Apache ServiceMix

Planning

- Overview and architecture
- Getting started
- Camel
- Normalized Message Router
- Java Business Integration

Planning

- Overview and architecture
- Getting started
- Camel
- Normalized Message Router
- Java Business Integration

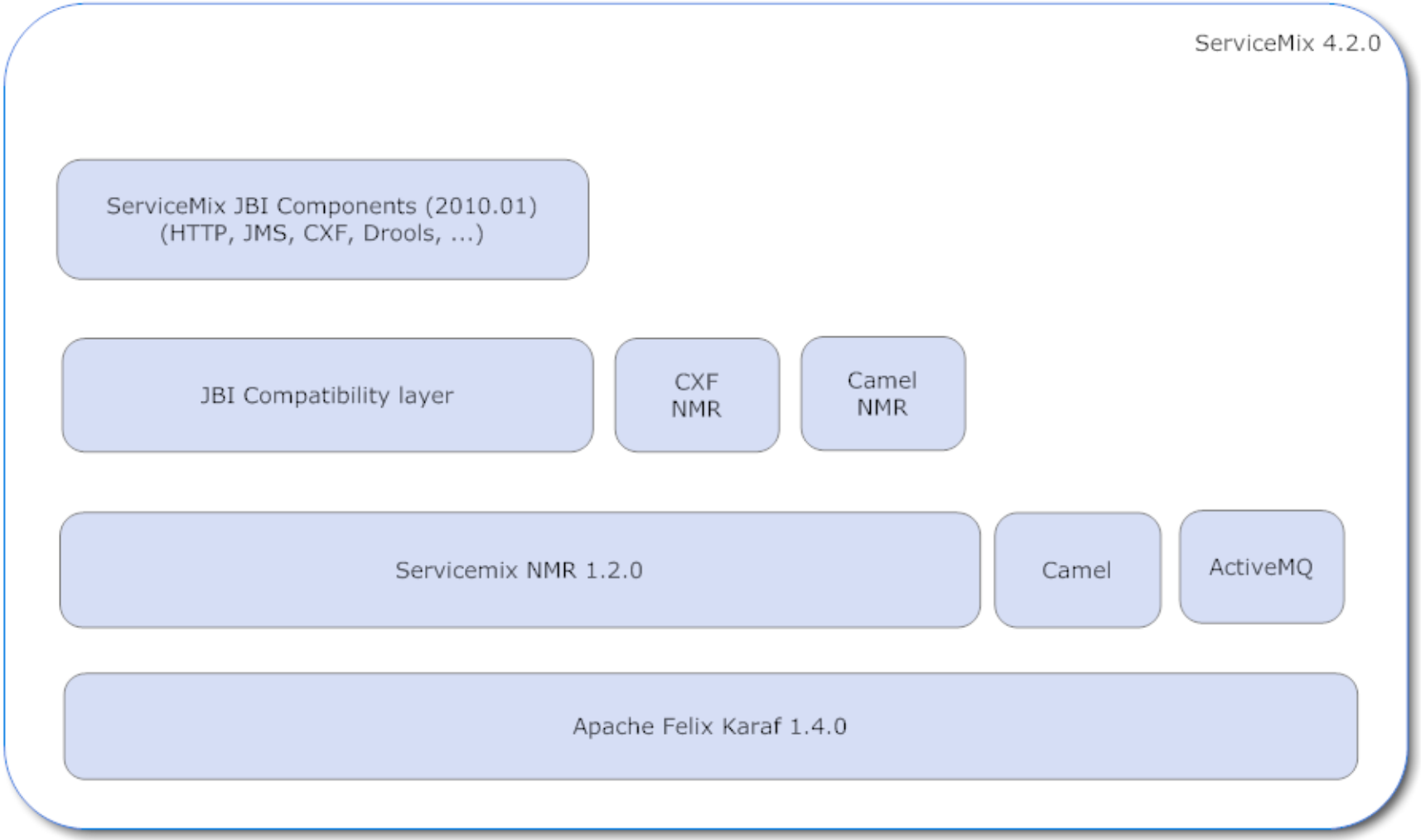
Overview and architecture

- ServiceMix 4
  - Open-source ESB
    - Apache Software License
    - Commercial support available
  - OSGi-based
  - Support for JBI 1.0

Overview and architecture

- ServiceMix 4 consists of
  - Apache Felix Karaf
  - Apache Camel
  - Apache ActiveMQ
  - NMR implementation
  - JBI implementation

Overview and architecture



Planning

- Overview and architecture
- Getting started
- Camel
- Normalized Message Router
- Java Business Integration

Getting started

- Download
- Installation
- Running the container
  - interactive
  - background
  - as a service

Getting started

- Download
  - from http://servicemix.apache.org
  - two archive formats
    - .tar.gz for Linux, Unix, Solaris and MacOS
    - .zip for Windows

Installation

- Requirements
  - 100 MB free disk space
  - JDK 1.5 or higher
- Installation
  - unzip the archive to the local disk

Running the container

- interactive
  - bin/servicemix starts ServiceMix
  - access the command shell console

```
apache-servicemix-4.2.0 gert$ bin/servicemix
 ____                  _          __  __ _
/ ___|  ___ _ ____   _(_) ___ ___|  \/  (_)_  __
\___ \ / _ \ '__\ \ / / |/ __/ _ \ |\/| | \ \/ /
 ___) |  __/ |   \ V /| | (_|  __/ |  | | |>  <
|____/ \___|_|    \_/ |_|\___\___|_|  |_|_/_/\_\

  Apache ServiceMix (4.2.0)

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.

karaf@root>
```

Running the container

- in the background
  - bin/start script starts ServiceMix
  - bin/stop scripts stops ServiceMix
  - access the console
    - using ssh
    - using the webconsole (when installed)

Running the container

- as a service
  - uses Tanuki Java Software Wrapper
  - start ServiceMix interactively
  - in the console
    - features:install wrapper
    - wrapper:install
  - follow instructions to complete installation

