# Pragmatic Service-orientated Integration – Camel just got CXFy

Adrian Trenaman,
CamelOne, Washington DC,
May 23rd, 2011

twitter : adrian_trenaman | LinkedIn: adrian.trenaman
http://trenaman.blogspot.com

FuseSource
A Progress Software Company

# This presentation, in a nutshell.

- CXF was, is, and continues to be a great framework for REST and SOAP web services.
  - Make the *right* implementation choice for your services
    - Straight up business logic / DB access / API : ‘Code it up in Java / Scala’!
    - Go with the flow: EIP-based implementation and orchestration with Camel!
- CXF’s integration with Camel makes EIP-based implementations *easy.*
  - And, it’s very, very popular.

# Meet … me.

- 15 years IT consulting experience
  - IONA Technologies, Progress Software Corp, FuseSource
  - Committer, Apache Karaf
  - SOA, ESB, Open Source, BPM, Web Services, CORBA, …
- Solution focused: architecting, mentoring, speaking, engineering, doing…
- PhD Artificial Intelligence
  - Dip. Business Development
  - BA Mod Computer Science


http://trenaman.blogspot.com
http://slideshare.net/trenaman
twitter: adrian_trenaman
LinkedIn: adrian.trenaman



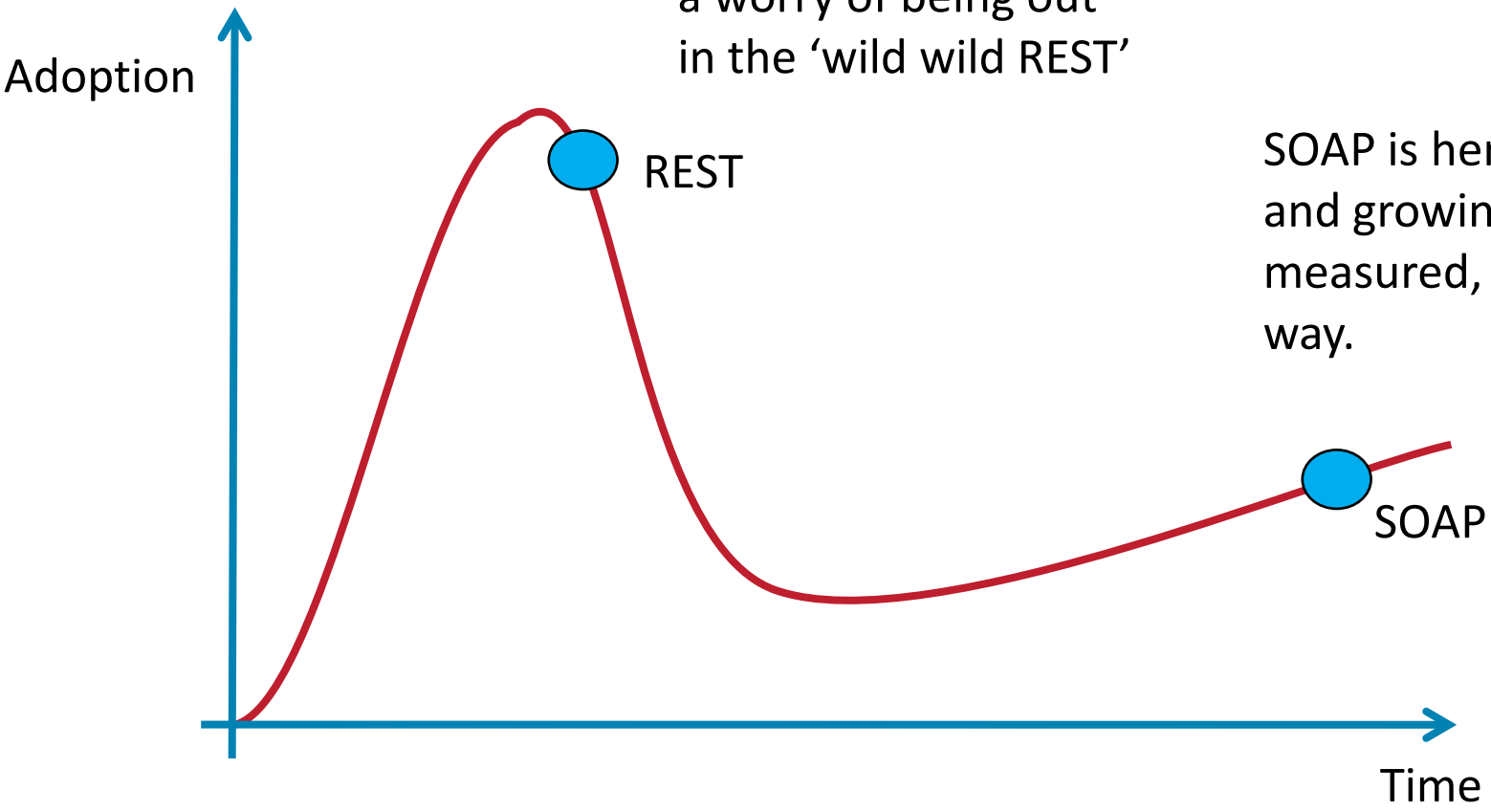

FuseSource
A Progress Software Company

# My love affair with CXF – Dude, get a room!

- Love at first sight…
  - Celtx (2004), CXF (2005 – wrote first CXF training course, consulted ever since)
- What's not to love?
  - Standards-based, extensible, open-source, Apache-licensed, fast, light, modular, supported.
- We've grown closer over time.
  - Tight integration with ServiceMix and Camel.
  - REST, WS-*, JMS, …
  - CXF Webinar Series at FuseSource .com

# REST & SOAP on the enterprise adoption hype curve

REST is everyone's darling; still, there is a worry of being out in the 'wild wild REST'




SOAP is here to stay, and growing in a measured, pragmatic way.

