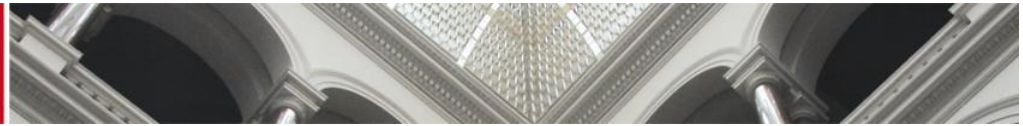
Technische Universität Berlin

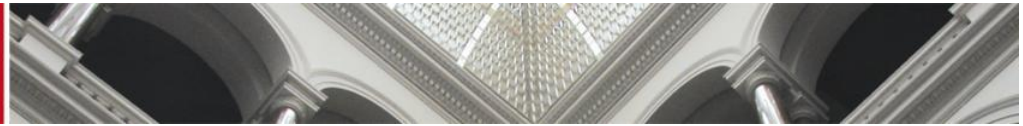
Distributed Systems

Exercise 4: EAI with Apache Camel



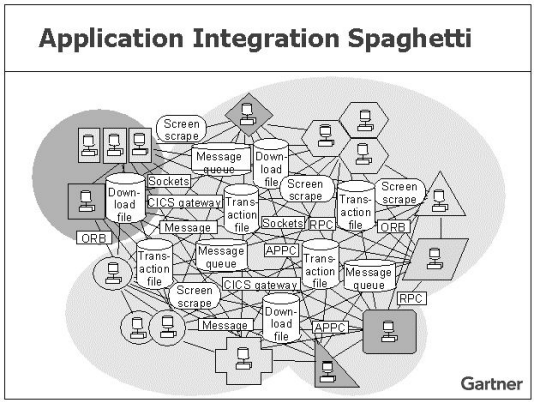
Agenda

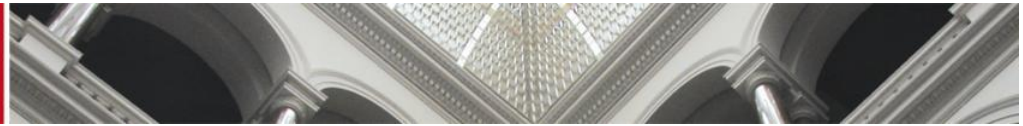
- **Motivation and EAI Patterns**
- **Apache Camel**
  - **Basics**
  - Java DSL based Integration (self-study ISIS videos, slides, provided example)
  - Integration Example (self-study ISIS videos, slides, provided example)
- **Notes regarding exercise 4**
- **Evaluation of selected groups for exercise 3**



Enterprise Application Integration (EAI) …

"… is the task of making disparate applications work together to produce a unified set of functionality" [1]





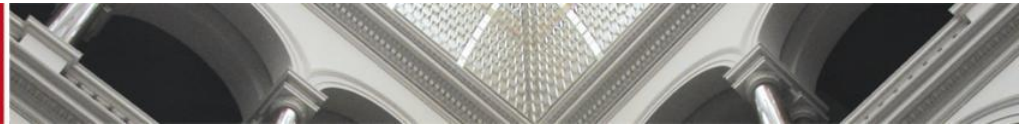
EAI - Motivation

Enterprises often comprised of several (hundreds, thousands, …) applications

- Applications are custom built, developed by third party, part of legacy systems or a combination
- Defining a clear separation between the applications is difficult -> business processes often span multiple systems (remember SOA)
  - If Alice & Bob reject their bill due to unjustified total/items, is this a customer care or a billing function?

How can this be possible? Management issue?

- Developing single applications that suit the needs of the entire business is almost impossible
- Using several applications, a business gains flexibility to choose the most suitable application
  - Instead of choosing one big solution that performs well but not optimal for each use case
- Businesses evolve, so their applications do as well
  - Existing applications are continuously adapted, new applications are introduced



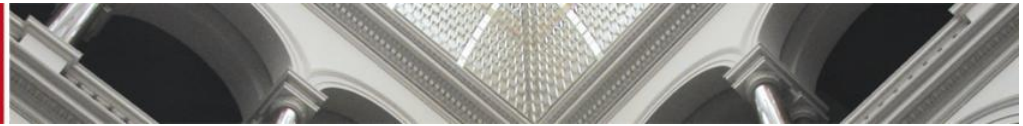
EAI - Patterns

Why do some people write better code/find better solutions than others do?

- It's all about experience:
  - If you have solved enough problems, you can compare new problems
  - You can identify "patterns" of problems and related solutions
  - Patterns are common solutions that help to cope with common type of problems
    - Patterns evolved over time by trial-and-error
  - Patterns are often related to other patterns and need to be combined to solve complex problems (Pattern language)

Design patterns are one of the most important concepts when talking about software-design:

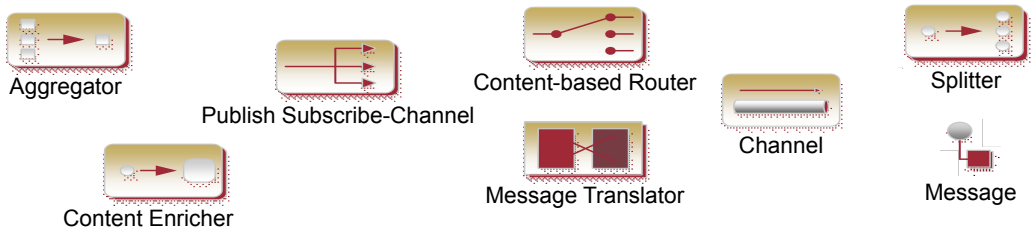
- Examples: Observer, Model View Controller, Lazy Load, Singleton, Iterator, Factory Method



