Enterprise Integration using Apache Camel

PV207 – Business Process Management

Mgr. Ivo Bek

Senior Product Manager

Red Hat

Introduction to Enterprise Integration

Recapitulation

- Implementing business processes connecting to enterprise systems
- Core principles of interoperability in software systems

  Pub-sub messaging, message routing, event streaming, data integration, api management, workflow orchestration
- Handling the differences between Requester and Provider
- Message processing
- Integration patterns

Integration Project Task #1

- Interface catalog in the project analysis document

Implementing business processes connecting to enterprise systems

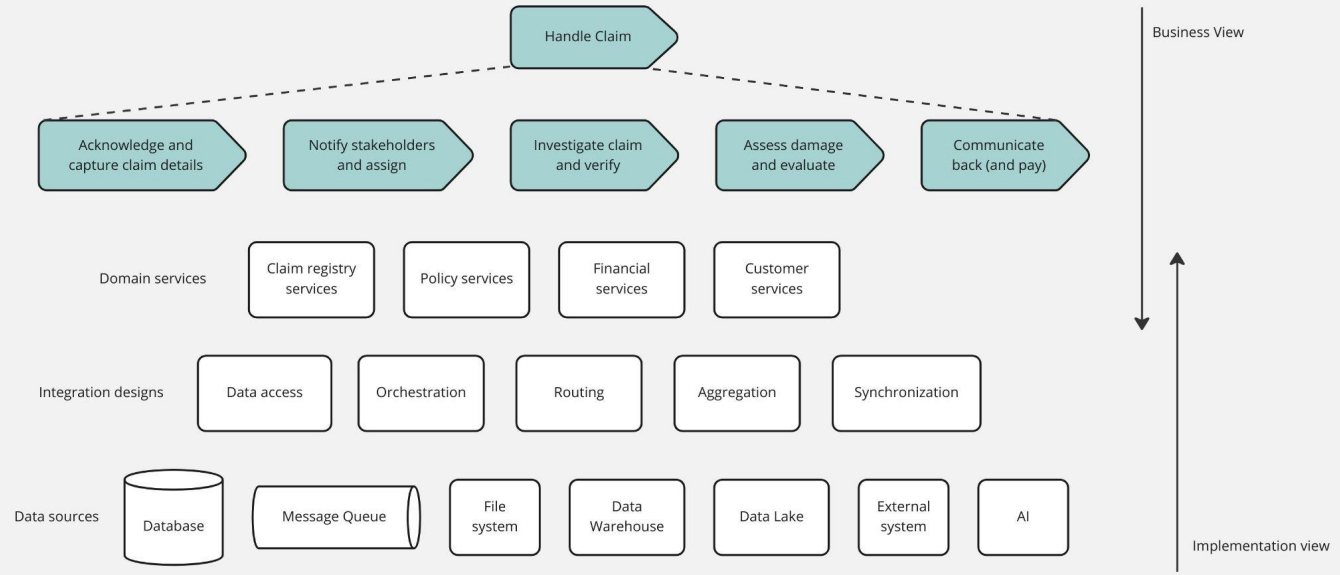
Investigation and Assessment

Business analyst

- 1. Create high level process model
- 2. Establish data objects used by process
- 5. Establish functional usage (operations) of data

