SWITCHYARD

SYSTEM INTEGRATION WITH JBOSS

Tomáš Rohovský = trohovsk@redhat.com

Tomáš Turek - tturek@redhat.com

AGENDA

- What is SwitchYard?
- Service Component Architecture
- Cross cutting concerns
- Component implementations
- Clustering
- Testing

SWITCHYARD

"A lightweight service delivery framework providing full lifecycle support for developing, deploying, and managing service-oriented applications."

JBoss Community

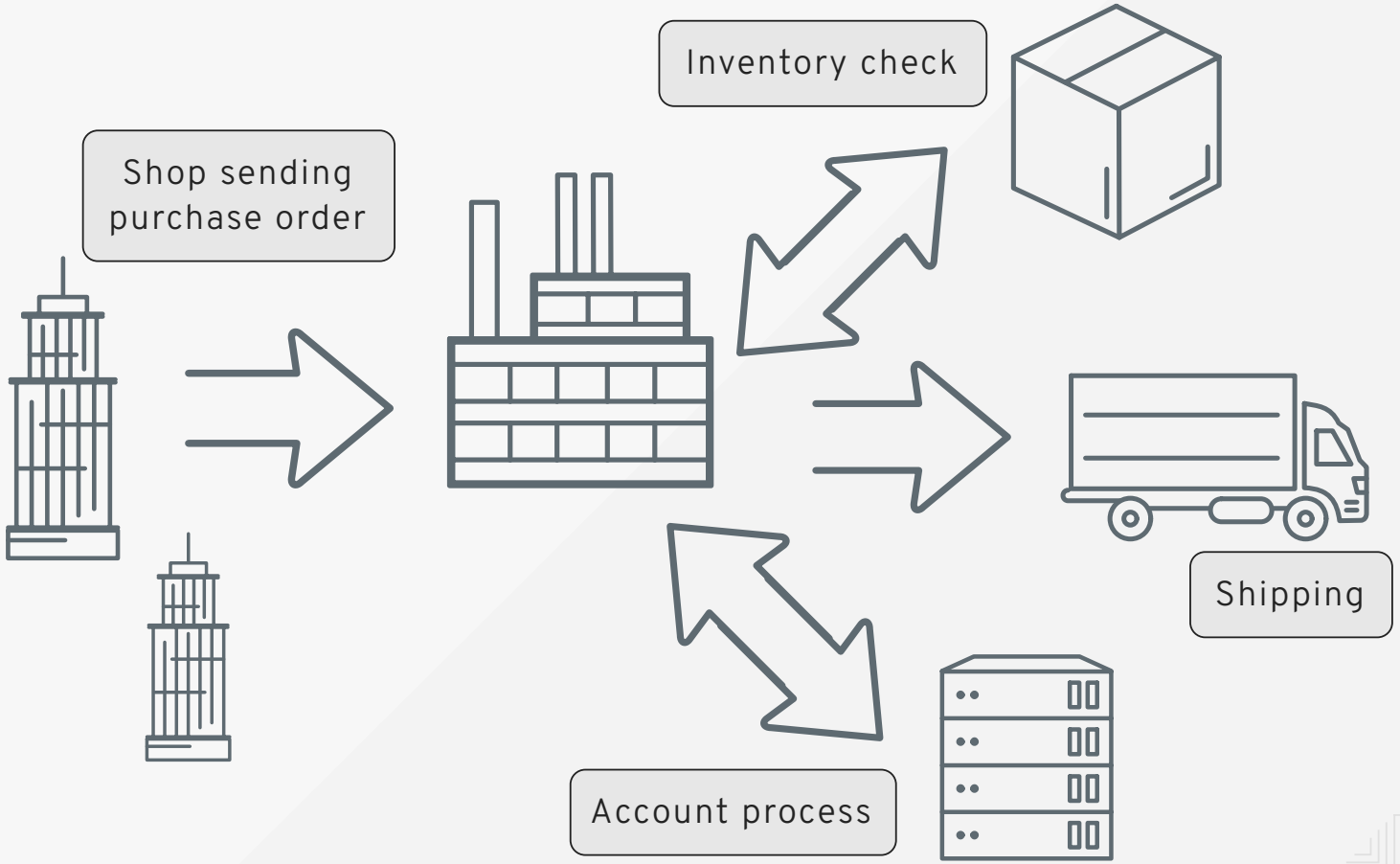
Integration framework based on Camel

Supports various protocols and implementations

Implements Service Component Architecture specification

Can run in WildFly/EAP (J2EE) or Karaf/Fuse (OSGi)

