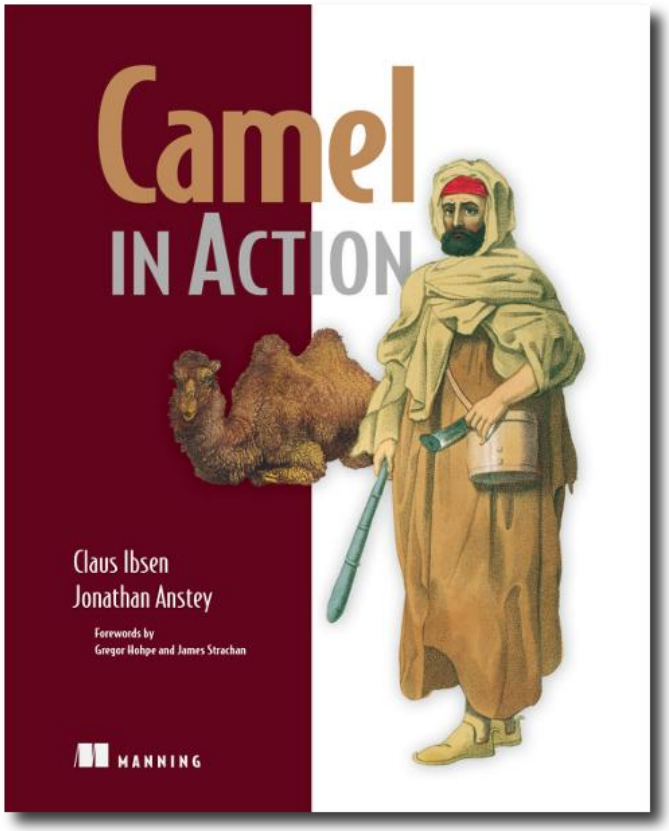
Using Apache Camel in ServiceMix

Jonathan Anstey
Principal Engineer, FuseSource
May 15, 2012

FuseSource
integration everywhere

Your Presenter is: Jonathan Anstey

- Principal Engineer at FuseSource
- Apache Camel, ActiveMQ, and ServiceMix committer
- Co-author of Camel in Action



- Twitter: @jon_anstey
- Email: jon@fusesource.com
- Blog: http://janstey.blogspot.com

FuseSource
integration everywhere

Agenda

- What is Apache ServiceMix?
- Tips for deploying Camel apps into ServiceMix
- Rider Auto Parts example and demo

What is Apache ServiceMix?