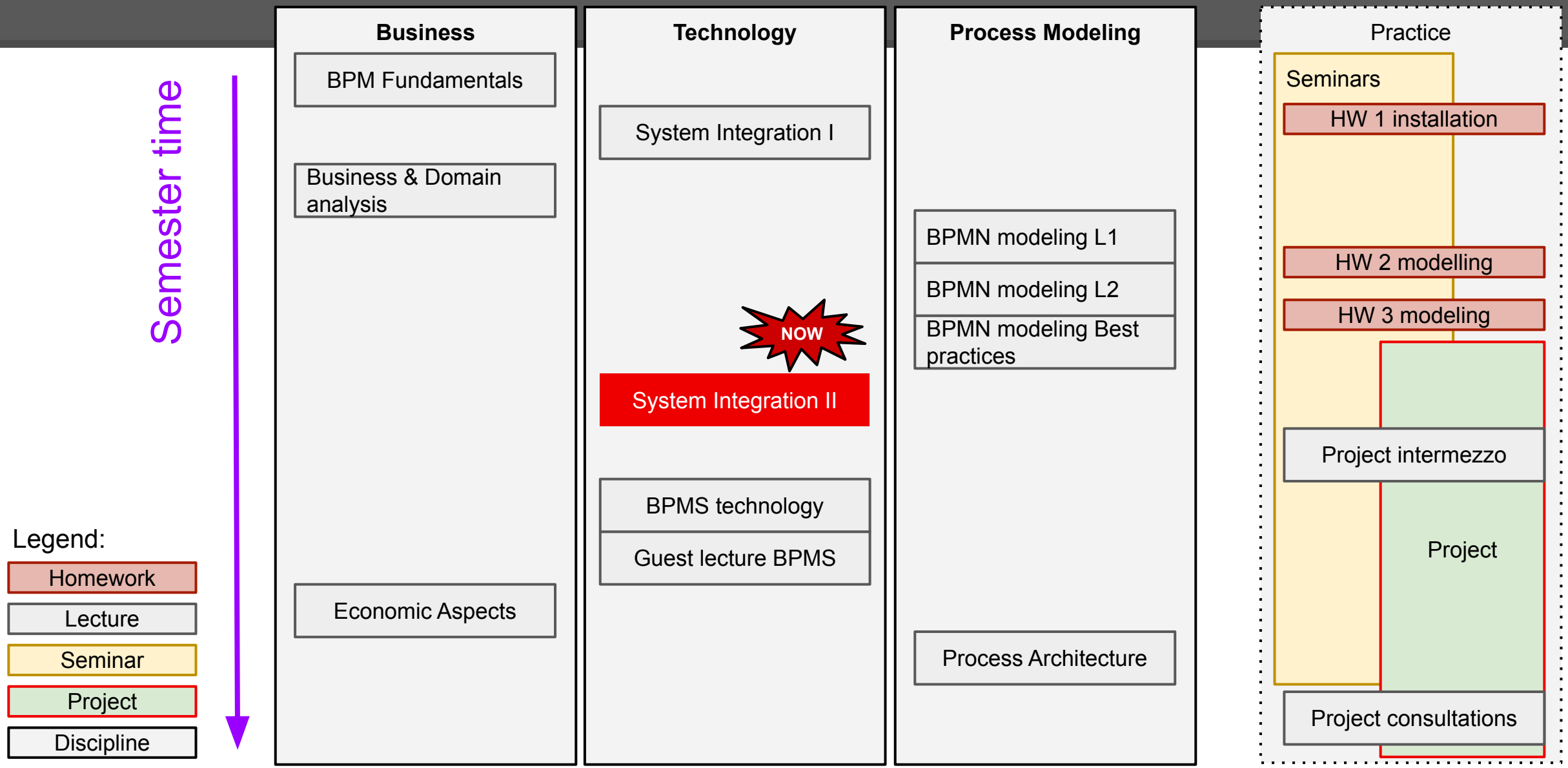
Integration Specialist

- 3. Establish systems containing the data objects
- 4. Establish technical interfaces exposing the data