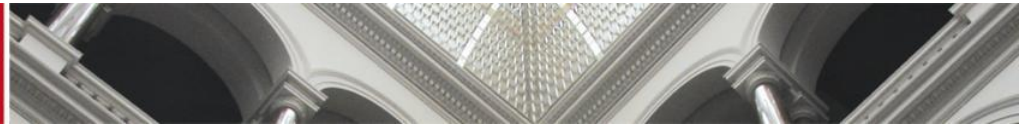
EAI - Patterns

A set of EAI patterns that help us to solve integration problems exist

- Refer to the lecture slides for a more comprehensive introduction to the topic
- Consult the book *Enterprise Integration Patterns* by Gregor Hohpe and Bobby Woolf for a detailed description of the available patterns and their relation to each other
- Today, we will introduce Apache Camel as a framework to implement integrations solutions based on the patterns

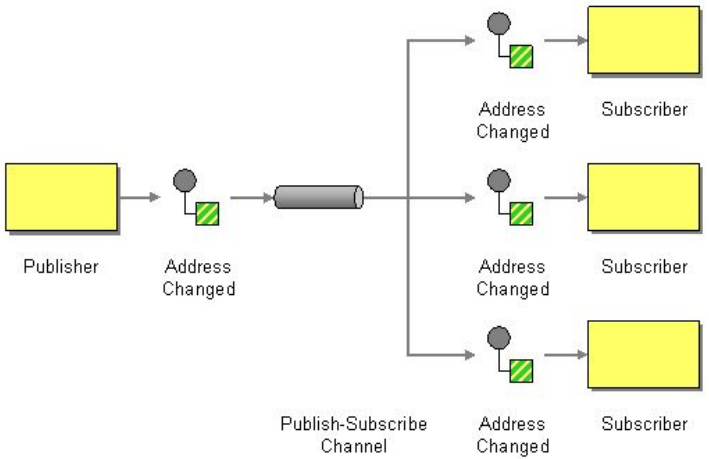


[1]: **Gregor Hohpe, Bobby Woolf:** - *Enterprise Integration Patterns;* Addison-Wesley Educational Publishers Inc (2003), ISBN13: 978-0-321-20068-6

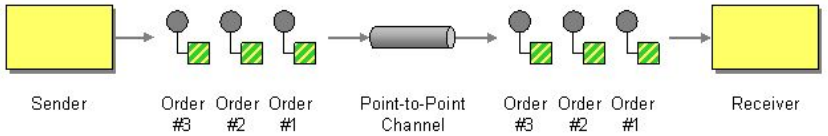


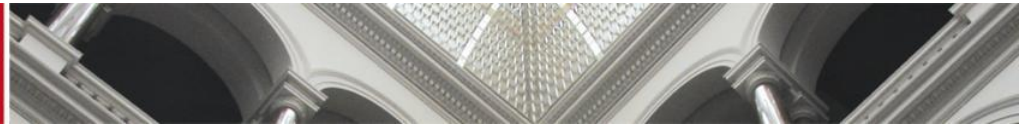
EAI - Patterns (must be used in exercise)

Publish-Subscribe Channel (Category: Messaging Channels): Send the event on a Publish-Subscribe Channel, which delivers a copy of a particular event to each receiver (as in exercise 3)



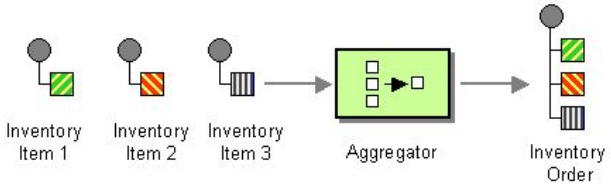
Point-to-Point Channel (Category: Messaging Channels): Send the message on a Point-to-Point Channel, which ensures that only one receiver will receive a particular message



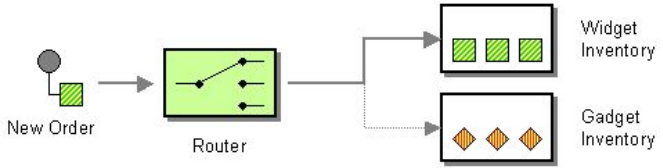


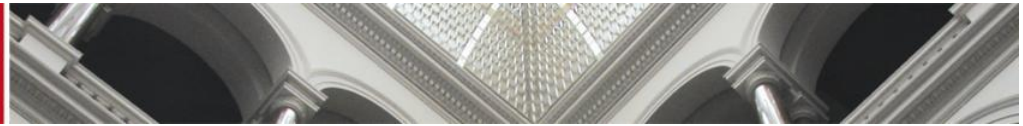
EAI - Patterns (must be used in exercise)

Aggregator (Category: Message Routing): Use a stateful filter, an Aggregator, to collect and store individual messages until a complete set of related messages has been received. Then, the Aggregator publishes a single message distilled from the individual messages



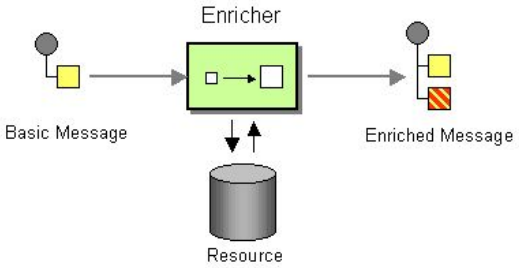
Content-Based Router (Category: Message Routing): Allows to route messages to the correct destination based on the contents of the message exchanges