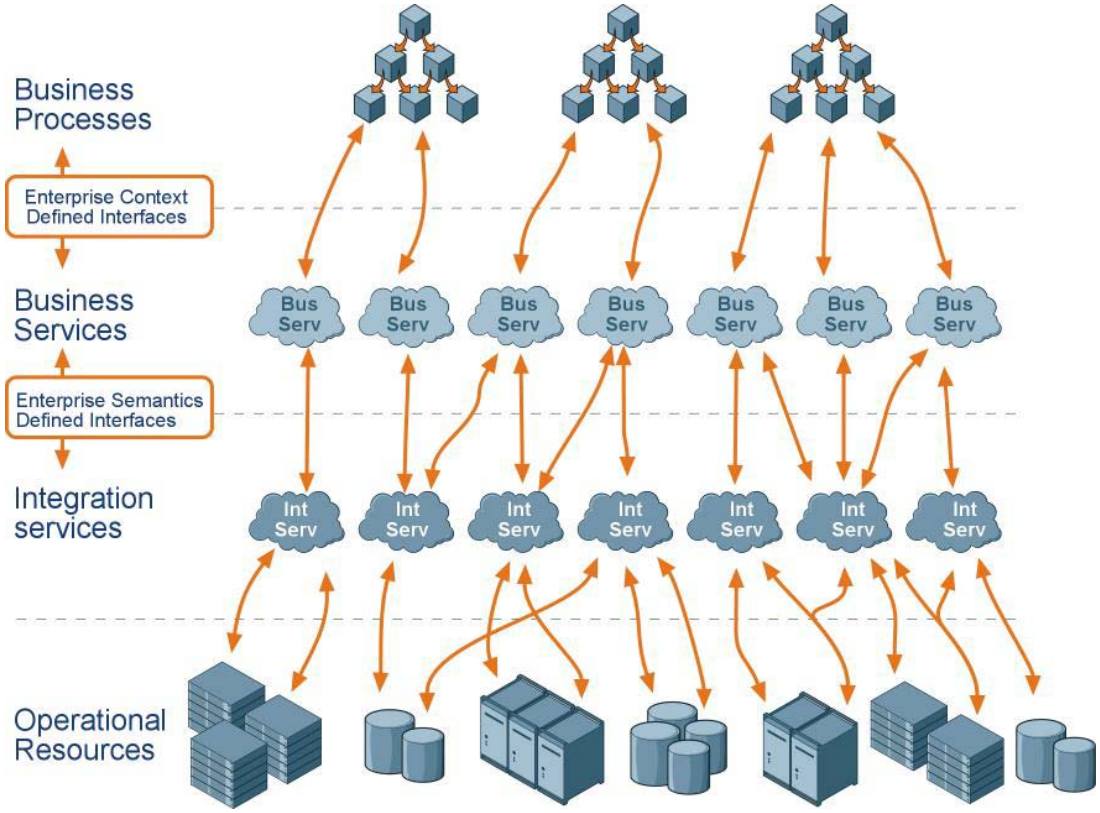
*Source: gut feeling on the road.*

# Observations on adoption of Service Oriented Architecture

- WSDL/SOAP has moved from hype to pragmatic, no-fuss adoption.
- WSDL was (and still is) way too hard.
  - There are very view people who can design a good contract.
  - But it's worth it - benefits of technology agnostic interface are tangible.
- Pragmatic adoption despite big SOA scare tactics.
  - "You need a $$$ registry / repository!"
- The world and its auntie loves REST
  - However, few are engaging in 'high REST' (HMATEOS)
  - Some are yearning for a return to contract-driven SOA
- Simplicity? +1. It's no longer acceptable to be complicated and misunderstood.
  - From "your software is so difficult to understand – you're awesome!"
  - To: "I couldn't be bothered trying to figure out your sucky stack."

# Pragmatic ‘Service Oriented Integration’

- ‘Everything is a web service’ philosophy is dangerous.
  - Time consuming. Slow. Unrealistic. Tightly coupled. Synchronous. Silly… yet promoted by the classic SOA architecture where “BPEL is the glue”