Running the container

```
Karaf

C:\tools\apache-servicemix-4.2.0>bin\servicemix.bat

  Apache ServiceMix (4.2.0)

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.

karaf@root> features:install wrapper
karaf@root> wrapper:install
Creating file: C:\tools\apache-servicemix-4.2.0\bin\karaf-wrapper.exe
Creating file: C:\tools\apache-servicemix-4.2.0\etc\karaf-wrapper.conf
Creating file: C:\tools\apache-servicemix-4.2.0\bin\karaf-service.bat
Creating file: C:\tools\apache-servicemix-4.2.0\lib\wrapper.dll
Creating file: C:\tools\apache-servicemix-4.2.0\lib\karaf-wrapper.jar

Setup complete.  You may want to tweak the JVM properties in the wrapper configuration file:
        C:\tools\apache-servicemix-4.2.0\etc\karaf-wrapper.conf
before installing and starting the service.


To install the service, run:
  C:> C:\tools\apache-servicemix-4.2.0\bin\karaf-service.bat install

Once installed, to start the service run:
  C:> net start "karaf"

Once running, to stop the service run:
  C:> net stop "karaf"

Once stopped, to remove the installed the service run:
  C:> C:\tools\apache-servicemix-4.2.0\bin\karaf-service.bat remove

karaf@root>
```

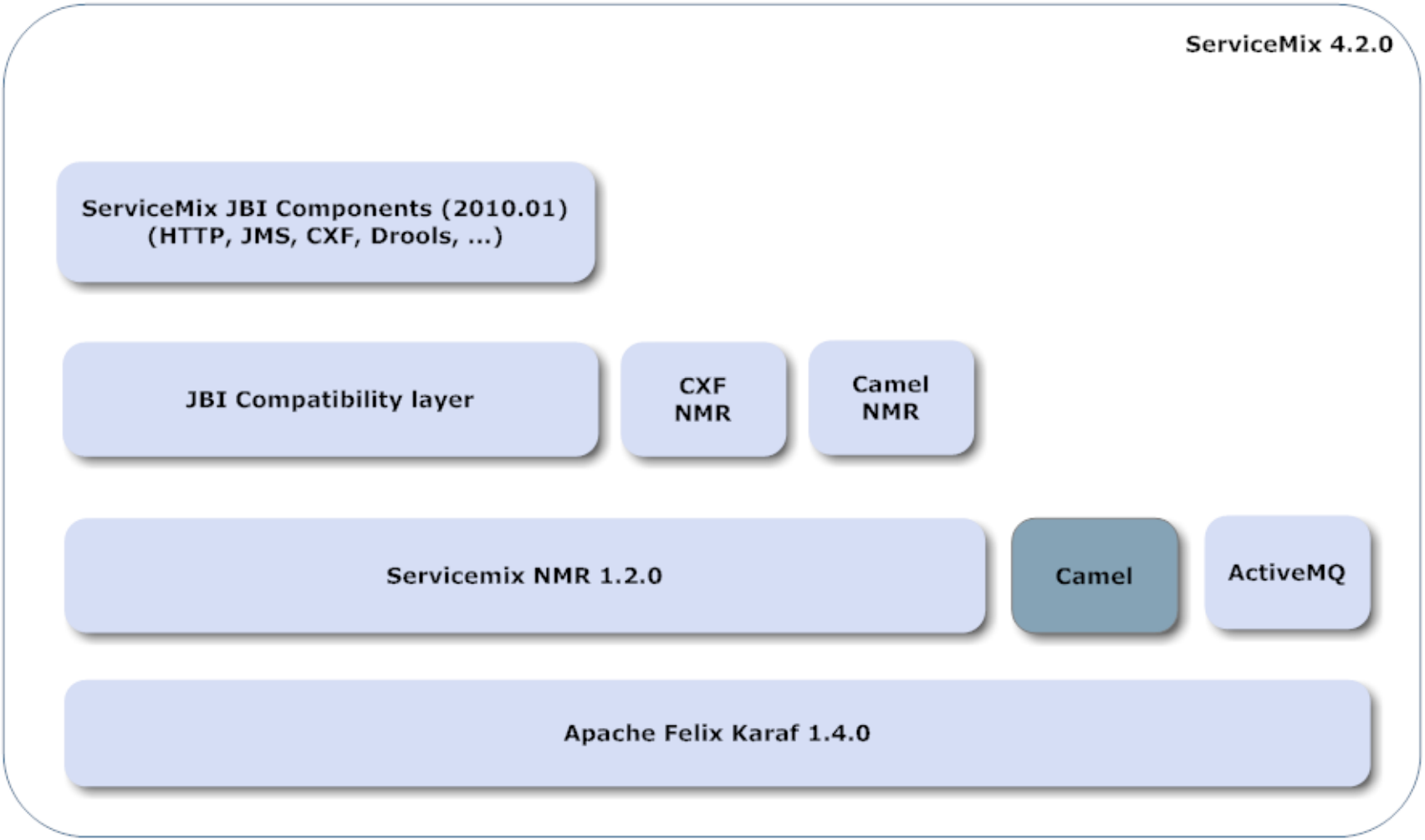
Planning

- Overview and architecture
- Getting started
- Camel
- Normalized Message Router
- Java Business Integration

Camel

- Introduction
- Installing Camel Components
- Deploying Camel Routes

Camel



Introduction

- What is Apache Camel?
  - open-source mediation and routing
  - based on Enterprise Integration Patterns
  - routes are defined in
    - Java DSL
    - Spring XML DSL
    - Scala DSL

Installing Camel Components

- Camel itself is installed by default
- Additional Camel Components
  - can be installed through features

```
karaf@root> features:list | grep camel
[installed  ] [2.2.0      ] camel                    repo-0
[installed  ] [2.2.0      ] camel-core               repo-0
[installed  ] [2.2.0      ] camel-spring-osgi        repo-0
[uninstalled] [2.2.0      ] camel-cxf                repo-0
[uninstalled] [2.2.0      ] camel-mina               repo-0
[uninstalled] [2.2.0      ] camel-jetty              repo-0
```

```
karaf@root> features:install camel-scala
karaf@root> features:install camel-tagsoup 2.2.0
```

