JUDCon

JBoss Users & Developers Conference

2012:India

Enterprise Services Made Easy

with SwitchYard

Keith Babo
Magesh Bojan

Introducing SwitchYard

- New JBoss community project
- Next generation Enterprise Service Bus
- What happened to JBoss ESB?
- Taking the next evolutionary step
  - Focus on consistent, intuitive user experience
  - Refactor core to eliminate known pain points
  - Leverage standards and complimentary technologies

Activity

- SwitchYard
  - Milestone 1 - February, 2011
  - 0.1 release - June, 2011
  - 0.2 release - August, 2011
  - 0.3 release - December, 2011
  - 0.*n* releases every 8-10 weeks
- JBoss ESB
  - SOA Platform 5.1 - March, 2011
  - JBoss ESB 4.10 - August 2011
  - SOA Platform 5.2 - November, 2011
  - SOA Platform 5.3 - In Planning

Enterprise Services

from a Developer's Perspective

- You have an application
- It contains services that other apps will use
- It may want to consume services elsewhere
- A structured development methodology would help
  - Something 'service-oriented' ... hmm
- Now for a platform that helps develop and run these type of applications
- <Insert ESB Here>

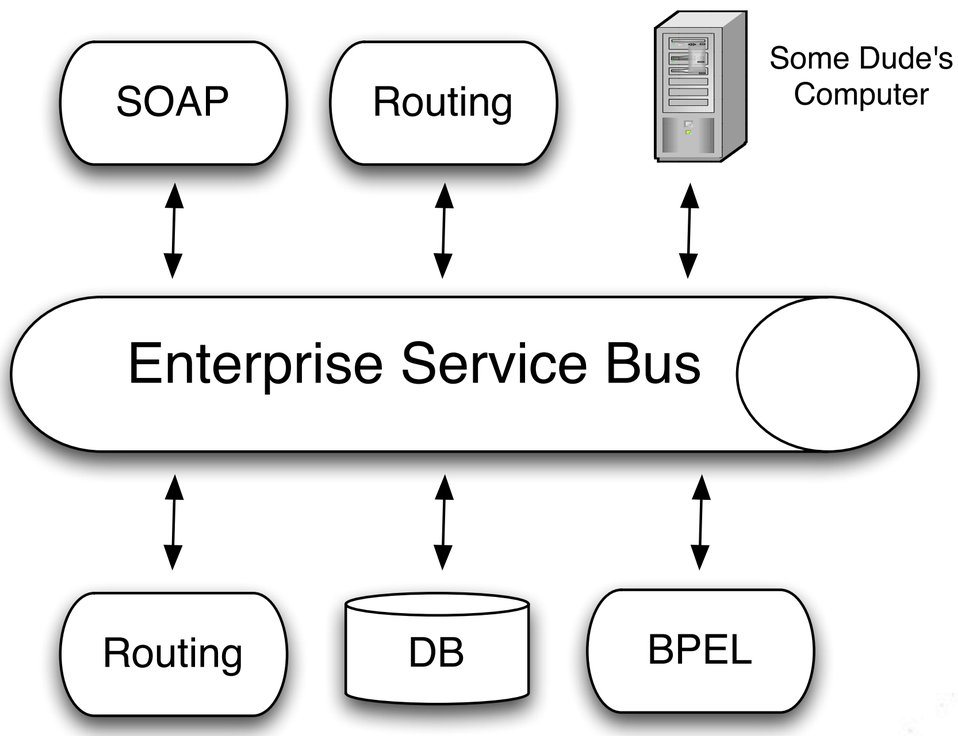
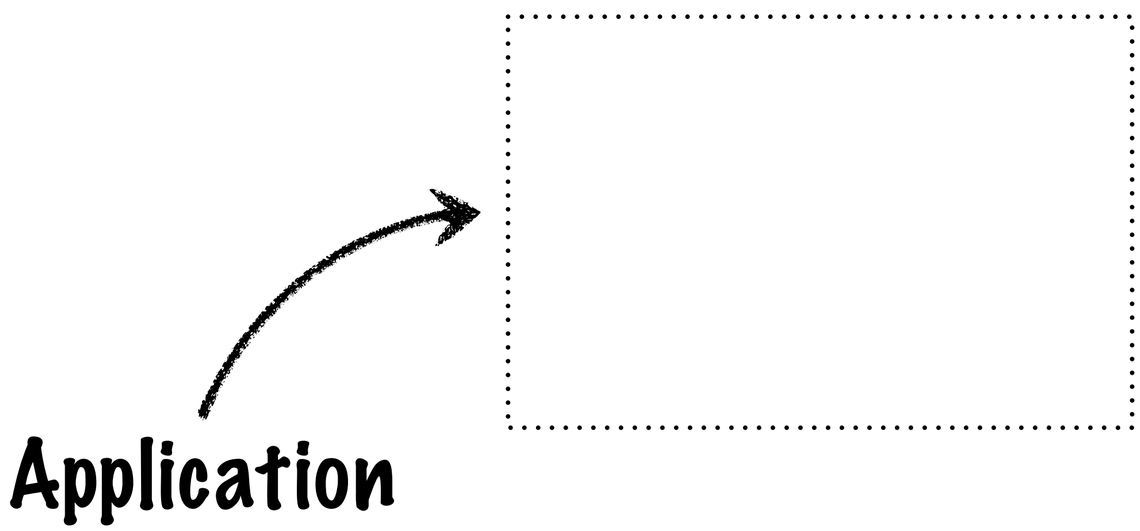
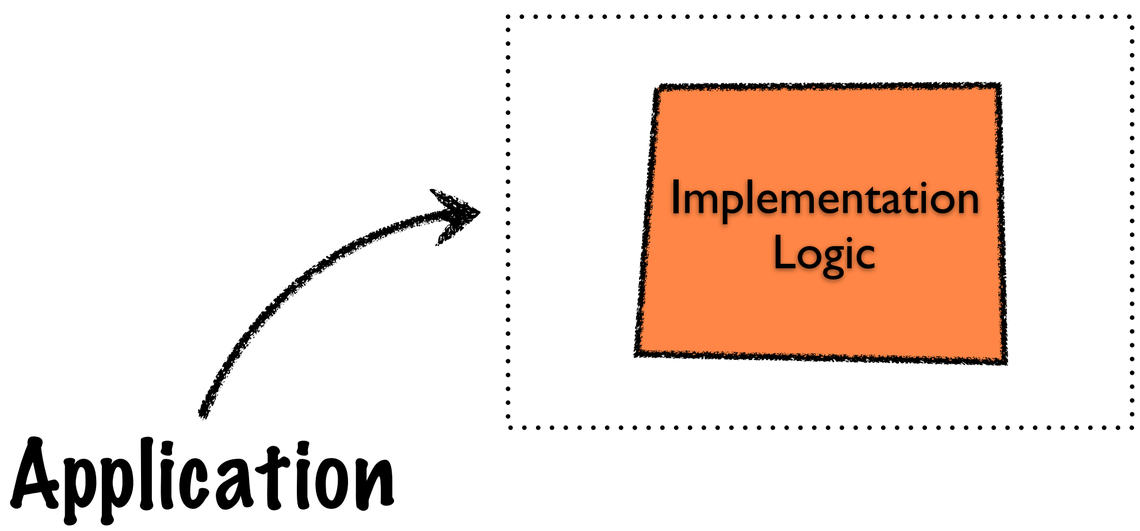