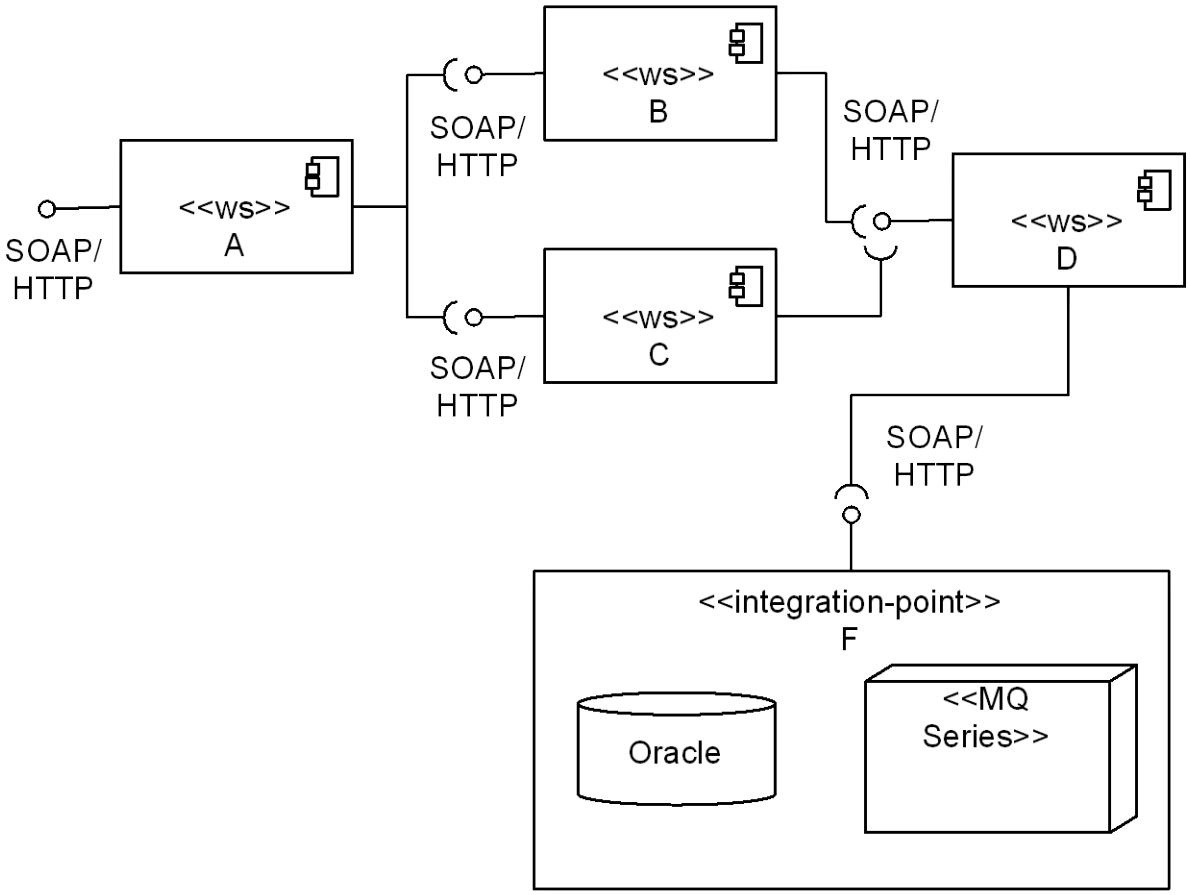
FuseSource
A Progress Software Company

# Pragmatic 'Service Oriented Integration'

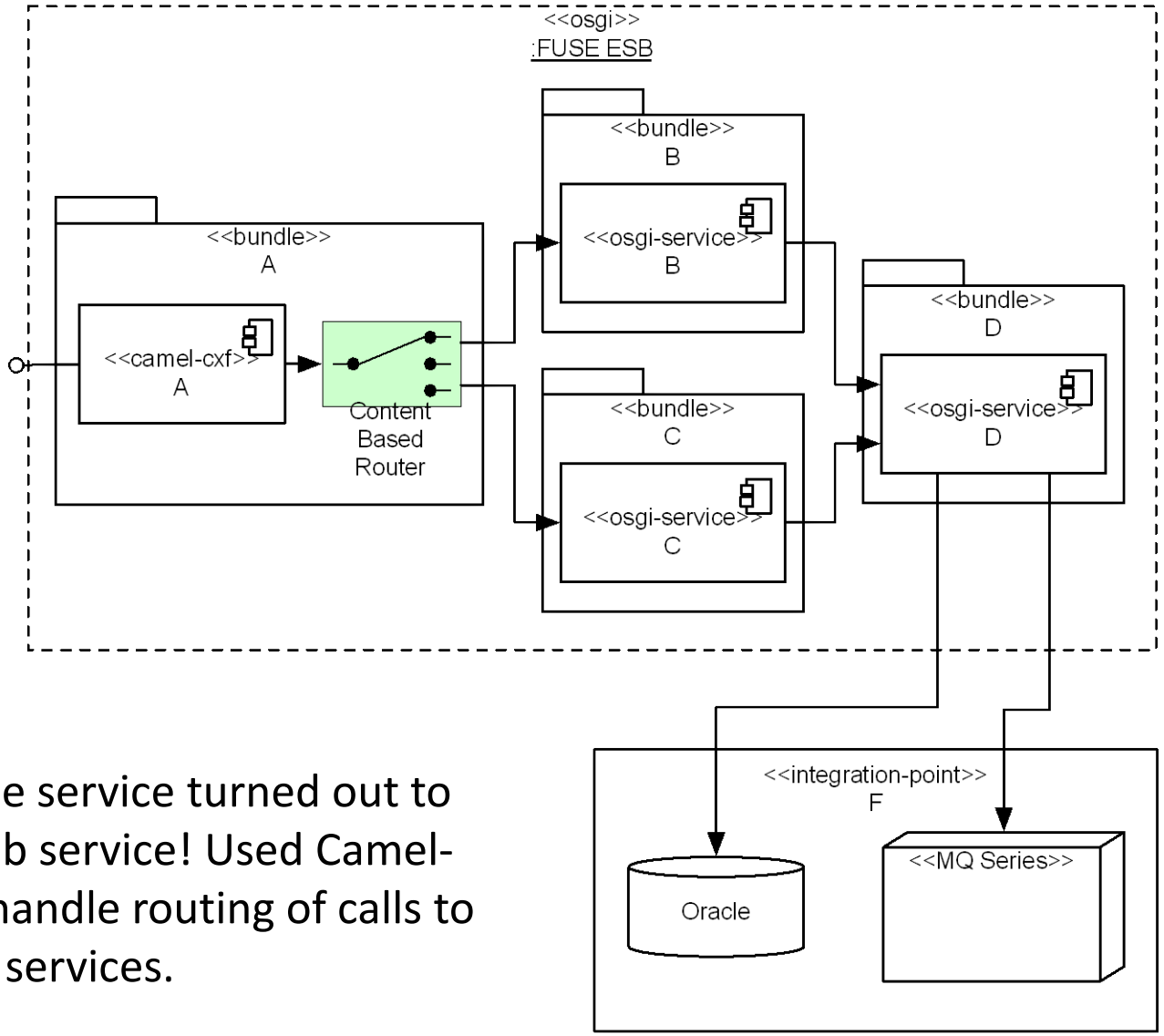
- Embrace heterogeneity in your middleware / integration stack.
  - Use web service as a *façade*
  - Behind the *façade*? *Do whatever you like. That's an implementation detail.*
- Let's take a real customer example.
  - A customer very new to SOA.
  - Initial design: 'module == web service' && 'orchestration = BPEL'.
    - Why? SWAG: Simple wild ass guess.
    - Innocence. No in-house experience with WSDL, XSD, or BPEL.
  - Final design:
    - Façade == web service
    - Module == JAVA service (OSGi)
    - Orchestration == Camel.

# Overuse of WS-* ☹

- Correct identification of modules; *incorrect* assignment of modularity to technology.



# Use-case 1: Orchestration via EIP




Only one service turned out to be a web service! Used Camel-CXF to handle routing of calls to distinct services.