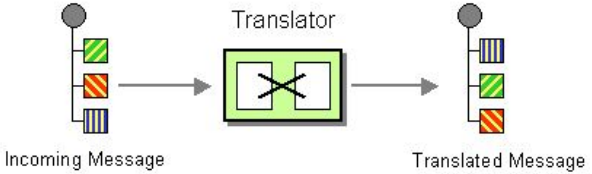


EAI - Patterns (must be used in exercise)

Content Enricher (Category: Message Transformation): Can be used to enrich individual messages with additional information that is required in downstream processing / applications

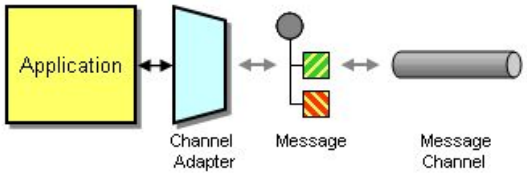


Message Translator (Category: Messaging Systems): Use a special filter, a Message Translator, between other filters or applications to translate one data format into another

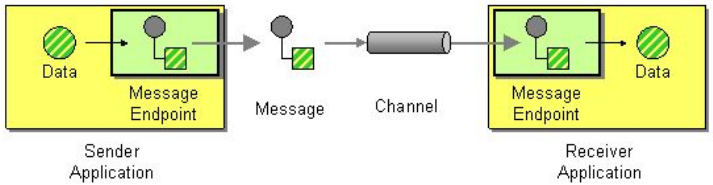


EAI - Patterns (must be used in exercise)

Channel Adapter (Category: Messaging Channels): Acts as a messaging client to the messaging system and invokes applications functions via an application-supplied interface. This way, any application can connect to the messaging system and be integrated with other applications as long as it has a proper Channel Adapter



Message Endpoint (Category: Messaging Systems): Connect an application to a messaging channel using a Message Endpoint, a client of the messaging system that the application can then use to send or receive messages



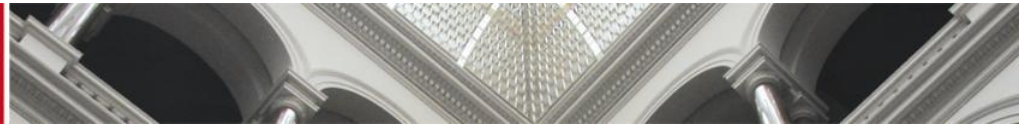
EAI – Apache Camel

Apache camel supports most of the Integration Patterns

- It allows to define routing and mediation rules based on the pattern -> thus, Apache Camel is a framework for EAI based on asynchronous messaging
- Camel is an open-source, Java-based project
  - The supported patterns are listed here

Camel enables the definition & execution of routes between endpoints

- Endpoints are sources/destinations of messages
- Routes define if and how messages are distributed
  - May involve duplication, alteration of messages



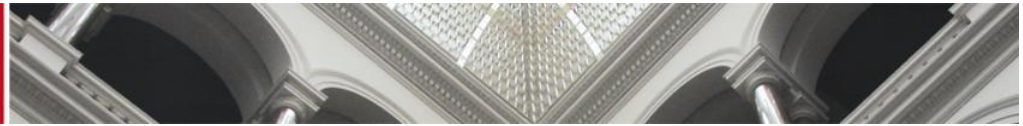
Apache Camel - Endpoints

The term “endpoint” may refer to different things depending on the context.

- e.g. machine, application, component within the application
- Address vs. application, heavyweight (machine, process) vs lightweight (object within application)

In Camel an endpoint is a possible source and/or destination for messages.

- There are different endpoint implementations for different technologies
  - JMS Queues, files, Java beans, FTP server, mail server, …
- What a message for an endpoint is generally depends on the implementation



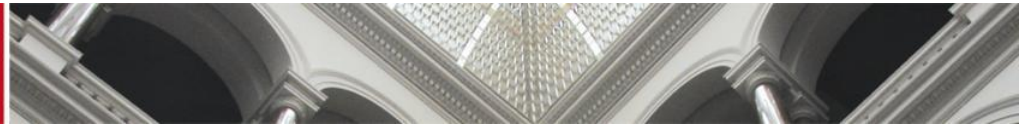
Apache Camel - URIs

Uniform Resource Identifiers (URIs) is a string identifying a physical or abstract resource.

- It can be either a unique name (e.g. ISBN) or a location (e.g. web address).
- A URI contains at least a scheme and a path
  - http://www.tu-berlin.de/ (scheme: http, path: www.tu-berlin.de/)
- Uniform Resource Locator (URL) are a special class of URIs

In Camel URLs are used to identify endpoints.

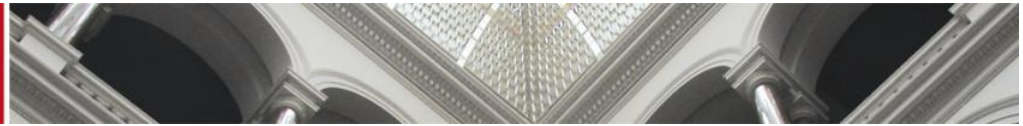
- Note, that the documentation states the use of URIs, but the implementation only allows URLs.
- The scheme is used to identify the endpoint implementation
  - e.g. jms:myqueue for a JMS queue, file:/home/user/ for access to the file system
  - See http://camel.apache.org/components.html for a list of supported technologies



Apache Camel – Components aka Endpoint Factories

In Camel each supported protocol/technology is implemented through a component.