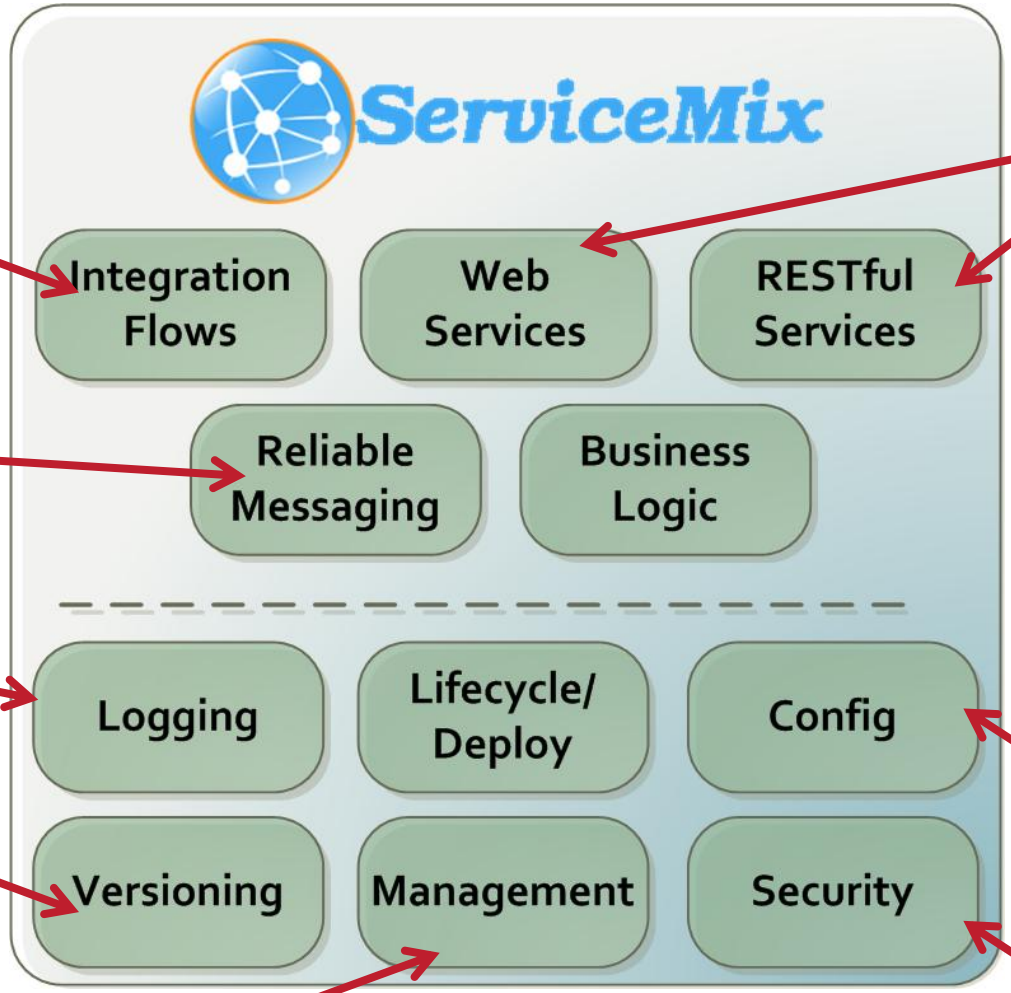
What is ServiceMix?

- Open source container useful for integration and SOA – an ESB.
  - EIP-style integration flows
  - SOAP & REST web services
  - Reliable messaging
- Fuse ESB Enterprise is based on Apache ServiceMix

What is ServiceMix?

- Support for various cross-functional concerns
  - Logging
  - Lifecycle and deployment
  - Configuration
  - Versioning & Dependency Mgmt
  - Management
  - Security
  - Transactions

ServiceMix – the Technologies



EIP via Camel

JAX-WS and JAX-RS via CXF

JMS via ActiveMQ

OSGi, JCL, JUL, Log4j, Slf4j

OSGi Config Admin

OSGi

JAAS, SSH, HTTPS, TLS, etc

JMX, web console, SSH

FuseSource
integration everywhere

Where do I start?

Start quickly with Maven archetypes

- Apache Maven archetypes are project templates
- Use camel-archetype-blueprint to create new blueprint-based Camel route
- Fuse IDE, Eclipse, IntelliJ support this

OSGi-ifying existing project

- Change Maven POM packaging type
  - <packaging>bundle</packaging>
- Use the maven-bundle-plugin to generate OSGi entries in the JAR's MANIFEST

```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <extensions>true</extensions>
</plugin>
```

- Alternatively, in Fuse ESB Enterprise you can use Fuse Application Bundles (FAB) to import existing Maven projects into the OSGi container.