Red Hat

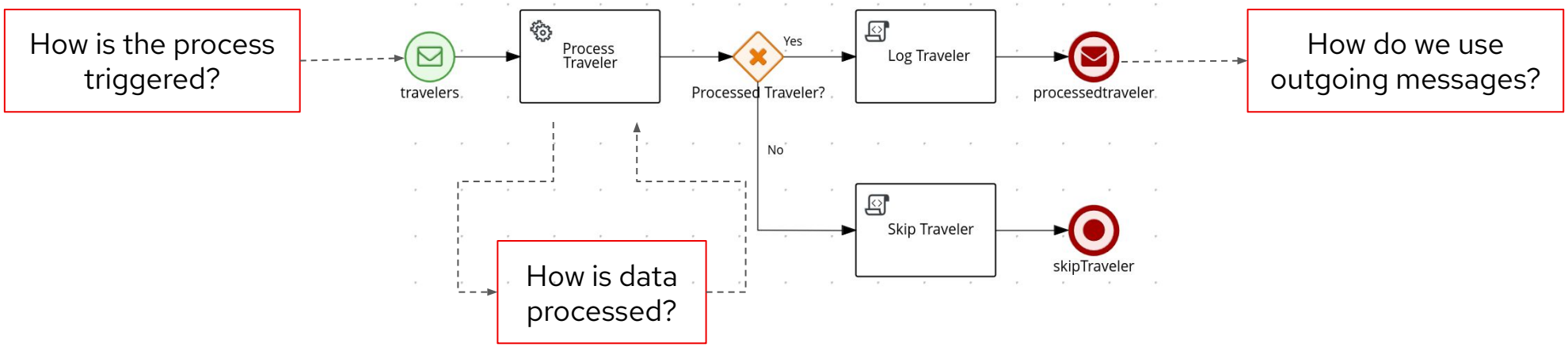
PV207 helicopter view



What we'll discuss today

- What's Apache Camel?
- Developing Camel integration services
- Camel implementations of enterprise integration patterns
- Conditional processing and data extraction with expression languages
- Data format transformations
- Custom processing
- Contract-first OpenAPI integrations
- Setup of Camel development environment

Integrating systems and services in business processes



Let’s build integration services

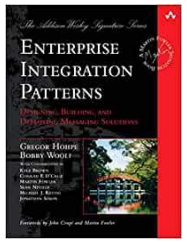
Red Hat

Apache Camel

That could connect to ~~any~~ almost any system

That can work on and off the cloud

This is
Apache Camel



With support for known integration patterns

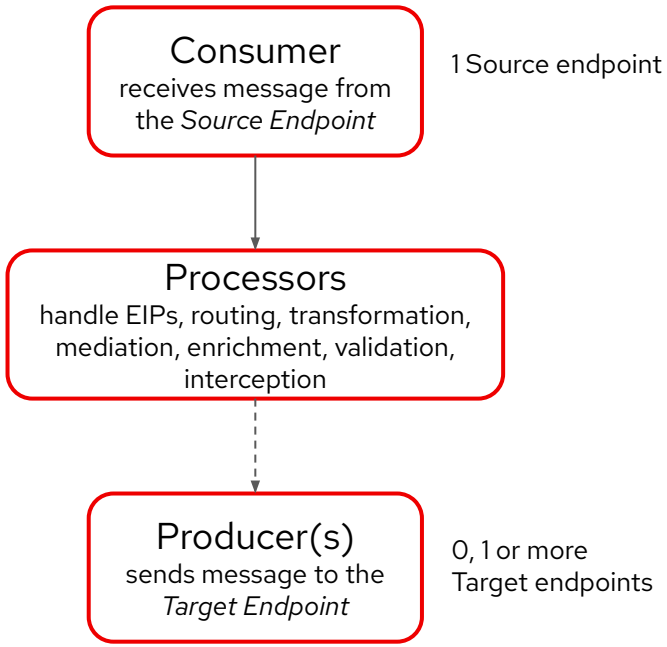
```
from("kafka:topic")
  .to("grpc:endpoint")
```

Integration defined in a simple language. Such as XML, YAML and Java

Red Hat

Camel Route

connect a source endpoint to a destination endpoint



Camel route describes the step-by-step movement of a Message from a source endpoint, through arbitrary types of decision-making routines (such as filters and routers) to a destination endpoint (if any)