Exhibit #1 : Not Helping

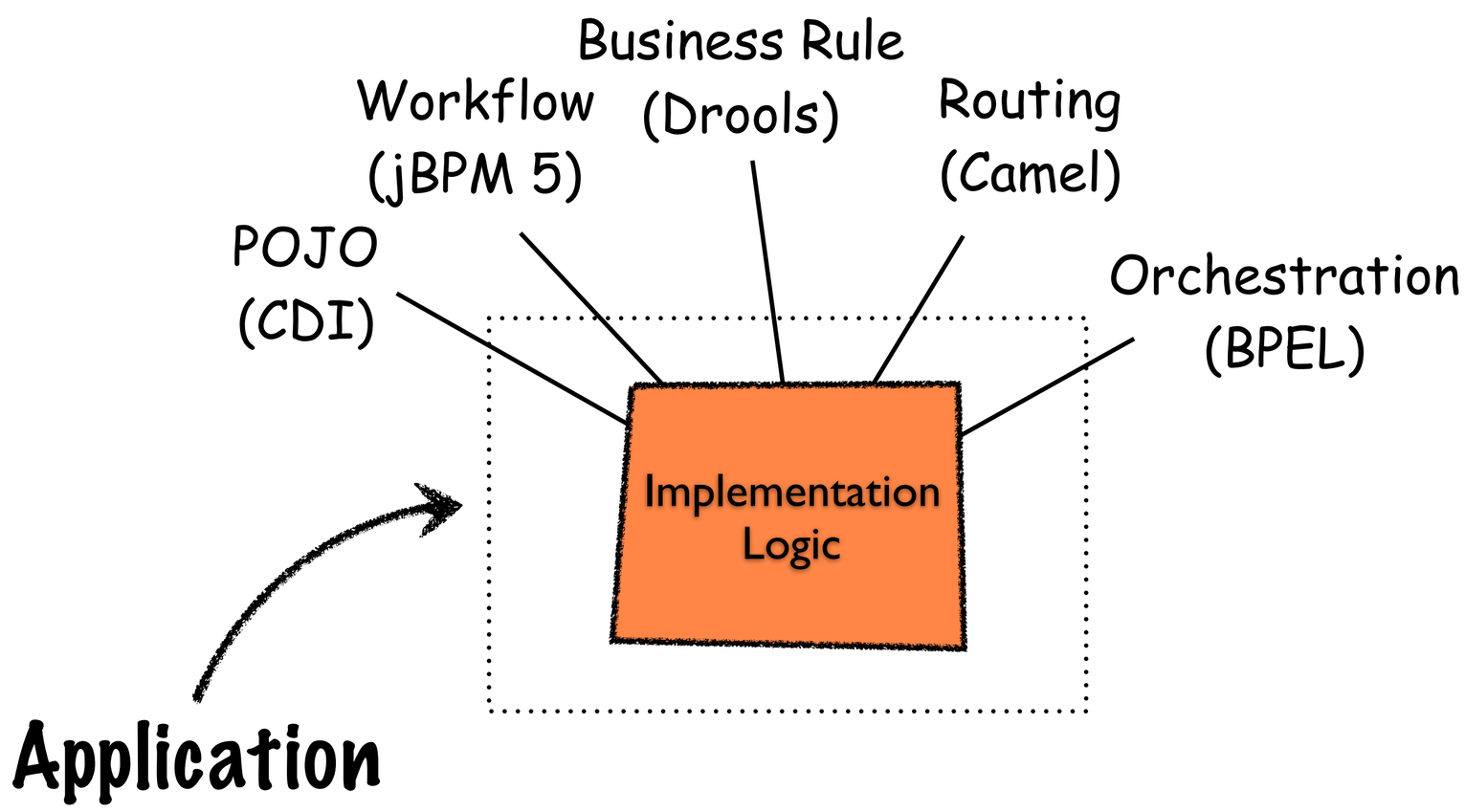
Application Architecture



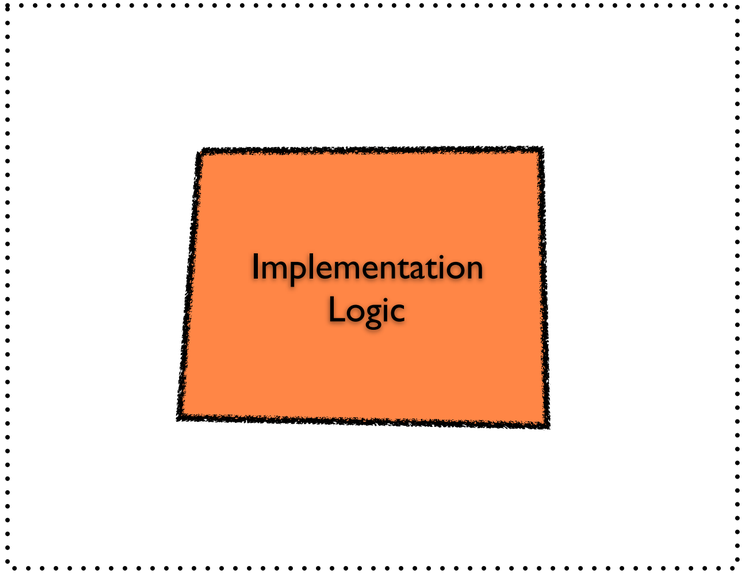
Application Architecture



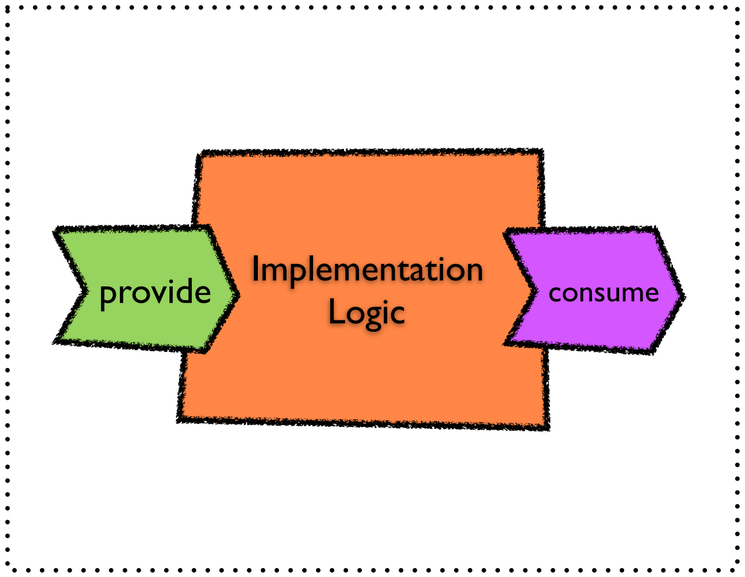
Application Architecture



Application Architecture



Application Architecture



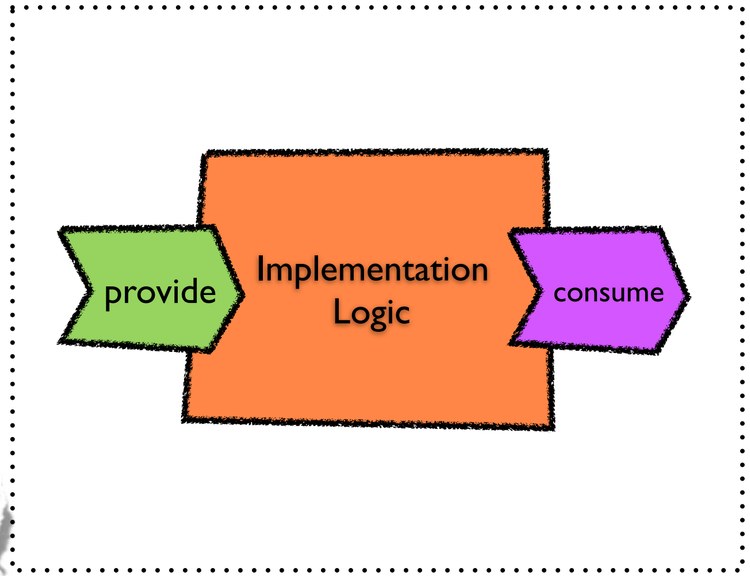
Application Architecture

Java

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

WSDL

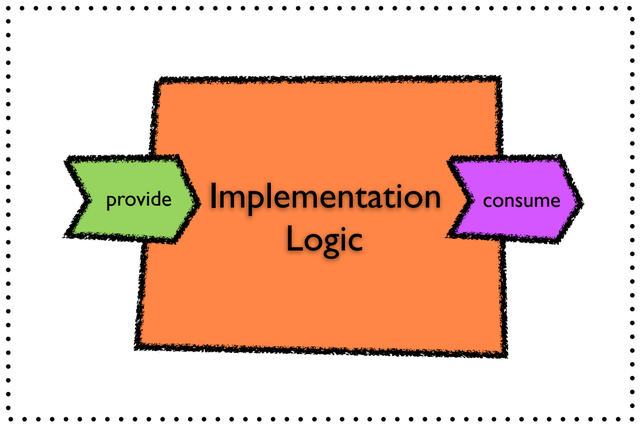
```
<portType name="OrderService">
  <operation name="submitOrder">
    <input message="tns:submitOrder"/>
    <output message="tns:submitOrderResponse"/>
  </operation>
</portType>
```



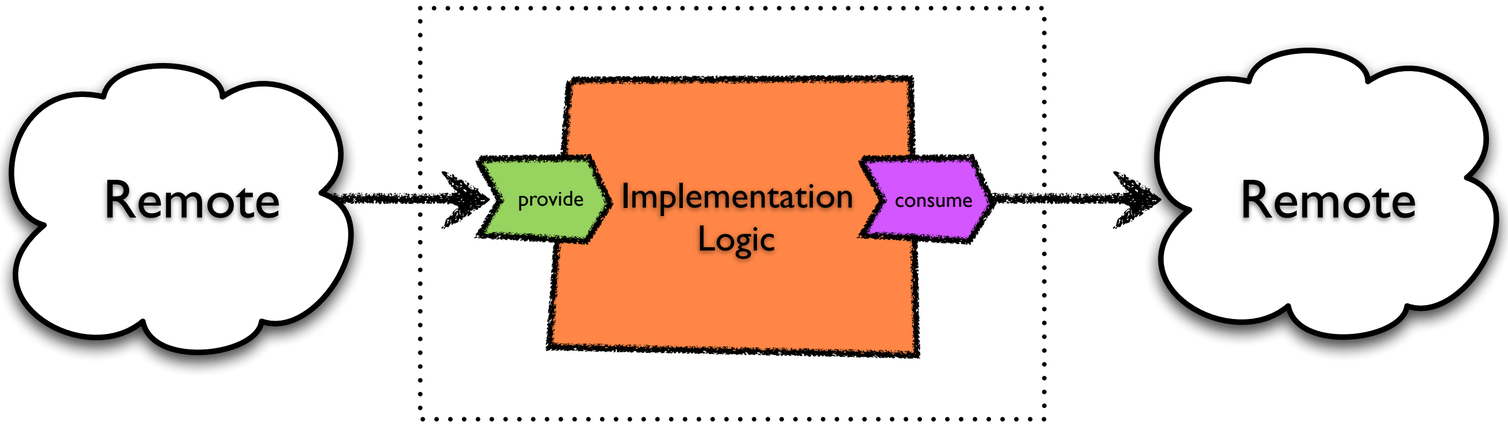
SCA

```
<component name="OrderService">
   <bean:implementation.bean
      class="org.example.OrderServiceBean"/>
   <service name="OrderService">
      <interface.java
       interface="org.example.OrderService"/>
   </service>
   <reference name="InventoryService">
      <interface.java
       interface="org.example.InventoryService"/>
   </reference>
</component>
```

