Enterprise Application Integration mit Apache Camel

JUG Augsburg, 17.11.2011

Andreas Prüller

Agenda

- Enterprise Application Integration
  - Definition
  - Enterprise Integration Patterns
  - Beispiel
- Apache Camel
  - Übersicht
  - DSL
  - Komponenten
  - Datentransformation
  - Errorhandling
  - Testing

http://www.tamilchrist.ch/gallery/displayimage.php?album=1&pos=1

Enterprise Application Integration

Enterprise Application Integration (EAI) is defined as the use of software and computer systems architectural principles to integrate a set of enterprise computer applications.

(http://en.wikipedia.org)

"EAI encompasses approaches, methodologies, standards, and technologies allowing very diverse but important systems to share information, processes, and behavior in support of the core business."

(David S. Linthicum)

Enterprise Integration Patterns

http://www.nlarchitects.nl/media/uploads/virtual_realities/thumbnails/Minimum_speed_200_kmh__jpg_1200x750_q85.jpg

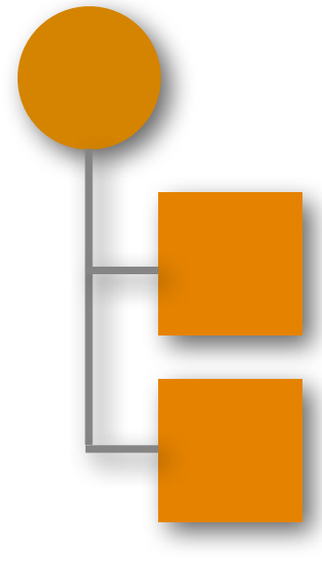
Enterprise Integration Patterns

- Message

  Nachricht, die über einen Channel verschickt werden kann. Besteht aus *Header* und *Body*.

  Es gibt verschiedene Typen von Messages:

  - Header
    Metadaten und Routinginformationen. Wird vom Messaging-System ausgewertet.
  - Body
    Payload, wird in der Regel vom Messaging-System ignoriert und einfach übertragen.



Enterprise Integration Patterns

- Message Exchange Pattern (MEP)

  Definiert verschiedene Typen von Interaktionen zwischen den Systemen.

  Unter anderem wird der Typ der Message-verbindung festgelegt:

  - One-way message (event message)
  - Request-Response

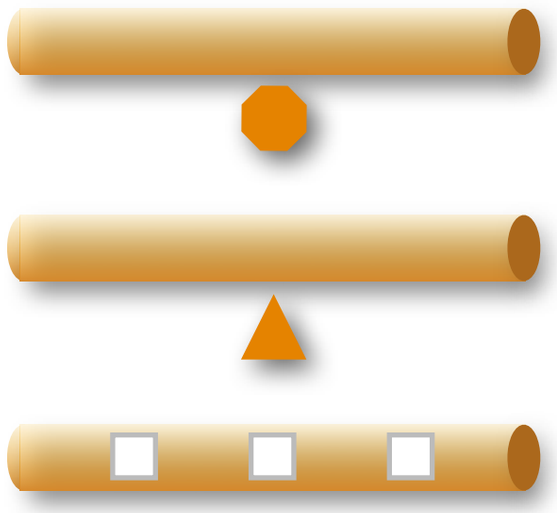
Enterprise Integration Patterns

- Message Channel

Messages werden über Message Channels übertragen. Der Message Channel (oder auch mehrere) werden vom Messagingsystem bereit gestellt.

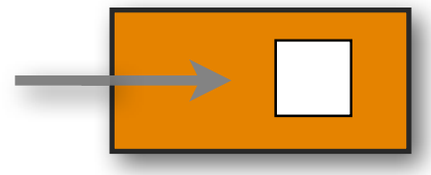
Spezialisierung:

- Dead Letter Channel
- Invalid Message Channel
- Datatype Channel



Enterprise Integration Patterns

- Message Endpoint



Schnittstelle für Systeme zu einem Message Channel um Nachrichten senden und empfangen zu können.

Der Message Endpoint kapselt das Messaging-System vom Rest der Applikation. Hier wird die abstrakte Messaging API auf den speziellen Anwendungfall hin implementiert.

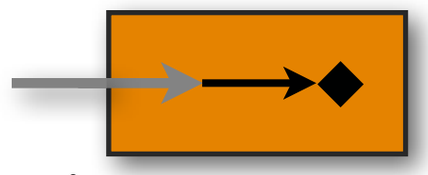
Bei Apache Camel sind in diesem Zusammenhang die Begriffe *Consumer* und *Producer* essentiell.

