JBoss Community

Service-Oriented Integration

JBoss Community

Agenda

- Implementing a Service
  - Bean Services
  - Camel Services
- Binding a Service
  - SOAP Gateway
  - Camel Gateway
  - HornetQ Gateway
- Data Transformation
- Testing Services

Implementing a Service

JBoss Community

Bean Services

- POJO = Service ... ‘nuff said
- Easy to use
  - Annotation-based
  - Config auto-generated
  - Service auto-registered
- Consistent with core principles
  - Services declare a service interface
  - References injected based on service interface
  - Dependencies are explicit

Bean Services

- Implemented as a CDI Extension
  - Standard programming model (JSR 299)
  - ”The *theme* of CDI is *loose-coupling with strong typing*.”
- Weld provides the implementation framework
  - Less work for us
  - More features for users
- Straightforward integration into the web tier

Providing a Service

- Create a Java interface representing the contract
- Create a Java class implementing the interface
- Add an @Service annotation

Service Interface

In-Only

```
public interface OrderService {
    void submitOrder(Order order);
}
```

In-Out

```
public interface OrderService {
    OrderAck submitOrder(Order order);
}
```

In-Out
with Fault

```
public interface OrderService {
    OrderAck submitOrder(Order order)
        throws OrderProcessingException;
}
```

Service Implementation

This is where the magic happens

```
@Service(OrderService.class)
public class OrderServiceBean implements OrderService {

    public OrderAck submitOrder(Order order) {
     ...
    }
}
```

Consuming a Service

- Add a field representing the consumed service
- Add an @Reference annotation
- Invoke methods on the injected reference

Service Reference

```
@Service(OrderService.class)
public class OrderServiceBean implements OrderService {

    @Inject @Reference
    private InventoryService inventory;

    public OrderAck submitOrder(Order order) {
        // Check the inventory
        Item orderItem = inventory.lookupItem(order.getItemId());
        ...
    }
}
```

Dealing with Context

```
@Service(OrderService.class)
public class OrderServiceBean implements OrderService {

    @Inject @Reference
    private InventoryService inventory;

    @Inject
    private Context context;

    public OrderAck submitOrder(Order order) {
        // Get order priority
        int priority = context.getProperty(“priority”).getValue();
        ...
    }
}
```

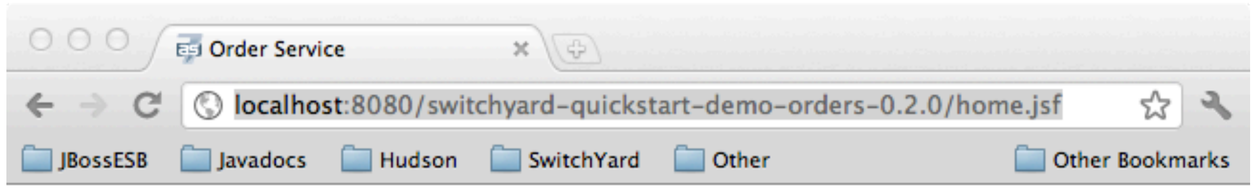
Into the Web Tier

```
<div id="content">
  <h1>New Order</h1>
  <h:form id="newOrder">
    <div>
     Order ID:
     <h:inputText id="orderID" value="#{order.orderId}" required="true"/><br/>
     Item ID:
     <h:inputText id="itemID" value="#{order.itemId}" required="true"/><br/>
     Quantity:
     <h:inputText id="quantity" value="#{order.quantity}" required="true"/><p/>
     <h:commandButton id="createOrder" value="Create" action="#{order.create}"/>
     </div>
  </h:form>
</div>
```

Into the Web Tier

```
@Named
@RequestScoped
public class Order implements Serializable {

    @Inject
    @Reference
    private OrderService orderService;

    public void create() {
        OrderAck serviceAck = orderService.submitOrder(this);
        FacesContext.getCurrentInstance().addMessage(null,
              new FacesMessage(serviceAck.toString()));
    }

    // Setters and getters omitted for brevity
}
```