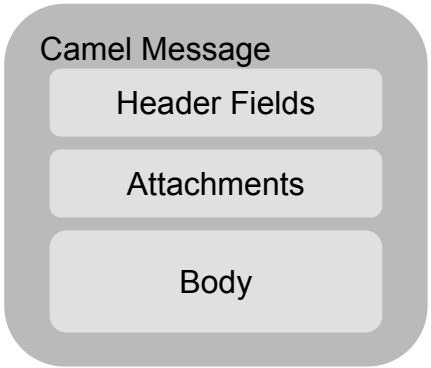
- A component is responsible for the creation of the appropriate endpoint and message instances.
- Camel comes with components for several technologies already implemented.
  - See http://camel.apache.org/components.html for a list.
- The component for an endpoint identified via an URL is selected through the scheme part.
  - “pop3” for the MailComponent, “file” for the FileComponent, etc.
  - This selection is automatic and thus it is often not necessary to explicitly deal with components.
- To add support for a new protocol/technology a custom component has to be implemented (and registered).



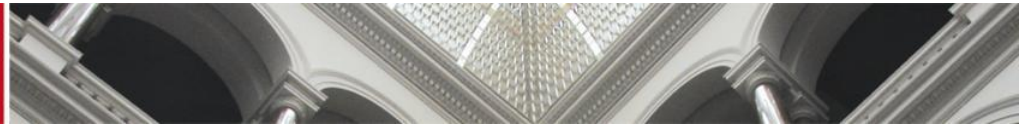
Apache Camel – Message Interface

Messages are the basic unit of communication between the individual components.

Camel assumes a simple general structure of messages and provides a corresponding interface.




```
Message msg = ...;


//Access header field "field1"
Object headerField1 = msg.getHeader("field1");


//Access message body
Object body = msg.getBody();
```



Apache Camel – Exchange Interface

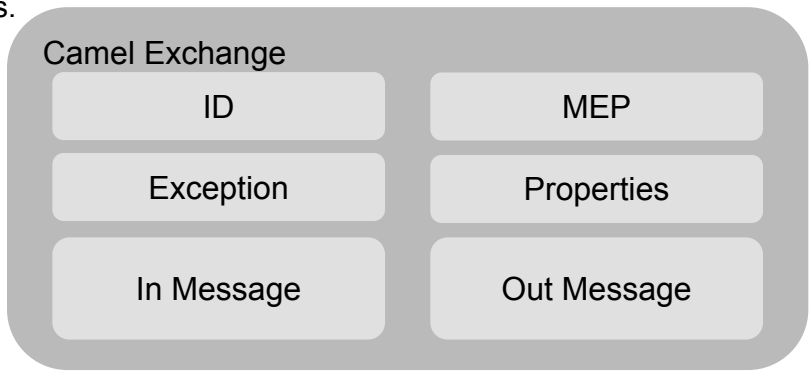
Often a message triggers a specific response.

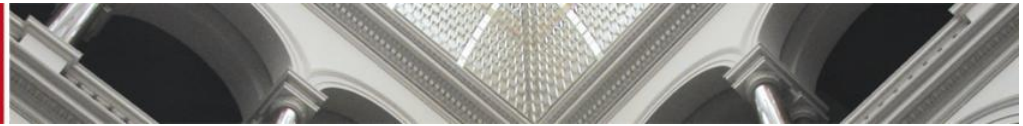
- e.g. a http request will be answered with a http response by a web server

Camel groups requests and a possible response/failure in an exchange object.

- Code handling messages usually deals with exchange objects.

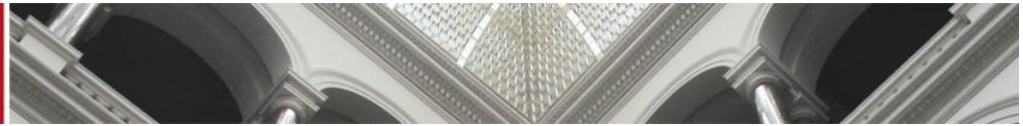
- ID – a unique id for the exchange
- MEP – Message Exchange Pattern.
  - e.g. `InOnly`, `InOut`
- Exception – exception, if any occurred during processing
- Properties – named properties of this exchange
- In Message – the request
- Out Message – the response (or failure message)





Apache Camel – Exchange Interface

```
Exchange exc = ...;

//access in message
Message inMsg = exc.getIn();

//access out message
Message outMsg = exc.getOut();

//access property "prop1"
Object prop1 = exc.getProperty("prop1");
exc.setProperty("prop1", prop1);
```



Apache Camel – Processor Interface

Camel enables the modification of messages that are passed between endpoints. This is realized through the `Processor` interface.

```
interface Processor {
  void process(Exchange exc) throws Exception;
}
```

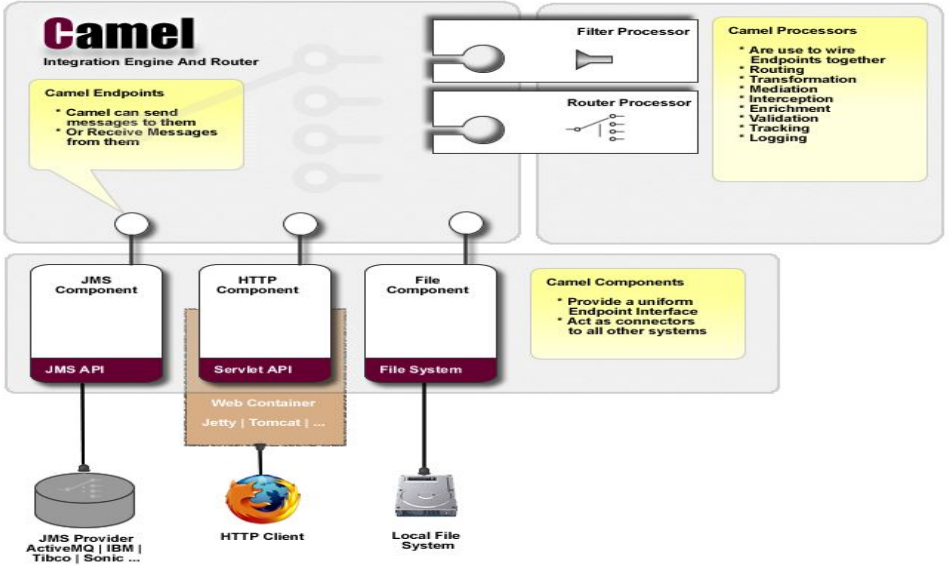
- Processors can be chained to model complex behavior.