Deploying Camel Routes

- 3 options for deployment
  - Plain Spring XML DSL deployment
  - Spring XML DSL in a bundle
  - Java/Scala DSL in a bundle

Deploying Camel Routes

- Plain Spring XML DSL deployment

```
<?xml version="1.0"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:camel="http://camel.apache.org/schema/spring"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans.xsd
        http://camel.apache.org/schema/spring
          http://camel.apache.org/schema/spring/camel-spring.xsd">

  <camelContext xmlns="http://camel.apache.org/schema/spring">
    <route>
      <from uri="timer:camel-on-smx4?period=3000" />
      <to uri="log:camel-on-smx4"/>
    </route>
  </camelContext>

</beans>
```

Deploying Camel Routes

- Create a bundle with
  - Spring XML file in META-INF/spring
- Routes can be defined
  - in the Spring XML file itself
  - in Java/Scala DSL RouteBuilders

Deploying Camel Routes

- Spring XML file with RouteBuilder classes

```
<?xml version="1.0"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:camel="http://camel.apache.org/schema/spring"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans.xsd
        http://camel.apache.org/schema/spring
          http://camel.apache.org/schema/spring/camel-spring.xsd">

  <camelContext xmlns="http://camel.apache.org/schema/spring">
    <packageScan>
        <!-- refer to package that contains RouteBuilders -->
        <package>be.anova.course.servicemix.camel</package>
        <excludes>**.Base*</excludes>
        <includes>**.*</includes>
    </packageScan>
  </camelContext>

</beans>
```

When to use Camel?

- Default choice for integration
  - easy to use
  - feature-rich
  - lots of components available
  - add other technology when necessary
    - ActiveMQ for reliable, async messaging
    - CXF for web services support
    - NMR
    - JBI

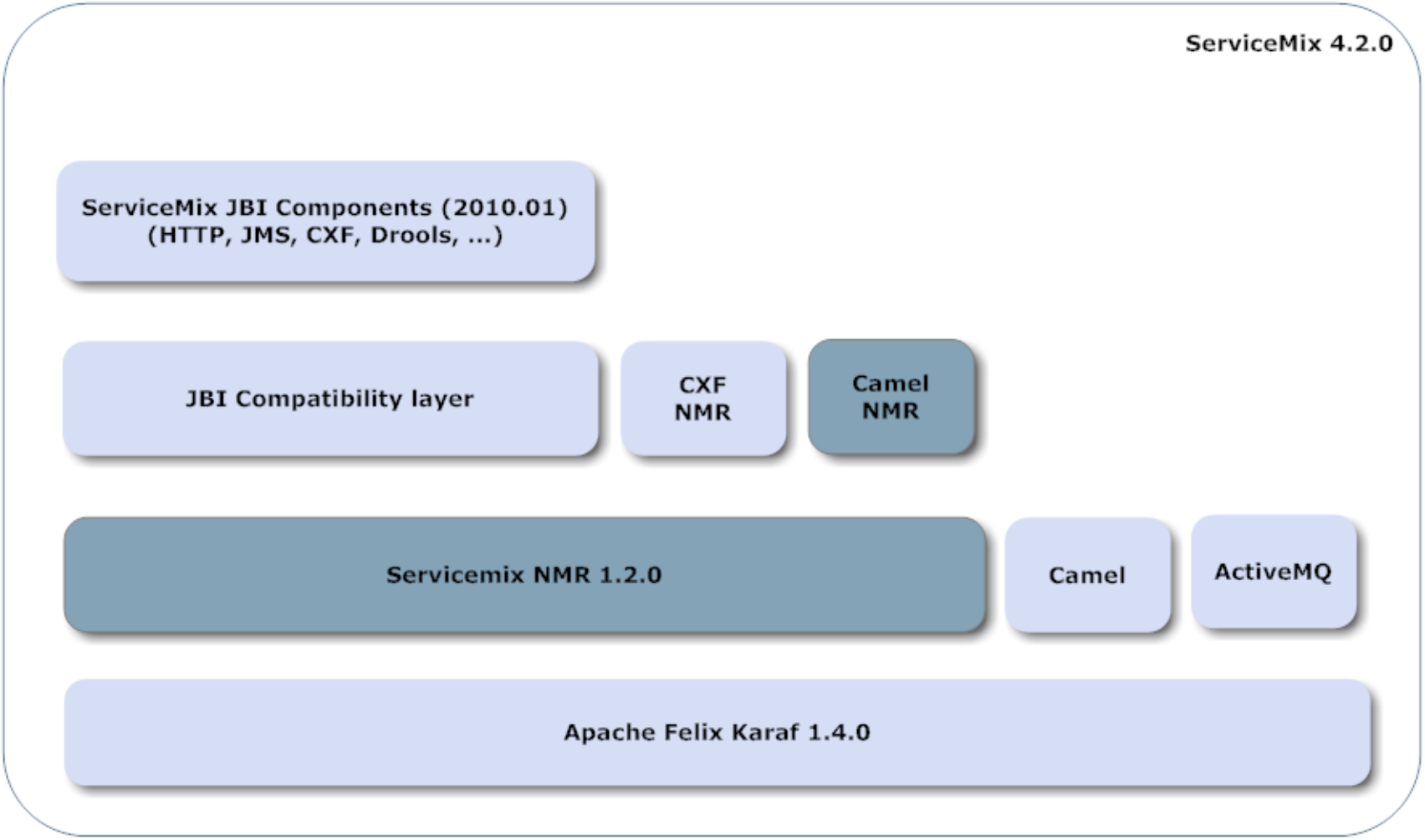
Planning

- Overview and architecture
- Getting started
- Camel
- Normalized Message Router
- Java Business Integration

NMR

- Introduction
- NMR API
- Camel NMR component
- When to use?
- Exercise

NMR



Introduction

- Normalized Message Router
  - Used for
    - foundation for JBI support
    - Camel NMR messaging
    - CXF NMR messaging
  - Messaging API with OSGi-based implementation
  - Ensure ServiceMix' independence of different messaging technologies