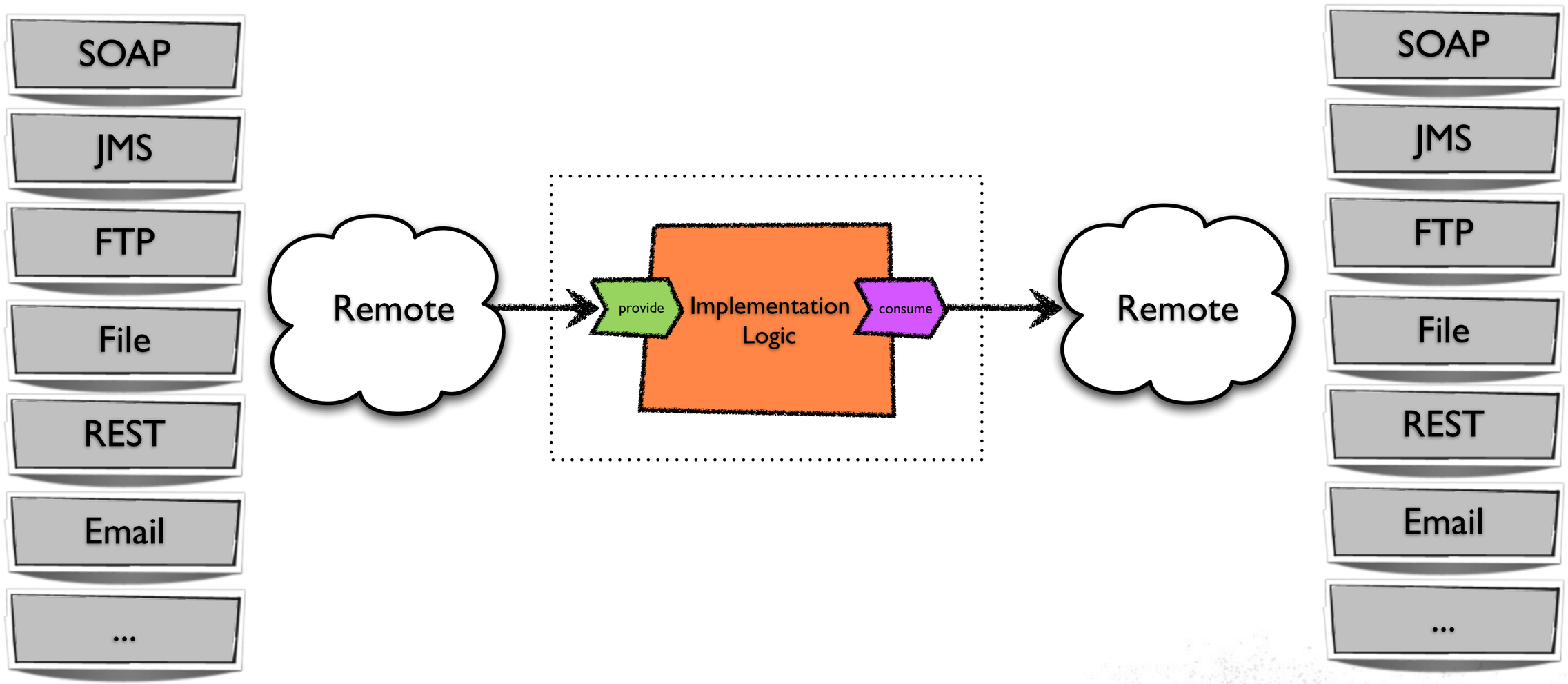
Application Architecture



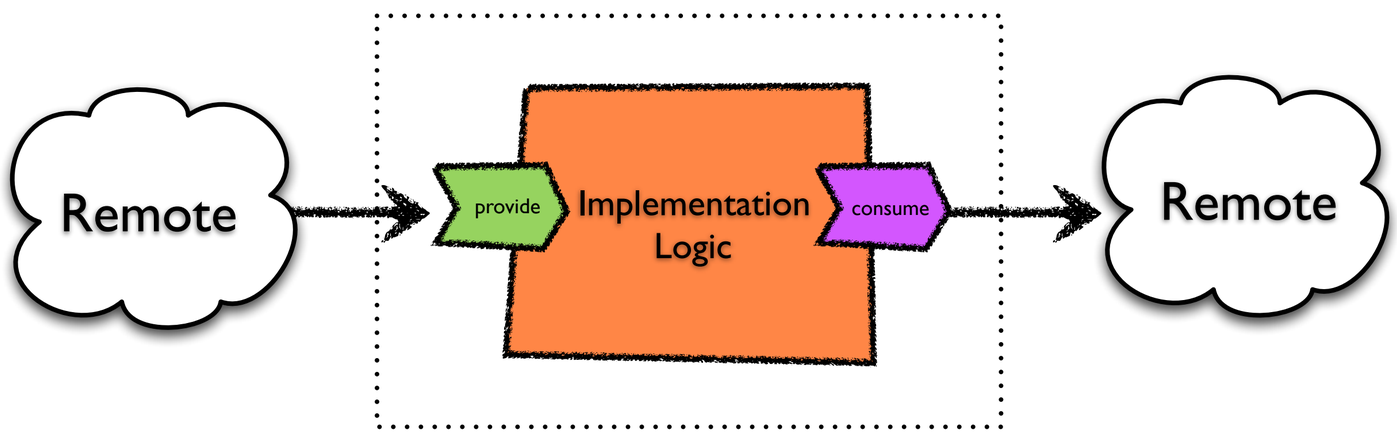
Application Architecture



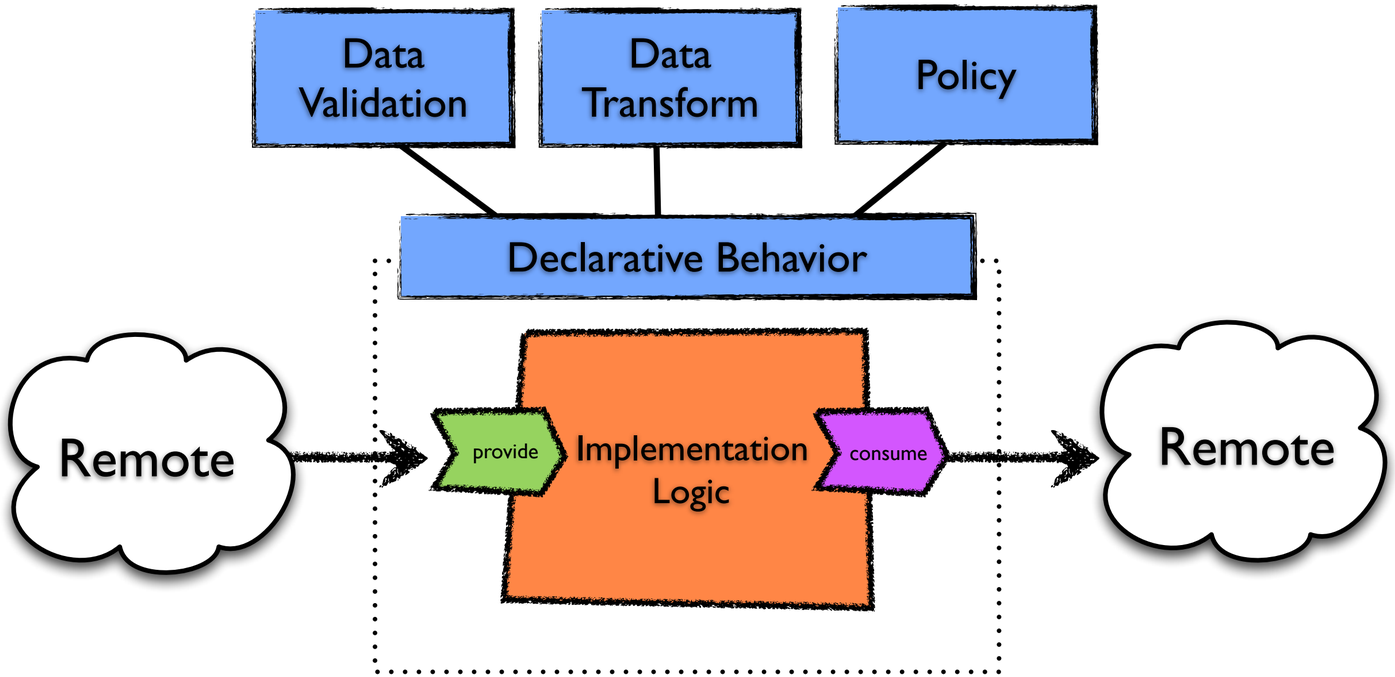
Application Architecture



Application Architecture



Application Architecture



Implementing a Service

Bean Services

- POJO = Service ... ‘nuff said
- Easy to use
  - Annotation-driven
  - Config auto-generated
  - Service auto-registered
- Based on CDI
  - Standard programming model (Java EE / JSR 299)
  - 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

Bean Service

Bean Service

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

Bean Service

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

```
public class OrderServiceBean implements OrderService {

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

Bean Service

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

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 {

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

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());
        ...
    }
}
```