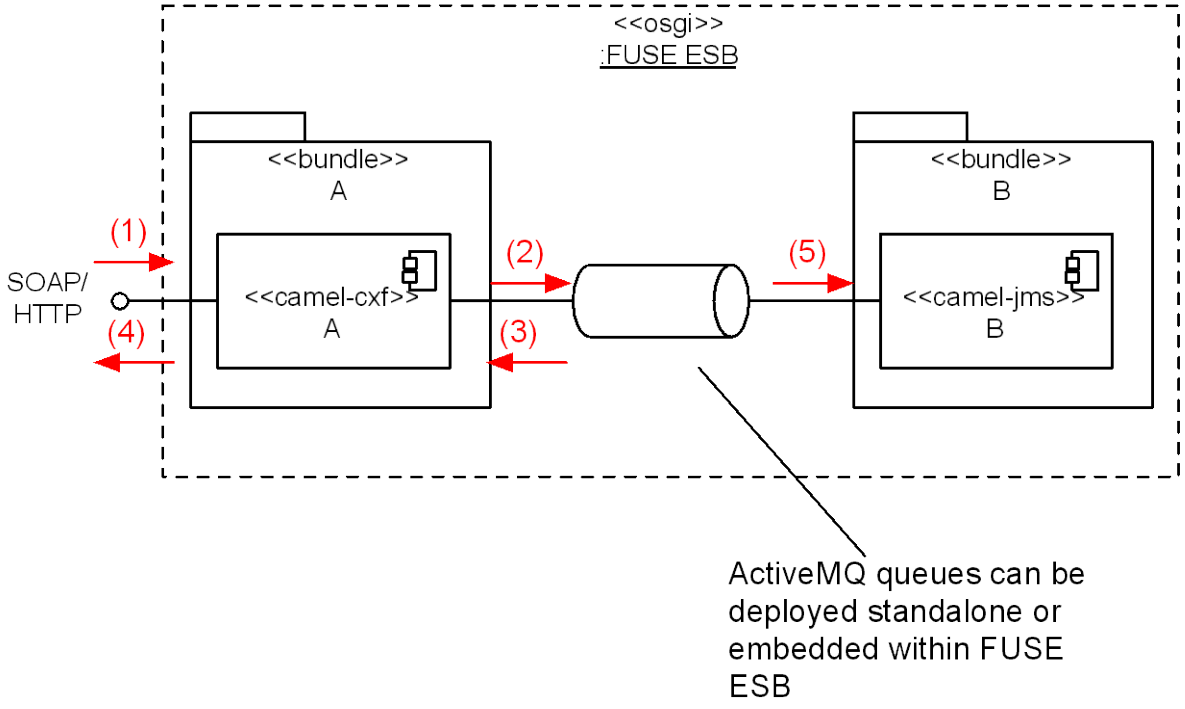
- Snippet of Camel for previous route shown below..

```
<camelContext xmlns="http://camel.apache.org/schema/spring">

    <route errorHandlerRef="noErrorHandler">
        <from uri="cxf:bean:MyWebService"/>
        <to uri="direct:productTypeRouter"/>
    </route>

    <route errorHandlerRef="noErrorHandler">
        <from uri="direct:productTypeRouter" />
            <choice>
                <when>
                    <methodCall bean="determineProductType" method="isProductTypeA" />
                    <to uri="productTypeA" />
                </when>
                <when>
                    <methodCall bean="determineProductType" method="isProductTypeB" />
                    <to uri="productTypeB" />
                </when>
                <otherwise>
                    <to uri="unknownProductTypeProcessor" />
                </otherwise>
            </choice>
    </route>

</camelContext>
```

Aside: these processors were deployed as OSGi services, & injected in via Spring DM configuration .

# Use-case 2: asynch processing from SOAP/HTTP to JMS

- A very common usage of Camel and CXF is to provide a web service that offloads work for asynchronous processing later.
  - Message is placed onto a JMS queue, and then an acknowledgement is returned.
  - Work is carried out later.



# Use-case 2: asynch processing from SOAP/HTTP to JMS (cont')

```
<cxf:cxfEndpoint id="customer-ws"
        address="http://0.0.0.0:9003/Customer"
        endpointName="c:SOAPOverHTTP"
        serviceName="c:CustomerService"
        wsdlURL="wsdl/CustomerService.wsdl"
        xmlns:c="http://demo.fusesource.com/wsdl/CustomerService/"/>

<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="cxf:bean:customer-ws?dataFormat=PAYLOAD"/>
    <choice>
      <when>
        <simple>${in.header.operationName} == 'updateCustomer'</simple>
                ...
      </when>
      <when>
        <simple>${in.header.operationName} == 'lookupCustomer'</simple>
                ...
      </when>
      <when>
        <simple>${in.header.operationName} == 'getCustomerStatus'</simple>
        <convertBodyTo type="org.w3c.dom.Node"/>
                ...
      </when>
    </choice>
  </route>
</camelContext>
```

No need for generated code! Just specify the WSDL location.

You *must* set the dataFormat to PAYLOAD in the URI.




FuseSource
A Progress Software Company

# Use-case 2: asynch processing from SOAP/HTTP to JMS (cont')

- Place incoming payload onto a reliable JMS queue for offline processing, and return an acknowledgment response.
  - Note usage of inOut and jmsMessageType when sending to the queue!
  - Note creation of response from inline XML – neat!

```
<when>
  <simple>${in.header.operationName} == 'updateCustomer'</simple>
  <log message="Placing update customer message onto queue."/>
  <inOnly uri="activemq:queue:CustomerUpdates?jmsMessageType=Text"/>
  <transform>
    <constant>
    <![CDATA[
      <ns2:updateCustomerResponse
          xmlns:ns2="http://demo.fusesource.com/wsdl/CustomerService/"/>
    ]]>
    </constant>
  </transform>
</when>
```

# Using Xpath and templates for request-response traffic

- Nice idea: inject values into a pre-packaged XML template.
  - Use XPath to extract useful data from the request and store as a header.
  - Inject response values into XML!

```
<when>
  <simple>${in.header.operationName} == 'getCustomerStatus'</simple>
  <convertBodyTo type="org.w3c.dom.Node"/>
  <setHeader headerName="customerId">
    <xpath resultType="java.lang.String">/cus:getCustomerStatus/customerId</xpath>
  </setHeader>
  <to uri="getCustomerStatus"/>
  <to uri="velocity:getCustomerStatusResponse.vm"/>
</when>
```

Processor does the work of getting the customer status, maybe from a DB or backend system.