Using what the ESB has to offer

Take advantage of OSGi Config Admin

- The Config Admin service provides an easy way of getting configuration into your bundle

```
<property-placeholder persistent-id="org.fusesource.camel.file"
  xmlns="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.0.0">
  <default-properties>
    <property name="fileEndpoint" value="file:target/placeorder" />
  </default-properties>
</property-placeholder>
```

- You can then use these properties in your routes

```
<camelContext id="rider-auto-file-poller"
  xmlns="http://camel.apache.org/schema/blueprint">
  <route id="file-to-jms">
    <from uri="{{fileEndpoint}}" />
    <to uri="jms:incomingOrders" />
  </route>
</camelContext>
```

Take advantage of OSGi Config Admin

- You can update properties in the command shell or by modifying a properties file.
- Updating the HTTP endpoint at runtime is simple:

```
FuseESB:karaf@root> config:edit org.fusesource.camel.file
FuseESB:karaf@root> config:propset fileEndpoint file:/tmp/my_directory
FuseESB:karaf@root> config:update
FuseESB:karaf@root> osgi:restart 216
```

Reference existing services

- You should reuse existing services rather than rolling your own
- Reference ActiveMQ ConnectionFactory for JMS messaging

```
<reference id="connectionFactory" interface="javax.jms.ConnectionFactory"
  filter="(name=localhost)" />

<bean id="jms" class="org.apache.activemq.camel.component.ActiveMQComponent">
  <property name="connectionFactory" ref="connectionFactory" />
</bean>
```

- Reference Aries TransactionManager for transactions

```
<reference id="transactionManager" interface="javax.transaction.TransactionManager" />
```

Decouple subsystems with JMS

- Decouple sub systems by using JMS queues hosted on ActiveMQ broker

```
<route id="file-to-jms">
  <from uri="{{fileEndpoint}}" />
  <to uri="jms:incomingOrders" />
</route>

<route id="normalize-message-data">
  <from uri="jms:incomingOrders" />
  <choice>
    <when>
      <simple>${header.CamelFileName} regex '^.*xml$'</simple>
      <unmarshal>
        <jaxb contextPath="org.fusesource.camel.model" />
      </unmarshal>
    </when>
    ...
```

Testing before deploying…

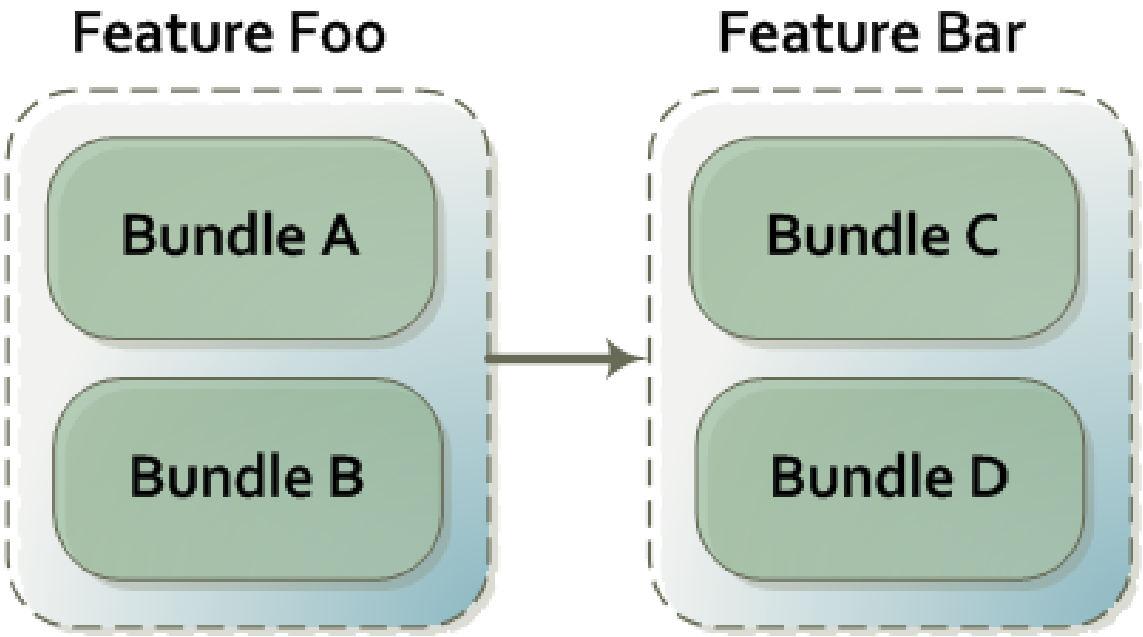
Testing

- Use camel-test-blueprint to define unit tests for your blueprint-based routes
- Use Pax Exam to test routes that require services from the container (like JMS broker, transaction manager, etc)

