Using Apache Camel In An OSGi World

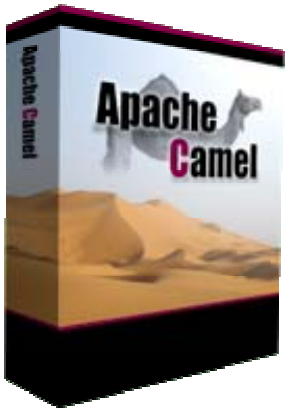
Dependency injection in OSGi, the Apache way.

Johan Edstrom
Jeff Genender

Johan and Jeff

savoir TECHNOLOGIES

The Apache Software Foundation
http://www.apache.org/



Agenda

- **Dependency injection in OSGi**
  - **Spring DM**
  - **Classloading issues**
  - **Aries Blueprint**
- **Adopting Aries Blueprint**
  - **Missing components and current development**
  - **Camel, CXF, Transactions, JPA, JNDI**
- **Using these libraries in a real world application**

Our Project


NCAR
NATIONAL CENTER FOR ATMOSPHERIC RESEARCH


FEDERAL AVIATION ADMINISTRATION


NNEW
NEXTGEN NETWORK ENABLED WEATHER

What is dependency injection?

```
<bean id="amqConnectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">
    <property name="brokerURL" value="${amq.connection.url}"/>
</bean>

<bean id="wcsriPooledConnectionFactory" class="org.apache.activemq.pool.PooledConnectionFactory">
    <property name="maxConnections" value="${amq.connection.maxConnections}"/>
    <property name="maximumActive" value="${amq.connection.maximumActive}"/>
    <property name="connectionFactory" ref="amqConnectionFactory"/>
</bean>
```

- **Quick**
- **Easy to follow**
- **Separation of concerns**
- **Simple easy “pojo” development model.**

Spring-DM... it's a bit like Trump's hair...

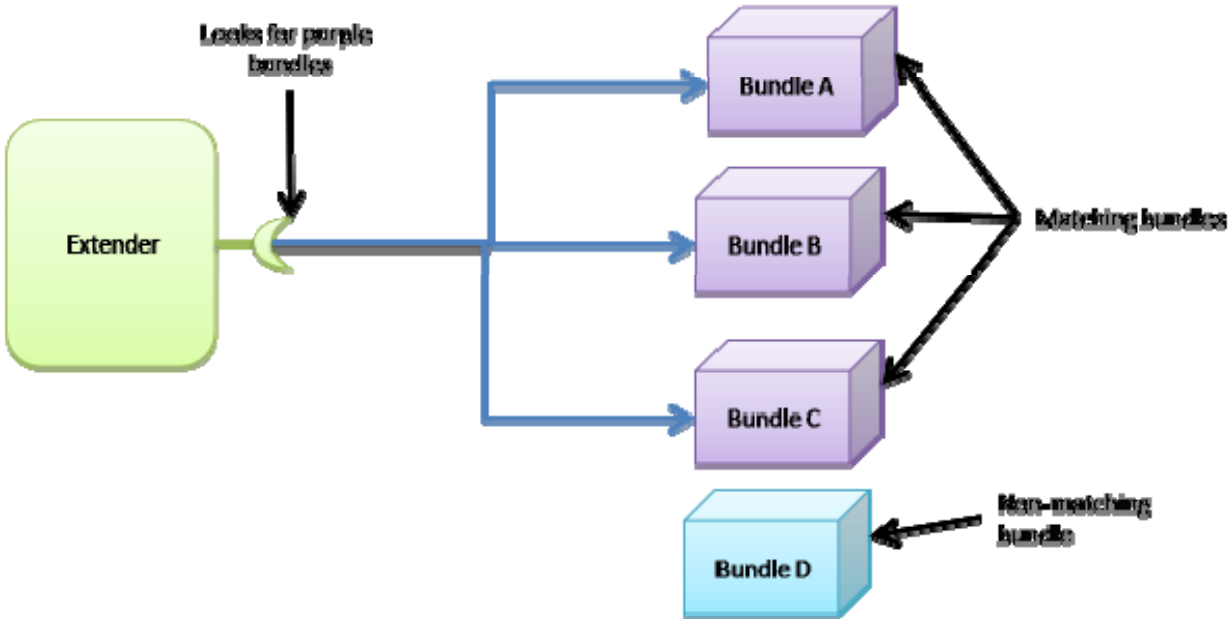
Its there… it covers quite a bit…
but it's kinda hard to recommend.