SYSTEM INTEGRATION



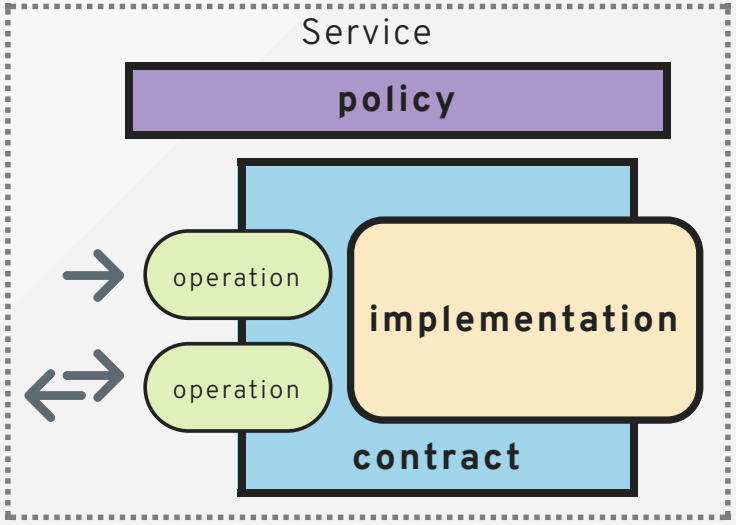
SERVICE

Is a logical representation of a repeatable business activity

Is self-contained

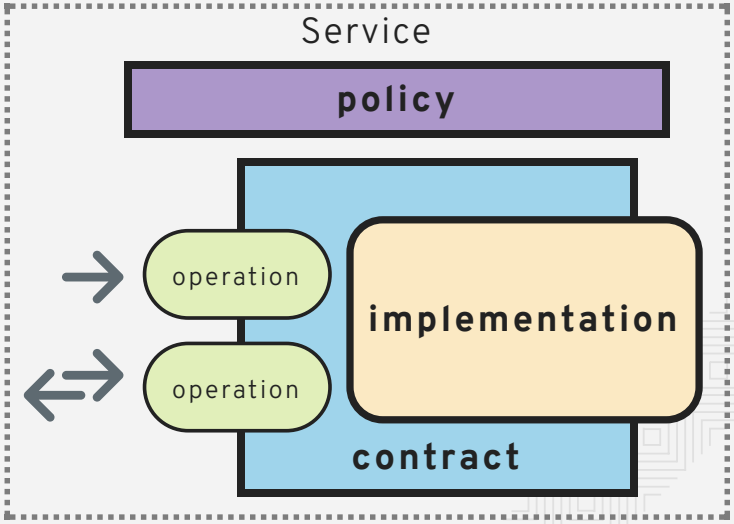
May be composed of other services

Is a "black box" to consumers of the service



SERVICE CONTRACTS

- Agreement between consumer and provider
- Contract-based development
  - Service Interface
  - Implementation Hiding



SCA

SERVICE COMPONENT ARCHITECTURE

A Set of specifications which describe a model for building applications and systems using a Service-Oriented Architecture (SOA).

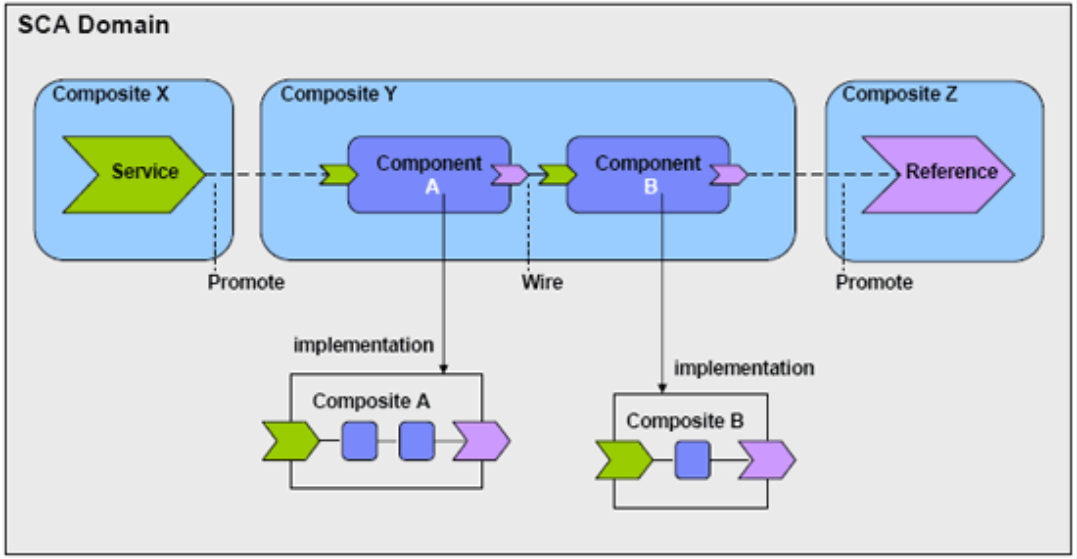
OASIS standard for creating service-oriented applications

- vendor neutral
- language neutral
- technology neutral

SCA

BASE ARTEFACTS

- Composite
- Components
- Services
- References
- Wires
- Properties



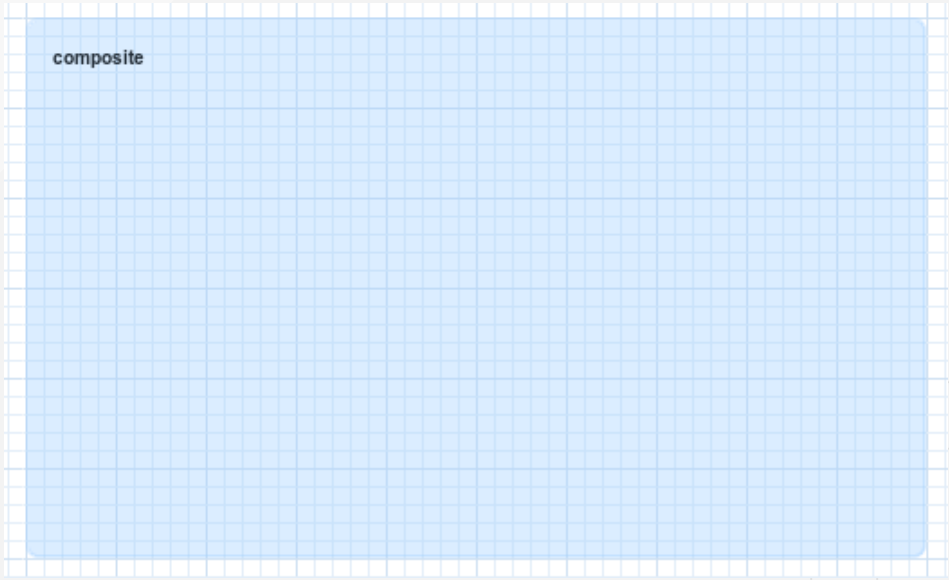
COMPOSITE

Define the application boundary

Application name (documentation)