More Magic

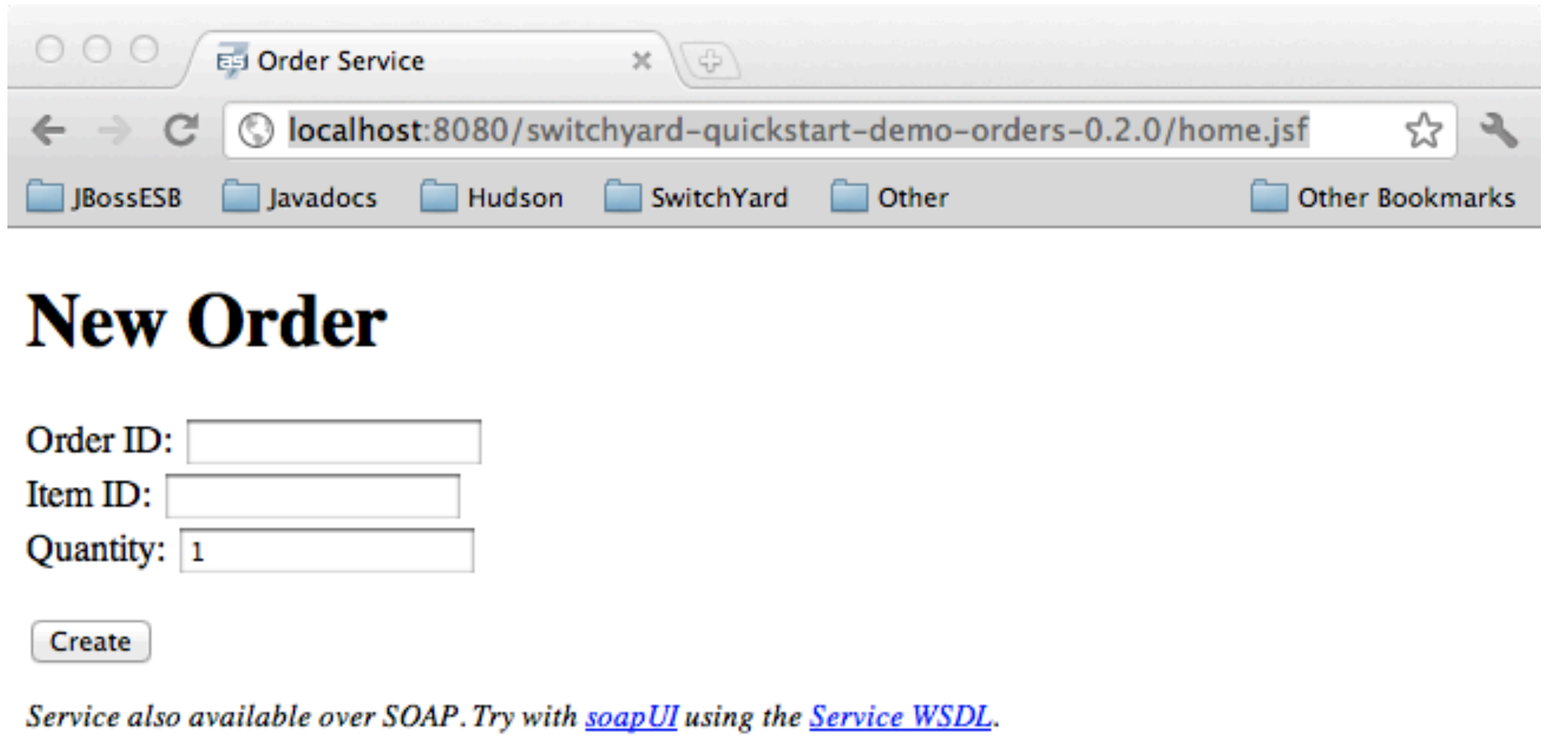
Into the Web Tier

```
<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>
```

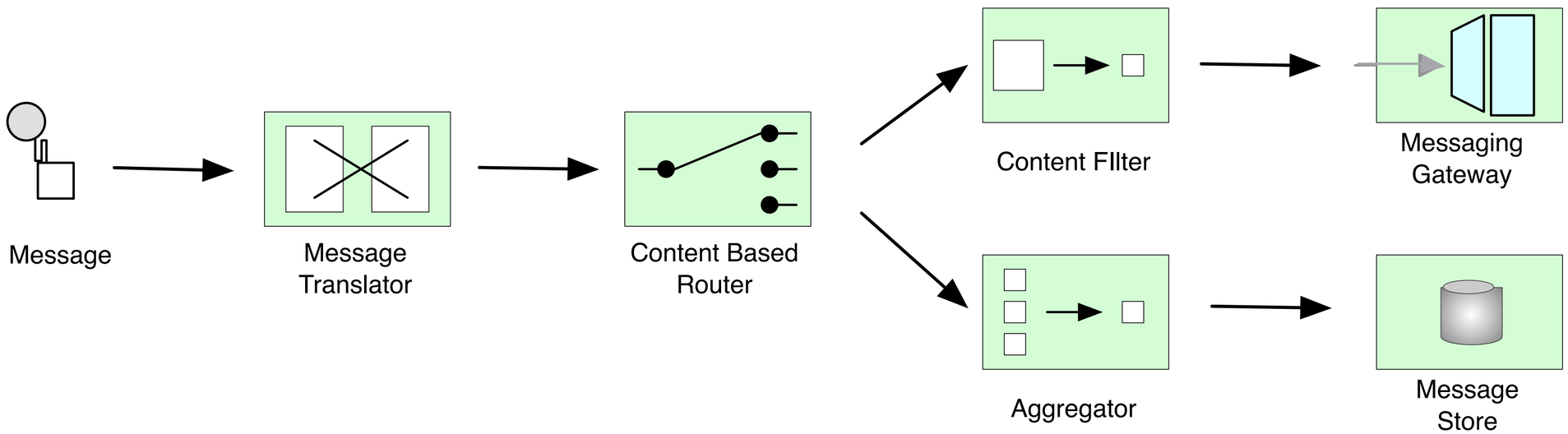
Into the Web Tier

```
@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()));
    }
    ...
}
```

JSF + CDI + SwitchYard



Routing


Message
Message Translator
Content Based Router
Content FIlter
Messaging Gateway
Aggregator
Message Store

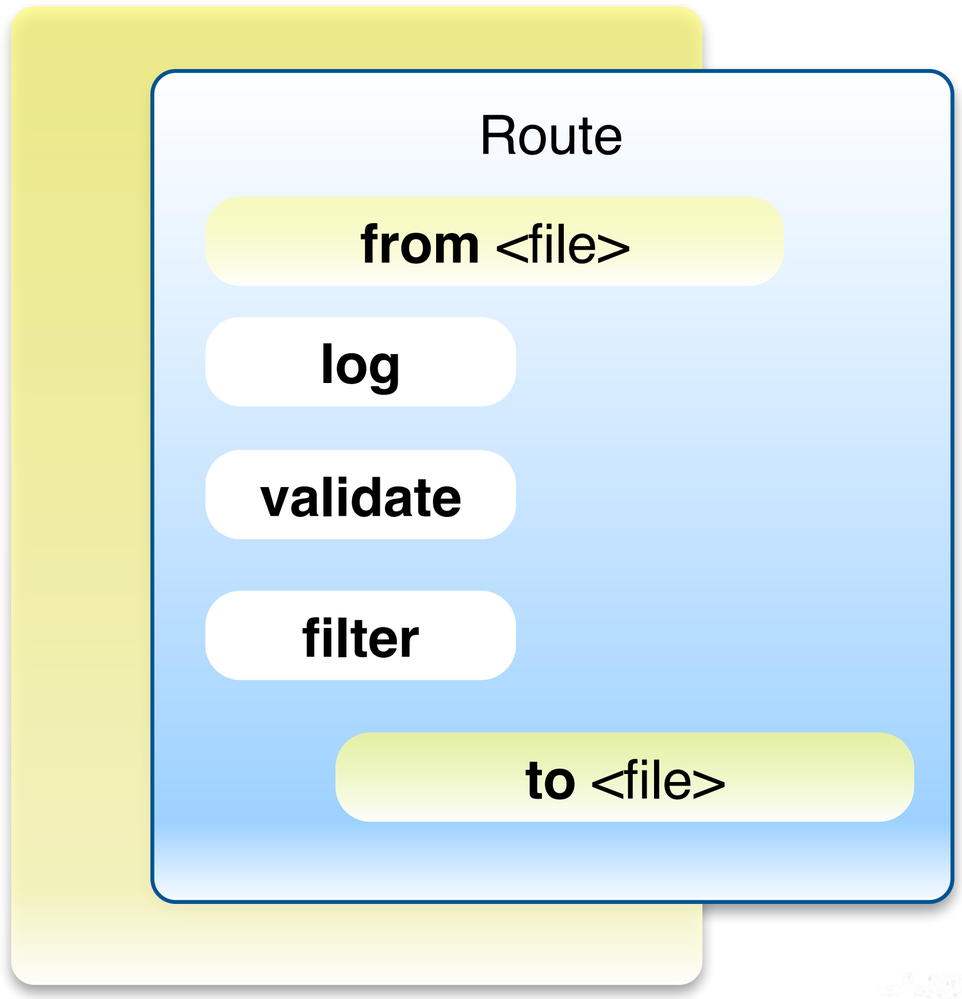
Routing Services

- 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

```
public class OrderServiceBuilder extends RouteBuilder {

    public void configure() {
        from("file://orders/in")
            .log("Order Received : ${body}")
            .to("bean:prioritize")
            .filter().xpath("/order[@priority=‘high’]”"))
            .to("file://shipping/in");
    }
}
```

Example Route



Route As A Service

```
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

```
@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");
    }
}
```

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");
    }
}
```

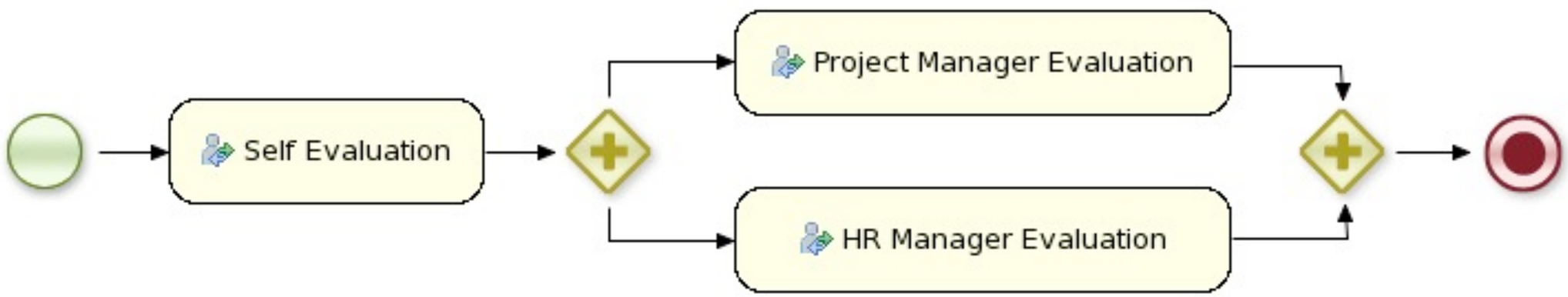
Kapow!