Looks great… but needs a lot of maintenance
or it becomes a big mess.

What does Spring-DM add to Spring?

- **A Spring application written in this way provides better separation of modules**
- **Ability to dynamically add, remove, and update modules in a running system,**
- **Ability to deploy multiple versions of a module simultaneously (and have clients automatically bind to the appropriate one),**
- **A dynamic service model.**

How is this done? The Extender Pattern.



* Extender registers itself as BundleListener* Bundle gets installed/started* Framework fires a BundleEvent* Extender picks up the BundleEvent (e.g. STARTING)* Extender reads metadata from the Bundle and does its work

Spring-DM example

```
<beans xmlns=http://www.springframework.org/schema/beans
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns:ctx=http://www.springframework.org/schema/context
       xmlns:osgi=http://www.springframework.org/schema/osgi
       xmlns:osgix="http://www.springframework.org/schema/osgi-compendium">

  <ctx:property-placeholder properties-ref="dcProps"/>

  <osgi:reference id="jmsConnectionPool" interface="javax.jms.ConnectionFactory"
                  filter="(wcsri.client=true)"/>

  <!-- For Configuration Administration -->
  <ctx:property-placeholder properties-ref="dcProps"/>

  <osgi:reference id="isoMetaGDSRequestHandlerRef"
                            bean-name="isoMetaGDSRequestHandler"
              interface="edu.ucar.ral.gds.handler.IsoMetaGDSRequestHandler"/>
```

OSGi class-loading and Spring-DM

- **Each bundle in OSGi has it's own ClassLoader, only one bundle can be exposed as the thread context ClassLoader at any time.**
- **This means that if a third-party library needs to see types that are distributed across multiple bundles, it isn't going to work as expected.**
- **Spring-DM “fixes” this by creating a ClassLoader that imports all the exported packages of every module in your application. This ClassLoader is then exposed as the thread context ClassLoader, enabling third-party libraries to see all the exported types in your application.**

Spring-DM Extender Bundle

- **Spring Dynamic Modules provides an OSGi bundle org.springframework.osgi.extender.**
- **This bundle is responsible for instantiating the Spring application contexts for your application bundles. It serves the same purpose as the ContextLoaderListener does for Spring web applications.**
- **In addition, it listens for bundle starting events and automatically creates an application context for any Spring-powered bundle that is subsequently started.**

Why does this become a problem?

- **Backwards compatibility**
  - **Spring really is designed for a monolithic JVM**
- **NamespaceHandler resolution**
  - **NamespaceHandlers are activated via resource files**
- **Global "ClassPath"**
- **Thread Context ClassLoader is not defined in OSGi**
  - **Spring solves this by "enabling" an everything is visible classloader, this will quickly break with legacy code using SPI, changing classloaders or when Spring-DM bundles are notified of context refreshes.**

So what are some of the workarounds?

- **Ordering bundle startups**
- **Embedding dependencies**
- **Forcing class-loader changes in Bundle Activators**
- **Not to mention a ton of tedious experimentation**

Aries to the rescue!!!