Nice use of the velocity component to generate response!




FuseSource
A Progress Software Company

# Using headers to transmit request information…

- A custom Processor can retrieve the customerId, and store response information as headers on the message.

```
public class GetCustomerStatus implements Processor
{
    public void process(Exchange exchng) throws Exception {
        String id = exchng.getIn().getHeader("customerId", String.class);

        // Maybe do some kind of lookup here!
        //

        exchng.getIn().setHeader("status", "Away");
        exchng.getIn().setHeader("statusMessage", "Going to sleep.");
    }
}
```

# Creating a response using Velocity

- Headers from the Camel Exchange are injected easily into a Velocity template using ${headers.<headerName>} place-holders.
  - Example velocity template (src/main/resources/getCustomerStatusResponse.vm):

```
<ns2:getCustomerStatusResponse
  xmlns:ns2="http://demo.fusesource.com/wsdl/CustomerService/">
  <status>${headers.status}</status>
  <statusMessage>${headers.statusMessage}</statusMessage>
</ns2:getCustomerStatusResponse>
```

# Use-case throttling access to third-party services

- Problem: invoke on a web service, ensuring that invocations *and* retries do not exceed  messages per time period…
  - Motivation: breaking the third-party web service's SLA involves $$$ penalties.
- Can use the throttle() EIP from Camel!
  - Note that to ensure throttle is applied to retries, we must 'spin' out the throttled code to separate route.
  - This is because retry logic is applied at the point of a failed processor, not at the entire route.

# Throttling access to third-party services (cont')

- Phase 1: Set up error handling for retry processing, then delegate invocation on web service as a second route.

```
from("timer:t?delay=0&period=15000")
.onException(WebServiceException.class)
  .maximumRedeliveries(3)
  .redeliverDelay(500)
  .to("direct:unableToProcess")
  .end()
.process(
    new Processor() {
      public void process(Exchange exchange) throws Exception {
        Exchange newExchange = exchange.copy();
        producer.send("direct:sendToCXFEndpoint", newExchange);
        if (newExchange.getException() != null) {
          System.out.println("Failure communicating with web service."
                             + newExchange.getException());
          throw newExchange.getException();
        }
      }
    });
```

# Throttling access to third-party services (cont')

- Phase 2: throttle the logic that invokes on the web service.

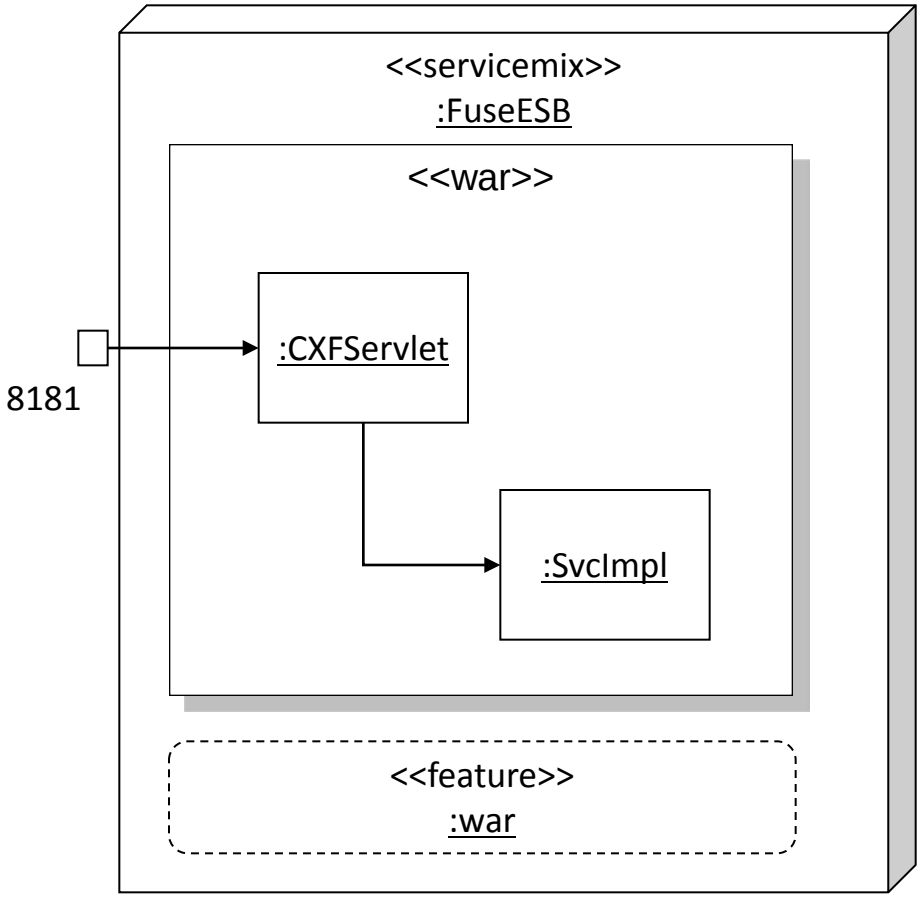
```
from("direct:sendToCXFEndpoint")
.throttle(1).timePeriodMillis(3000)
.process(new Processor() {
  public void process(Exchange exchange) throws Exception {
    Base64Binary image = new Base64Binary();
    image.setContentType("application/octet-stream");
    image.setValue(exchange.getIn().getBody(String.class).getBytes());

    System.out.println("ACTUALLY INVOKING ON THE SERVICE .");
    imageProcessor.processImage(image);
  }
});
```

CXF client proxy.