Enterprise Integration Patterns

- Consumer

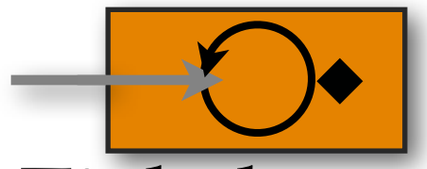
  Consumer empfangen die Nachrichten. Dies kann auf verschiedene Arten geschehen:

  - Event-Driven Consumer

    Wartet auf ein Event und tritt dann in Aktion; er ist damit ein asynchroner Empfänger.

  - Polling Consumer

    Fragt in bestimmten Zeitintervallen sein Ziel ab, ob etwas Neues bereit liegt. Damit ist er ein synchroner Empfänger.

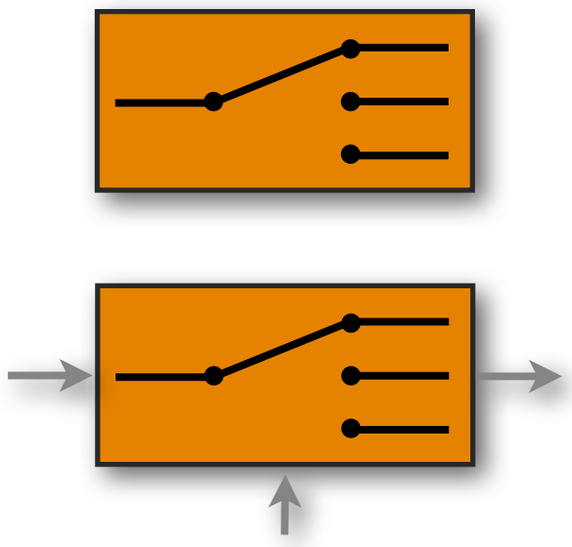
Enterprise Integration Patterns

- Message Router

  Verteilt Messages an Hand von bestimmten Regeln auf verschiedene Channels. Der Inhalt der Message wird dabei nicht verändert.

Typen von Routern:

- Content-Based Router
- Context-Based Router
- Dynamic Router



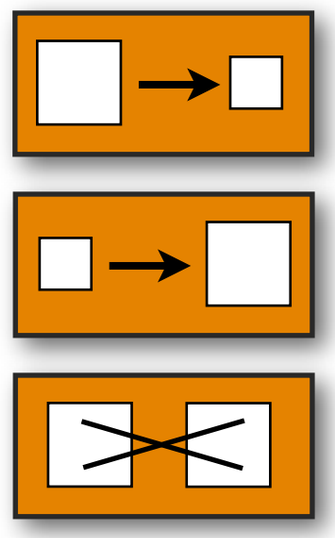
Enterprise Integration Patterns

- Processor

  Manipuliert bzw. bearbeitet eine Message. Dies kann auch bedeuten dass Inhalte von anderen Quellen hinzuaggregiert oder Information unwiederbringlich gelöscht werden.

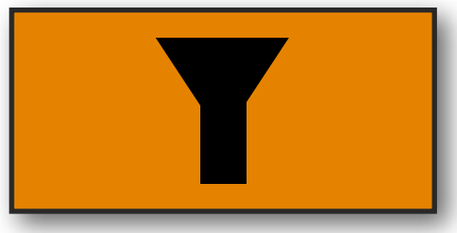
Standardprozessoren sind beispielsweise:

- Content Filter
- Content Enricher
- Message Translator



Enterprise Integration Patterns

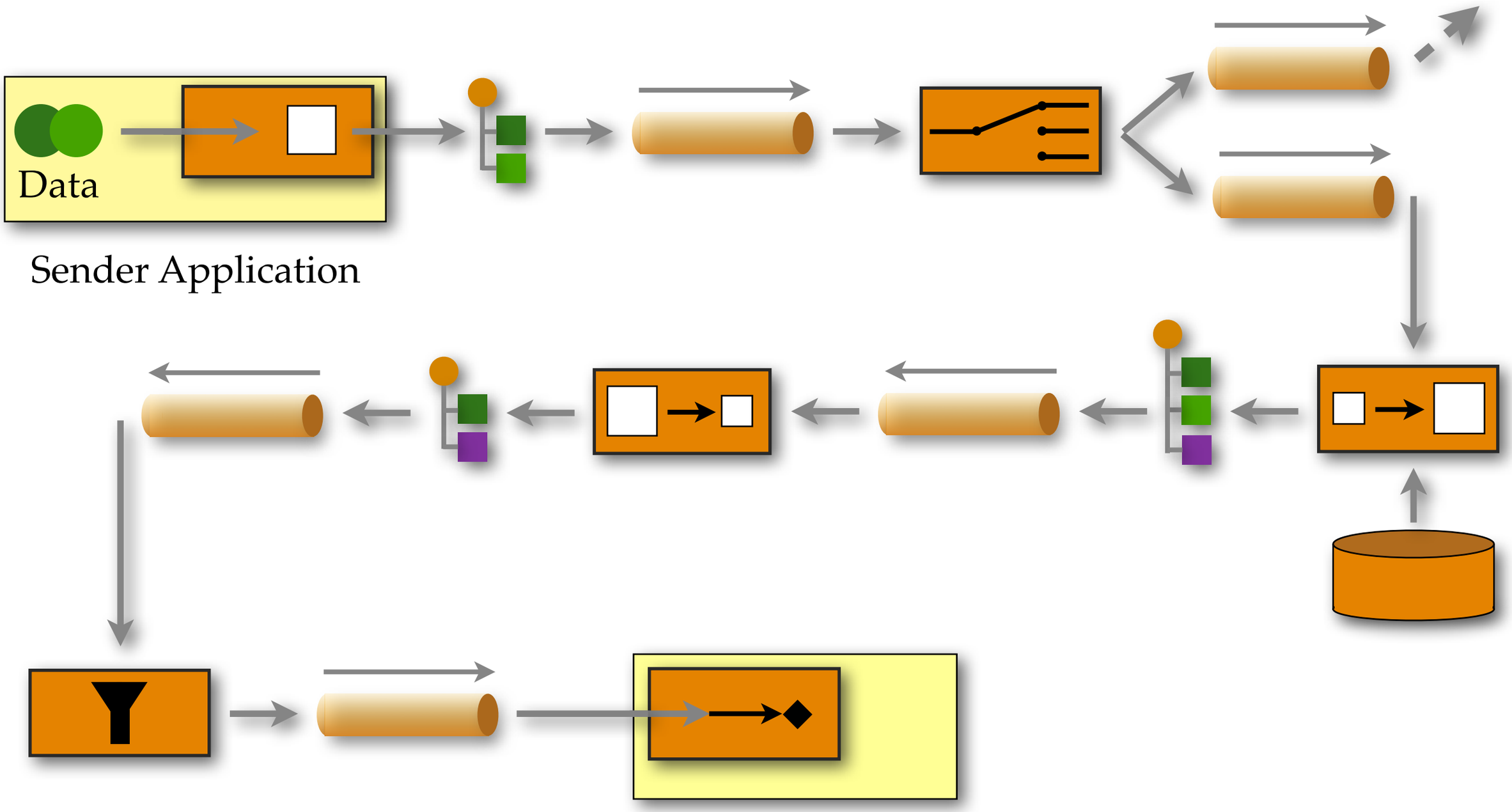
- Message Filter



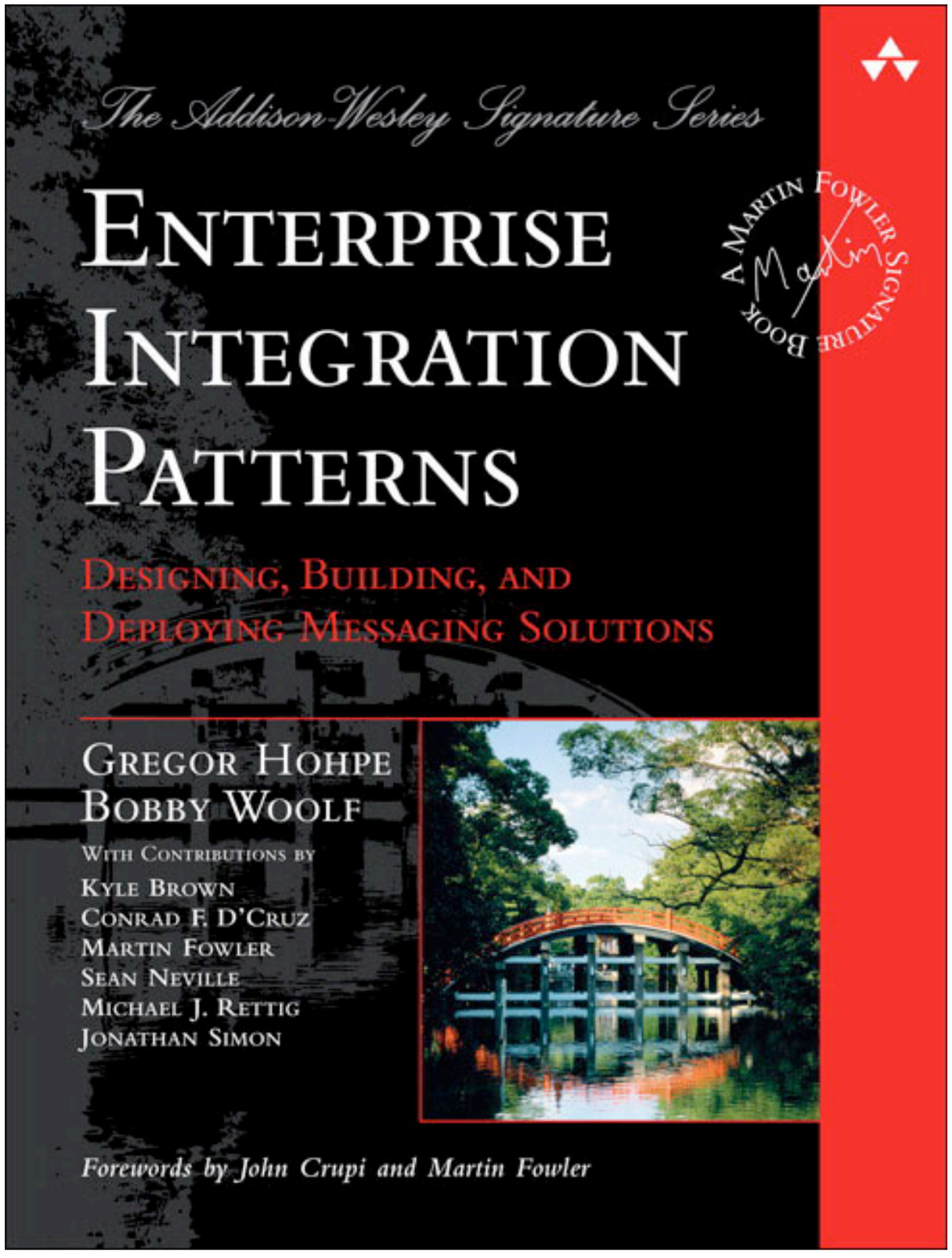
Den Message Filter kann man sich wie eine Spezialisierung des Content-Based Routers vorstellen, der ungewünschte Messages einfach verwirft (oder um beim Bild des Routers zu bleiben, auf den Null-Channel routet).

Message Filter stellen eine einfache Lösung dar um das Anlegen von sehr vielen Channels zu verhindern. Es wird jedoch nur die Last auf die verarbeitenden Komponenten gedrosselt, nicht so sehr auf dem Messagingsystem selbst.