- **The Aries project delivers a set of pluggable Java components enabling an enterprise OSGi application programming model.**
- **This includes implementations and extensions of application-focused specifications defined by the OSGi Alliance Enterprise Expert Group (EEG) and an assembly format for multi-bundle applications, for deployment to a variety of OSGi based runtimes.**

Apache Aries - Blueprint

- **BP provides a dependency injection framework for OSGi**
- **A bundle is a BP bundle if there are files in OSGI-INF/blueprint/ or specified explicitly in the Bundle-Blueprint manifest header.**
- **A BP container uses an extender pattern that handles**
  - **Parsing the Blueprint XML files**
  - **Instantiating the components**
  - **Wiring the components together**
  - **Registering services**
  - **Looking up service references**

What does BP look like?

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
                    xmlns:cm="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.0.0">
   <bean id="amqConnectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">
      <property name="brokerURL" value="${amq.connection.url}"/>
   </bean>
   <bean id="wcsriPooledConnectionFactory" class="org.apache.activemq.pool.PooledConnectionFactory">
      <property name="maxConnections" value="${amq.connection.maxConnections}"/>
      <property name="maximumActive" value="${amq.connection.maximumActive}"/>
      <property name="connectionFactory" ref="amqConnectionFactory"/>
   </bean>
   <bean id="resourceManager" class="org.apache.activemq.pool.ActiveMQResourceManager" init-method="recoverResource">
      <property name="transactionManager" ref="transactionManager"/>
      <property name="connectionFactory" ref="amqConnectionFactory"/>
      <property name="resourceName" value="activemq.default"/>
   </bean>
   <!-- Client pool config -->
   <bean id="amqClientConnectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">
      <property name="brokerURL" value="${amq.client.connection.url}"/>
   </bean>
   <!-- This may be the one to use - commenting this in case it may be needed... unfortunately it uses Spring -->
   <bean id="wcsriClientPooledConnectionFactory" class="org.springframework.jms.connection.CachingConnectionFactory">
      <property name="targetConnectionFactory" ref="amqClientConnectionFactory"/>
      <property name="sessionCacheSize" value="${amq.client.session.cache.size}"/>
      <property name="cacheConsumers" value="false"/>
   </bean>
</blueprint>
```

So.... Where is this BP stuff used?

- **Apache ServiceMix & Apache Karaf**
  - **Commands, components, instantiation**
  - **Basically what you see as "A Servicemix Container" is BP.**
- **Apache Geronimo**
  - **Aries came from here.**

What was “Missing” for community adaptation?

- **Aries**
  - **JPA**
  - **TX**
  - **JNDI**
- **Camel-Blueprint**
  - **Pretty stellar since camel 2.6, 2.7 added advanced properties use.**
  - **Camel-CXF : Endpoint in trunk (2.8)**
- **CXF- Blueprint**
  - **CXF Endpoint**
  - **CXF Client**
  - **Features : Addressing, Coloc**
  - **JaxRsServer**
  - **JaxRsClient**

What is a NamespaceHandler?

camel-blueprint (148) provides:
---
osgi.service.blueprint.namespace = http://camel.apache.org/schema/blueprint
objectClass = org.apache.aries.blueprint.NamespaceHandler
service.id = 257
---
osgi.blueprint.container.version = 2.6.0.fuse-00-00
osgi.blueprint.container.symbolicname = org.apache.camel.camel-blueprint
objectClass = org.osgi.service.blueprint.container.BlueprintContainer
service.id = 259

And that means?

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
   <camelContext xmlns="http://camel.apache.org/schema/blueprint">
      <route>
         <from uri="direct:start"/>
         <to uri="mock:result"/>
      </route>
   </camelContext>
</blueprint>
```

Beans and Camel

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
    <camelContext xmlns="http://camel.apache.org/schema/blueprint">
        <route>
            <from uri="mycomp:queue"/>
            <to uri="mock:result"/>
        </route>
    </camelContext>
    <bean id="mycomp" class="org.apache.camel.component.seda.SedaComponent"/>
</blueprint>
```

Slightly more advanced - and cooler than Spring!