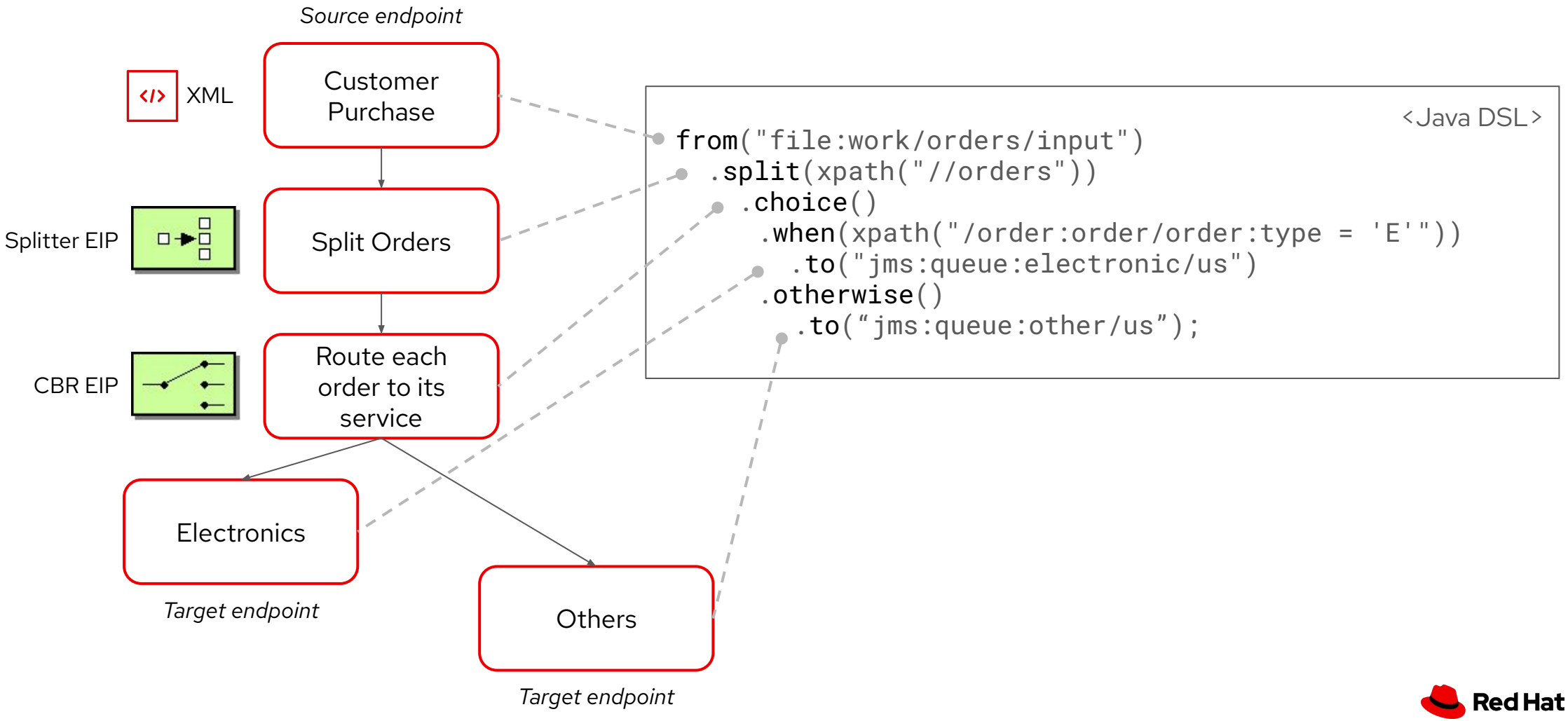
A **Camel DSL** wires endpoints and processors together to form routes

<XML DSL>

```
<camel> -- Camel context holds 1 to N routes
    <route>
        <from uri="timer:sec?period=1000"/>
        <setBody>
            <simple>Hello Camel from ${routeId}</simple>
        </setBody>
        <log message="${body}"/>
    </route>
</camel>
```


Red Hat

Example: Process and route XML orders



Red Hat

Camel Components

encapsulate APIs to enable it for routes.

Endpoint URI

```
component:resource[?options]
```

Core

```
direct:result

timer:name?period=1000

file:directoryName[?options]

bean:Class?method=myMethod
```

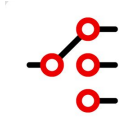
300+ extra (dynamically loaded)

```
jms:queue:order

kafka:myTopic

sql:select * from orders where…[?options]

smtp://mycompany.mailserver:30[?options]
```

Enterprise Integration Patterns

Apache Camel Implementations

Content-based Router

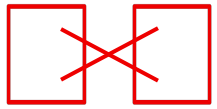


.choice()
  .when(*<expression>*)
    <steps>
  .otherwise()
    <steps>

Filter

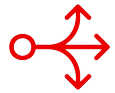
.filter(*<expression>*)
  <step(s)>
.end() -- *optional*

Message Translator



.setBody(*<expression>*)
.transform(*<expression>*)
.bean(*<customBean>*)
.to() -- *xslt, jslt, ..*