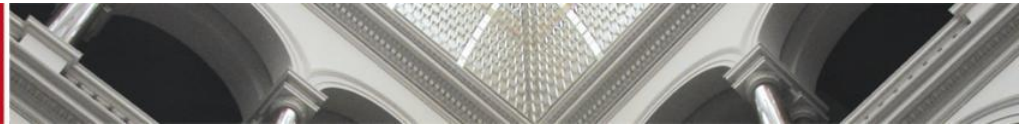
Most of the EIPs are implemented as a processor.

- e.g. `ChoiceProcessor`, `FilterProcessor`, ...
- It is still possible and sometimes necessary to write custom processors for specific behavior.

Apache Camel – Processor Interface

Before moving on to routing that implements the glue, we present the big picture





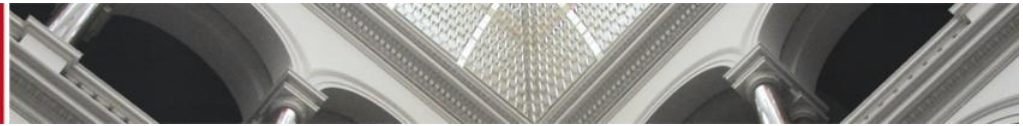
Apache Camel – Routes

Finally, the connection between endpoints and processors is given by routes.

- A route describes each step messages make from one endpoint through various processors to another endpoint.
- By using appropriate processors, such as a message router or filter, the routing may involve decision making and message altering.
- Routes can be seen as a network, connecting endpoints and processors.
- Camel can then run such a network, i.e. pass the messages along as described.

In camel routes can be describes in either XML or Java-Code.

- The library provides a easy-to-use Java-based domain specific language (DSL).

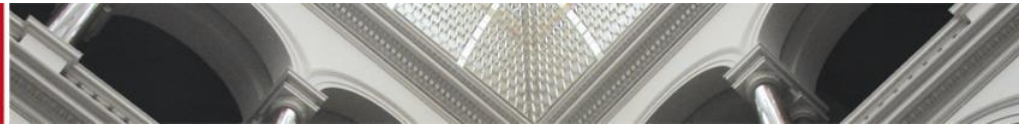


Apache Camel – Java DSL

One of the core features of camel is the Java DSL for route description.

- Routes are described through a series of Java method calls.
- Since it is still regular Java code most available tools (such as IDEs) can be used when writing the routes.
- Entrypoint to the DSL is the `RouteBuilder` class.

```
RouteBuilder route = new RouteBuilder() {
  public void configure() {
    from("file:/home/user/in/&noop=true").to("file:/home/user/out");

    from("apachemq:queue:myqueue1").choice()
      .when(header("foo").isEqualTo("bar")).to("apachemq:queue:myqueue2")
      .when(header("foo").isEqualTo("bar")).to("apachemq:queue:myqueue3")
      .otherwise().to("apachemq:queue:myqueue4");
  }
}
```

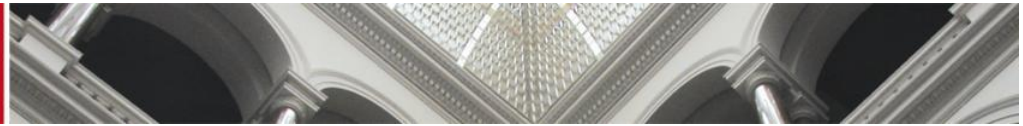


Apache Camel - CamelContext

The Camel runtime system is instantiated and controlled through the `CamelContext` class.

- The runtime manages threads for all endpoints and processors.
- It is also responsible for associating components with URI schemes.
- For typical applications the following steps are required:
  1. Create a CamelContext object.
  2. Register (additional) endpoints/components.
  3. Add routes connecting the endpoints.
  4. Call the `start()` method of the context.
  5. Before the application closes, call the `stop()` method of the context.

```
DefaultCamelContext camelContext = new DefaultCamelContext();

camelContext.addRoutes(route);

camelContext.start();
//wait for application to end
camelContext.stop();
```



Apache Camel - CamelTemplate

In some cases (e.g. testing) it can be useful to manually inject messages into endpoints.

- The injected message is then handled like any other message and passed along the network.
- To send messages a `ProducerTemplate` object must be used.

```
ProducerTemplate ptpl = camelContext.createProducerTemplate();
Object obj = ...;
ptpl.sendBody("direct:objectlink", obj);
```

It is also possible to receive messages from endpoints.

- The corresponding class is `ConsumerTemplate`.

```
ConsumerTemplate ctpl = camelContext.createConsumerTemplate();
Object obj = ctpl.receiveBody("direct:objectlink");
```

Apache Camel - Example

As an example consider a vote counting application.