# CXF Deployment Models

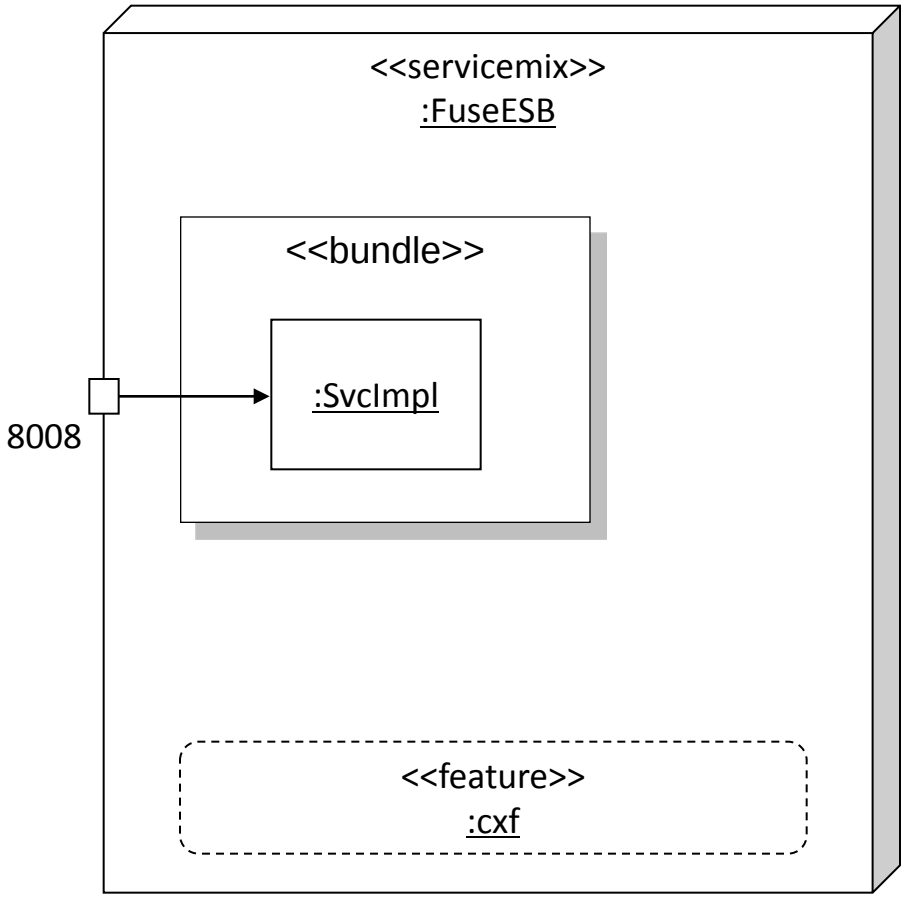
# Deploying web services using JEE web archives (WARs)

- ServiceMix supports WARs using the Jetty servlet engine.
- You must install the ServiceMix 'war' feature…
  - … and copy the 'war' file to the <servicemix-base>/deploy directory.
- Benefits:
  - Simple WAR deployment
  - Works for Tomcat and J£E servers.
- Drawbacks
  - 'Fat' deployment, approx 8Mb per service.



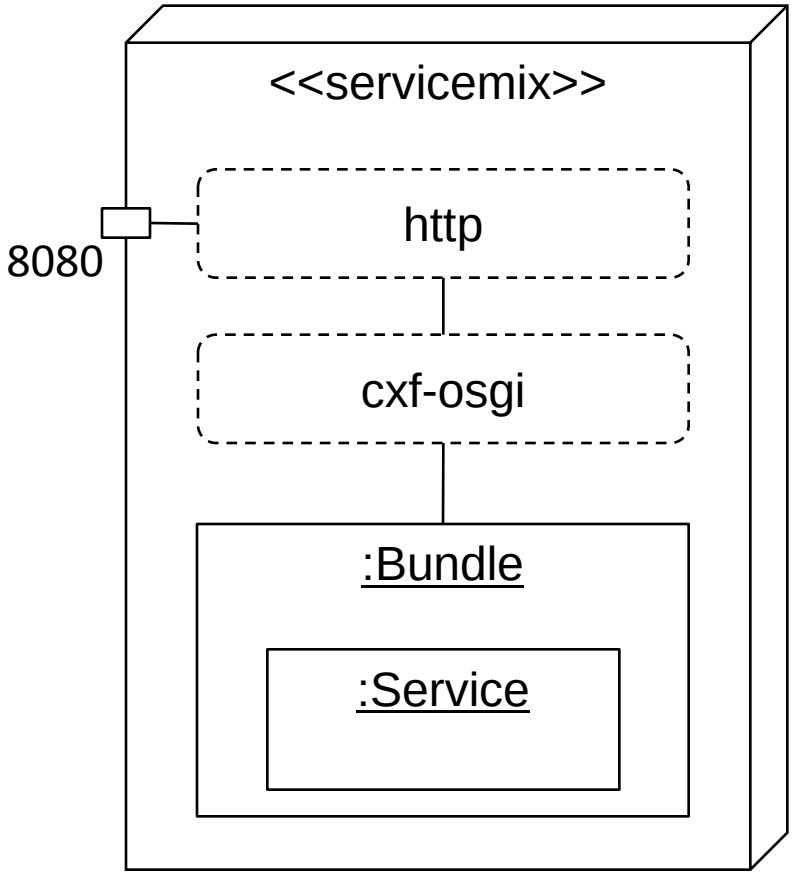
# Deploying web services using OSGi bundles

- Package as an OSGi bundle with Spring-DM or 'Blueprint' meta-data.
  - Service can use it's own HTTP Jetty engine *or* share the OSGi HTTP service (see next slide)
- Benefits:
  - Adopt modular OSGi architecture with shared services and shared code.
  - Versioned artifacts
  - Lightweight deployables – approx 16k (500 times smaller than equivalent WAR!)
  - Allows 'per-service' control over HTTP port configuration



# Simplify configuration of CXF – leave it to the container!

- Fuse ESB enables HTTP access using the OSGi HTTP Service, implemented using 'pax-web'
  - Install the 'http' feature – fully configurable HTTP stack powered by Jetty
- Uses port 8080 by default.
  - All HTTP options (including security) configured by etc/org.ops4j.pax.web.cfg
  - Example on next slide.
- CXF can 'piggy-back' onto that port.
  - Install the 'cxf-osgi' feature.