CDI Bean

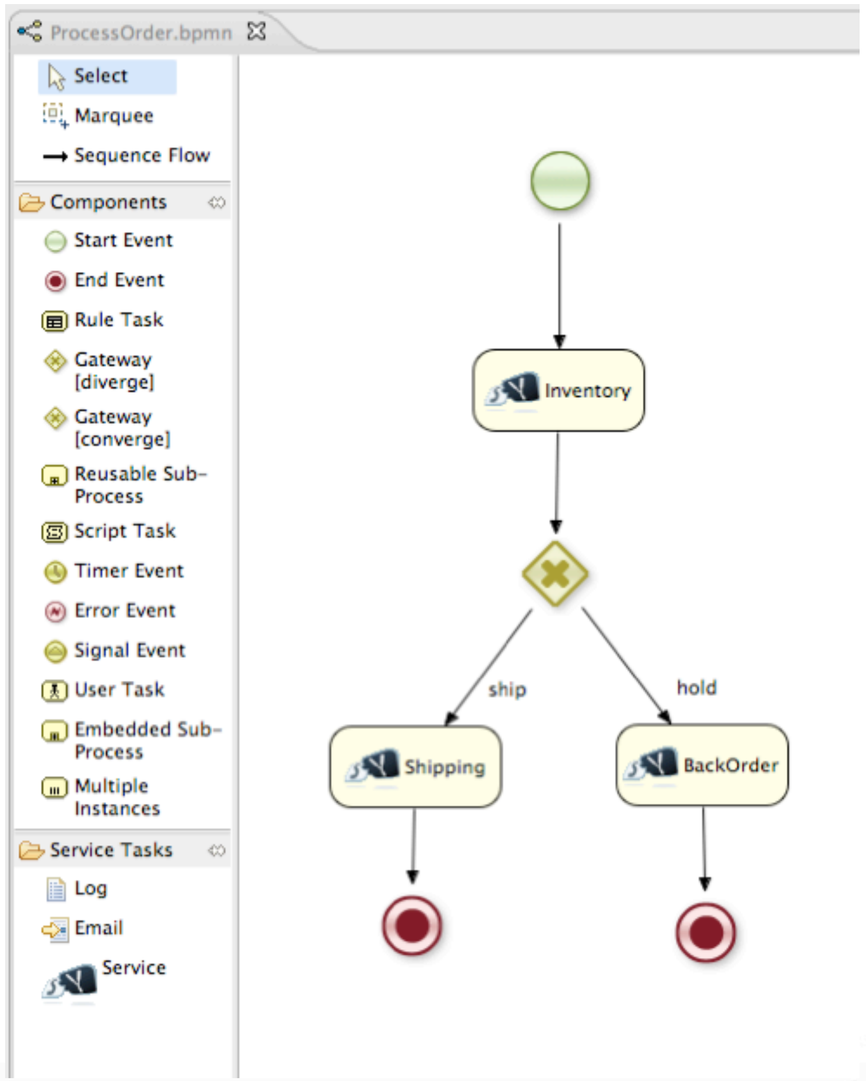
```
@Named("MyBean")
@ApplicationScoped
public class SomeBean {
  public void foo() {
    …
  }
}
```

Workflow Services

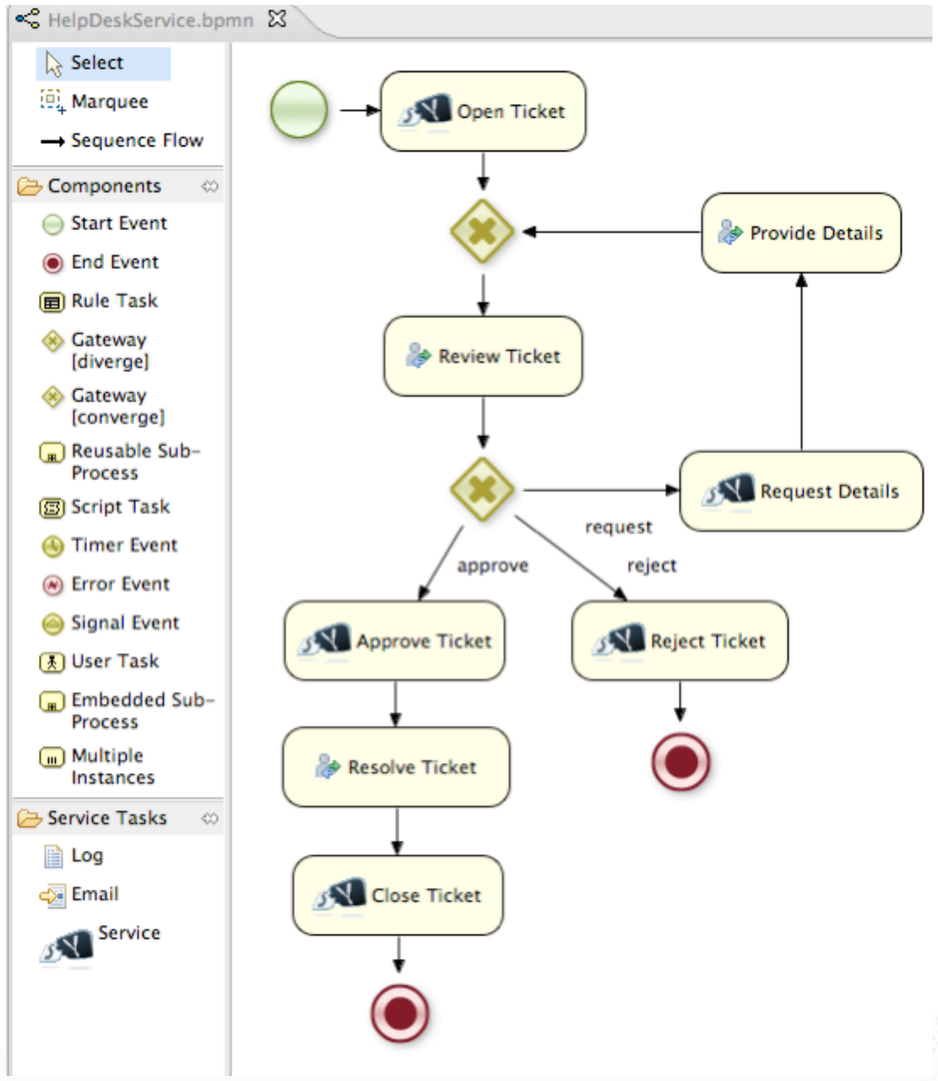
- Provides business process and human workflow support
- Based on jBPM 5
- Native integration in BPMN2 modeler



Service Orchestration



Integrated Workflow



Decision Services

- Business Rules as Services
- Based on Drools
- Provides
  - Bootstrap of Knowledge Runtime and Session
  - Explicit contract for decision service
  - Binding agnostic fact insertion
  - Data format isolation

An Example

interview.drl

```
package org.example

rule "Is of valid age"
    when
        $a : Applicant( age > 17 )
    then
        $a.setValid( true );
end

rule "Is not of valid age"
    when
        $a : Applicant( age < 18 )
    then
        $a.setValid( false );
end
```

Interview.java

```
public interface Interview {
    public void verify(Applicant applicant);
}
```

switchyard.xml

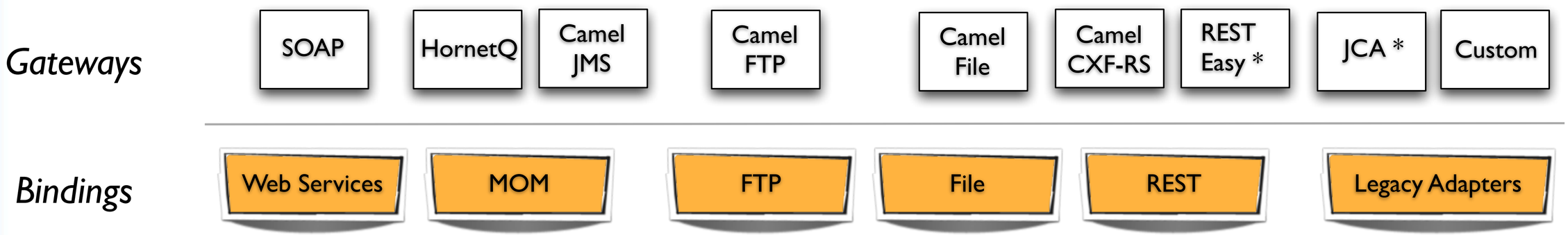
```
<implementation.rules stateful="false">
   <rulesAction name="verify" type="EXECUTE_RULES"/>
   <resource
      location="/rules/interview/Interview.drl"
      type="DRL"/>
</implementation.rules>
```