```
<blueprint xmlns=http://www.osgi.org/xmlns/blueprint/v1.0.0
        xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
        xmlns:cm=http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.0.0
        xsi:schemaLocation="
        http://www.osgi.org/xmlns/blueprint/v1.0.0 http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">

   <!-- OSGI blueprint property placeholder -->
   <cm:property-placeholder id="myblueprint.placeholder" persistent-id="camel.blueprint">
     <!-- list some properties for this test -->
      <cm:default-properties>
         <cm:property name="result" value="mock:result"/>
      </cm:default-properties>
   </cm:property-placeholder>

   <camelContext xmlns="http://camel.apache.org/schema/blueprint">
      <!-- using Camel properties component and refer to the blueprint property placeholder by its id -->
      <propertyPlaceholder id="properties" location="blueprint:myblueprint.placeholder"/>

      <!-- in the route we can use {{ }} placeholders which will lookup in blueprint -->
      <route>
         <from uri="direct:start"/>
         <to uri="mock:foo"/>
         <to uri="{{result}}"/>
      </route>
   </camelContext>

</blueprint>
```

What about those web services?

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:cm="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.0.0"
           xmlns:camel-cxf="http://camel.apache.org/schema/blueprint/cxf"
           xmlns:cxfcore="http://cxf.apache.org/blueprint/core"
           xsi:schemaLocation="
             http://www.osgi.org/xmlns/blueprint/v1.0.0 http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">

      <camel-cxf:cxfEndpoint id="routerEndpoint"
                   address="http://localhost:9001/router"
                   serviceClass="org.apache.servicemix.examples.cxf.HelloWorld">
        <camel-cxf:properties>
           <entry key="dataFormat" value="MESSAGE"/>
        </camel-cxf:properties>
      </camel-cxf:cxfEndpoint>

      <camel-cxf:cxfEndpoint id="serviceEndpoint"
                     address="http://localhost:9000/SoapContext/SoapPort"
                   serviceClass="org.apache.servicemix.examples.cxf.HelloWorld">
      </camel-cxf:cxfEndpoint>

     <camelContext xmlns="http://camel.apache.org/schema/blueprint">
        <route>
           <from uri="routerEndpoint"/>
           <to uri="log:request"/>
        </route>
     </camelContext>

</blueprint>
```

And what about CXF only?

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:jaxws="http://cxf.apache.org/blueprint/jaxws"
        xmlns:coloc="http://cxf.apache.org/binding/coloc"
        xmlns:object="http://cxf.apache.org/blueprint/binding/object"
        xmlns:cxf="http://cxf.apache.org/blueprint/core"
        xmlns:soap="http://cxf.apache.org/blueprint/bindings/soap"
        xmlns:p="http://cxf.apache.org/policy"
        xmlns:wsp-200607="http://www.w3.org/2006/07/ws-policy"
        xmlns:wsp-200409="http://schemas.xmlsoap.org/ws/2004/09/policy"
        xsi:schemaLocation="
        http://www.osgi.org/xmlns/blueprint/v1.0.0
                http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">

 <cxf:bus>

      <cxf:features>
         <cxf:logging/>
      </cxf:features>

   </cxf:bus>
```

Policy

```
<p:engine
    enabled="true”
    ignoreUnknownAssertions="true">

    <p:alternativeSelector>

        <bean class="org.apache.cxf.ws.policy.selector.MaximalAlternativeSelector"/>

    </p:alternativeSelector>

</p:engine>
```

More endpoint configs!

```
<bean class="org.apache.servicemix.examples.cxf.HelloWorldImpl" id="impler"/>