Application namespace (important)

One per application



COMPONENT

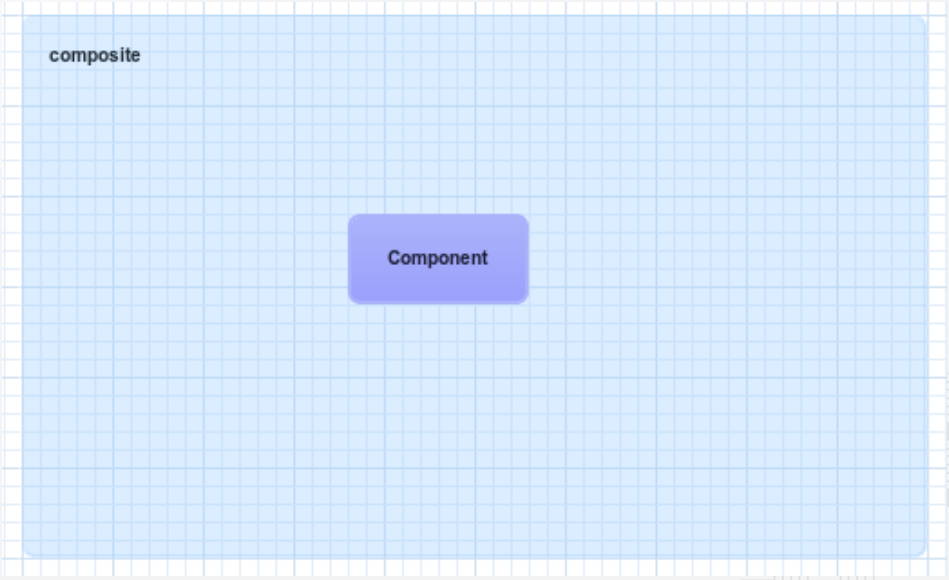
Container for application logic

Worthless without an implementation

0 ... 1 services

0 ... * references

1 implementation



IMPLEMENTATION

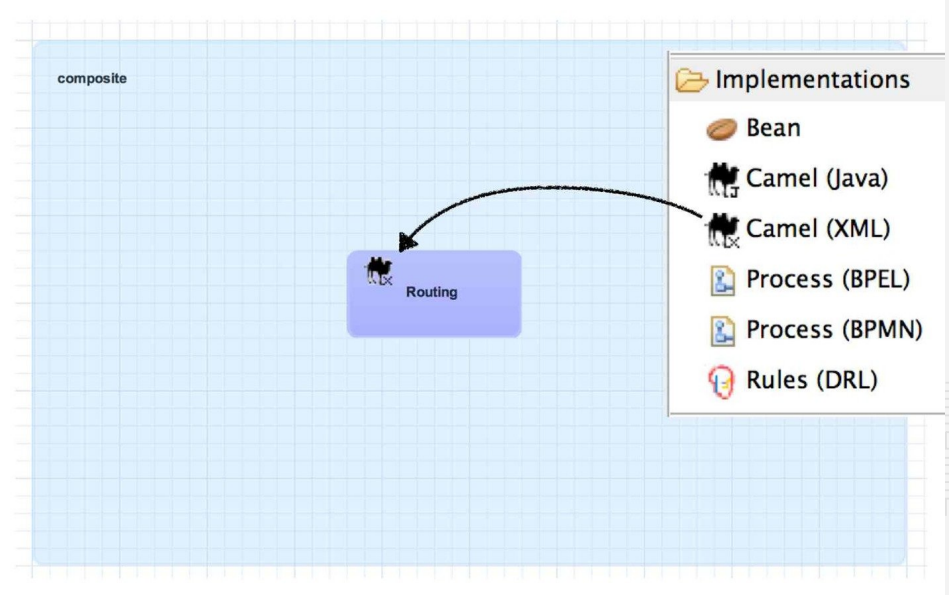
Adds intelligence/action to a component

Private to an application

First step in bottom-up approach

Implementations

- Bean
- Camel
- BPMN
- BPEL
- Rules



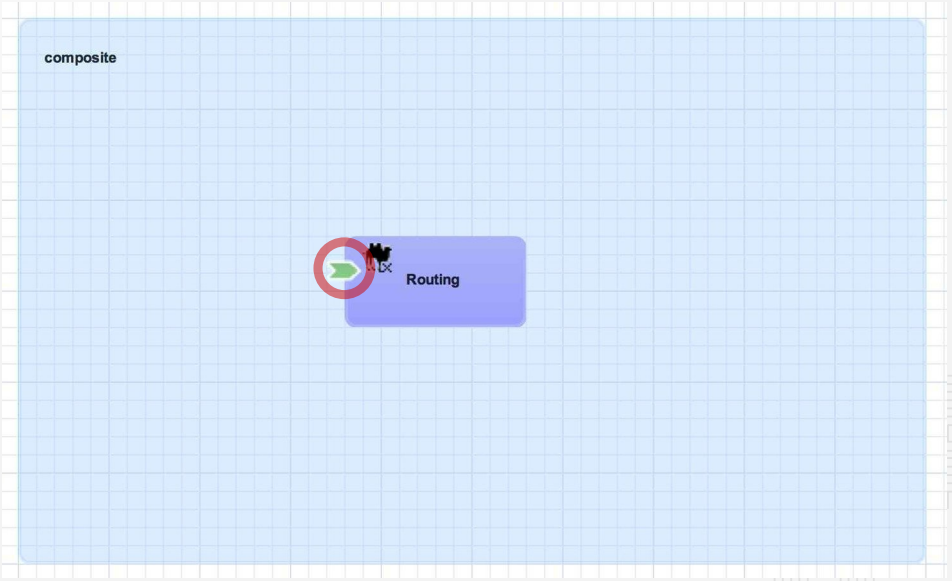
COMPONENT SERVICE

Exposes functionality of a component as a service

Application private

Name

Contract



SERVICE CONTRACT

Agreement between consumer and provider

Collection of operations

Each operation defines exchange pattern and message types

Java, WSDL, ESB

EXAMPLE CONTRACTS

JAVA

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

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

WSDL

ESB

```
<interface.esb
   inputType="json:orders:Order"
   outputType="java:org.example.OrderAck"
   faultType="{urn:example}error"/>
```