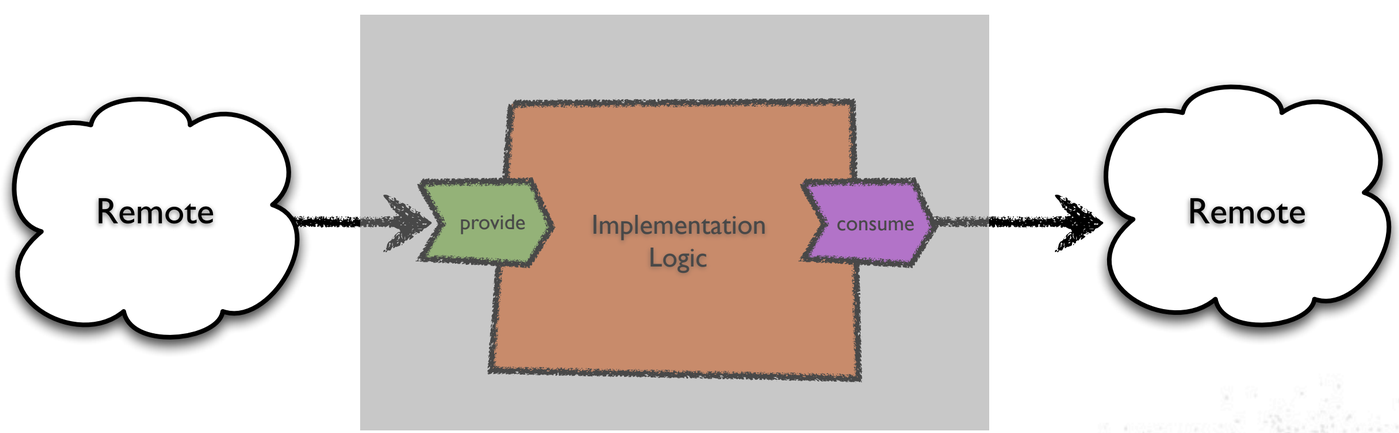
Orchestration Services

- Orchestrate web services using BPEL
- Backed by Riftsaw
  - Apache ODE base
  - GWT-based console
  - Short-lived and long-running processes
  - Process persistence and recovery
  - Process versioning
- WSDL-based contracts a natural fit for BPEL

Binding a Service

Service Bindings





* coming soon

SOAP Gateway

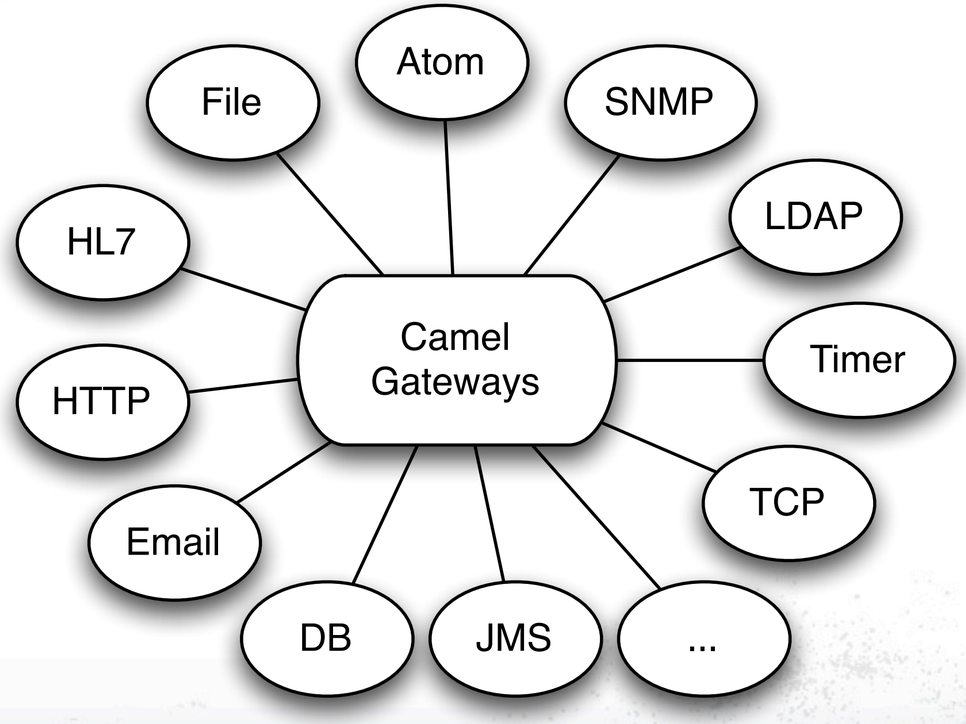
- SOAP binding for services and references
- Service contract based on WSDL
- Message payload is XML
  - XML through the bus
- Implemented as JAX-WS provider
- Binding configuration in SCA descriptor
- Forge tooling support

Binding Configuration

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

<reference name="InventoryService" promote="InventoryService">
   <binding.soap>
      <wsdl>wsdl/InventoryService.wsdl</soap:wsdl>
   </soap:binding.soap>
</reference>
```

Camel Gateway

- Allows Camel components to be used as gateways
- XML or URI-based endpoint configuration
- File, Timer, and JMS included with AS7 distribution
  - Others can be added as modules



Binding 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>
```

HornetQ Gateway

- Bind 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>
```

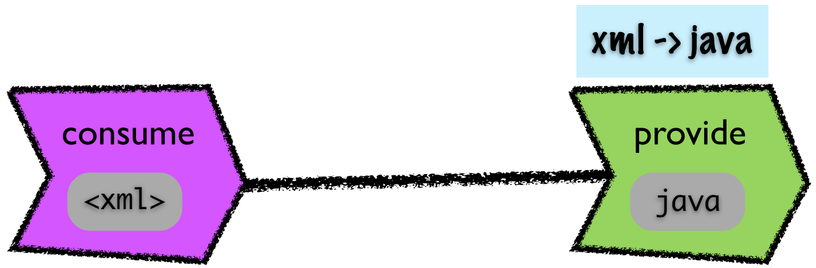
Declarative Behavior

Transformation

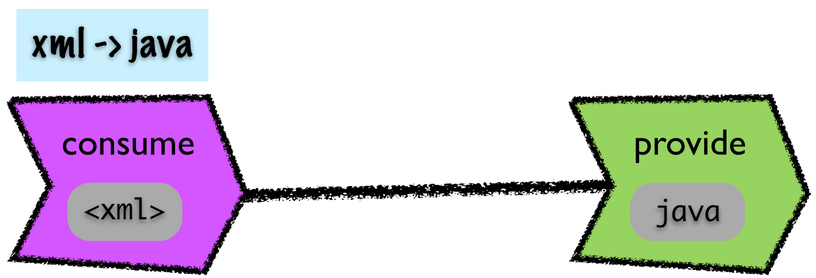
- Ubiquitous challenge in application integration and SOA
- Change in data representation
  - java.io.Reader -> java.lang.String
- Change in data format
  - CSV -> XML
- Change in data itself
  - Enrichment
- Multiple ways to handle this requirement

Where To Transform

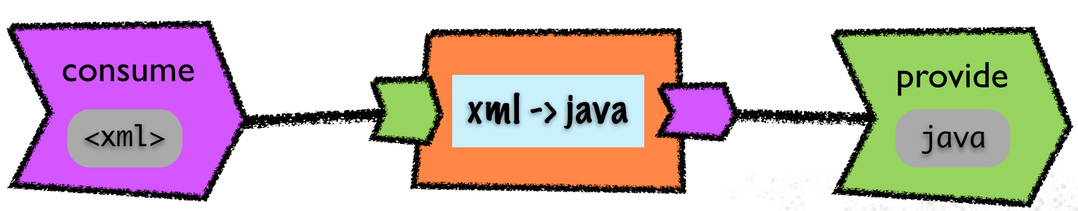
- In the provider?



- In the consumer?



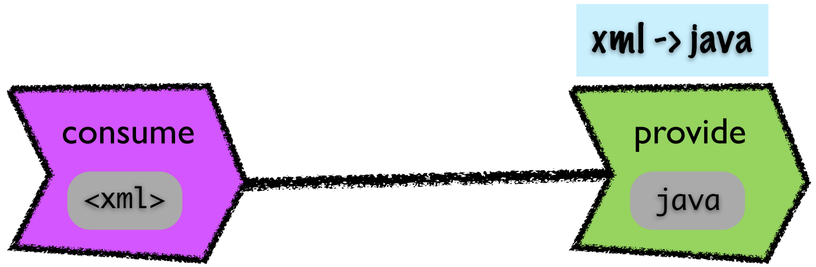
- Add a routing service?



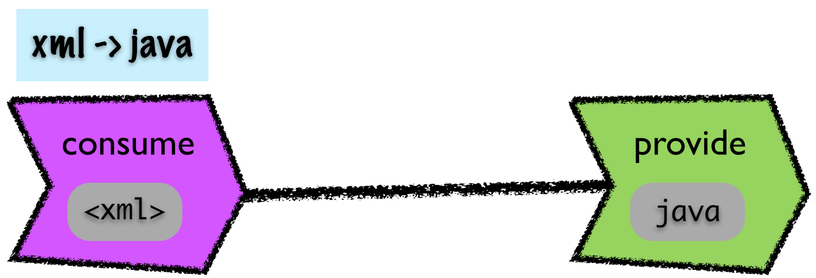
Where To Transform

- In the provider?

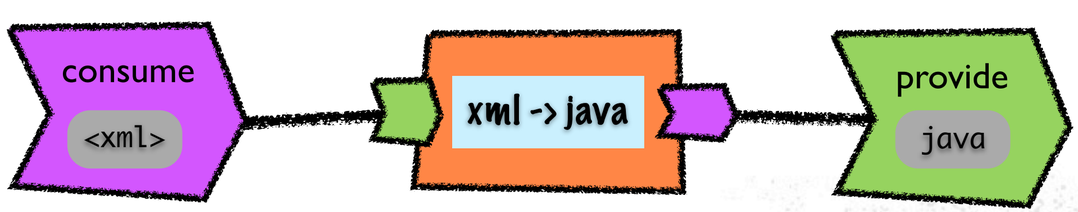
  NO!



- In the consumer?



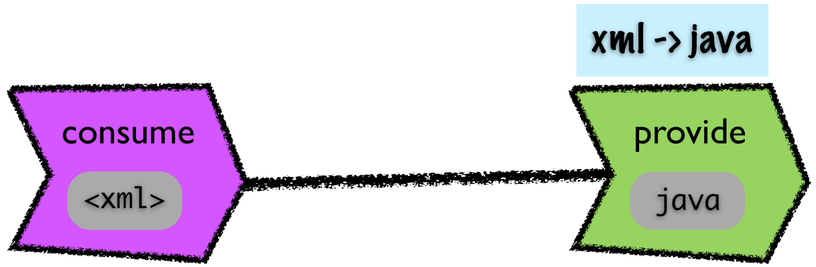
- Add a routing service?



Where To Transform

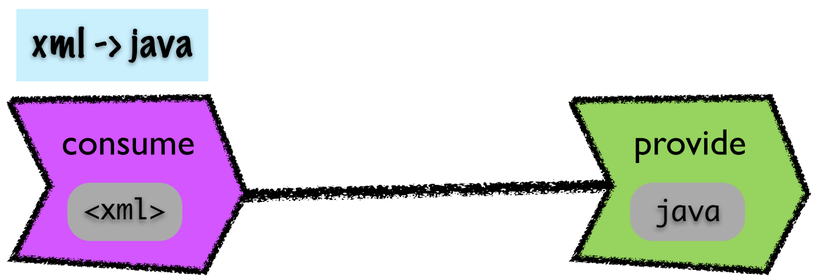
- In the provider?

NO!

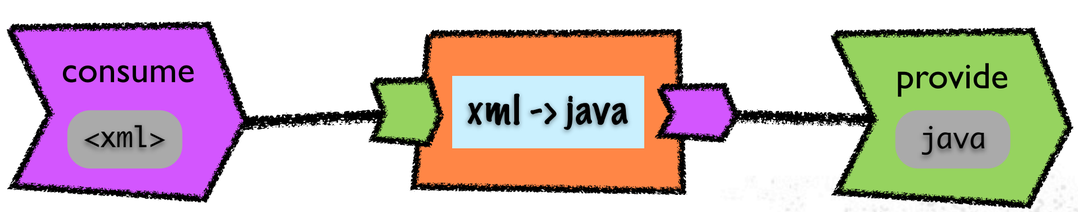


- In the consumer?

NO!



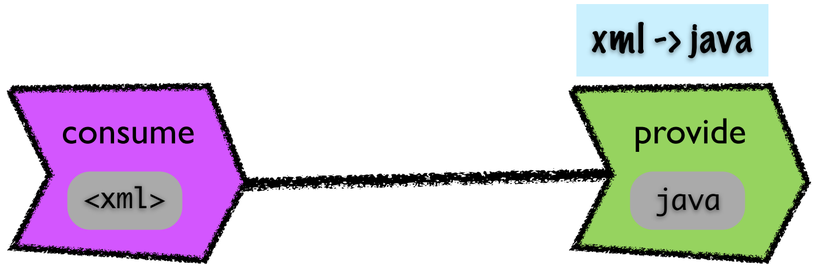
- Add a routing service?



Where To Transform

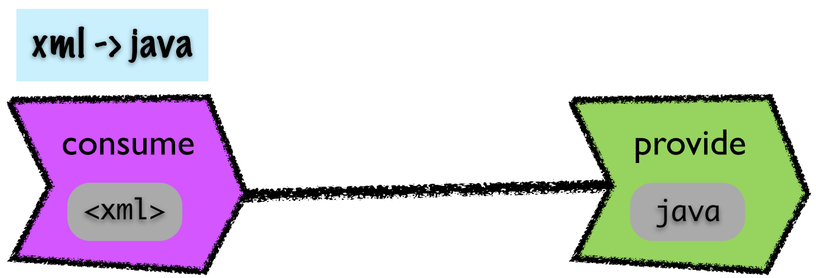
- In the provider?

NO!



- In the consumer?

NO!

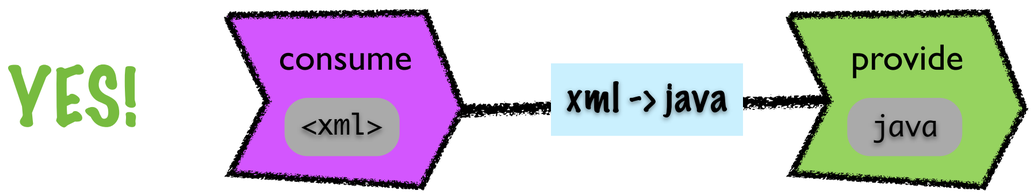


- Add a routing service?

NO!

Transformers

- Transformation is wired into SwitchYard core
  - Types declared via service contract
  - Transformer resolved dynamically at runtime
- Declarative, not procedural



- Java, JAXB, XSLT, JSON, and Smooks

Java Transformer

- Config generated from annotation