Introduction

- Other features
  - Endpoint wiring
  - Listener interfaces
    - ExchangeListener
    - EndpointListener
  - Shell for managing the NMR

NMR API

- Exchange
  - used for exchanging messages between endpoints
  - contains
    - ID
    - MEP
    - properties
    - in, out and fault Message

NMR API

- Message
  - body
    support for both XML and Object payloads
  - headers
  - attachments

NMR API

- Example: Exchange and Message

```
public void process(Exchange exchange) {
    String id = exchange.getId();
    Pattern pattern = exchange.getPattern();
    String property =
           exchange.getProperty("property.key", String.class);

    Message in = exchange.getIn();
    Object object = in.getBody();
    String body = in.getBody(String.class);

    Object attached = in.getAttachment("attachment.name");
    Object header = in.getHeader("header.key");
}
```

NMR API

- Listener interfaces
  - ExchangeListener
  - EndpointListener
- Use OSGi whiteboard pattern
  - register Listener interfaces in OSGi Service Registry
  - NMR will invoke methods on all registered instances

NMR API

- Example: ExchangeListener

```
private class MyExchangeListener implements ExchangeListener {
  public void exchangeDelivered(Exchange exchange) {
  }
  public void exchangeFailed(Exchange exchange) {
  }
  public void exchangeSent(Exchange exchange) {
  }
}
```

NMR API

- Example: EndpointListener

```
public class MyEndpointListener implements EndpointListener {
  public void endpointRegistered(InternalEndpoint endpoint) {
  }
  public void endpointUnregistered(InternalEndpoint endpoint) {
  }
}
```

NMR API

- Example: Blueprint service registration

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <bean id="MyEndpointListener" class="listener.MyEndpointListener" />

  <service ref="MyEndpointListener">
    <interfaces>
      <value>
        org.apache.servicemix.nmr.api.event.EndpointListener
      </value>
      <value>org.apache.servicemix.nmr.api.event.Listener</value>
    </interfaces>
  </service>

</blueprint>
```

Camel NMR component

- Camel Component
  - URI to interact with NMR
    - sending exchanges to NMR endpoint
    - register NMR endpoint to receive exchanges
  - URI syntax: nmr:EndpointName
- Add to Spring XML file with import

```
<import
  resource="classpath:org/apache/servicemix/camel/nmr/camel-nmr.xml" />
```

Camel NMR component

- Example: Using the Camel NMR component

```
// route with provider endpoint
// to send NMR Exchange
from("direct:endpoint")
  .to("nmr:Orders");