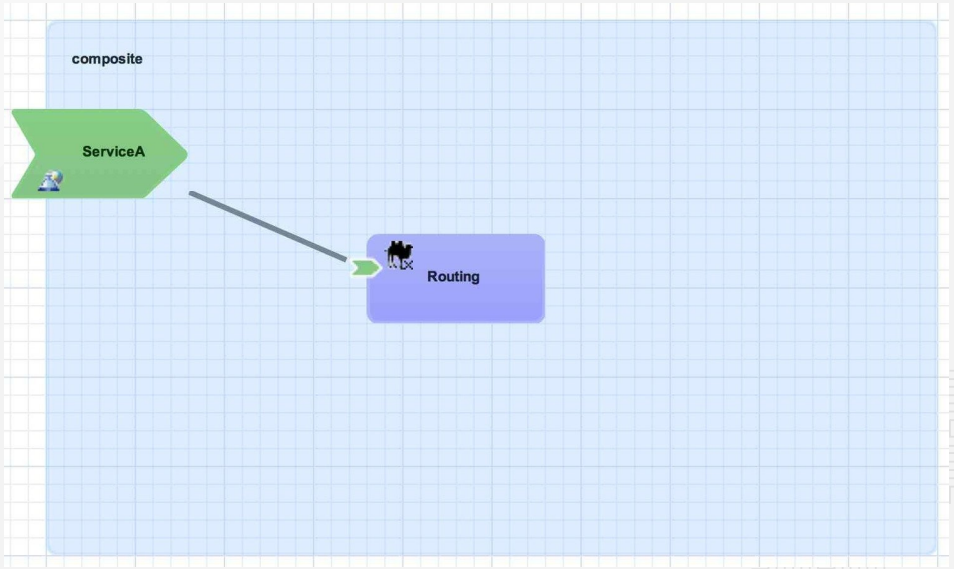
COMPOSITE SERVICE

Service available outside an application

“Promotes” a component service

Name

Contract



SERVICE BINDINGS

How external consumers communicate with a service

A composite service can have multiple bindings

A component service never has bindings

Conforms to service contract

- Camel, Ftp, File, Http, JCA, JMS, JPA, Mail, TCP, UDP, REST, SOAP, SQL, ...

COMPONENT REFERENCE

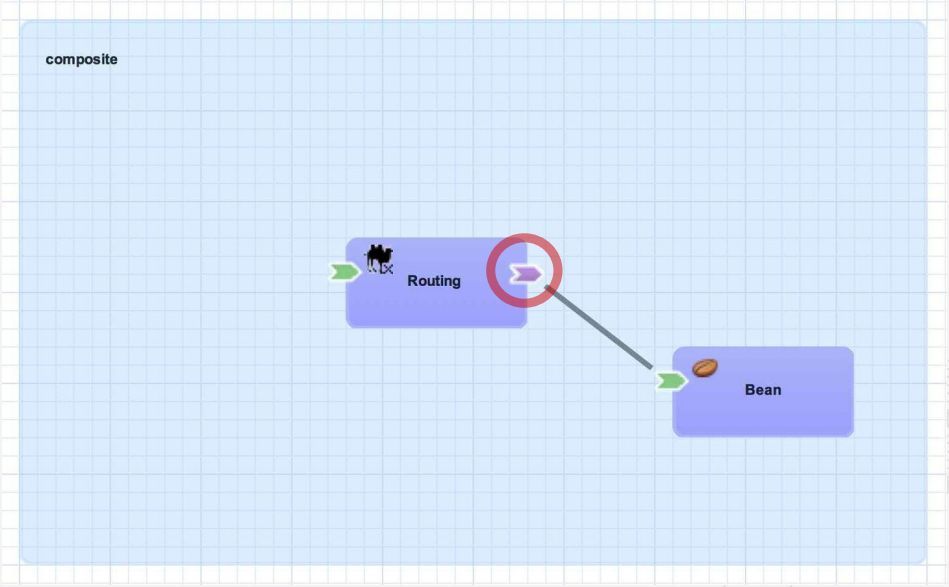
Allow a component to consume other services

Can be wired to services inside and outside an application

Implementation dependency

Name

Contract



COMPOSITE REFERENCE

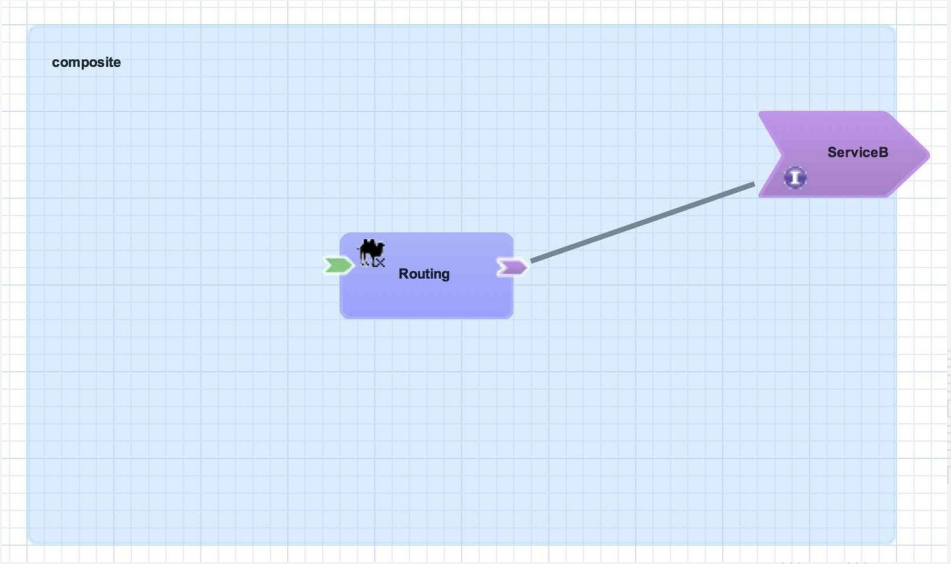
Wires to services outside an application