```
@Transformer(from = "{urn:switchyard-example: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”)));
}
```

XSLT Transformer

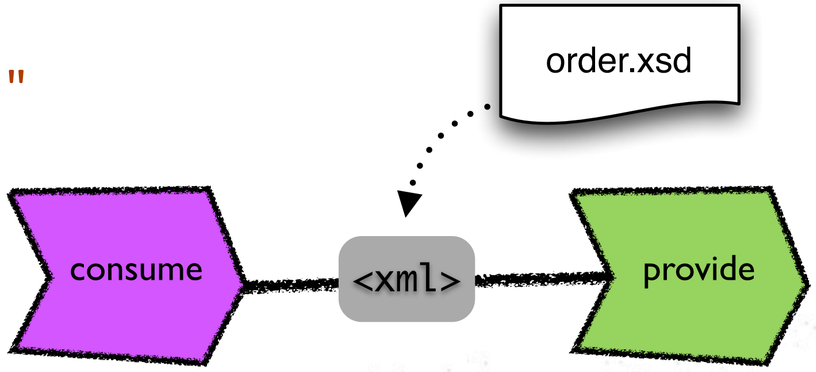
- Declared in application descriptor

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

Validators

- Declarative validation
- Supports XML Schema and Java validation
- Executes pre and post transformation

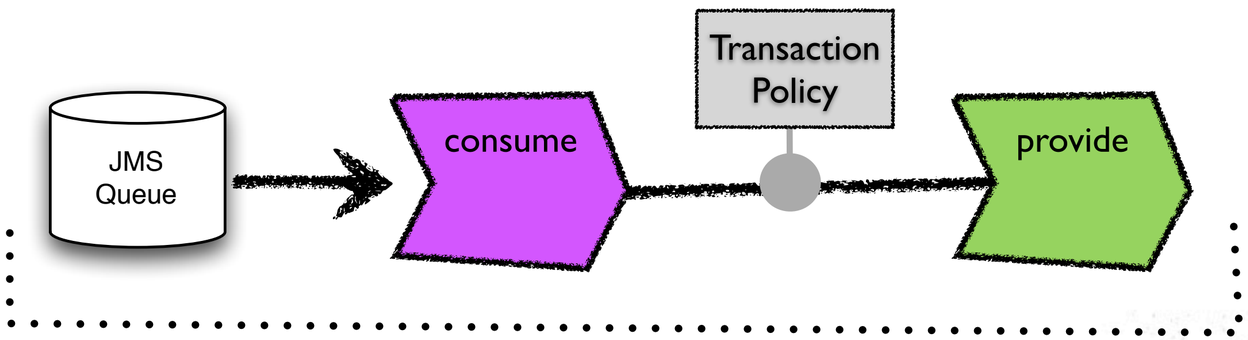
```
<validate.xml
     schemaType="XMLSCHEMA"
     name="{urn:example:purchasing}order"
     schemaFile="xsd/order.xsd"/>
```



Policy

- Declarative policy
  - Requirements attached to service definition
  - Runtime injects policy enforcement point

```
<service name="OrderService requires="propagatesTransaction">
  ...
</service>
```



desired transaction scope

Testing Services

Testing

- Big Bang testing of SOA applications must stop!
- Develop and test your project iteratively
  - Service, transformation, binding, etc.
- SwitchYardRunner
  - Bootstraps runtime, components, and application
- MixIns
  - Enriches test case via composition vs. extension
  - CDI, HTTP, Smooks, BPM, HornetQ
- Arquillian

Service Test

```java
@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

```
@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());
    }
}
```

Injects a reference to a service operation

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());
    }
}
```

Injects a reference to a service operation

Sends and receives messages over the bus

Transformation Test

Injected MixIn class provides helper methods