JSF + CDI + SwitchYard



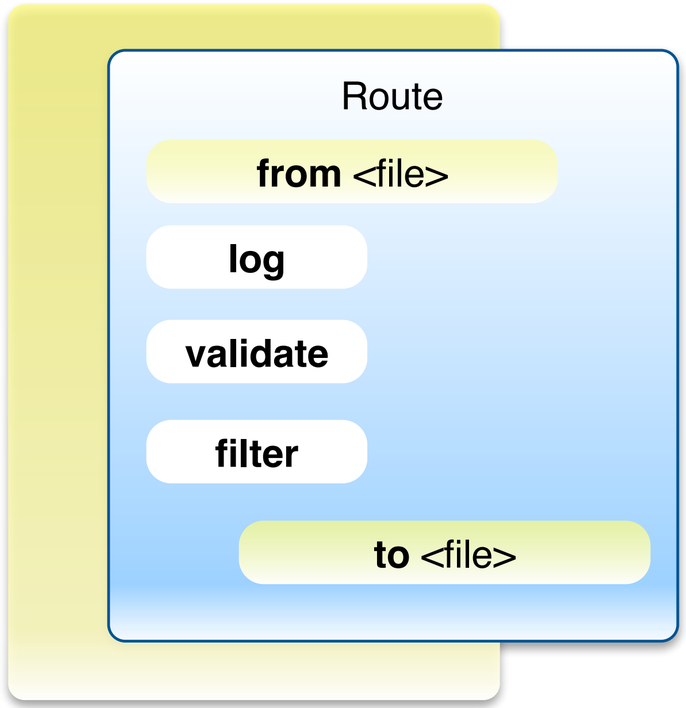
Camel Services

- Integrates Apache Camel with SwitchYard
- Camel provides
  - Routing engine and language(s)
  - Loads of EIP
  - Cornucopia of components
- Camel as a service
  - Routes provide pipeline orchestration
  - Service interface
  - Service references resolved independent of binding

Example Route

```
<route>
  <from uri="file://orders/in"/>
  <log message="Order Received : ${body}"/>
  <to uri="prioritize”/>
  <filter>
    <xpath>/order[@priority='high']</xpath>
    <to uri="file://shipping/in"/>
  </filter>
</route>
```

Example Route



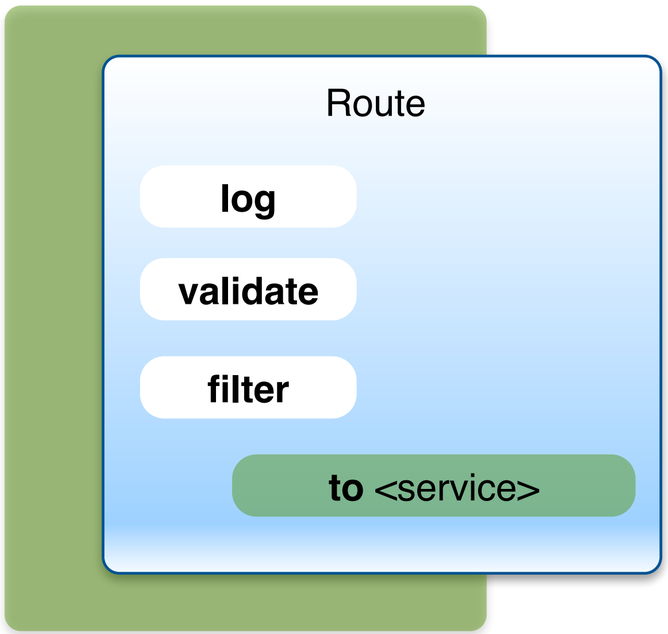
Route As A Service

```
<sca:component name="CamelComponent">

   <sca:service name="OrderService" >
      <sca:interface.java interface="org.example.OrderService"/>
   </sca:service>

   <sca:reference name="ShippingService">
      <sca:interface.java interface="org.example.ShippingService"/>
   </sca:reference>

   <implementation.camel>
      <route>
        <log message="Order Received : ${body}"/>
        <to uri="prioritize”/>
        <filter>
          <xpath>/order[@priority='high']</xpath>
          <to uri="switchyard://ShippingService"/>
        </filter>
      </route>
   </implementation.camel>

</sca:component>
```