“Promotes” a component reference

Application dependency

Name

Contract



REFERENCE BINDING

Defines how external services are accessed

Composite reference can have only one binding

Component reference never has bindings

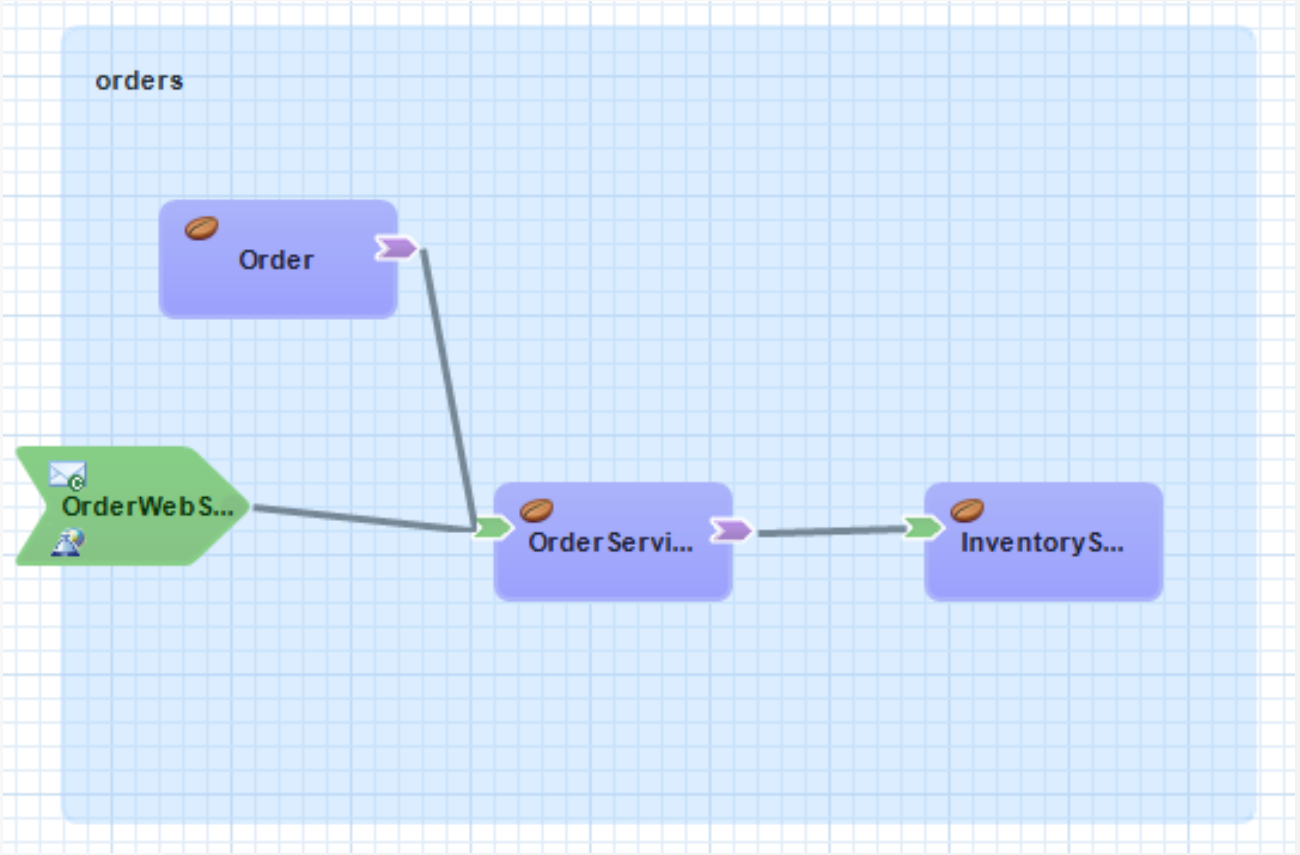
Conforms to reference contract

- Camel, Ftp, File, Http, JCA, JMS, JPA, Mail, TCP, UDP, REST, SOAP, SQL, ...

APPLICATION DESCRIPTOR

```
<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" name="orders"
     targetNamespace="urn:switchyard-quickstart-demo:orders:0.1.0">
   <service name="OrderService" promote="OrderService">
      <interface.wsdl interface="wsdl/OrderService.wsdl#wsdl.porttype(OrderService)"/>
      <binding.soap xmlns="urn:switchyard-component-soap:config:1.0">
         <wsdl>wsdl/OrderService.wsdl</wsdl>
         <socketAddr>:18001</socketAddr>
         <contextPath>demo-orders</contextPath>
      </binding.soap>
   </service>
   <component name="InventoryService">
      <implementation.bean class="org.example.InventoryServiceBean"/>
      <service name="InventoryService">
         <interface.java interface="org.example.InventoryService"/>
      </service>
   </component>
   <component name="Order">
      <implementation.bean class="org.example.Order"/>
      <reference name="OrderService">
         <interface.java interface="org.example.OrderService"/>
      </reference>
   </component>
   <component name="OrderService">
      <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>
</composite>
```

APPLICATION MODEL



KEY BENEFITS OF SCA

Loose Coupling: components integrate without need to know how othe rs are implemented