Grouping bundles…

Deploy with features

- Features group bundles into a logical unit of deployment
- Installing feature "Foo" would install bundles A, B, C and D



Deploy with features

- You should specify existing features in ServiceMix to depend on rather than individual bundles.

```
<features name="rider-auto-osgi"
  xmlns="http://karaf.apache.org/xmlns/features/v1.0.0">
    <feature version="${project.version}" name="rider-auto-osgi">
        <feature>camel-core</feature>
        <feature>camel-blueprint</feature>
        <feature>camel-activemq</feature>
        <feature>camel-jaxb</feature>
        <feature>camel-bindy</feature>
        <feature>camel-cxf</feature>
        <bundle>mvn:org.fusesource.examples/rider-auto-common/${project.version}</bundle>
        <bundle>mvn:org.fusesource.examples/rider-auto-file/${project.version}</bundle>
        <bundle>mvn:org.fusesource.examples/rider-auto-ws/${project.version}</bundle>
        <bundle>mvn:org.fusesource.examples/rider-auto-normalizer/${project.version}</bundle>
        <bundle>mvn:org.fusesource.examples/rider-auto-backend/${project.version}</bundle>
    </feature>
</features>
```

- You can then SSH into ServiceMix and use the features shell to install the feature.

Managing runtime routes using Karaf Camel commands…

Karaf Camel Commands

- Start/stop routes and contexts deployed in ESB
- View route XML and stats
- Many commands available

```
FuseESB:smx@root> camel:
camel:context-info      camel:context-list      camel:context-start
camel:context-stop      camel:route-info        camel:route-list
camel:route-resume      camel:route-show        camel:route-start
camel:route-stop        camel:route-suspend
```

Let’s look at the example…

Rider Auto Parts: Problem to be Solved

- Frontend receives messages from web store via SOAP/HTTP and FTP
- Message payloads can be CSV or XML from the FTP
- Backend service needs POJO payload
- There is a no downtime requirement when replacing backend