<jaxws:endpoint id="bob”
                implementor="#impler”
                address="http://localhost:9090/hello_world">
    <jaxws:binding>
      <soap:soapBinding mtomEnabled="true" version="1.2"/>
    </jaxws:binding>
    <jaxws:inInterceptors>
       <bean class="org.apache.cxf.interceptor.LoggingInInterceptor"/>
    </jaxws:inInterceptors>
    <jaxws:outInterceptors>
       <bean class="org.apache.cxf.interceptor.LoggingOutInterceptor"/>
    </jaxws:outInterceptors>
    <jaxws:properties>
      <entry key="mtom_enabled" value="true"/>
    </jaxws:properties>
    <jaxws:features>
      <wsa:addressing xmlns:wsa="http://cxf.apache.org/blueprint/ws/addressing"/>
       <coloc:enableColoc/>
       <p:policies namespace="http://schemas.xmlsoap.org/ws/2004/09/policy">
          <wsp-200409:Policy/>
       </p:policies>
    </jaxws:features>
</jaxws:endpoint>
```

Well, what about the other things we use in Spring?

- **Transactions?**
  - **xmlns:tx="http://aries.apache.org/xmlns/transactions/v1.1.0"**

```
<bean id="messageDAO" class="edu.ucar.ral.wcsri.pubsub.persistence.dao.jpa.MessageJPADAO">
   <jpa:context property="entityManager" unitname="wcsri-jpa"/>
   <tx:transaction method="*" value="Required"/>
</bean>
```

- **JPA**
  - **Mark your bundle as a persistence provider**
    - **Meta-Persistence: META-INF/persistence.xml**
- **Accessing services as JNDI?**
  - **Servicemix has had support for this a long time, aries adds to it.**

Persistence.xml with OpenJPA

```
<persistence xmlns=http://java.sun.com/xml/ns/persistence
        xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
        xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
        http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd" version="2.0">
  <persistence-unit name="wcsri-jpa" transaction-type="JTA">
    <provider>org.apache.openjpa.persistence.PersistenceProviderImpl</provider>
    <jta-data-source>blueprint:comp/jta</jta-data-source>
    <non-jta-data-source>osgi:service/javax.sql.DataSource/(transactional=false)</non-jta-data-source>
    <class>edu.ucar.ral.wcsri.pubsub.persistence.domain.Subscription</class>
      <exclude-unlisted-classes>true</exclude-unlisted-classes>
    <validation-mode>NONE</validation-mode>
    <properties>
      <property name="openjpa.Multithreaded" value="true"/>
      <property name="openjpa.TransactionMode" value="managed"/>
      <property name="openjpa.ConnectionFactoryMode" value="managed"/>
      <property name="openjpa.LockManager"
value="pessimistic(VersionCheckOnReadLock=true,VersionUpdateOnWriteLock=true)"/>
      <property name="openjpa.LockTimeout" value="30000"/>
      <property name="openjpa.jdbc.MappingDefaults" value="ForeignKeyDeleteAction=restrict,
JoinForeignKeyDeleteAction=restrict"/>
      <property name="openjpa.LockManager"
value="pessimistic(VersionCheckOnReadLock=true,VersionUpdateOnWriteLock=true)"/>
      <property name="openjpa.Log" value="DefaultLevel=INFO, Runtime=INFO, Tool=INFO, SQL=INFO"/>
    </properties>
  </persistence-unit>
```

What are those datasources?

- **Glad you asked!**
  - **Aries JNDI provides for easy use of BP / JNDI naming**
    - **Importing the service we saw :**

```
<reference id="jta"  interface="javax.sql.DataSource”
          filter="(transactional=true)" availability="mandatory"/>
```

  - **It is an XA datasource, re-exported and used as BP component.**

Sample XA Datasource

```
<service interface="javax.sql.DataSource" ref="jta">
    <service-properties>
      <entry key="transactional" value="true"/>
    </service-properties>
  </service>

  <service interface="javax.sql.DataSource" ref="nonJTA">
    <service-properties>
      <entry key="transactional" value="false"/>
    </service-properties>
  </service>

  <!-- ###### JDBC Data Source ###### -->
  <reference id="txManager" interface="javax.transaction.TransactionManager" availability="mandatory"/>

  <bean id="dataSource" class="org.enhydra.jdbc.standard.StandardXADataSource">
    <property name="driverName" value="${jdbc.driverClassName}"/>
    <property name="url" value="${jdbc.url}"/>
    <property name="user" value="${jdbc.username}"/>
    <property name="password" value="${jdbc.password}"/>
    <property name="minCon" value="${jdbc.initialSize}"/>
    <property name="maxCon" value="${jdbc.maxActive}"/>
  </bean>