- Binary votes need to be verified using an established system and finally counted.
- The votes are given in several text files, containing unique ids for the voters and their vote.

```
1028388-yes
1020300-no
1020303-yes
...
```

```
6239922-no
7372999-no
3848288-yes
...
```

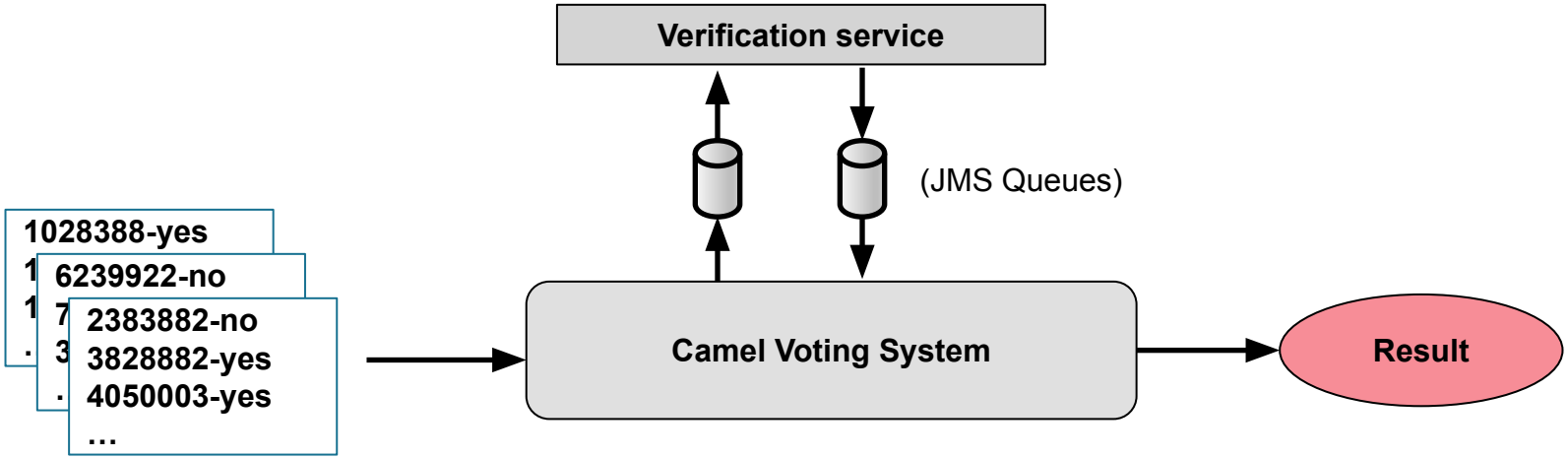
```
2383882-no
3828882-yes
4050003-yes
...
```

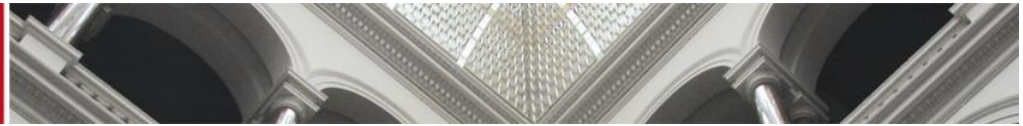
- Within the system, votes must be represented using a simple data type.

```
class Vote implements Serializable {
  private String voterId;
  private Boolean vote;

  ...
}
```

Apache Camel - Example

- Additionally, there is a service, which can be used to verify votes. The service uses JMS queues for communication. The verification result is given as a message property.

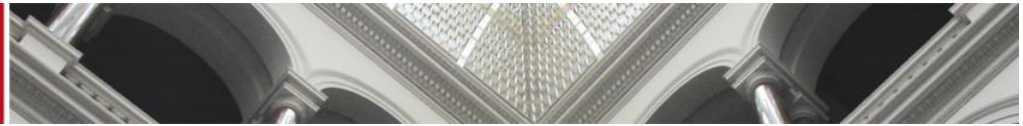




Apache Camel - Example

- Additionally, there is a service, which can be used to verify votes. The service uses JMS queues for communication. The verification result is given as a message property.

```
...
MessageConsumer consumer = session.createConsumer(inQueue); //validationIn
consumer.setMessageListener(new MessageListener() {
  public void onMessage(Message message) {
      Vote vote = (Vote)((ObjectMessage)message).getObject();

      ObjectMessage answer = session.createObjectMessage(vote);
      boolean validated = Math.random() > 0.5;
      answer.setBooleanProperty("validated", validated);

      MessageProducer producer = session.createProducer(outQueue); //validationOut
      producer.send(answer);
  }
});
...
```



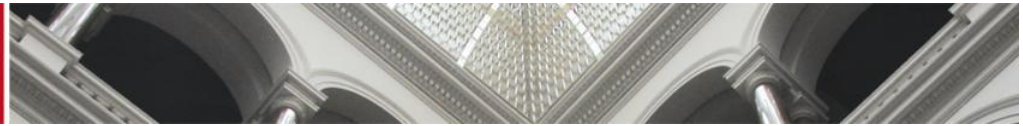
Apache Camel - Example

Camel can now be used to connect the different endpoints (voting files, verification service queues) and perform additional transformations and routing decisions.

- First the obligatory setup.

```
DefaultCamelContext ctxt = new DefaultCamelContext();

ActiveMQComponent activeMQComponent =
    ActiveMQComponent.activeMQComponent("tcp://localhost:61616");
activeMQComponent.setTrustAllPackages(true);

ctxt.addComponent("activemq", activeMQComponent);
```

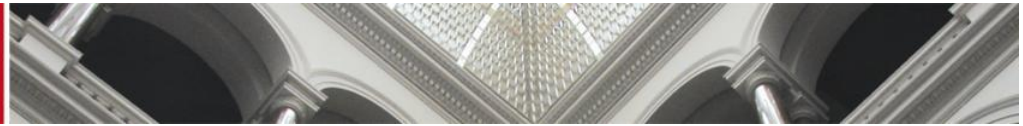
- Note, that the component responsible for connecting ActiveMQ to Camel is explicitly instantiated and loaded into the `CamelContext`.