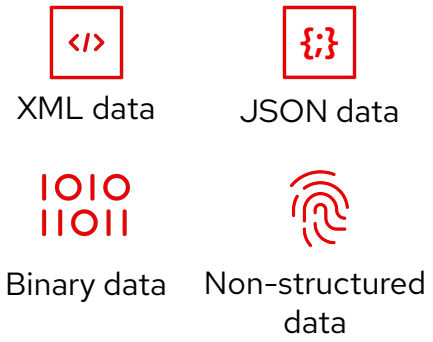
Splitter

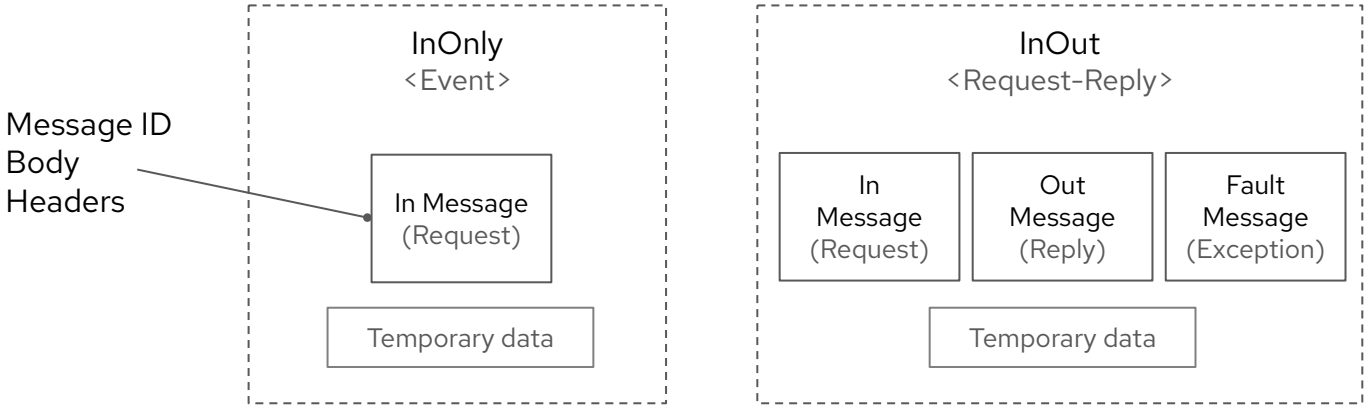


.split(*<expression>*)

And more at
https://camel.apache.org/components/4.4.x/eips/enterprise-integration-patterns.html

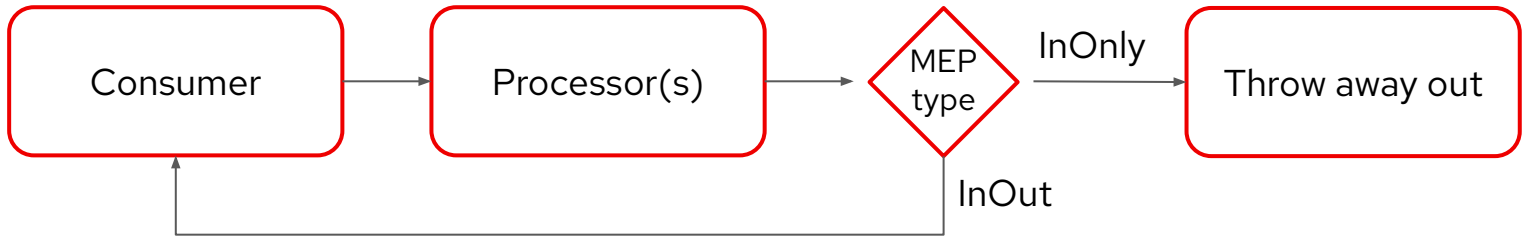
Message Exchange Pattern (MEP)





`?exchangePattern=InOnly`

`?exchangePattern=InOut`



Red Hat

Conditional processing and data extraction

with Expression languages

when routing, filtering and transforming

Simple language for basic expressions and conditions

```
.choice().when(simple("${body} contains 'Camel'"))
.filter(simple("${header.type'} == 'gold'"))
.setBody(simple("The today is ${date:now:yyyyMMdd} and it is a great day."))
```

XPath for navigating and extracting parts of XML documents

```
.choice().when(xpath("/customer/type = 'Premium'"))
.filter(xpath("/person[@name='James']"))
.setBody(xpath("/customer/name/text()"))
```

JSONPath for navigating and extracting parts of JSON structures

```
.choice().when(jsonpath("$.orderType == 'Online'"))
.filter(jsonpath("$.isActive == true"))
.setBody(jsonpath("$.customer.name"))
```

And more at https://camel.apache.org/components/4.4.x/languages/index.html

Convert message content from one format to another

Data Format Transformations

Data Type Converters

Automatic or explicit conversion of message bodies and headers from one type to another. Built-in converters significantly reduce the need for custom conversion logic

```
.convertBodyTo(String.class)
.transform(<dataType>)
```