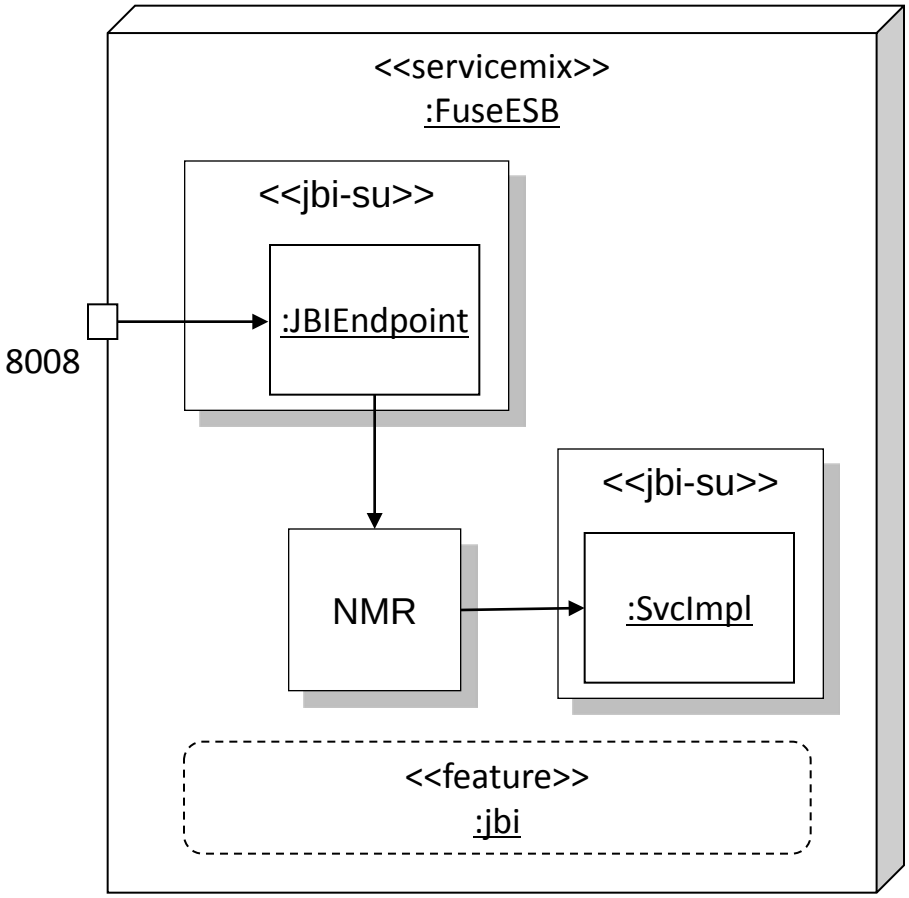
# org.ops4j.pax.web.cfg

- Sample configuration below *disables* HTTP and enables HTTPS
  - See http://wiki.ops4j.org/display/paxweb/Basic+Configuration for more.

```
org.osgi.service.http.enabled=false
org.osgi.service.http.port=8080

org.osgi.service.http.port.secure=8443
org.osgi.service.http.secure.enabled=true

org.ops4j.pax.web.ssl.keystore=./etc/samwise.jks
org.ops4j.pax.web.ssl.password=samwise
org.ops4j.pax.web.ssl.keypassword=samwise

org.ops4j.pax.web.listening.addresses=samwise.local
```

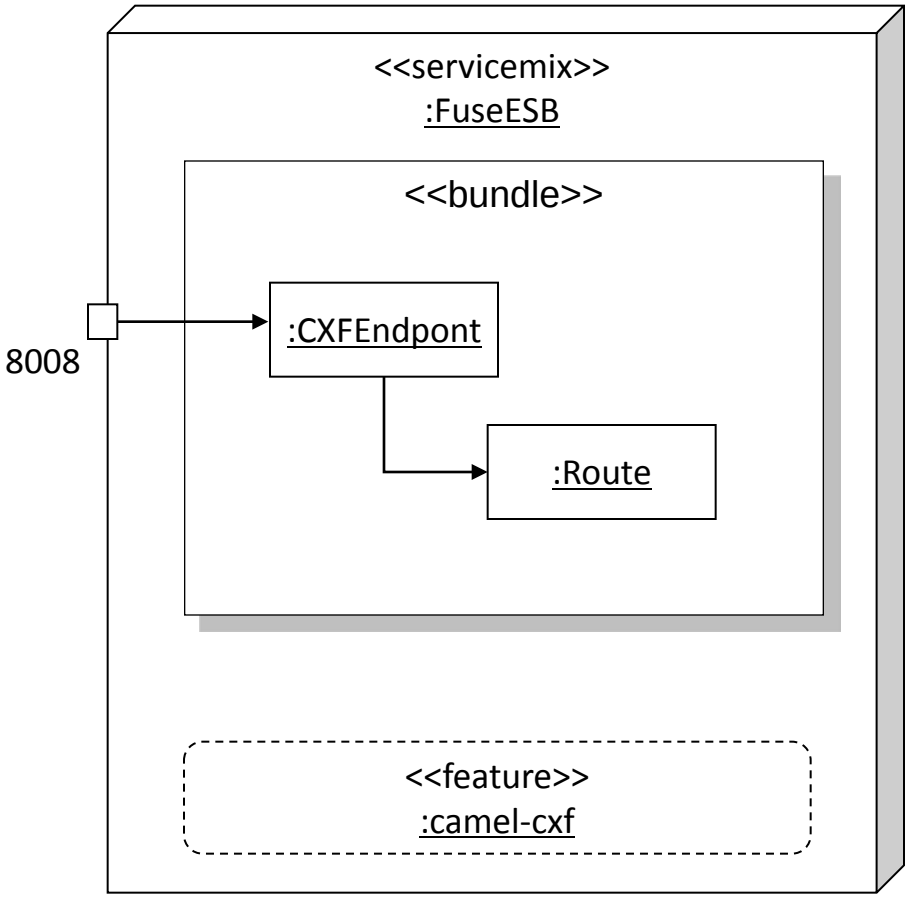
# Deploying web services using JBI service assemblies

- Can configure the transport using the CXF 'binding component' and the implementation using the CXF 'service engine'.
- Benefits:
  - Can integrate with other JBI components.
- Drawbacks:
  - JBI packaging often overly complicated for most cases.
- *Recommendation*: prefer the OSGi, WAR or Camel (next slide!) approaches.



