
}
```

```
@Remote
public interface ManagedRemote
{
   public void doRequest();
}
```

```
@Service (objectName="jboss.example:service=Managed")
public class ManagedBean implements Managed, ManagedRemote
{
   @Resource Datasource ds;
   @EJB AccountBean account;

   @Depends ({"jboss:service=VersionMBean"})
   VersionMBean mbean;

   int requestCount;

   public void doRequest(){ requestCount++; ... }
   public int getRequestCount() { return requestCount; }
}
```


@Service Interceptors

```
@Service (objectName="jboss.example:service=Managed")
@Interceptors({AuditInterceptor.class})
public class ManagedBean implements Managed, ManagedRemote
{
   ...
   @AroundInvoke
   public Object sendNotification(InvocationContext ctx) throws Exception{
      ...
      return ctx.proceed();
   }
}
```

Method, args, bean instance, shared state etc.

```
public class AuditInterceptor {
   @Resource EJBContext ejbctx;
   @Resource(name="DefaultDS", mappedName="java:DefaultDS") DataSource ds;

   @AroundInvoke
   public Object intercept(InvocationContext ctx) throws Exception{
      Connection conn = ds.getConnection();
      //Write audit log to Db
      return ctx.proceed();
   }
}
```

MicroContainer POJO Services

25

JBoss MicroContainer

- A standalone/lightweight Dependency Injection framework, used by:
  - ✓ Embedded EJB3 (JNDI, JCA, JTA/UserTransaction)
  - ✓ Seam/Tomcat
  - ✓ JBoss 5
- Accessible in JB4 through -beans.xml deployer
  - ✓ Creates objects for you
  - ✓ Initializes their properties
  - ✓ Registers object in internal registry

26

Bean Configuration

```
public class Version
{
  String version;

  public void setVersion(String version) {
    this.version = version;
  }

  public String getVersion() {
    return version;
  }
  ...
}
```

MC beans are POJOs!

my-beans.xml

```
<deployment>
  <bean name="Version" class="org.acme.Version">
    <property name="version">4.0.4</property>
  </bean>
</deployment>
```

27

Setter Injection

```
public class VersionedService {
  Version vh;

  public void setVersion(Version vh) {
    this.vh = vh;
  }

  public void start() {
    log("Started " + vh.getVersion());
  }
  ...
}
```

my-beans.xml

```
<deployment>
<bean name="VersionedService" class="org.acme.VersionedService">
    <property name="Version"><inject bean="Version"/></property>
</bean>
</deployment>
```

28

Constructor Injection

```
public class VersionedService {
  Version vh;

  public VersionedService(Version vh) {
    this.vh = vh;
  }
  ...
}
```

my-beans.xml

```
<deployment>
<bean name="VersionedService" class="org.acme.VersionedService">
  <constructor>
    <parameter><inject bean="Version"/></parameter>
  </constructor>
</bean>
</deployment>
```

29

Static Factory Injection

```
public class VersionedServiceFactory {

  public static VersionedService create(Version vh) {
    return new VersionedService(vh);
  }
}
```

my-beans.xml

```
<deployment>
<bean name="VersionedService" class="org.acme.VersionedService">
  <constructor factoryClass="org.acme.VersionedServiceFactory"
               factoryMethod="create">
    <parameter><inject bean="Version"/></parameter>
  </constructor>
</bean>
</deployment>
```

30

Bean Factory Injection

```
public class VersionedServiceFactory {
    Version vh;
    public void setVersion(Version vh){ this.vh = vh; }

    public static VersionedService create(Version vh) {
      return new VersionedService(vh);
    }
}
```

my-beans.xml

```
<deployment>
<bean name="Factory" class="org.acme.VersionedServiceFactory">
  <property name="Version"><inject bean="Version"/></property>
</bean>
<bean name="VersionedService" class="org.acme.VersionedService">
  <constructor factoryMethod="create">
    <factory bean="Factory"/>
  </constructor>
</bean>
</deployment>
```


Property Types

- Primitives, String
- Collections: (any Map, List, Set type)

```
<bean name="VersionedService" class="org.acme.VersionedService">
  <property name="mappedVersions">
    <map class="java.util.concurrent.ConcurrentHashMap"
      keyClass="java.lang.String" valueClass="java.lang.String">
      <entry>
        <key>4.0.4 final</key>
        <value>4.0.4.GA</value>
      </entry>
    </map>
</bean>
```


Property Types (2)

- Referenced beans
- Inlined beans [with custom schema]