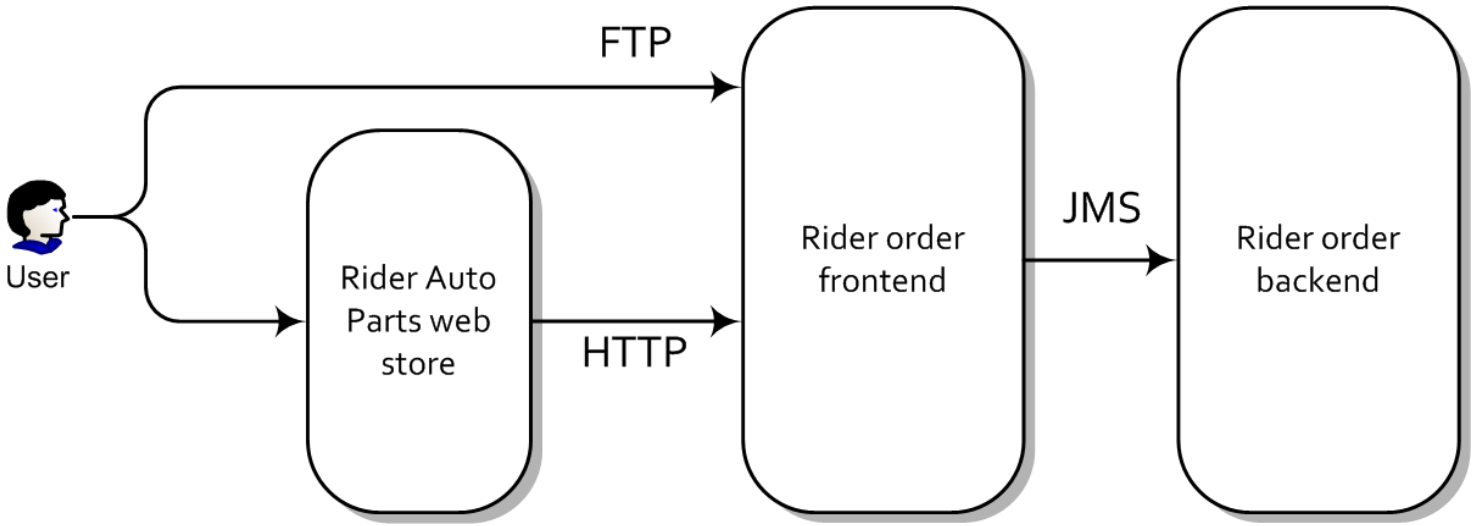
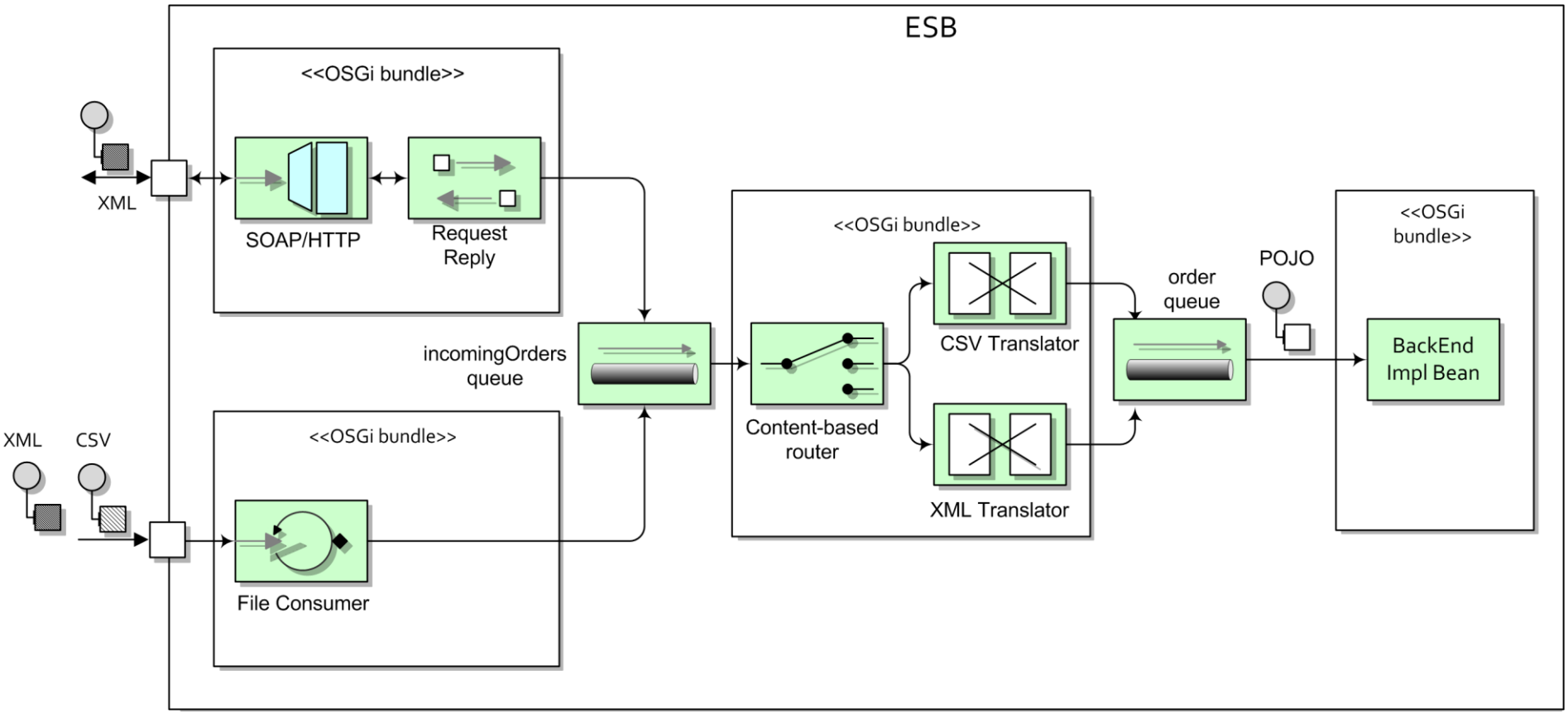