<bean id="jta" class="org.enhydra.jdbc.pool.StandardXAPoolDataSource">
    <property name="dataSource" ref="dataSource"/>
    <property name="transactionManager" ref="txManager"/>
    <property name="user" value="${jdbc.username}"/>
    <property name="password" value="${jdbc.password}"/>
    <property name="minSize" value="${jdbc.initialSize}"/>
    <property name="maxSize" value="${jdbc.maxActive}"/>
  </bean>
```

Sample non-JTA datasource

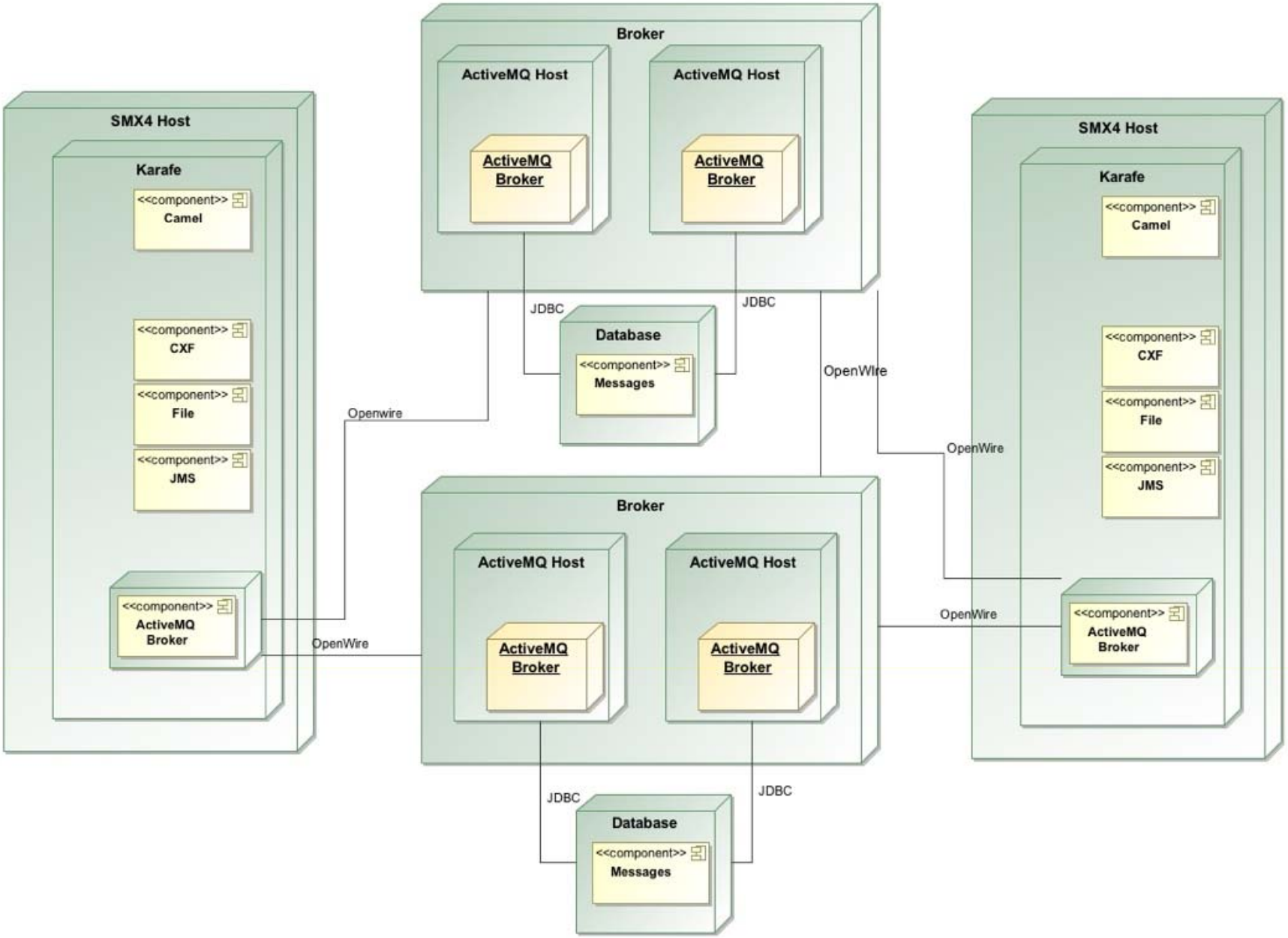
```
<service interface="javax.sql.DataSource" ref="nonJTA">
    <service-properties>
      <entry key="transactional" value="false"/>
    </service-properties>
</service>