```
@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");
    }
}
```

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"

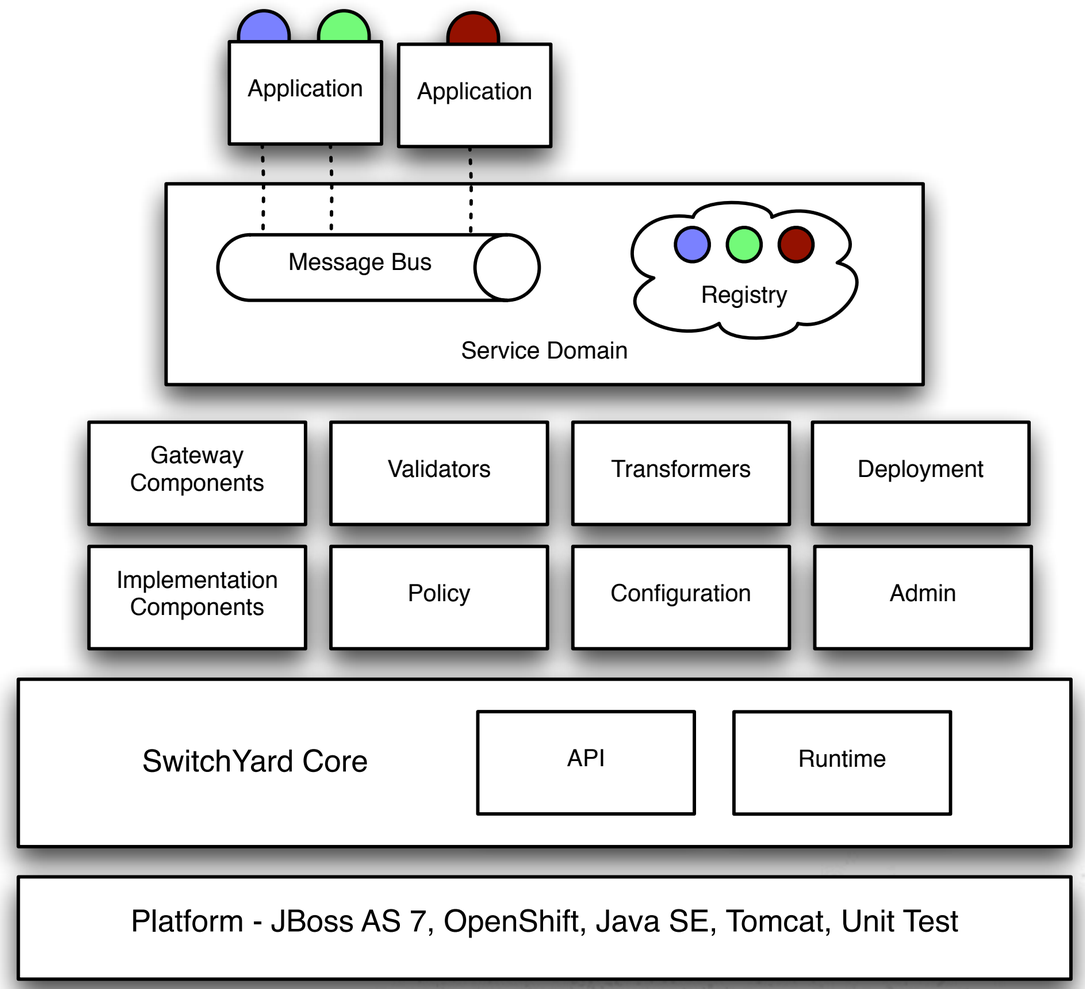
Runtime

Runtime Options

- Supported Containers
  - JBoss AS 7 (7.1.0.CR1b)
  - OpenShift (7.0.2.Final)
  - JBoss AS 6 (Phased out from 0.3)
  - Java SE - unit test or standalone
  - Tomcat
  - OSGi, and others coming soon ...
- Application Deployment Options
  - JAR, WAR, EAR

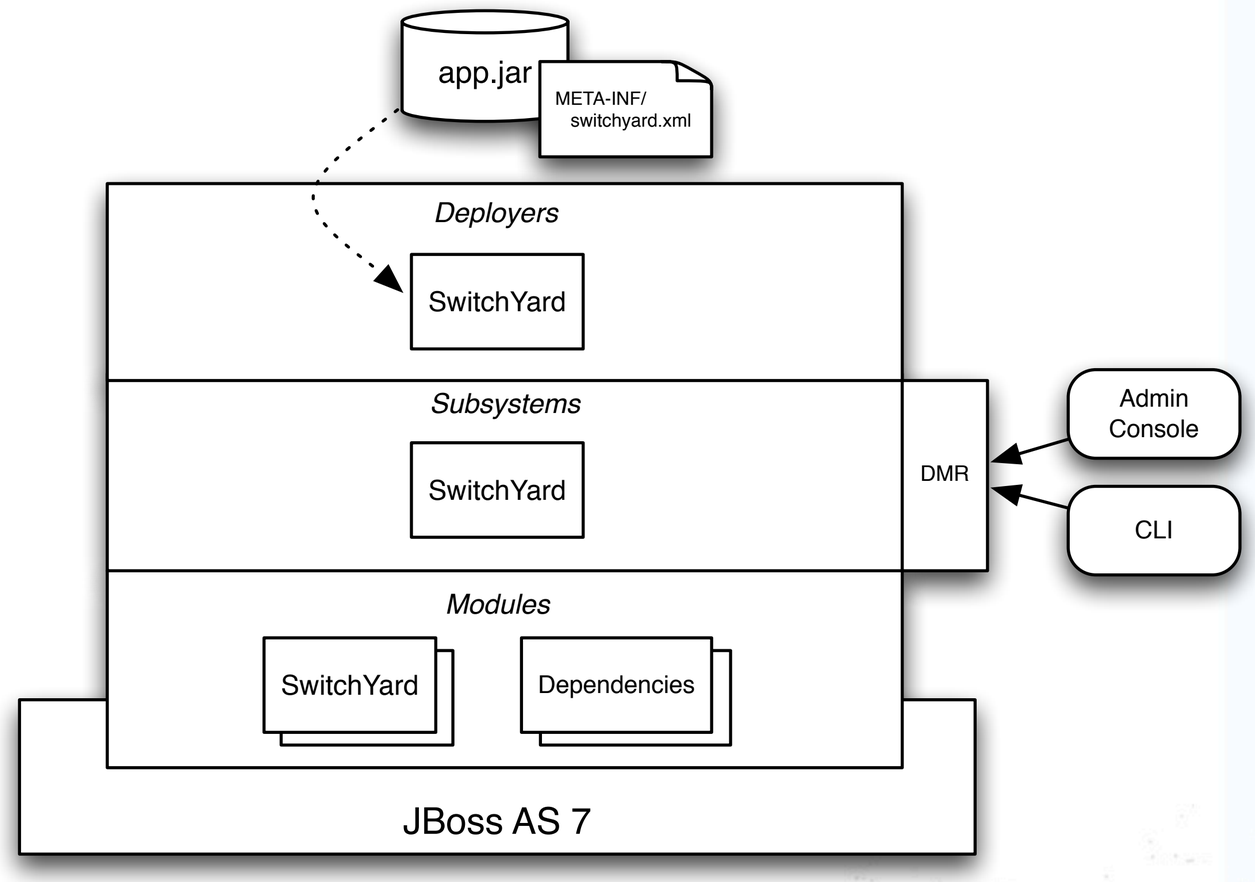
Runtime Architecture

- Small, extensible core (150kb)
- Features are modules
- Service Domain
  - Service endpoint registry
  - Message Bus
  - Application services
- Multiple service domains on the roadmap
  - Isolate application services
  - Share policy, configuration, etc.



AS7 Runtime Architecture

- JBoss AS 7
  - Blazingly fast (< 3s startup)
  - Lightweight
  - Modular core
- SwitchYard on AS7
  - Module Service Container
    - Core - api, config, deploy, runtime, transform, validate
    - Components - bean, bpel, camel, rules, soap, hornetq, bpm
    - Dependencies - Camel, Smooks, jBPM, Drools, XStream



AS7 Configuration

```
<server xmlns="urn:jboss:domain:1.1">
...
   <extensions>
       ...
       <extension module="org.switchyard"/>
   </extensions>
...
   <subsystem xmlns="urn:jboss:domain:switchyard:1.0">
       <modules>
           <module identifier="org.switchyard.component.bean" implClass="org.switchyard.component.bean.deploy.BeanComponent"/>
           <module identifier="org.switchyard.component.soap" implClass="org.switchyard.component.soap.deploy.SOAPComponent">
               <properties>
                   <socketAddr>:18001</socketAddr>
               </properties>
           </module>
           <module identifier="org.switchyard.component.camel" implClass="org.switchyard.component.camel.deploy.CamelComponent"/>
           <module identifier="org.switchyard.component.rules" implClass="org.switchyard.component.rules.deploy.RulesComponent"/>
           <module identifier="org.switchyard.component.bpm" implClass="org.switchyard.component.bpm.deploy.BPMComponent"/>
           <module identifier="org.switchyard.component.bpel" implClass="org.switchyard.component.bpel.deploy.BPELComponent"/>
           <module identifier="org.switchyard.component.hornetq" implClass="org.switchyard.component.hornetq.deploy.HornetQComponent"/>
       </modules>
   </subsystem>
...
</server>
```

Cloud

- OpenShift (Go beyond the Clouds)
  - Free PaaS from Red Hat
  - Java, Perl, PHP, Python, Ruby and ...
  - SwitchYard!
- SwitchYard on OpenShift
  - Runs on JBoss AS 7
  - Simple bootstrap using our template application

    http://github.com/jboss-switchyard/switchyard-openshift

  - Sample application included in the template project

Tooling

Tooling

- Seam Forge
  - Rapid application development tool
  - Ease of a wizard, power of a shell
- IDE Support
  - Maven support provides baseline functionality across IDEs
  - Specific tooling features for Eclipse

Forge

```
ExampleService$ forge
     _____
    |  ___|__  _ __ __ _  ___
    | |_ / _ \| `__/ _` |/ _ \  \\
    |  _| (_) | | | (_| |  __/  //
    |_|  \___/|_|  \__, |\___|
                      |___/

[ExampleService] ExampleService $
```

Add SwitchYard to your application

```
[ExampleService] ExampleService $ project install-facet switchyard.bpm
[ExampleService] ExampleService $ project install-facet switchyard.soap
```

Create a BPMN 2 workflow service

```
[ExampleService] ExampleService $ bpm-service --serviceName ExampleService
```

Bind the service to SOAP / HTTP

```
[ExampleService] ExampleService $ switchyard promote-service --serviceName ExampleService
[ExampleService] ExampleService $ soap-binding bind-service
          --serviceName ExampleService --wsdl wsdl/Example.wsdl
```

Join Us!

- Learn
  - http://www.jboss.org/switchyard
- Play
  - http://github.com/jboss-switchyard/quickstarts
- Chat
  - chat.freenode.net #switchyard
- Fork
  - http://github.com/jboss-switchyard

Questions?

JUDCon

JBoss Users & Developers Conference

2012:India