// route with consumer endpoint
// registers NMR Endpoint to receive
// Exchanges
from("nmr:Orders")
  .beanRef("OrderService", "processOrder");
```

When to use the NMR?

- Use the NMR
  - for linking Camel routes across bundles (with the Camel NMR component)
  - for linking CXF endpoints across bundles (with the CXF NMR transport)
  - if you require support for pluggable ExchangeListener/EndpointListener (e.g. auditing or BAM)

Exercise

- Connect the sender and receiver Camel router bundles
  - Add import to Spring XML
  - Add NMR endpoints to connect two routes
- Implement an ExchangeListener bundle
  - Log exchange IDs and properties
  - Ensure that the listener can be dynamically (de)activated by stopping/starting the bundle

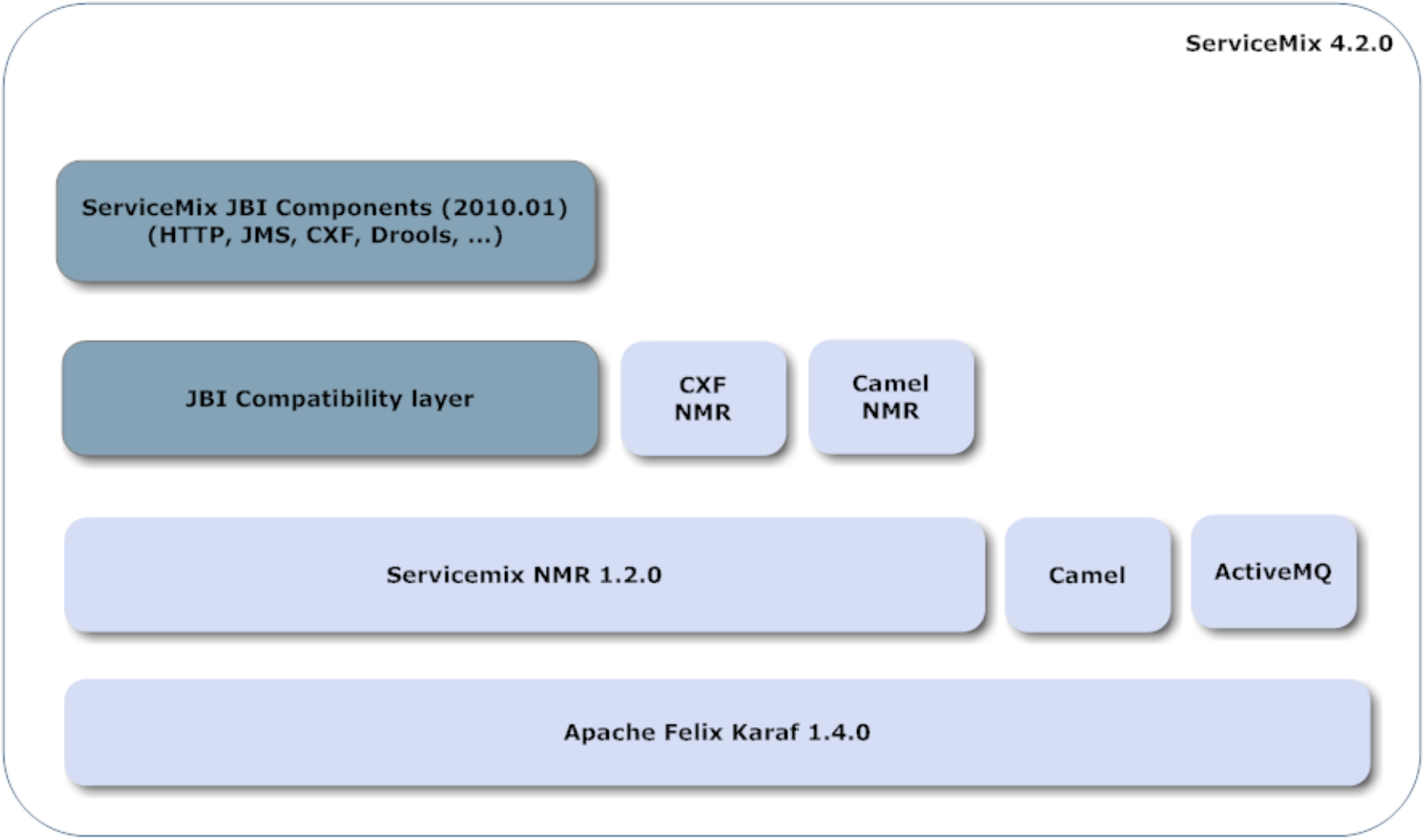
Planning

- Overview and architecture
- Getting started
- Camel
- Normalized Message Router
- Java Business Integration

JBI

- Introduction
- JBI 1.0 in depth
- Packaging options
- ServiceMix JBI Components
- When to use JBI?
- Exercise

JBI



Introduction

- Java Business Integration (JSR-208)
  - Pluggable architecture for integration systems
  - Components interoperate through mediated message exchange
    - based on the WSDL message exchange model
  - Implementations:
    ServiceMix, OpenESB, Petals ESB

Introduction

- Java Business Integration (JSR-208)
  - ServiceMix JBI Container
    - version 3.x is a pure JBI container
    - version 4.x is OSGi-based implementation
  - ServiceMix JBI Components
    - version 2010.01 is latest
    - compatible with both versions

JBI 1.0 in depth

- JBI 1.0 API defines
  - SPI for JBI Component developers
  - Messaging API
  - Mechanism for exchanges to flow between components
  - Standard for packaging components and for packaging the services deployed on them
  - Administration and management hooks to allow for standard tools

JBI 1.0 in depth

- 2 kinds of components
  - Service Engine (SE)
    Allow implementing business logic or services on the ESB
    e.g. servicemix-drools or servicemix-bean
  - Binding Components
    Provide connectivity to external services (transport, normalization, ...)
    e.g. servicemix-ftp or servicemix-http

JBI 1.0 in depth

- Messaging API
  - MessageExchange contains
    - in, out and fault NormalizedMessage
    - exchange properties
    - metadata for routing
      - exchange id
      - exchange pattern
      - error
      - service, endpoint, operation, interface
      - role and status

JBI 1.0 in depth

- Messaging API
  - NormalizedMessage contains
    - XML message content
    - headers
    - attachments
    - security subject

JBI 1.0 in depth

- Example: Messaging API

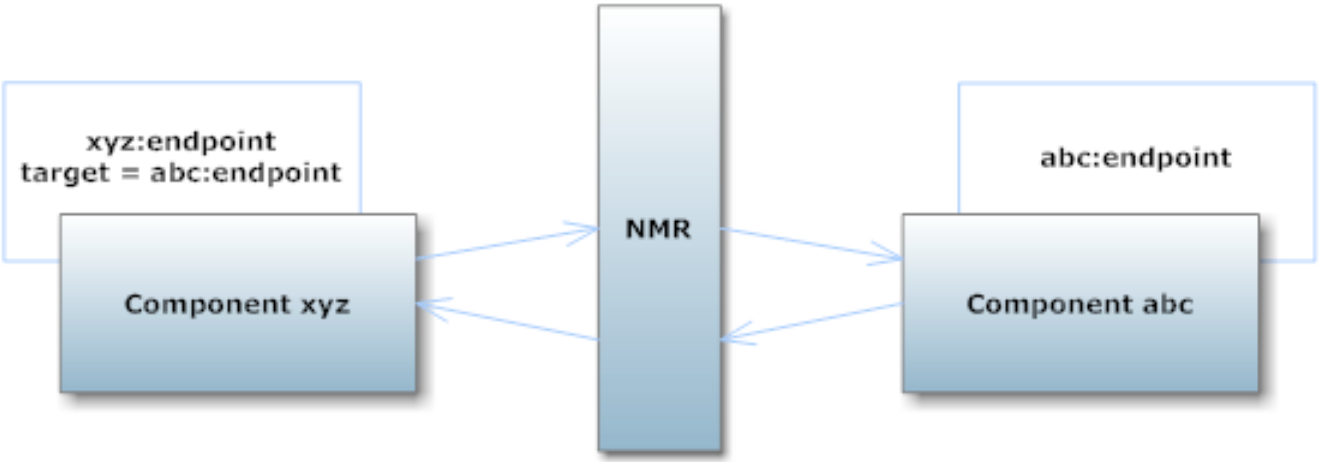
```
public void process(MessageExchange exchange) {
  if (exchange.getStatus() == ExchangeStatus.ACTIVE) {
    String id = exchange.getExchangeId();
    Object property = exchange.getProperty("property.key");

    NormalizedMessage in = exchange.getMessage("in");