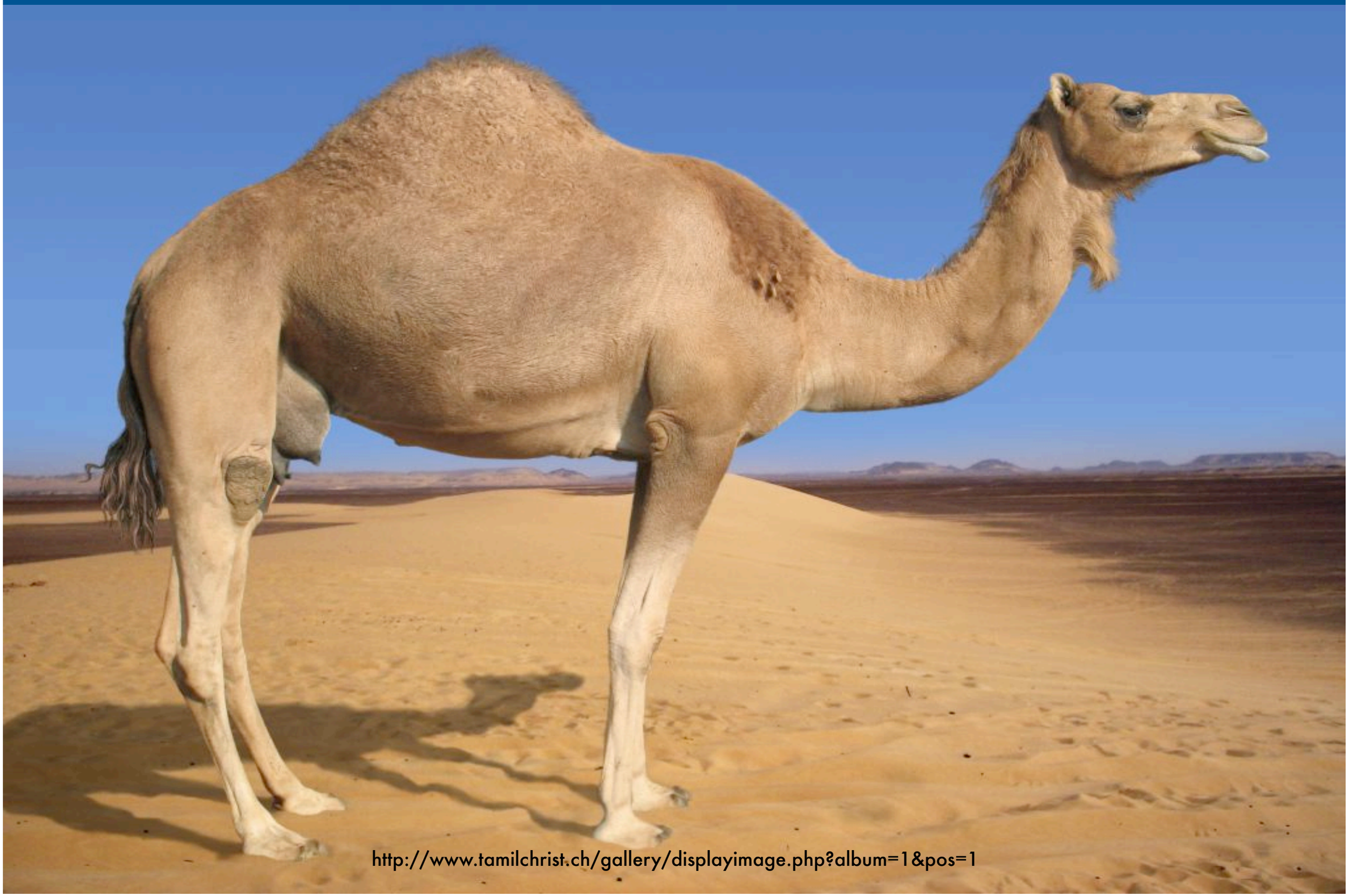
Enterprise Integration Patterns - Beispiel



Enterprise Integration Patterns



Apache Camel



http://www.tamilchrist.ch/gallery/displayimage.php?album=1&pos=1

Apache Camel

- Was ist Apache Camel?
  - Open Source
  - Integration Framework
  - Routing-Engine Builder

- Was ist Apache Camel nicht?
  - kein Enterprise Service Bus (ESB)

- Wo bekomme ich Apache Camel?
  - http://camel.apache.org

Apache Camel

- Was ist der Kern von Apache Camel?
  - Modulare und erweiterbare Architektur
  - POJO model
  - Riesige Komponentenbibliothek
  - Enterprise Integration Patterns
  - Domain Specific Language (DSL)
    Wobei nicht nur eine DSL angeboten wird, sondern diverse, wie z.B. Java, Spring oder Scala.

Apache Camel - Producer

- Konzepte in Camel: Producer

  Ein Producer ist eine Abstraktion in Camel. Er dient dazu Message Endpoints mit zu ihnen kompatiblen Inhalten zu versorgen. Damit kümmert sich der Producer um alle Konvertierungslogiken des End-points. Producer werden vom Endpoint erzeugt.

  - FileProducer: schreibt den Body einer Nachricht in eine Datei
  - JmsProducer: konvertiert die Camel-Message in eine Jms-Message und versendet diese

Apache Camel - Exchange

- Konzepte in Camel: Exchange

  Container für eine Message auf dem Channel und enthält zusätzlich MEP-Informationen.

  - Exchange ID
  - MEP
  - Exception
  - Properties (ähnlich dem Message Header, aber während der gesamten Exchange-Lebensdauer gültig)
  - In Message
  - Out Message (optional)

Apache Camel - Komponenten

- über 60 unterstützte Standardkomponenten, u.a.:
  - ActiveMQ
  - Atom
  - Apache CXF
  - DNS
  - EJB 3.0
  - File
  - FTP
  - Google App Engine
  - Hibernate
  - HTTP
  - Jetty
  - JMX
  - LDAP
  - Lucene
  - Nagios
  - Quartz
  - Restlet
  - RSS
  - SMTP
  - TCP / UDP
  - XMPP
  - XSLT

Apache Camel - DSL

- Apache Camel und DSLs - Beispiele:
  - Java:

```
from("file:data/inbox").to("jms:queue:order")
```