<bean class="org.apache.commons.dbcp.BasicDataSource" id="nonJTA">
    <property name="driverClassName" value="${jdbc.driverClassName}"/>
    <property name="url" value="${jdbc.url}"/>
    <property name="username" value="${jdbc.username}"/>
    <property name="password" value="${jdbc.password}"/>
    <property name="initialSize" value="${jdbc.initialSize}"/>
    <property name="maxActive" value="${jdbc.maxActive}"/>
    <property name="maxIdle" value="${jdbc.maxIdle}"/>
    <property name="defaultAutoCommit" value="false"/>
    <property name="removeAbandoned" value="${jdbc.removeAbandoned}"/>
    <property name="removeAbandonedTimeout" value="${jdbc.removeAbandonedTimeout}"/>
    <property name="logAbandoned" value="${jdbc.logAbandoned}"/>
  </bean>
```

FAA - WCSRI

- **OGC-defined Web Coverage Service (WCS)**
  - **Reference implementation for weather data in the ERAM2 system**
  - **Originally web services in a Tomcat container**
- **Deployment goals**
  - **Publish subscribe over JMS or similar.**
  - **Deployable in a clustered environment.**
  - **Provide for many concurrent consumers and scaling.**
  - **Modular deployment.**

Basic use-case and system boundaries

Basic deployment



Core components - initial design

- **The core platform for deployment is based on OSGi, the container use is depending on configuration Apache Felix or Eclipse Equinox.**
- **The container framework is utilizing Apache ServiceMix and Apache Karaf to provide a runtime environment suitable for EE and SOA applications.**
- **In it's first iterations the WCSRI reference implementation was built around Spring-DM, Hibernate, Spring TX support**

Core components - current design

- **ServiceMix as the container**
- **Apache Camel for all messaging**
- **100% Aries Blueprint**
  - **Operational stability**
  - **Significant speed improvements**
- **OpenJPA for persistence**
- **Aries TX support**
- **Aries JPA support**
- **CXF Blueprint**
  - **The current implementation in CXF 2.4.x is from the WCSRI project**

Final outcome

- **The whole transition took about 2 weeks**
- **An additional week was spent weeding out bugs and writing additional blueprint namespacehandler support**
- **An overall aspect was “EE” like system stability**
- **Removing context refreshes led to very maintainable life-cycles for all components**
- **Less classloading issues**
- **Far, far easier to upgrade the components**

Complete component list

- **Container and queueing**
  - **Apache ServiceMix**
  - **Apache Camel**
  - **Apache CXF**
  - **Apache ActiveMQ**
  - **Apache Felix**
  - **Apache Aries (Blueprint, Transaction, JPA, JNDI)**
  - **Apache OpenJPA**
  - **Apache Derby**
  - **Eclipse Equinox**
- **Databases**
  - **Oracle RDBMS**
  - **Postgresql**
  - **MySQL**
  - **HSQLDB**
- **Monitoring**

Questions?

jgenender at savoirtech.com
jgenender at apache.org

jedstrom at savoirtech.com
joed at apache.org

savoir
TECHNOLOGIES