    Source content = in.getContent();
    DataHandler attachment = in.getAttachment("attachment.name");
    Object header = in.getProperty("header.key");
  }
}
```

JBI 1.0 in depth

- Endpoints
  - Internal endpoints
    - exposed within the JBI environment
    - examples:
      file sender endpoint
      bean endpoint
  - External endpoints
    - endpoints 'outside' of JBI environment
    - examples:
      file poller endpoint
      jms consumer endpoint

JBI 1.0 in depth

- MessageExchange routing
  - static with service (and endpoint) name
  - static with interface name
  - dynamic with endpoint reference (EPR)

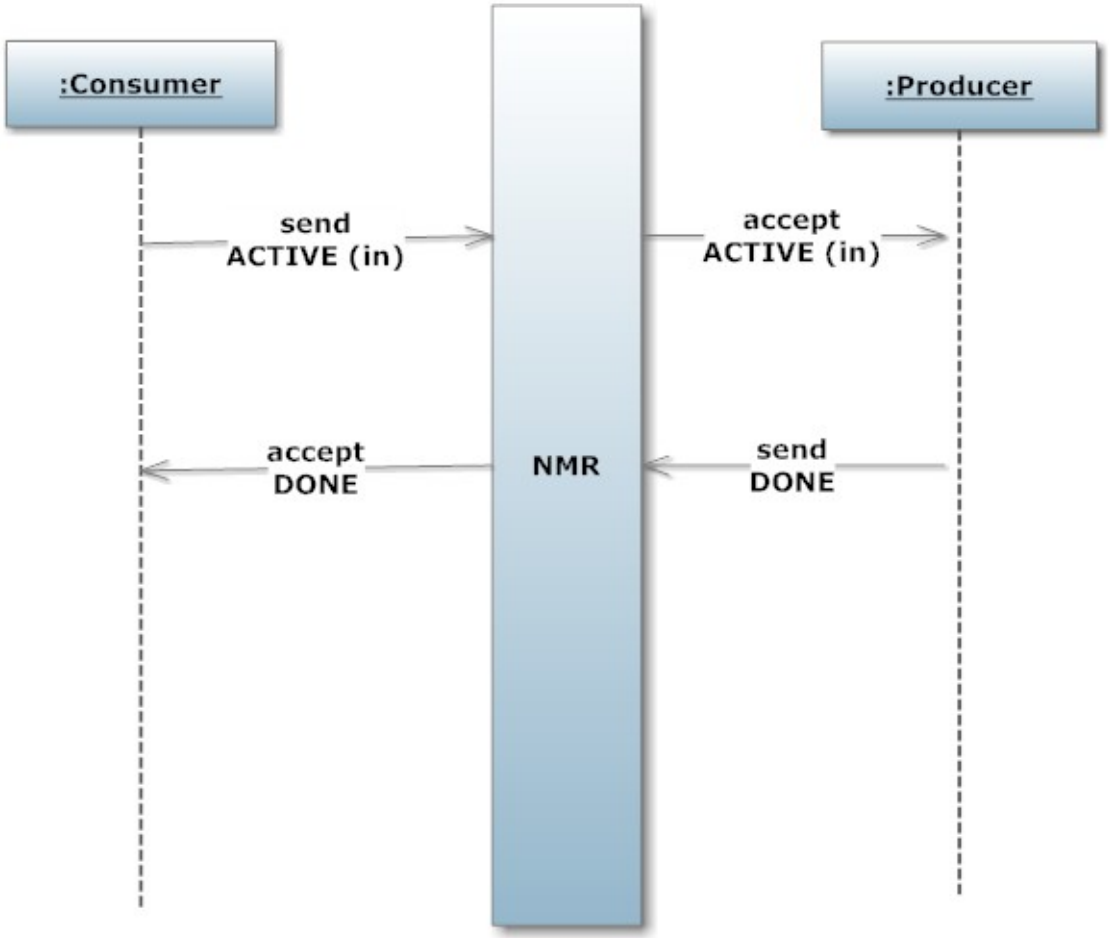


JBI 1.0 in depth

- MEPs
  - MEPs start in ACTIVE status
    - InOnly : in message
    - InOut : in message and out (or fault) message
    - RobustInOnly : in message, optionally fault message on in
    - InOptionalOut : in message, optionally out (or fault), optionally fault message on out
  - All MEPs end with a DONE or ERROR Message

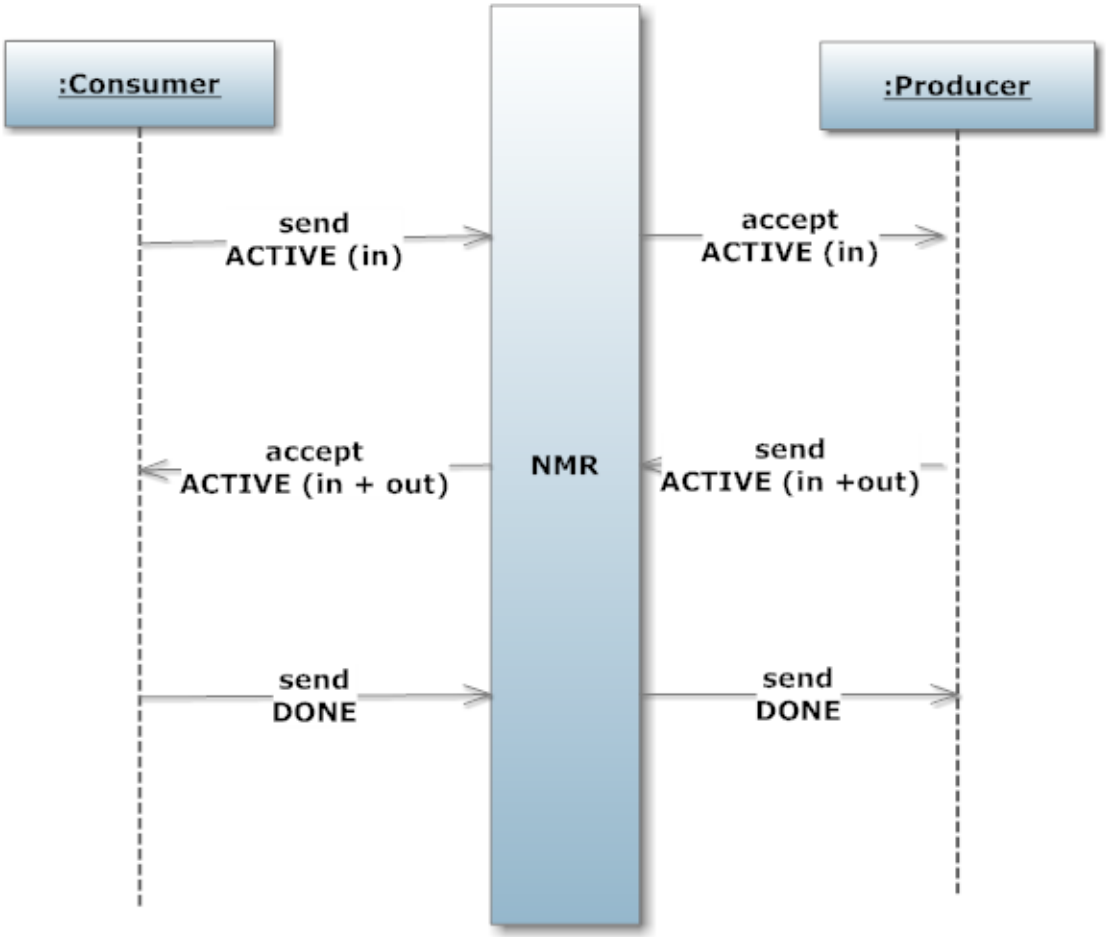
JBI 1.0 in depth

- Example : InOnly MEP



JBI 1.0 in depth

- Example : InOut MEP



Packaging options

- Options for deploying endpoints on ServiceMix 4
  - JBI Packaging (Service Assembly)
  - Spring XML deployment
    - deploy plain XML file
    - deploy XML file in bundle

Packaging options

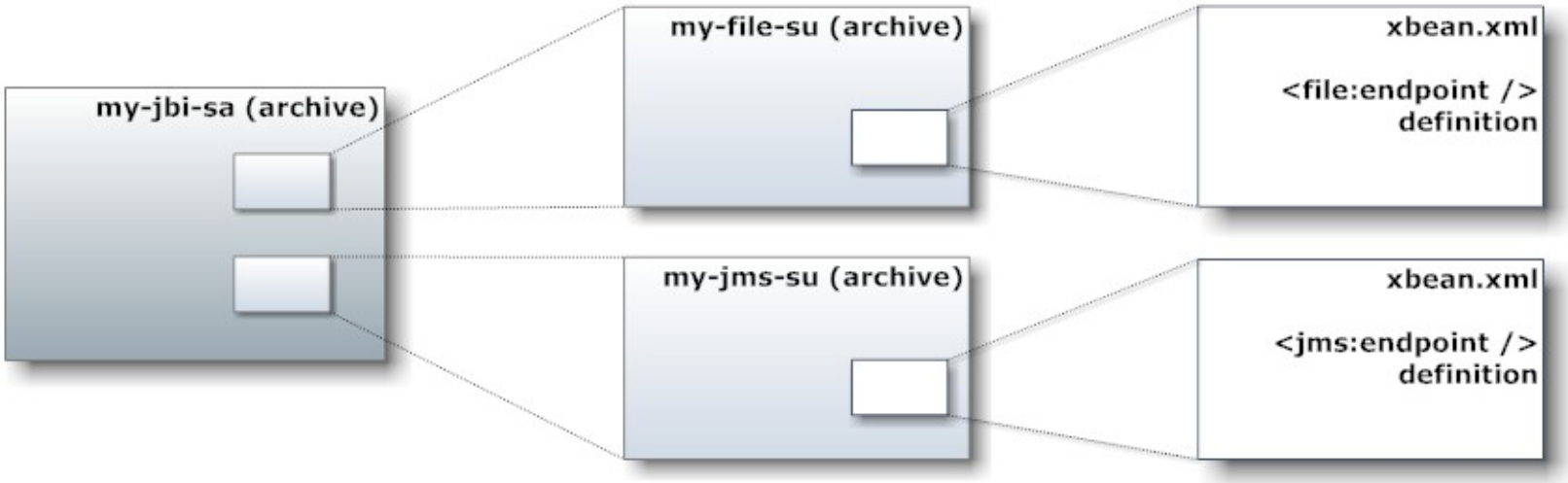
- JBI Packaging - Service Assembly (SA)
  - ZIP archive that contains
    - META-INF/jbi.xml
    - Service Units
  - Deployment descriptor specifies target components for every SU

Packaging options

- JBI Packaging – Service Unit (SU)
  - ZIP archive that contains
    - META-INF/jbi.xml
    - component-specific endpoint descriptions
    - extra classes, JARs, ... used by SU

Packaging options

- JBI Packaging
  - jbi-maven-plugin to create JBI packages



Packaging options

- Single Spring XML file
  - add all endpoints in a single Spring XML
  - add o.a.s.common.osgi.EndpointExporter to register JBI endpoints
- OSGi Bundle with Spring XML File
  - add XML file to META-INF/spring folder

Packaging options

- Example: single XML file for JBI endpoints