Same Route ...

```
@Route(OrderService.class)
public class OrderServiceBuilder extends RouteBuilder {

    public void configure() {
        from("switchyard://OrderService")
            .log("Order Received : ${body}")
            .to("bean:prioritize")
            .filter().xpath("/order[@priority='high']""))
            .to("switchyard://ShippingService");
    }
}
```

Route As A Service



Beans In Camel

- Allows Java objects to be called inside a route
- Very useful for fine-grained integration tasks
  - EIP configuration - route, split, etc.
  - Metadata access
  - Bolt-on logic
- Bean registry is pluggable

CDI Beans In Camel

- Keep the same programming model you use for services
- Wired to Camel route based on @Named annotation

Camel Route

```
@Route(MyService.class)
public class MyServiceBuilder extends RouteBuilder {
  public void configure() {
    from("switchyard://MyService")
      .split(body(String.class).tokenize("\n"))
      .filter(body(String.class).startsWith("item:"))
      .to("bean:MyBean");
    }
}
```

Kapow!

CDI Bean

```
@Named("MyBean")
@ApplicationScoped
public class SomeBean {
  public void foo() {
    …
  }
}
```

Binding a Service

JBoss Community

SOAP Gateway

- Provides SOAP/HTTP binding for services and references
- Service contract based on WSDL
- Content model is XML
  - No messy composition configuration required
  - Composition can be configured though :-)
- Implemented as JAX-WS provider
- Binding configuration in SCA descriptor
- Forge tooling support

Binding Configuration

```
<service name="OrderService" promote="OrderService">
   <interface.wsdl interface="wsdl/OrderService.wsdl#wsdl.porttype(OrderService)"/>
   <binding.soap>
      <wsdl>wsdl/OrderService.wsdl</wsdl>
      <serverPort>18001</serverPort>
   </binding.soap>
</service>
```

Camel Gateway

- Allows Camel components to be used as gateways
- Flexible schema allows for XML or URI endpoint configuration

XML-Based

```
<camel:binding.file>
   <camel:operationSelector operationName="print" />
   <camel:consume>
      <camel:inputDir>/tmp/in</camel:inputDir>
      <camel:autoCreate>true</camel:autoCreate>
      <camel:initialDelay>10</camel:initialDelay>
      <camel:delete>true</camel:delete>
   </camel:consume>
</camel:binding.file>
```

URI-Based

```
<camel:binding.camel
   configURI="file://tmp/in?autoCreate=true&amp;initialDelay=10&amp;delete=true">
   <camel:operationSelector operationName="print"/>
</camel:binding.camel>
```

JBoss Community

HornetQ Gateway

- Bind services and references to HornetQ destinations
- 8.2 million messages per second in SpecJMS
- Two different ways to use it
  - Camel Gateway Component - JMS
  - SwitchYard HornetQ Component - Core API

```
<service name="GreetingService" promote="GreetingService">
   <hornetq:binding.hornetq>
      <hornetq:operationSelector operationName="greet"/>
      <hornetq:config>
         <hornetq:connector>
            <hornetq:factoryClass>org.hornetq.core.remoting.impl.invm.InVMConnectorFactory</hornetq:factoryClass>
         </hornetq:connector>
         <hornetq:queue>jms.queue.GreetingServiceQueue</hornetq:queue>
      </hornetq:config>
   </hornetq:binding.hornetq>
</service>
```

JBoss Community

Data Transformation

JBoss Community

Transformers

- Transformation is wired into SwitchYard core
  - Types declared via service contract
  - Transformer resolved dynamically at runtime
- Bring on the canonical data models
- Current Transformers
  - Java, Smooks, XSLT, JAXB, JSON

Java Transformer

- Implement transformation directly in Java
- Two options
  - Implement `org.switchyard.transform.Transformer`
  - Annotate one or more methods with `@Transformer`
- Provides greatest flexibility, but least implementation help
  - Useful for converters and basic transformations