# Using CXF with Camel in ServiceMix

- Can use the `camel-cxf` component to create integration flows that provide and consume SOAP or REST interfaces.
- Benefits
  - Easily route marshaled (JAX-B) or unmarshaled (DomSource / SoapMessage) content.
  - Build elegant integration flows based on Enterprise Integration Patterns (EIPs)
  - Can integrate with ServiceMix 4's NMR for scalability, flexibility and clustering.



# All these choices…

- Apache ServiceMix thrives on innovation and experimentation.
  - … it's not surprising that ServiceMix provides many ways to implement web services.
- My recommendations:
  - If you want to using/implement web services using Java programming, then use CXF's JAX-WS support and package as OSGi bundles.
  - If you want to route SOAP traffic with little marshalling overhead, then use Camel's camel-cxf component.

# Parting words

# Adopting CXF? Your team needs the following…

- Maven / Ant skills (code generation + build + packaging)
- JAX-WS / JAX-B
- XSD
- WSDL
- Either:
  - Good Java,
  - Spring Framework, or
  - Blueprint
- Either:
  - OSGi packaging
  - WAR packaging







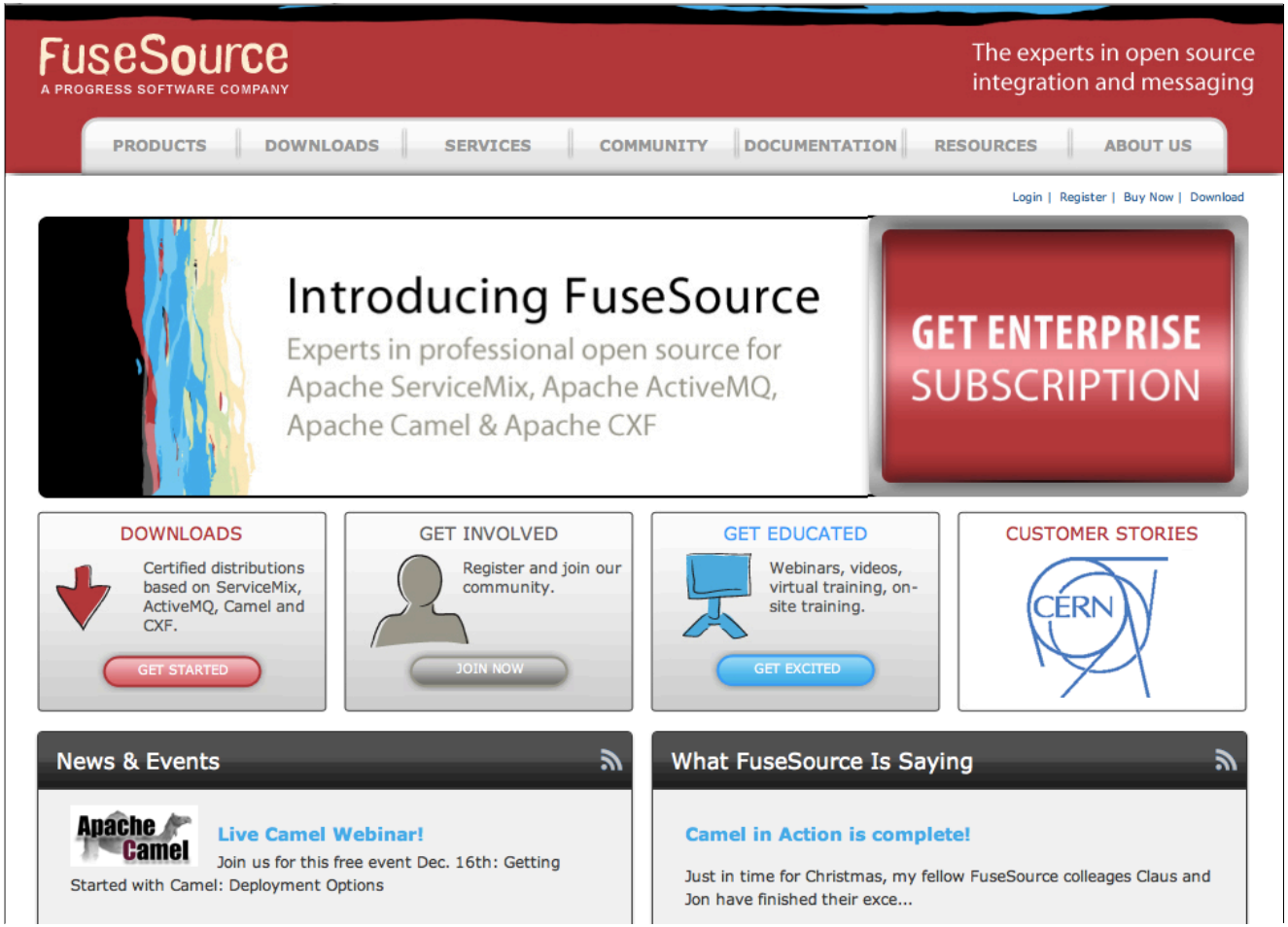


FuseSource
A Progress Software Company

# Summary

- Camel gives new EIP-based techniques for implementing web services.
  - Camel-CXF gives you the smart endpoint technology.
    - Also: camel-restlet, camel-http, camel-jetty, and camel-jaxrs are all relevant!
  - Camel DSL gives you elegant EIPs: content-based router, transformer, protocol switch, throttle, …
- Camel lets your choose the *right* integration technology for the job at hand.
  - Use WS/REST for your external entry points.
  - Use EIPs and other camel components for orchestrating and implementing your integration flows!

# Learn More at http://fusesource.com








FuseSource
A Progress Software Company

# Pragmatic Service-orientated Integration – Camel just got CXFy

Adrian Trenaman,
CamelOne, Washington DC,
May 23rd, 2011

twitter : adrian_trenaman | LinkedIn: adrian.trenaman
http://trenaman.blogspot.com

FuseSource
A Progress Software Company