Flexibility: components can easily be replaced by other components

Composition of solutions: clearly described

Productivity: easier to integrate components to form composite applic ation

Heterogeneity: multiple implementation languages, communication m echanisms

Declarative application of infrastructure services

Simplification for all developers, integrators and application deployers

COMPONENT IMPLEMENTATIONS

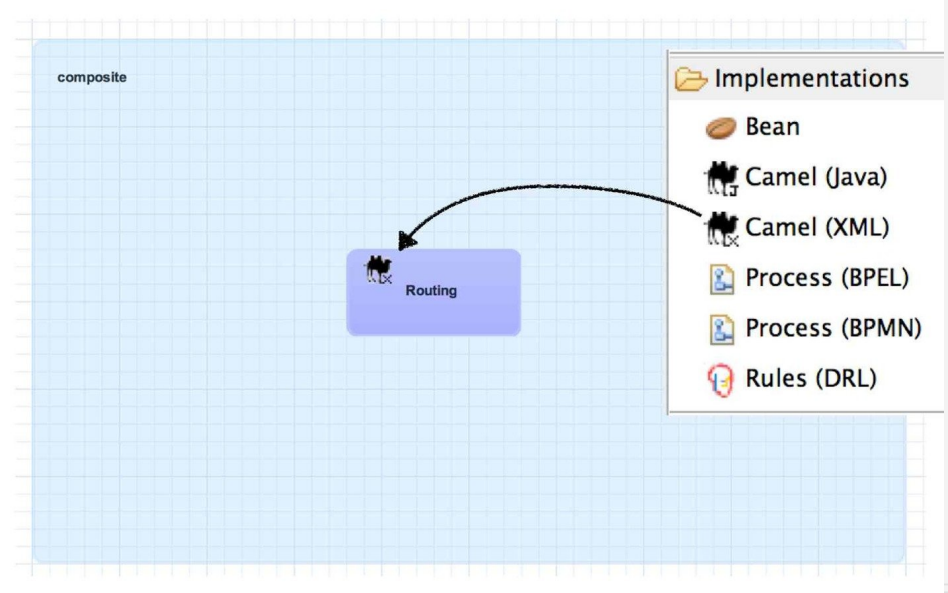
BEAN

CAMEL

BPEL

BPM

RULES



BEAN IMPLEMENTATION

Standard CDI beans with a few extra annotations

```
@Service(GreetingService.class)
public class GreetingServiceBean implements GreetingService {

  @Inject
  @Reference
  private LanguageService languageService;
  // private ReferenceInvoker languageService;

  public void greet(String name) {
    String greeting = languageService.getGreeting("EN");
    // languageService.newInvocation("getGreeting").setProperty("type", "informal").invoke("EN");
    return greeting + " " + name;
  }
}
```

BEAN IMPLEMENTATION

BASIC

```
@WebService
public class OrderServiceBean {

   @Resource(mappedName = "jms/ConnectionFactory")
   private static ConnectionFactory cf;

   @WebMethod
   public OrderAck submitOrder(Order order) {
   …
   }
}
```

SWITCHYARD

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

   @Inject @Reference
   private InventoryService _inventory;

   @Override
   public OrderAck submitOrder(Order order) {
      …
   }
}
```

CAMEL IMPLEMETATION

A route can be created in JAVA or XML DSL

Camel provides

- Routing engine
- Languages (Simple, XPath, scripting languages)
- Loads of EIP

EXAMPLE CAMEL ROUTE

CAMEL ROUTE

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

CAMEL IN SWITCHYARD