Java Transformer

```
@Transformer(from = "{urn:switchyard-quickstart-demo:orders:1.0}submitOrder")
  public Order transform(Element from) {
     return new Order()
          .setOrderId(getElementValue(from, "orderId”))
          .setItemId(getElementValue(from, "itemId”))
          .setQuantity(Integer.valueOf(getElementValue(from, "quantity”)));
  }
```

JAXB Transformer

- Translate from XML to Java and vice versa
- Based on JAXB ObjectFactory
  - Easy to generate using xjc, wsconsume, etc.
- Automatic registration
  - If object used in service contract contains JAXB annotations, runtime automatically picks it up during deployment

Document Schema

```
<xsd:schema
    targetNamespace="urn:switchyard-quickstart:transform-jaxb:1.0"
    xmlns:tns="urn:switchyard-quickstart:transform-jaxb:1.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="order" type="tns:order"/>
    <xs:complexType name="order">
        <xs:sequence>
            <xs:element name="orderId" type="xs:string"/>
            <xs:element name="itemId" type="xs:string"/>
            <xs:element name="quantity" type="xs:int"/>
        </xs:sequence>
    </xs:complexType>
</xsd:schema>
```

ObjectFactory

```
@XmlRegistry
public class ObjectFactory {

    @XmlElementDecl(
       namespace = "urn:switchyard-quickstart:transform-jaxb:1.0",
       name = "order")
    public JAXBElement<Order> createOrder(Order value) {
        return new JAXBElement<Order>(_Order_QNAME, Order.class, null, value);
    }
}
```

Payload Object

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
    "orderId",
    "itemId",
    "quantity"
})
@XmlRootElement(name = "order")
public class Order {

    @XmlElement(required = true)
    protected String orderId;
    @XmlElement(required = true)
    protected String itemId;
    protected int quantity;

    ...
}
```

Service Interface

```
public interface OrderService {

    OrderAck submitOrder(Order order);

}
```

XML->Order Transformer
Is Created Automatically

XSLT Transformer

- Convenient mechanism for XML to XML transformation
- Declared in application descriptor

```
<transform.xslt
      from="{http://acme/}A"
      to="{http://acme/}B"
      xsltFile="com/acme/xslt/A2B.xslt"/>
```

JSON Transformer

- JSON marshalling
  - JSON -> Object, Object -> JSON
- Based on Jackson Data Binding
- Works well for JavaBean payloads

```
<trfm:transform.json
    from="java:org.switchyard.quickstarts.transform.json.OrderAck"
    to="{urn:switchyard-quickstart:transform-json:1.0}orderResponse"/>
```

Smooks Transformer

- Extensive support for XML and non-XML mapping
  - CSV, EDI, Java, ...
- Beyond basic transformation
  - Enrichment
  - Validation
  - Persistence
- Multiple expressions
  - Freemarker, Groovy, MVEL, ...

Testing Services

JBoss Community

Service Test

```
@RunWith(SwitchYardRunner.class)
@SwitchYardTestCaseConfig(mixins = CDIMixIn.class)
public class InventoryServiceTest {

    @ServiceOperation("InventoryService.lookupItem")
    private Invoker lookupItem;

    @Test
    public void testItemLookupExists() throws Exception {
        final String ITEM_ID = "BUTTER";
        Item item = lookupItem
            .sendInOut(ITEM_ID)
            .getContent(Item.class);

        Assert.assertNotNull(item);
        Assert.assertEquals(ITEM_ID, item.getItemId());
    }
}
```

Service Test

Bootstraps SwitchYard runtime and handles test injection

```
@RunWith(SwitchYardRunner.class)
@SwitchYardTestCaseConfig(mixins = CDIMixIn.class)
public class InventoryServiceTest {

    @ServiceOperation("InventoryService.lookupItem")
    private Invoker lookupItem;

    @Test
    public void testItemLookupExists() throws Exception {
        final String ITEM_ID = "BUTTER";
        Item item = lookupItem
            .sendInOut(ITEM_ID)
            .getContent(Item.class);

        Assert.assertNotNull(item);
        Assert.assertEquals(ITEM_ID, item.getItemId());
    }
}
```

Service Test

Bootstraps SwitchYard runtime and handles test injection

Helper methods for CDI including Bean Scanning