```
<beans xmlns:amq="http://activemq.apache.org/schema/core"
       xmlns:file="http://servicemix.apache.org/file/1.0"
       xmlns:jms="http://servicemix.apache.org/jms/1.0"
       xmlns:course="urn:be:anova:course:servicemix:jbi">

  <file:poller service="course:poller" endpoint="orders"
               targetService="course:orders"
               file="file:/home/orders" />

  <jms:producer service="course:orders" endpoint="endpoint"
                connectionFactory="#connectionFactory"
                destinationName="queue.orders" />

  <amq:connectionFactory id="connectionFactory"
                         brokerURL="tcp://localhost:61616"/>

  <bean class="org.apache.servicemix.common.osgi.EndpointExporter" />

</beans>
```

ServiceMix JBI Components

- Binding Components

servicemix-cxf-bc
servicemix-file
servicemix-ftp
servicemix-http
servicemix-jms
servicemix-mail
servicemix-smpp
servicemix-snmp
servicemix-truezip
servicemix-vfs
servicemix-xmpp

ServiceMix JBI Components

- servicemix-file
  - namespace uri http://servicemix.apache.org/file/1.0
  - <file:poller/> for reading files
  - <file:sender/> for writing files

ServiceMix JBI Components

- Example: File SU xbean.xml
  - file poller reads from orders/input
  - file sender writes to orders/output

```
<beans xmlns:file="http://servicemix.apache.org/file/1.0"
       xmlns:course="urn:be:anova:course:servicemix:jbi">