Apache Camel - Example

To describe the messaging logic, new routes have to be defined.

```
RouteBuilder route = new RouteBuilder() {
  public void configure() throws Exception {


  }
};
```

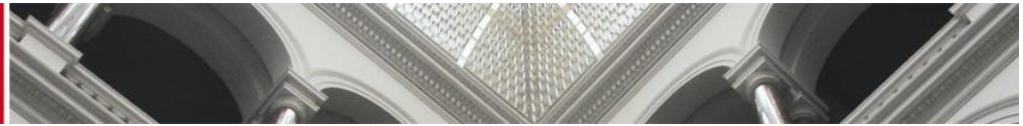


Apache Camel - Example

To describe the messaging logic, new routes have to be defined.

```
RouteBuilder route = new RouteBuilder() {
  public void configure() throws Exception {
    from("file:votes?noop=true")




  }
};
```

A `file` endpoint, which emits all files in the directory *votes* as messages.



Apache Camel - Example

To describe the messaging logic, new routes have to be defined.

```
RouteBuilder route = new RouteBuilder() {
  public void configure() throws Exception {
    from("file:votes?noop=true")
      .split(body().tokenize("\n"))




  }
};
```

Split messages and let each line be a seperate new message.

Apache Camel - Example

To describe the messaging logic, new routes have to be defined.

```
RouteBuilder route = new RouteBuilder() {
  public void configure() throws Exception {
    from("file:votes?noop=true")
      .split(body().tokenize("\n"))
      .process(voteFactory)
  }
};
```

```
Processor voteFactory = new Processor() {
  public void process(Exchange exchange) throws Exception {
    String[] parts = exchange.getIn().getBody($string.class).split("\\-");
    String voterId = parts[0];
    Boolean vote = parts[1].equalsIgnoreCase("yes");
    exchange.getIn().setBody(new Vote(voterId, vote));
  }
};
```

Use a custom `Processor` to transform each line into a `Vote`.

Apache Camel - Example

To describe the messaging logic, new routes have to be defined.

```
RouteBuilder route = new RouteBuilder() {
  public void configure() throws Exception {
    from("file:votes?noop=true")
      .split(body().tokenize("\n"))
      .process(voteFactory)
      .to("activemq:queue:validationIn");




  }
};
```

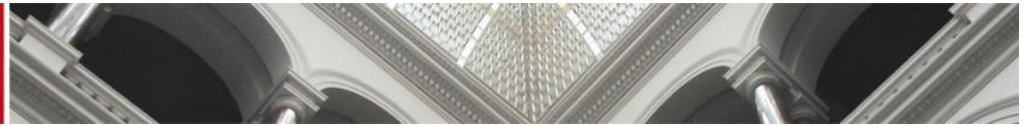
Send votes to the JMS queue named `validationIn` (and thus to the validation service)

Apache Camel - Example

To describe the messaging logic, new routes have to be defined.

```
RouteBuilder route = new RouteBuilder() {
  public void configure() throws Exception {
    from("file:votes?noop=true")
      .split(body().tokenize("\n"))
      .process(voteFactory)
      .to("activemq:queue:validationIn");


    from("activemq:queue:validationOut")


  }
};
```

Read votes from the JMS queue named `validationOut` (from the validation service)



Apache Camel - Example

To describe the messaging logic, new routes have to be defined.

```
RouteBuilder route = new RouteBuilder() {
  public void configure() throws Exception {
    from("file:votes?noop=true")
      .split(body().tokenize("\n"))
      .process(voteFactory)
      .to("activemq:queue:validationIn");



    from("activemq:queue:validationOut")
      .choice()
        .when(header("validated"))

  }
};
```

Look in header to see if vote has been confirmed.

Apache Camel - Example

To describe the messaging logic, new routes have to be defined.

```
RouteBuilder route = new RouteBuilder() {
  public void configure() throws Exception {
    from("file:votes?noop=true")
      .split(body().tokenize("\n"))
      .process(voteFactory)
      .to("activemq:queue:validationIn");

    VoteFilter voteFilter = new VoteFilter();

    from("activemq:queue:validationOut")
      .choice()
        .when(header("validated"))
          .filter(method(voteFilter, "isYesVote"))
  }
};
```

```
class VoteFilter {
  public boolean isYesVote(Vote vote) {
    return vote.getVote();
  }
}
```

Use custom filter method to only get yes-votes.

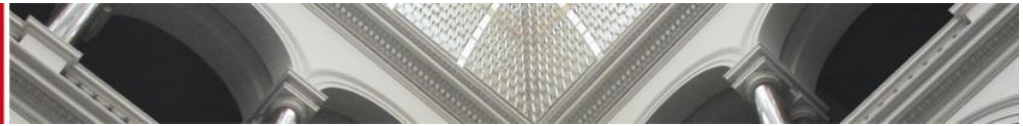
Apache Camel - Example

To describe the messaging logic, new routes have to be defined.

```
RouteBuilder route = new Routel
  public void configure() thro
    from("file:votes?noop=true
      .split(body().tokenize("
      .process(voteFactory)
      .to("activemq:queue:vali

    VoteFilter voteFilter = new

    from("activemq:queue:validationOut")
      .choice()
        .when(header("validated"))
          .filter(method(voteFilter, "isYesVote"))
          .aggregate(constant(0),new CountingAggregation()).completionInterval(5)
  }
};
```

```
class CountingAggregation implements AggregationStrategy {
  private int count = 0;

  public Exchange aggregate(Exchange ex1, Exchange ex2) {
    count++;
    ex2.getIn().setBody(count);
    return ex2;
  }
}
```

Use a custom `AggregationStrategy` to count the remaining votes and produce a result every 5 ms.