```
@RunWith(SwitchYardRunner.class)
@SwitchYardTestCaseConfig(mixins = CDIMixIn.class)
public class InventoryServiceTest {

    @ServiceOperation("InventoryService.lookupItem")
    private Invoker lookupItem;

    @Test
    public void testItemLookupExists() throws Exception {
        final String ITEM_ID = "BUTTER";
        Item item = lookupItem
            .sendInOut(ITEM_ID)
            .getContent(Item.class);

        Assert.assertNotNull(item);
        Assert.assertEquals(ITEM_ID, item.getItemId());
    }
}
```

Service Test

Bootstraps SwitchYard runtime and handles test injection

Helper methods for CDI including Bean Scanning

Injects a reference to a service operation

```
@RunWith(SwitchYardRunner.class)
@SwitchYardTestCaseConfig(mixins = CDIMixIn.class)
public class InventoryServiceTest {

    @ServiceOperation("InventoryService.lookupItem")
    private Invoker lookupItem;

    @Test
    public void testItemLookupExists() throws Exception {
        final String ITEM_ID = "BUTTER";
        Item item = lookupItem
            .sendInOut(ITEM_ID)
            .getContent(Item.class);

        Assert.assertNotNull(item);
        Assert.assertEquals(ITEM_ID, item.getItemId());
    }
}
```

Service Test

Bootstraps SwitchYard runtime and handles test injection

Helper methods for CDI including Bean Scanning

Injects a reference to a service operation

```
@RunWith(SwitchYardRunner.class)
@SwitchYardTestCaseConfig(mixins = CDIMixIn.class)
public class InventoryServiceTest {

    @ServiceOperation("InventoryService.lookupItem")
    private Invoker lookupItem;

    @Test
    public void testItemLookupExists() throws Exception {
        final String ITEM_ID = "BUTTER";
        Item item = lookupItem
            .sendInOut(ITEM_ID)
            .getContent(Item.class);

        Assert.assertNotNull(item);
        Assert.assertEquals(ITEM_ID, item.getItemId());
    }
}
```

Sends and receives messages over the bus

Service Test

Bootstraps SwitchYard runtime and handles test injection

Helper methods for CDI including Bean Scanning

Injects a reference to a service operation

Declares the expected return type - triggers declarative transformation

Sends and receives messages over the bus

```
@RunWith(SwitchYardRunner.class)
@SwitchYardTestCaseConfig(mixins = CDIMixIn.class)
public class InventoryServiceTest {

    @ServiceOperation("InventoryService.lookupItem")
    private Invoker lookupItem;

    @Test
    public void testItemLookupExists() throws Exception {
        final String ITEM_ID = "BUTTER";
        Item item = lookupItem
            .sendInOut(ITEM_ID)
            .getContent(Item.class);

        Assert.assertNotNull(item);
        Assert.assertEquals(ITEM_ID, item.getItemId());
    }
}
```

Transformation Test

```
@RunWith(SwitchYardRunner.class)
@SwitchYardTestCaseConfig(mixins = SmooksMixIn.class)
public class SmooksTransformationTest {

    private SmooksMixIn smooksMixIn;

    @Test
    public void testOrderTransform() throws Exception {
        // Verify the Order_XML.xml Smooks Java->XML binding
        smooksMixIn.testJavaXMLReadWrite(
                Order.class,
                "/smooks/Order_XML.xml",
                "/xml/order.xml");
    }
}
```

Injected MixIn class provides helper methods

Binding Test

Load the application descriptor and CDI services

```
@RunWith(SwitchYardRunner.class)
@SwitchYardTestCaseConfig(
        config = SwitchYardTestCaseConfig.SWITCHYARD_XML,
        mixins = {CDIMixIn.class, HTTPMixIn.class})
public class WebServiceTest {

    private HTTPMixIn httpMixIn;

    @Test
    public void invokeOrderWebService() throws Exception {
       httpMixIn.postResourceAndTestXML(
            "http://localhost:18001/OrderService",
            "/xml/soap-request.xml",
            "/xml/soap-response.xml");
    }
}
```

This tests the service from the "outside"

Questions?

JBoss Community

JBoss Community