Bundle deployment at state machine level

Aleš Justin
JBoss, a division of Red Hat
ales.justin@jboss.org
JBoss Microcontainer

• **How we started?**
  • Replacement for JMX based Kernel
  • Dependency State Machine
  • POJO Component Model

• **Where are we now?**
  • Aspectized Deployers
  • Classloader Model
  • OSGi Integration
  • VFS
  • MBeans Support
  • AOP Integration
  • Management Model
  • Spring Integration
• **Controller**
  - Dependency State Machine
  - ControllerStates – define your own states
  - Install context receive callbacks when dependencies satisfied

• **ControllerContext**
  - Represents a component
  - Has dependencies
  - Receives callbacks to implement the model when dependencies are satisfied for a state transition
  - ControllerMode – automatic, manual, on demand

• **DependencyInfo and DependencyItem**
  - Define your own dependencies
  - Write your own dependency item
    - Standard implementations available
The basic API

public interface Controller {
    install(ControllerContext c);
    uninstall(Object name);
}

public interface ControllerContext {
    Object getName();
    DependencyInfo getDependencyInfo();
    void install(ControllerState s);
    void uninstall(ControllerState s);
}

public interface DependencyInfo {
    boolean resolveDependencies(Controller c, ControllerState s);
}

public interface DependencyItem{
    ControllerState getWhenRequired();
    boolean isResolved();
}
• **Existing implementations**
  • POJO – KernelControllerContext
    • IOC component model
  • JMX - ServiceControllerContext
    • Legacy JBoss service model
• **Example transitions** – implemented by ControllerContexts
  • Instantiate
    • POJO – new POJO();
    • JMX – registerMBean();
  • Configure
    • POJO – pojo.setProperty(value);
    • JMX – mbeanServer.setAttribute(objectName, property, value);
  • Start lifecycle callback
    • POJO – pojo.start();
    • JMX – mbeanServer.invoke(objectName, “start”, null, null);
Deployer Integration

• **Structural Deployers**
  - VFS usage
  - Recognise deployment types
    - User defined – META-INF/jboss-structure.xml
    - Specification defined – jar, war, ear, etc.
  - Defines the structure
    - Where is the metadata? META-INF or WEB-INF, etc.
    - Where are the classes? / or WEB-INF/classes, etc.
    - What are the subdeployments?

• **Aspectized Deployers**
  - Each Deployer does one thing well
  - Easy to control how much gets done
    - Off-line tool like the admin console only wants to do parsing
    - Runtime does everything
  - Easy to swap out behaviour – e.g. change the classloader
• **Deployers are Staged**
  • Deployments are processed width first

• **Aspects**
  • Parsing Deployers
    • Turns xml into a metadata model attachment
    • e.g. my-beans.xml -> KernelDeployment
  • ClassLoading Deployers
    • Creates classloaders from metadata
    • e.g. Uses the information from the StructureDeployers
  • Component Deployers
    • Splits complicated deployments into units
    • e.g. KernelDeployment -> BeanMetaDatas
  • Real Deployers
    • Does the real work of deployment
    • e.g. BeanMetaData -> controller.install()
• **Deployment Attachments**
  • Each deployment has attachments
    • e.g. deployment.getAttachment(KernelDeployment.class)
  • Two types of attachment
    • Predetermined – overridden by the user, e.g. Profile service
    • Transient – parsed by the parsing Deployer
  • Predetermined overrides transient
• **Attachments are not used linearly**
  • Example JCA RAR deployment
    • Parsing done by RAR Deployer – creates RARMetaData
    • RAR Component Deployer creates ServiceMetaData
    • No real RAR Deployer, it uses the JMX real Deployer
• **Alternate Real Deployers Strategy**
  • Doesn't have to use a Microcontainer recognised component model
  • e.g. Log4j real Deployer could do META-INF/log4j.xml -> log4j config directly
Classloader Model Rewrite

- **Simplifying SPI**
  - Hidden Base
  - Loader interface
  - ClassLoaderPolicy
  - ParentPolicy

- **Goals**
  - Simple OSGi implementation
  - Backward compatible
  - Clean extension
OSGi Integration

- New ControllerContexts
  - Introduction of Dependency to Deployers
  - OSGi Services Support

- New Deployers
  - OSGi Manifest.MF MetaData
  - DeploymentResolver
  - OSGi Classloader

- Other features
  - OSGi Core API as Façade
  - Declarative Services Support
  - OBR usage