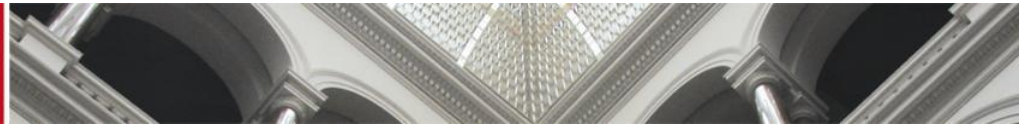


Apache Camel - Example

To describe the messaging logic, new routes have to be defined.

```
RouteBuilder route = new RouteBuilder() {
  public void configure() throws  Exception {
    from("file:votes?noop=true")
      .split(body().tokenize("\n"))
      .process(voteFactory)
      .to("activemq:queue:validationIn");

    VoteFilter voteFilter =  new VoteFilter();

    from("activemq:queue:validationOut")
      .choice()
        .when(header("validated"))
          .filter(method(voteFilter, "isYesVote"))
          .aggregate(constant(0),  new CountingAggregation()).completionInterval(5)
          .to("stream:out")
          .end()
  }
};
```

Print the resulting numbers to the standard output and end the conditional block.

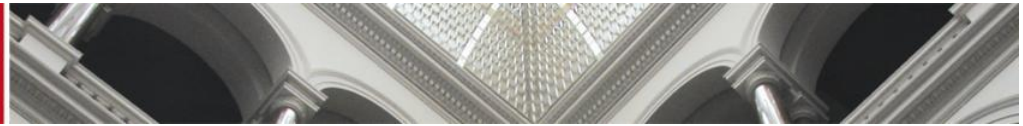


Apache Camel - Example

To describe the messaging logic, new routes have to be defined.

```
RouteBuilder route = new RouteBuilder() {
  public void configure() throws  Exception {
    from("file:votes?noop=true")
      .split(body().tokenize("\n"))
      .process(voteFactory)
      .to("activemq:queue:validationIn");

    VoteFilter voteFilter =  new VoteFilter();

    from("activemq:queue:validationOut")
      .choice()
        .when(header("validated"))
          .filter(method(voteFilter, "isYesVote"))
          .aggregate(constant(0),  new CountingAggregation()).completionInterval(5)
          .to("stream:out")
          .end()
        .endChoice().otherwise()
          .to("stream:err");
  }
};
```

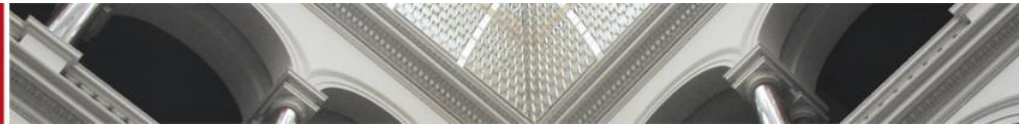
Continue the choice definition. If the vote could not be verfified print it to the error output.



Apache Camel - Example

To describe the messaging logic, new routes have to be defined.

```
RouteBuilder route = new RouteBuilder() {
  public void configure() throws  Exception {
    from("file:votes?noop=true")
      .split(body().tokenize("\n"))
      .process(voteFactory)
      .to("activemq:queue:validationIn");

    VoteFilter voteFilter =  new VoteFilter();

    from("activemq:queue:validationOut")
      .choice()
        .when(header("validated"))
          .filter(method(voteFilter, "isYesVote"))
          .aggregate(constant(0),  new CountingAggregation()).completionInterval(5)
          .to("stream:out")
          .end()
        .endChoice().otherwise()
          .to("stream:err");
  }
};
```



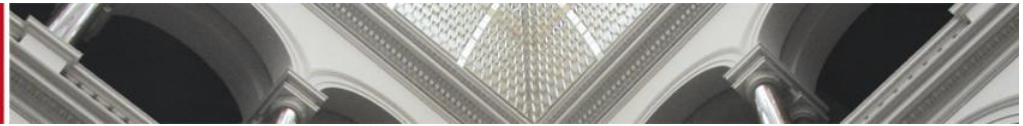
Apache Camel - Example

With the route definition added the Camel runtime system can be signaled to start execution.

```
ctxt.addRoutes(route);

ctxt.start();
System.in.read();
ctxt.stop();
```

To run the example, several libraries have to be added to the classpath during compilation and execution.

- A minimal set of required jar files is provided on the ISIS platform. These need to be added to the classpath.
- The process can be simplified through the use of a build management system like Maven.



Technische Universität Berlin

Notes regarding exercise 4

Focus of the exercise are the EIPs and their application.

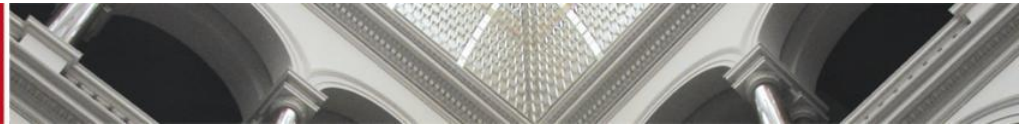
- It is necessary to understand the involved patterns before using them in code.

The overall idea is to connect otherwise independent services, which should then exchange messages.

- Any solution must include such services as independent applications.
- Services/applications then use indirect communication to form a complete system.
  - This can for example be realized using JMS via ActiveMQ.

All requested patterns can be easily implemented/used in Camel.

- See http://camel.apache.org/enterprise-integration-patterns.html for reference.



Evaluation of exercise 3

First: Group 5 and 10 (in parallel)

After ~10-15 minutes: Group 17 and 20 (in parallel)

Again, after ~10-15 minutes: Group 24 and Student 466797 (in parallel)

If you are part of a waiting group, please wait outside until you are asked to enter the room