```
<route>
   <from uri="switchyard//OrderService"/>
   <log message="Order Received : ${body}"/>
   <to uri="OrderValidator”/>
   <filter>
      <xpath>/order[@priority='high']</xpath>
      <to uri="switchyard://ShippingService?operation
   </filter>
</route>
```

WORKFLOW SERVICES

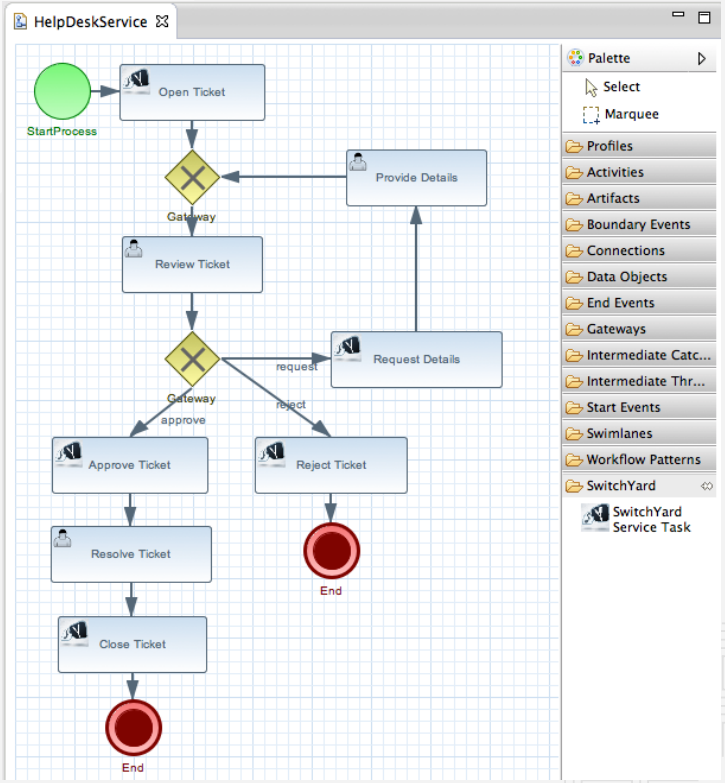
BPMN 2

Integrated with BPMN 2 modeler

Expose workflow as a service

Invoke services as part of a workflow

Flexible mapping between process and message



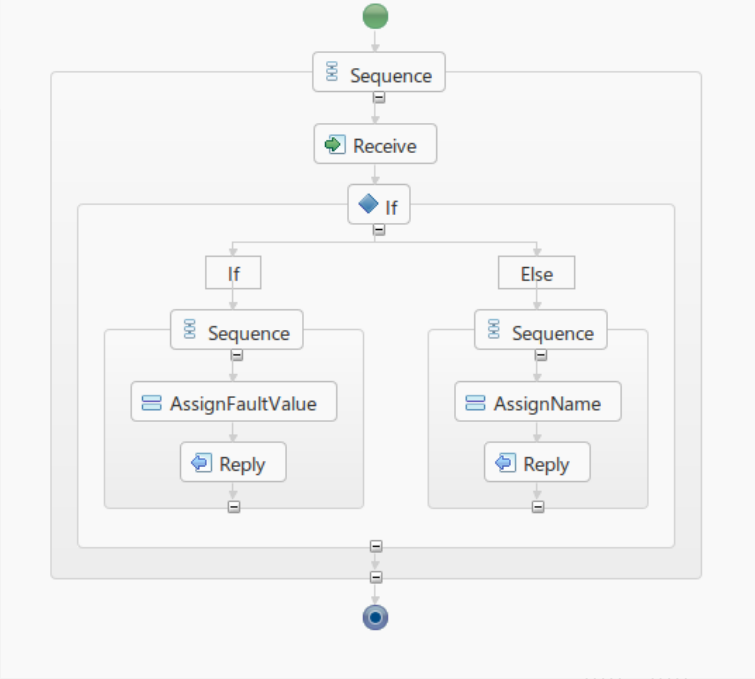
SERVICE ORCHESTRATION

BPEL

Orchestrate web services

WSDL contract

Multiple bindings



DEMO

CROSS CUTTING CONCERNS

- Transformation
- Validation
- Policy

Specified declaratively

- Isolation from application logic
- Reuse
- Clear view

TRANSFORMATION

Ubiquitous challenge in application integration and SOA

Three flavors of transformation

- Change in data representation
  - Conversion
- Change in data format
  - Translation
- Change in data itself
  - Enrichment

TRANSFORMATION OF REPRESENTATION

Change in representation

Representation = Java type

Transformation is simply a type conversion

No semantic knowledge required

TRANSFORMATION OF DATA FORMAT

Requires semantic understanding of data types

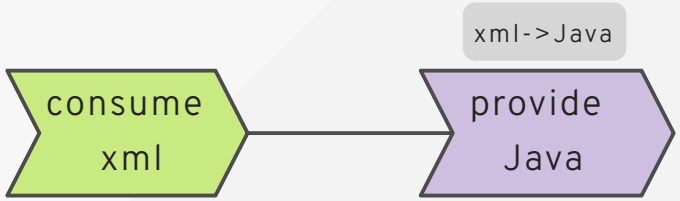
Machines cannot do this on their own

```
HR;1;ISSUED;
CUS;user1;Herry;
ORD;1;Pulp Fiction;
ORD;5;Pencil;
```

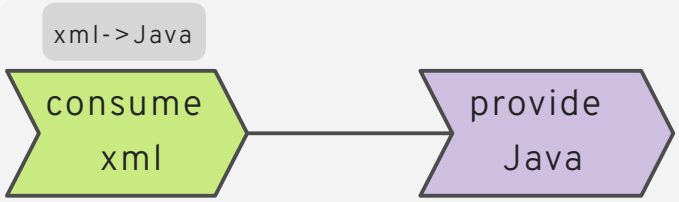
```
public class Order {
   private Header header;
   private List<OrderItem> orderItem;
}