(De)Serialization

Convert messages to and from various data formats (e.g., JSON, XML, CSV) to Java objects, facilitating the exchange of data between components that expect different formats

```
.mashal().jaxb() - XML to Pojo
.unmarshal().jaxb() - Pojo to XML
```

Advanced Transformations

Perform complex data transformations that go beyond simple format conversions or basic manipulations.

```
XML➡XML - XSLT
JSON➡JSON - JSLT
XML⬅➡JSON - XSLT|XJ
Custom processor
```


Red Hat

Ensure messages meet specified criteria

Data Format Validation

Validate EIP for basic expressions and conditions

```
.validate().simple("${body} regex '^.+@.+\\..+$'")
    .to("direct:validEmails")
    .onException(PredicateValidationException.class)
        .handled(true)
        .to("direct:invalidEmails");
```

XML Schema Definition (XSD) defines required structure and constraints of XML messages

```
.to("validator:messageSchema.xsd")
    .to("direct:valid")
    .onException(ValidationException.class)
        .handled(true)
        .to("direct:invalid");
```

JSON Schema defines required structure and constraints of the JSON messages

```
.to("json-validator:messageSchema.json")
    .to("direct:valid")
    .onException(ValidationException.class)
        .handled(true)
        .to("direct:invalid");
```

Custom processing using Beans

- Extend routing logic with custom Java code
- For complex integrations, custom transformations and business logic implementations
- Binding annotations - body, exchange, header, variable, ...
- Bind or lookup beans in Registry

```
public class MyBean {
    public String process(String body) {
        return body.toUpperCase();
    }
    public boolean isGoldCustomer(Exchange exchange) {
        // ...
    }
}

.bean(MyBean.class, "process")
.bean(OrderService.class) -- calls a method annotated with @Handler
.filter().method(MyBean.class, "isGoldCustomer")
.bean("foo") -- call a bean from Registry
```

Make applications externally configurable

and define placeholders instead of the actual values

```
db.host = 127.0.0.1
db.port = 8080
db.user = ibek
db.pwd = 12345
file.path = file.json
```

```
.to("file:foo?fileName={{file.path:/some/path}}") -- with default value

.to("file:foo?bufferSize={{?myBufferSize}}") -- optional

.log("What {{configmap:myconfig/drink}} do you want?")

.from("file:{{bean:foo.bar}}")

.to("{{env:HOME}}")

.to("{{sys:file.separator}}")
```

in **properties file, environment variables, configmaps, secrets**, etc.

Placeholder functions - `env, sys (jvm properties), bean, service`
Kubernetes placeholder functions - `configmap, secret`

Contract-first OpenAPI Integrations

Get or create
OpenAPI contract

[Simple online OpenAPI designer]

[Apicurio Studio]

Map `operationId` to
`.from("direct:operationId")`
route

Run & export integration using
camel CLI with
`--open-api api.json`
(automatically generates REST DSL)

Tips:

```
?exchangePattern=InOut
log:api?showAll=true&amp;multiline=true
${headers.name} -- get name from path or query
```