  - Spring DSL

```
<route>
  <from uri="file:data/inbox"/>
  <to uri="jms:queue:order"/>
</route>
```

  - Scala DSL

```
from "file:data/inbox" -> "jms:queue:order"
```

Apache Camel - Komponenten

- Einfaches Beispiel
  - alle Dateien aus `.../src/data/inbox` lesen
  - die Dateien nach `.../src/data/outbox` schreiben

  Mit Camel muss man keine einzige Zeile Code schreiben (wenn man will), es reichen etwa 30 Zeilen Spring-Konfiguration.

  siehe: Example 1

Apache Camel - Komponenten

- erweitern wir das Beispiel:
  - alle Dateien aus `.../src/data/inbox` lesen
  - die Datei nach `.../src/data/outbox/xml` schreiben, falls es eine XML-Datei ist
  - die Datei nach `.../src/data/outbox/csv` schreiben, falls es eine CSV-Datei ist

Lösung für das Problem: der MessageFilter

siehe: Example 2 (Spring)

siehe: Example 3 (POJO)

Apache Camel - Datentransformation

- Unterstützte Datenformate

Bindy	CSV, FIX, fixed length
Castor	XML
Crypto	Any
CSV	CSV
Flatpack	CSV
GZip	Any
HL7	HL7
JAXB	XML
Jackson	JSON
Protobuf	XML
SOAP	XML
Serialization	Object
TidyMarkup	HTML
XmlBeans	XML
XMLSecurity	XML
XStream	XML, JSON
Zip	Any

Apache Camel - Datentransformation

- Datentransformation mit XSLT
  - XSLT ist „nur“ eine Komponente, damit einfache Integration
  - Beispiel:

```
<route>
  <from uri="file://data/inbox"/>
  <to uri="xslt://resources/transform.xslt"/>
  <to uri="activemq:queue:transformed/>
</route>
```

  - XSL-Stylesheets können aus dem Classpath, vom Dateisystem oder von einer URL geladen werden.

Apache Camel - Datentransformation

- Datentransformation mit JAXB
  - Definition der Entität etwa über Annotationen
  - Bespiel einer Route:

```
<dataFormats>
  <jaxb id="jaxb" contextPath="demo"/>
</dataFormats>
<route>
  <from uri="file://data/inbox"/>
  <marshal ref="jaxb"/>
  <to uri="activemq:queue:order"/>
</route>
```

Apache Camel - Errorhandling

- Unterscheidung in zwei Fehlertypen:
  - recoverable: der Fehler löst sich eventuell schnell von selbst (z. B. kurzzeitig nicht vorhandene Netzwerkverbindung)
  - irrecoverable: der Fehler wird auch nach diversen Versuchen bestehen bleiben (z. B. Parsingfehler)
- kann während des Lebenszyklus eines Exchanges angewandt werden
- kann context-scoped oder route-scoped definiert werden

Apache Camel - Errorhandling

- Apache Camel bietet folgende Errorhandler:
  - DefaultErrorHandler
    Dieser ist automatisch aktiv und muss nicht konfiguriert werden.
  - DeadLetterChannel
    Implementierung des DeadLetter-Pattern
  - TransactionErrorHandler
    DefaultErrorHandler erweitert um transaction-awareness

Diese drei erweitern den RedeliveryErrorHandler.

Apache Camel - Errorhandling

- NoErrorHandler
  Schaltet das Errorhandling komplett ab.

- LoggingErrorHandler
  Logt die Exception, keine weitere Fehlerbehandlung

Diese zwei haben nur eine eingeschränkte Funktionalität, z. B. kein Redelivery.

Der DefaultErrorHandler ist so konfiguriert, dass er kein Redelivery versucht und die Exception zurück an den Aufrufer gibt.

Apache Camel - Errorhandling

- Beispiel zum Errorhandling:
  - Lesen eines Files
  - Scoping der Errorhandler
  - DefaultErrorHandler und DeadLetterChannel
  - Redelivery

siehe: Example 4

Apache Camel - Errorhandling

- Exception policies
  - gesonderte Behandlung von spezifischen Exceptions
  - dynamische Beeinflussung der Exceptionbehandlung zur Laufzeit
  - Umleitung von Messages im Fehlerfall
  - sehr mächtig, kann allerdings auch schnell zu einer Exception-Policy-Wüste führen

Apache Camel - Errorhandling

- Beispiel zu Exception policies:

```
<camelContext ...>
<onException>
  <exception>javax.jms.JmsException</exception>
  <handled><constant>true</constant></handled>
  <process ref="jmsExcptionHandler"/>
</onException>

<route> ... </route>
</camelContext ...>

<bean id="jmsExceptionHandler" class="..."/>
```