Diagram from Camel in Action.

Rider Auto Parts: Implementation Details

- Fuse ESB Enterprise (based on ServiceMix) used as the deployment container
- Each subsystem deployed as OSGi bundle
- Camel used as the integration framework
- CXF used to provide web service support
- Leverage ActiveMQ to decouple subsystems

Rider Auto Parts: Deployment Architecture



FuseSource
integration everywhere

Demo time!

Tweaking your ESB…

Deploying only what you need

- You should reduce the boot features to only what you need.
  - featuresBoot property in etc/org.apache.karaf.features.cfg
- Vanilla install of Apache ServiceMix and Fuse ESB Enterprise loads over 200 bundles

Making sure you don’t need internet access at deploy time

- Maven is great for development time as you never have to go out and download a library yourself – it just downloads from repositories on the Internet.
- In a production environment however, you should make sure all libraries are already available locally to the ESB.
  - You may not have Internet access in your environment
  - Having all libraries locally available reduces the risk of failure at deploy time
- Easy way: use the "full" distribution
  - Contains libraries for all features in system directory

Making sure you don't need internet access at deploy time

- Use the features-maven-plugin to package up all 3rd party dependencies of your application.

```
<plugin>
  <groupId>org.apache.karaf.tooling</groupId>
  <artifactId>features-maven-plugin</artifactId>
  <executions>
    <execution>
      <id>add-features-to-repo</id>
      <phase>generate-resources</phase>
      <goals>
        <goal>add-features-to-repo</goal>
      </goals>
      <configuration>
        <descriptors>
          <descriptor>mvn:org.apache.camel.karaf/apache-camel/${camel-version}/xml/features</descriptor>
          <descriptor>mvn:org.apache.servicemix/apache-servicemix/${servicemix-version}/xml/features</descriptor>
          <descriptor>mvn:org.apache.activemq/activemq-karaf/${activemq-version}/xml/features</descriptor>
          <descriptor>file:${basedir}/target/classes/features.xml</descriptor>
        </descriptors>
        <features>
          <feature>rider-auto-osgi</feature>
        </features>
        <repository>target/repo</repository>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Making sure you don’t need internet access at deploy time

- These dependencies should then be made available to ServiceMix by adding its URL to the org.ops4j.pax.url.mvn.repositories property in etc/org.ops4j.pax.url.mvn.cfg
  - Could be a local file system directory or a repository manager that you import the archive into.

Useful references

- Apache ServiceMix – http://servicemix.apache.org
- Apache Camel – http://camel.apache.org
- Fuse ESB Enterprise - http://fusesource.com/products/fuse-esb-enterprise
- Fuse IDE - http://fusesource.com/products/fuse-ide
- Camel in Action - http://manning.com/ibsen
- Customizing ESB libs - http://fusesource.com/docs/esbent/7.0/esb_deploy_osgi/Locate-CustomRepo.html
- Example source - https://github.com/janstey/rider-auto-osgi

Any Questions?

No vendor lock-in

Free to redistribute

Enterprise class….

FuseSource
integration everywhere