public class Header {
   private String orderId;
   private String status;
}

public class OrderItem {
   private in quantity;
   private String productId;
}
```

WHERE TO TRANSFORM

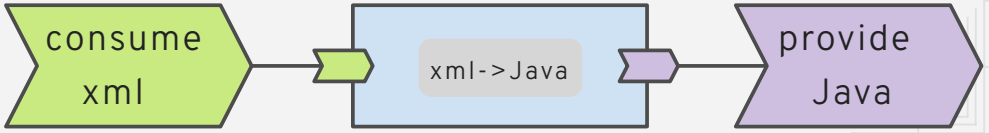
In the provider?



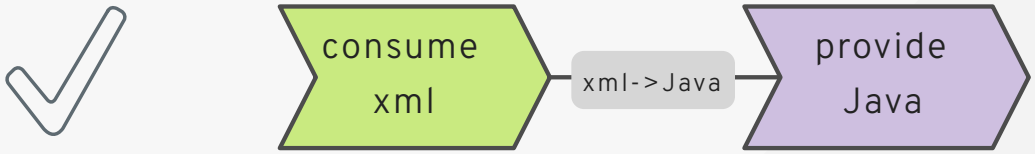
In the consumer?



Add a routing service?



DECLARATIVE TRANSFORMATION



Transformation is wired into SwitchYard core

- Types declared via service contract
- Transformer resolved dynamically at runtime

Declarative, not procedural

Dozer, Java, JAXB, JSON, XSLT, Smooks

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

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

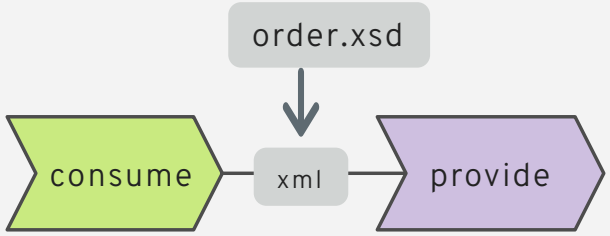
VALIDATION

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

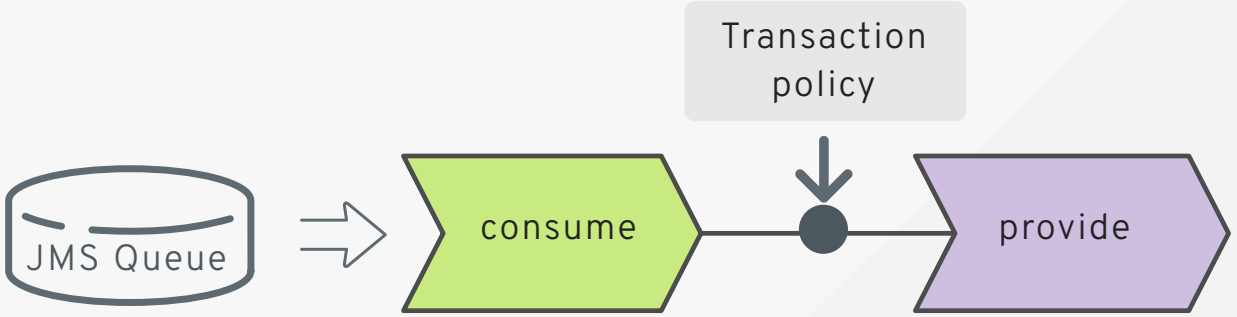
Declare the ‘what’, defer the ‘how’

Common service policies

- Transactional behavior
- Security

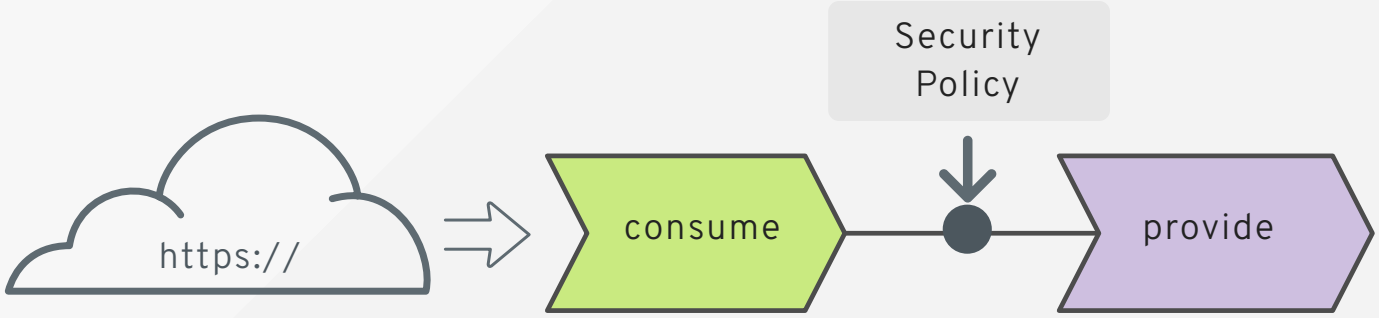
TRANSACTION POLICY

```
<service name="WorkService" promote="WorkService">
  <camel:binding.camel configURI="jms://policyQSTransacted?connectionFactory
</service>
<component name="WorkService">
  <implementation.bean class="org.example.WorkServiceBean"
   requires="managedTransaction.Global"/>
  <service name="WorkService" requires="propagatesTransaction">
    ...
  </service>
</component>
```



SECURITY POLICY

```
<sca:component name="OrderService">
   <sy:implementation.bean class="org.example.OrderServiceBean"
      requires="authorization"/>
   <sca:service name="OrderService"
      requires="confidentiality clientAuthentication" sy:security="default">
      <sca:interface.java interface="org.example.OrderService"/>
   </sca:service>
</sca:component>
```



TESTING

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, Transaction, JCA

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());
   }
}
```

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");
   }
}
```

BINDING TEST

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

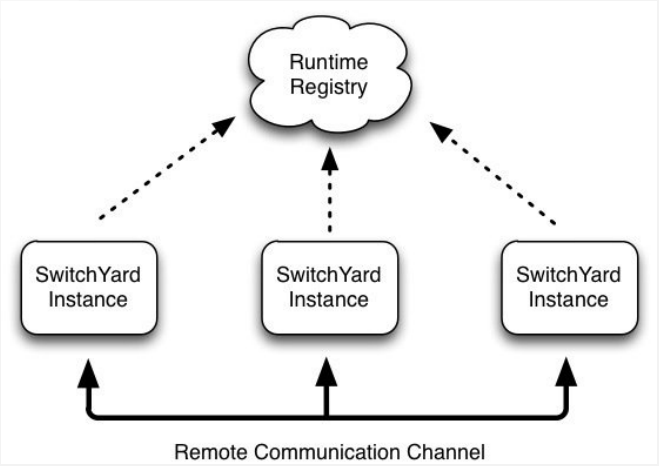
CLUSTERING

SHARED RUNTIME REGISTRY

- distributed Infinispan cache
- published remote service endpoints

REMOTE COMMUNICATION CHANNELS

- internal communication protocol (HTTP)
- SCA binding



CLUSTERING

EXAMPLE

```
<sca:service name="Goodbye" promote="GoodbyeBean/Goodbye">
  <sca:interface.java interface="com.example.Goodbye"/>
  <sca:binding.sca sy:clustered="true"/>
</sca:service>


<sca:reference name="Goodbye" multiplicity="0..1"
   promote="GreetingBean/Goodbye">
  <sca:interface.java interface="com.example.Goodbye"/>
  <sca:binding.sca sy:clustered="true"
</sca:reference>
```

MORE INFO

SWITCHYARD

http://switchyard.jboss.org/

GITHUB

https://github.com/jboss-switchyard

REDHAT

https://access.redhat.com/products/red-hat-jboss-fuse-service-works/

SCA

http://osasis-open.org/committees/sca-assembly

redhat.

THANK YOU!