Apache Camel - Testing

- Unterstützung beim Testing:
  Apache Camel Test Kit
  - Unterstützung bei der Erstellung von JUnit-Tests mit der Standard-JUnit-API
  - Mock component zur einfachen Erstellung von Komponenten-Mocks
  - Producer template zum einfachen Senden von Nachrichten in Testfällen

Apache Camel - Testing

- org.apache.camel.test.junit4.TestSupport:
  Abstrakte Testklasse mit zusätzlichen Assertions
- org.apache.camel.test.junit4.CamelTestSupport:
  Basistestklasse zum Test von Camel-Routen
- org.apache.camel.test.junit4.
  CamelSpringTestSupport:
  Basistestklasse zum Test von Camel-Routen, die in der Spring-DSL definiert sind. Enthält zusätzlich zu CamelTestSupport Spring-relevante Methoden.

Apache Camel - Testing

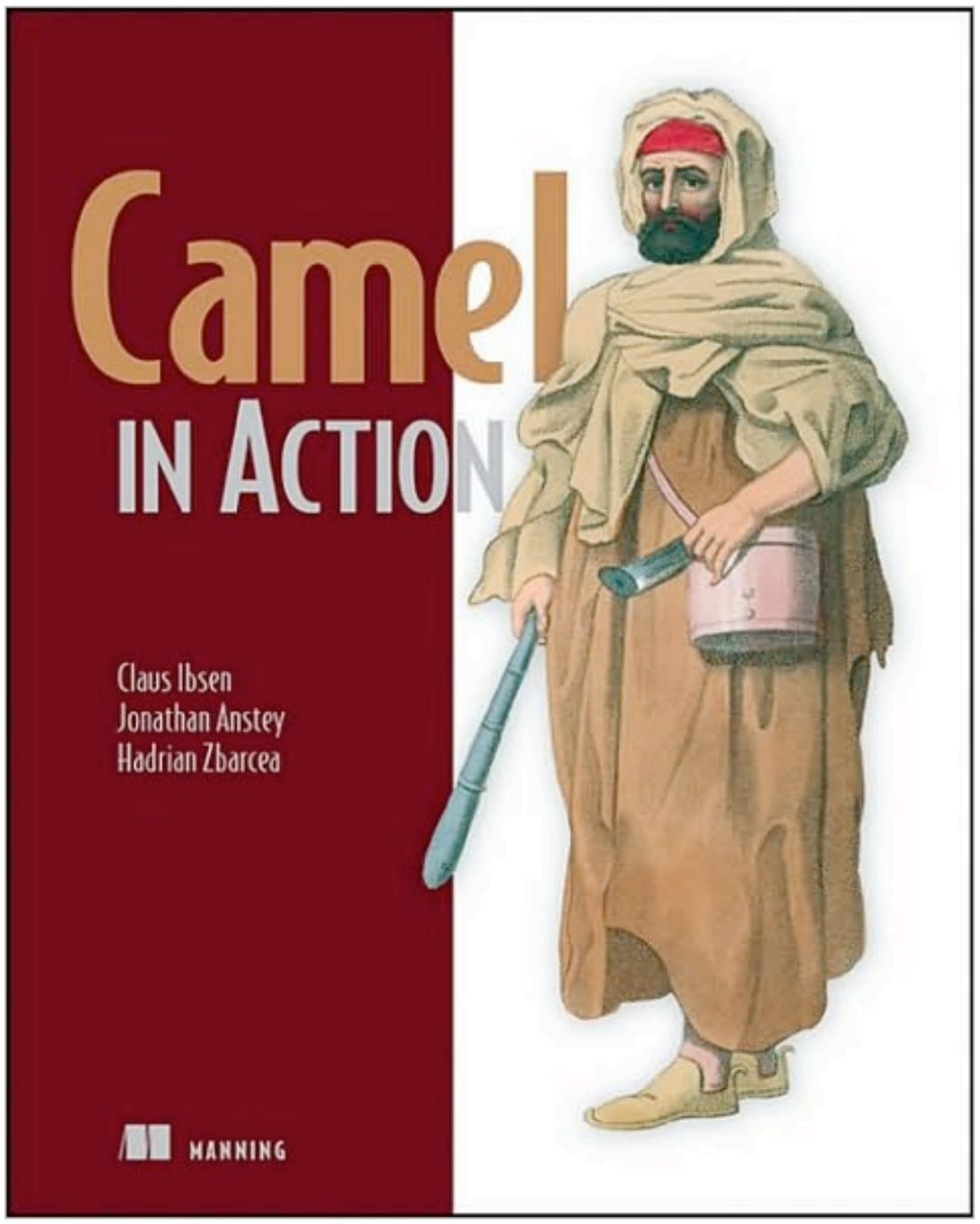
- Beispiel eines einfachen Tests mit der Routendefiniton in einem POJO