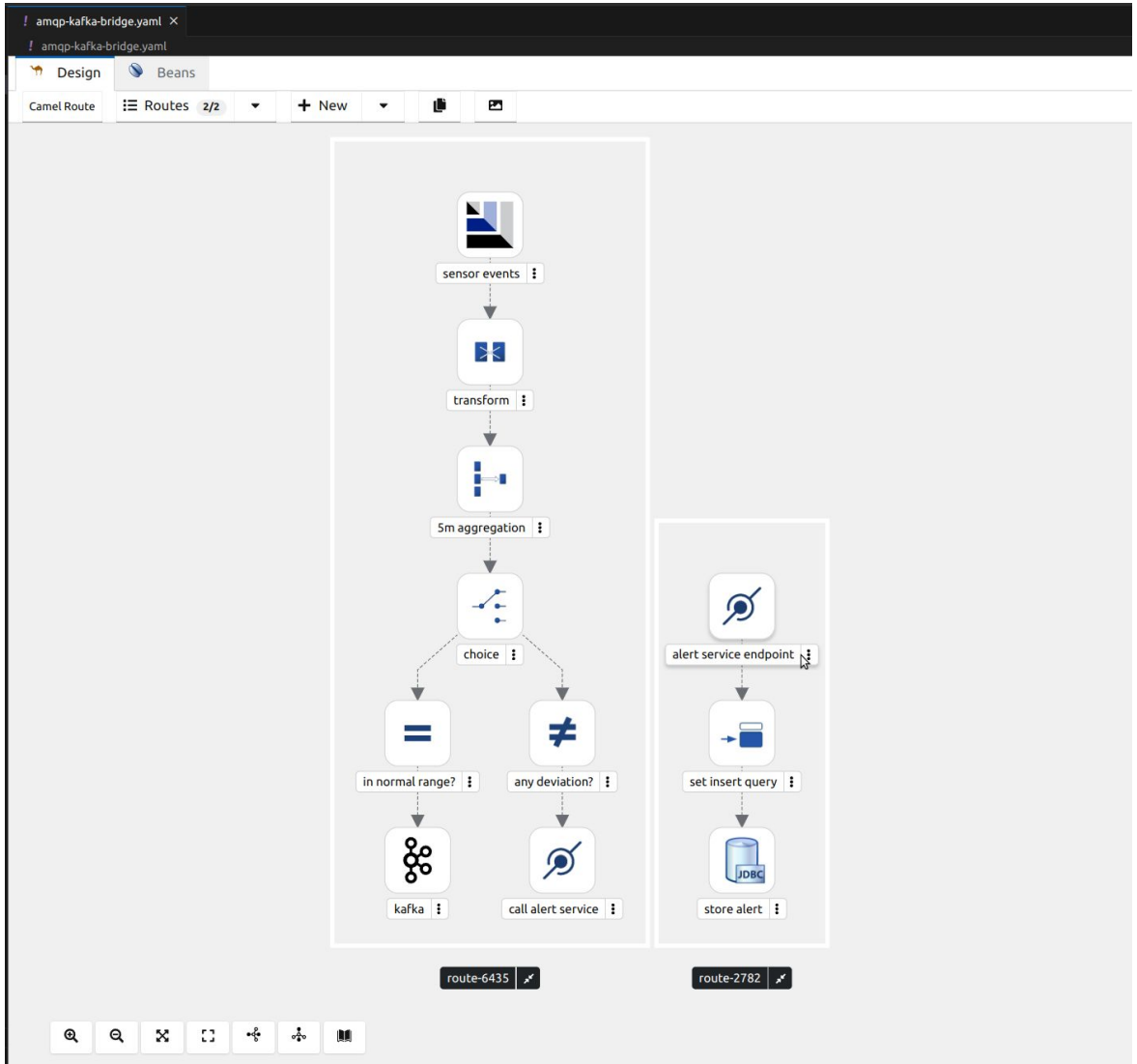
Setup of Camel development environment

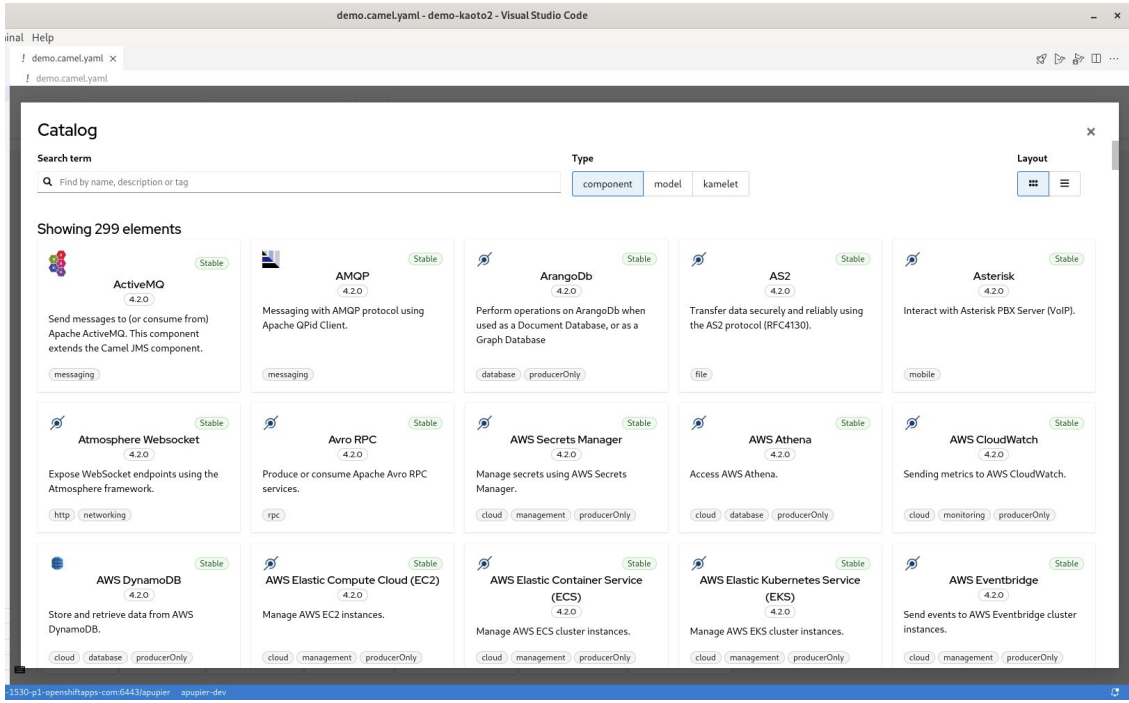
[Extension pack for Apache Camel]


```
# Linux/Mac - Install JBang

curl -Ls https://sh.jbang.dev | bash -s - app setup

# Windows - Install JBang

iex "& { $(iwr https://ps.jbang.dev) } app setup"

# Install Camel CLI

jbang app install camel@apache/camel

# Check

camel version

# Output: Camel JBang version: 4.Y.Z
```