```
<bean name="VersionedService" class="org.acme.VersionedService">
  <property name="complexVersion">
    <mybean:bean xmlns:mybean="urn:jboss:mbeans">
      <mybean:major>4</mybean:major>
      <mybean:minor>0</mybean:minor>
      <mybean:patch>4</mybean:patch>
      <mybean:suffix>GA</mybean:suffix>
    </mybean:bean>
  </property>
</bean>
```


Dependencies

- Bean not created/started until dependencies are created/started
- When a new bean is deployed, unstarted beans checked to see if their dependencies are resolved

```
<bean name="BeanA" class="org.acme.BeanA">
 <depends>BeanB</depends>
 <property name="AnotherBean">
   <inject bean="BeanC"/>
 </property>
</bean>

<bean name="BeanB" class="..">
 ...
</bean>
...
```


MicroContainer & AOP

Microcontainer and JBossAOP

- JBoss AOP an optional plug-in to MC
- IoC for defined Aspects
  - ✓ Aspect configuration
- Aspect dependency propagation
  - ✓ <bean> inherits aspect's dependencies
- Per <bean> aspects
  - ✓ Weaved or proxied
- Per <bean> annotation overrides
  - ✓ Introductions

JBoss AOP advice example

```
public class LogInterceptor implements org.jboss.aop.advice.Interceptor
{
   public String getName() { return "LogInterceptor";}

   public Object invoke(org.jboss.aop.joinpoint.Invocation inv) throws Throwable {
      System.out.println("Calling " + getJoinpoint(inv));
      try {
         invocation.invokeNext();
      }
      finally {
         System.out.println("Called " + getJoinPoint(inv));
      }
   }

   private AccessibleObject getJoinpoint(Invocation inv) {
      if (inv instanceof MethodInvocation) {
         return ((MethodInvocation)inv).getMethod();
      }
      else if (inv instanceof FieldReadInvocation)
      ...
   }
}
```


Binding the Advice using AOP

```
<aop>
  <interceptor name="log" class="org.acme.LogInterceptor"/>
  <bind pointcut="all(@org.acme.Traceable)">
     <interceptor-ref name="log"/>
  </bind>
</aop>
```

```
<deployment>
  <bean name="Version" class="org.acme.Version">
    <property name="version">4.0.4</property>
    <annotation name="@org.acme.Traceable"/>
  </bean>
  <bean name="NotAnnotated" class=".."/>
</deployment>
```

- AOP is MC agnostic, but the underlying metadata repository is shared

JBoss AOP advice example (2)

```
public class RequiresNewTxInterceptor implements org.jboss.aop.advice.Interceptor
{
   TransactionManager tm;
   public String getName() { return "TxInterceptor";}

   public void setTransactionManager(TransactionManager tm) { this.tm = tm; }

   public Object invoke(org.jboss.aop.joinpoint.Invocation inv) throws Throwable {
      Transaction oldTx = tm.suspend();
      tm.begin();
      try {
         invocation.invokeNext();
         tm.commit();
      }
      catch(Exception e) {
         tm.rollback();
      }
      finally {
         if (oldTx != null) tm.resume(oldTx);
      }
   }
}
```


Binding the Advice using MC

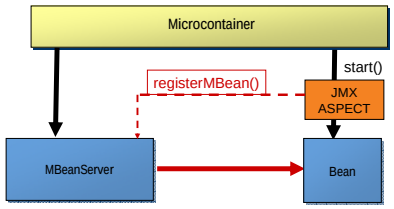
```
<deployment>
  <bean name="TxMananager" class="org.jboss.tm.TxManager">

  <bean name="TransactionalService" class="org.acme.TransactionalService">
    <annotation name="@org.acme.TxRequiresNew"/>
  </bean>

  <aop:aspect xmlns:aop="urn:jboss:aop-beans:1.0"
          name="RequiresNewTxInterceptor"
          class="org.acme.RequiresNewTxInterceptor"
          pointcut="execution(* @org.acme.TxRequiresNew->*(..))">
    <property name="transactionManager"><inject bean="TxManager"/></property>
  </aop:aspect>

</deployment>
```

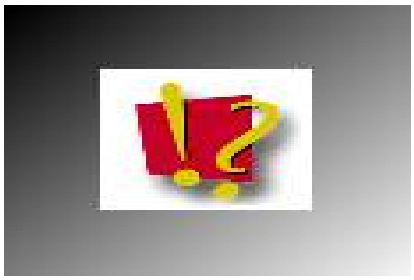

MC and JMX



Conclusion

- MicroContainer manages beans
  - ✓ injection/lifecycle/dependencies
- Services configured by MC can be deployed anywhere the MC can be run
  - ✓ "JBoss Everywhere!"
- MicroContainer core of JBoss 5.0
  - ✓ Services will be repackaged
  - ✓ MBean services will still work

dimitris@jboss.com
kkhan@jboss.com



www.jboss.com/products/jbossmc

43