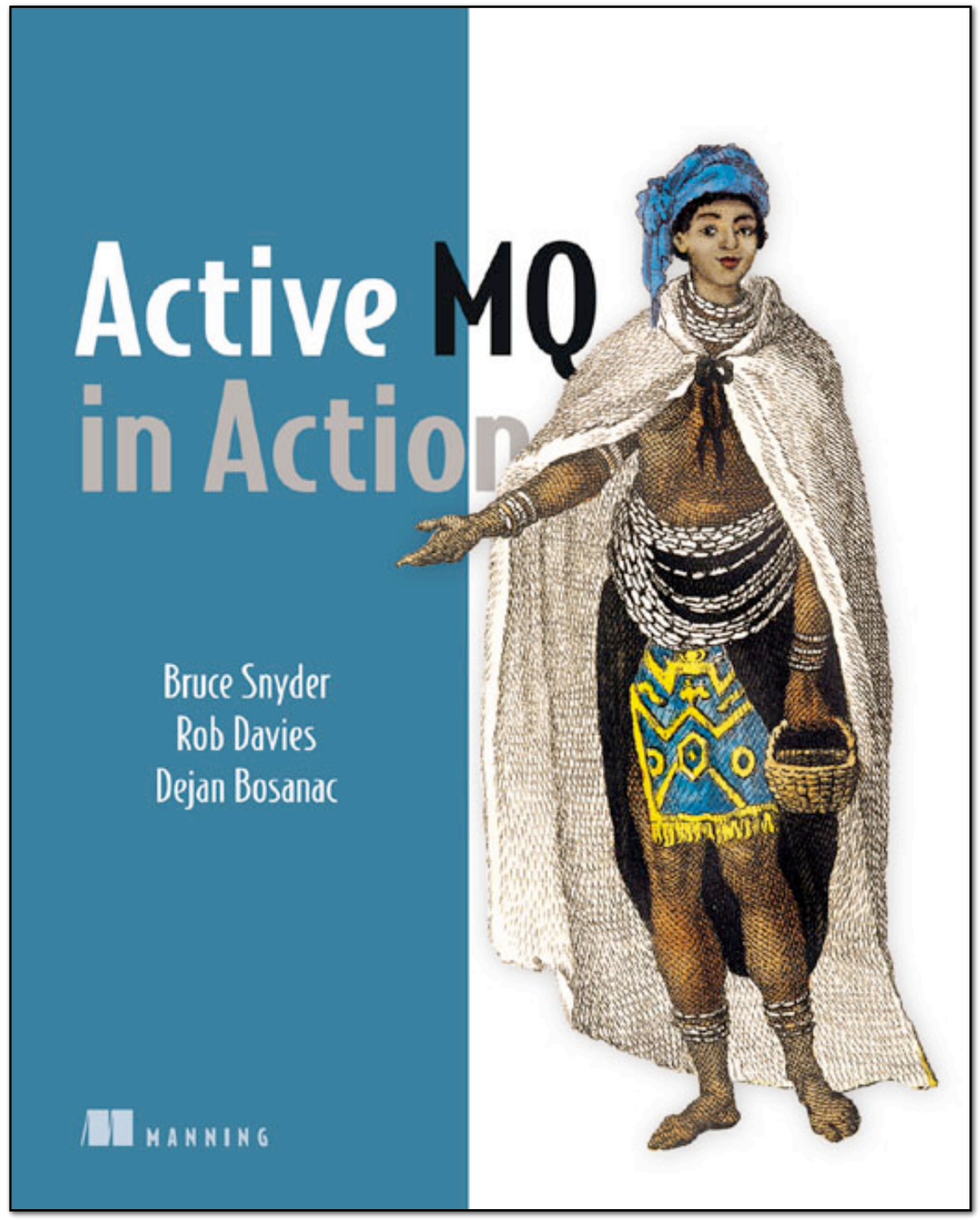
  siehe: Example 5

- Beispiel eines Tests mit Routendefinition in Spring

  siehe: Example 6

Apache Camel





Ende

Fragen?

Fragen!