Kaoto in Visual Studio Code

Visual Camel editor





Generates Camel YAML DSL

Red Hat

Labs today

- Developing integration routes using Apache Camel

Store the last order

- OpenAPI /order
- File
  - **`?fileName=tmp/last-order.json`**
- return **unchanged** json

```
$ camel init process-order.xml

$ camel run process-order.xml transformers/order_to_processed_order.jslt --dev --open-api apis/order.json

$ curl -H "Content-Type: application/json; charset=UTF-8" --data-binary @inputs/order.json
http://0.0.0.0:8080/order
```

Route to standard or expedited delivery

- **Content-based router EIP** (Choice)
  - Standard condition: jsonpath **`$[?(@.order_total < 100)]`**
    - Store last order to tmp/standard/last_order.json
  - Expedited: otherwise
    - Store last order to tmp/expedited/last_order.json
- return **unchanged** json

```
$ curl -H "Content-Type: application/json; charset=UTF-8"
--data-binary @inputs/order.json http://0.0.0.0:8080/order

$ curl -H "Content-Type: application/json; charset=UTF-8"
--data-binary @inputs/order_premium.json http://0.0.0.0:8080/order
```

Filter orders with unknown payment method

- **Filter EIP**
  - jsonpath **`$[?(@.payment_method !== 'Credit Card')]`**
  - direct:other-payment
  - log "Unknown payment method"
- return **unchanged** json

```
$ curl -H "Content-Type: application/json; charset=UTF-8"
--data-binary @inputs/order.json http://0.0.0.0:8080/order

$ curl -H "Content-Type: application/json; charset=UTF-8"
--data-binary @inputs/order_unknown.json http://0.0.0.0:8080/order
```

Transform order to processed order json

- JSLT Transformation
  - **`transformers/order_to_processed_order.jslt`**
- **Message Translator EIP** (Transform)
  - JQ - add new delivery property in the routes of the previous Content-based router
    - **`.delivery = "standard"`**
    - **`.delivery = "premium"`**
  - Marshal back to json **`<marshal><json /></marshal>`**
- return **transformed** json

```
$ curl -H "Content-Type: application/json; charset=UTF-8"
--data-binary @inputs/order.json http://0.0.0.0:8080/order

{
    "id":"899ac6c6-5939-4a91-8d0b-f0281e6b63ee",
    "customer":"Alex Smith",
    "items":{"Wireless Mouse":1,"Keyboard":1},
    "delivery":"standard"
}
```

The lab solution

- https://github.com/ibek/pv207-camel-lab/tree/main
- Use as inspiration for the **Integration task #2** in your projects:

  Implement at least 1 integration service based on Camel

  Make a working service call to the Camel integration service from a business process

Learn more about Camel

- https://camel.apache.org
- More integration pattern examples at:
  https://camel.apache.org/components/4.4.x/eips/enterprise-integration-patterns.html
- More components to integrate with:
  https://camel.apache.org/components/

A tip: ChatGPT works really well as an integration assistant for Apache Camel

Next lesson

- Business and domain analysis

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

facebook.com/redhatinc

youtube.com/OpenShift

twitter.com/OpenShift

Red Hat