  <file:poller service="course:poller"
               endpoint="poller"
               targetService="course:sender"
               file="file:orders/input" />

  <file:sender service="course:sender"
               endpoint="sender"
               directory="file:orders/output"/>

</beans>
```

ServiceMix JBI Components

- Sidetrack: Marshalers
  - convert transport specific data format into XML (and back again)
  - default implementations available for XML, implement your own for other data formats

ServiceMix JBI Components

- Sidetrack: Marshalers
  - Example: CSV extends DefaultFileMarshaler

```
String getOutputName(MessageExchange, NormalizedMessage) {
   //TODO: determine file name
}

void writeMessage(MessageExchange, NormalizedMessage,
          OutputStream, String)  {
   //TODO: write NormalizedMessage content to OutputStream as CSV
}

void readMessage(MessageExchange, NormalizedMessage,
                InputStream, String) {
   //TODO: read CSV from InputStream and convert to
   //      NormalizedMessage content (Source)
}
```

ServiceMix JBI Components

- servicemix-jms
  - namespace URI http://servicemix.apache.org/jms/1.0
  - endpoints
    - jms:consumer, jms:soap-consumer and jms:jca-consumer for receiving JMS messages
    - jms:provider and jms:soap-provider for sending JMS messages
  - support plain XML and SOAP/JMS payloads

ServiceMix JBI Components

- Example: JMS SU xbean.xml

```
<beans xmlns:amq="http://activemq.apache.org/schema/core"
       xmlns:jms="http://servicemix.apache.org/jms/1.0"
       xmlns:course="urn:be:anova:course:servicemix:jbi">

  <jms:consumer service="course:consumer" endpoint="endpoint"
                connectionFactory="#connectionFactory"
                destinationName="input.orders"
                targetService="course:producer" />

  <jms:producer service="course:producer" endpoint="endpoint"
                connectionFactory="#connectionFactory"
                destinationName="output.orders" />

  <amq:connectionFactory id="connectionFactory"
                         brokerURL="tcp://localhost:61616"/>

</beans>
```

ServiceMix JBI Components

- Service Engines

servicemix-bean	servicemix-osworkflow
servicemix-camel	servicemix-quartz
servicemix-cxf-se	servicemix-saxon
servicemix-drools	servicemix-scripting
servicemix-eip	servicemix-validation
servicemix-exec	servicemix-wsn2005

ServiceMix JBI Components

- servicemix-bean
  - allows processing MessageExchange with Java beans
  - endpoints:
    - <bean:endpoint /> to define POJO endpoint
    - auto-deployed endpoints using @Endpoint annotation
  - provides a set of annotations e.g. @Resource for injecting DeliveryChannel

ServiceMix JBI Components

- Example: the bean code

```
public class ProcessingBean implements MessageExchangeListener {

  @Resource
  private DeliveryChannel channel;

  public void onMessageExchange(MessageExchange exchange)
                                          throws MessagingException {
    if (exchange.getStatus() == ExchangeStatus.ACTIVE) {
      Source content = exchange.getMessage("in").getContent();

      // processing the XML Source here...

      exchange.setStatus(ExchangeStatus.DONE);
      channel.send(exchange);
    }
  }
}
```

ServiceMix JBI Components

- Example: Bean SU xbean.xml

```
<beans xmlns:bean="http://servicemix.apache.org/bean/1.0"
       xmlns:course="urn:be:anova:course:servicemix:jbi">

  <!-- this is a Spring bean -->
  <bean id="processingBean"
        class="be.anova.course.servicemix.jbi.ProcessingBean" />

  <bean:endpoint service="course:bean" endpoint="processing"
                 bean="#processingBean" />

</beans>
```

ServiceMix JBI Components

- servicemix-camel
  - use Camel for routing inside JBI container
  - Spring CamelContext for deployment
  - URI for interacting with JBI
    - jbi:service:<namespace><service>
    - jbi:endpoint:<namespace><service><endpoint>
    - use : or / as separator

ServiceMix JBI Components

- servicemix-camel
  - from("jbi:...")
    - exposes an internal endpoint
    - can be used as target service for other endpoints
  - to("jbi:...")
    - send to another JBI endpoint from within your Camel route

ServiceMix JBI Components

- Example: Java DSL RouteBuilder

```
public class MyRouteBuilder extends RouteBuilder {

  @Override
  public void configure() throws Exception {
    from("timer:events?period=5000")
      .setBody().constant("<hello/>")
      .to("jbi:service:urn:be:anova:servicemix:jbi:forwarder");

    from("jbi:service:urn:be:anova:servicemix:jbi:forwarder")
      .to("jbi:endpoint:urn:be:anova:servicemix:jbi:target:endpoint");

    from("jbi:endpoint:urn:be:anova:servicemix:jbi:target:endpoint")
      .to("log:events");

  }
}
```

ServiceMix JBI Components

- Example: Spring camelContext.xml file

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

  <camelContext xmlns="http://camel.apache.org/schema/spring">
    <packageScan>
      <!-- Use Java/Scala DSL classes for defining routes... -->
      <package>be.anova.course.servicemix.jbi</package>
    </packageScan>

    <!-- ... or use the Spring XML DSL -->
    <route>
      <from
        uri="jbi:endpoint:urn:be:anova:servicemix:jbi:target:endpoint" />
      <to uri="log:target-received"/>
    </route>
  </camelContext>

</beans>
```

When to use JBI?

- When to use JBI in ServiceMix 4?
  - When use of external standards is required
  - To leverage existing investments in JBI
  - To leverage third-party JBI components
  - For building WSDL-oriented SOA applications
  - Support for BPEL using ODE's JBI deployment

Exercise

- In the JBI packaging example
  - create a Camel route to link the two file endpoints and do the XSL transformation
- Define the endpoints in bridge.xml to create a HTTP to JMS bridge
- In the file-to-bean project
  - add the file endpoint
  - display the exchange id and properties from bean endpoint

Planning

- Overview and architecture
- Getting started
- Camel
- Normalized Message Router
- Java Business Integration