logica

Herzlich Willkommen!

Implementing Enterprise Integration Patterns with Apache Camel

Über den Referenten

Eduard Hildebrandt

IT Consultant

EAI

SOA

MDA

+49 (0711) 72846627

+49 (0160) 8870983

eduard.hildebrandt@logica.com

Definitionen

Enterprise Application Integration (EAI) ist ein Konzept zur unternehmensweiten Integration der Geschäftsfunktionen entlang der Wertschöpfungskette, die über verschiedene Applikationen auf unterschiedliche Plattformen verteilt sind, und im Sinne der Daten- und Geschäftsprozessintegration verbunden werden können.

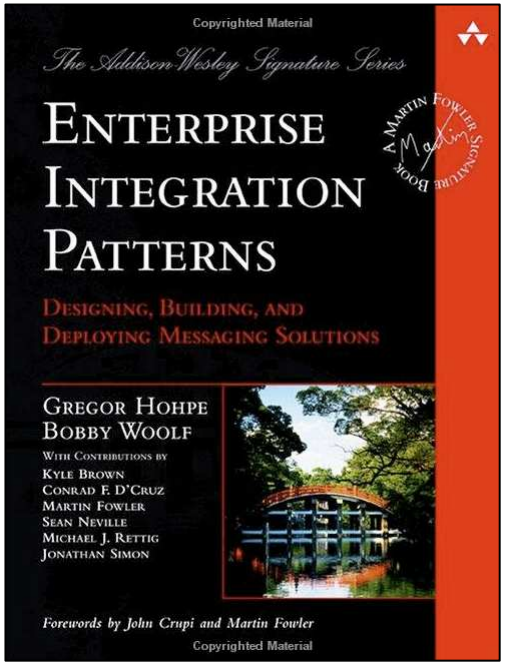
+

Design Patterns beschreiben bewährte Lösungs-Schablonen für ein Entwurfsproblem. Sie stellen damit wiederverwendbare Vorlagen zur Problemlösung dar, die in einem spezifischen Kontext einsetzbar sind.

=

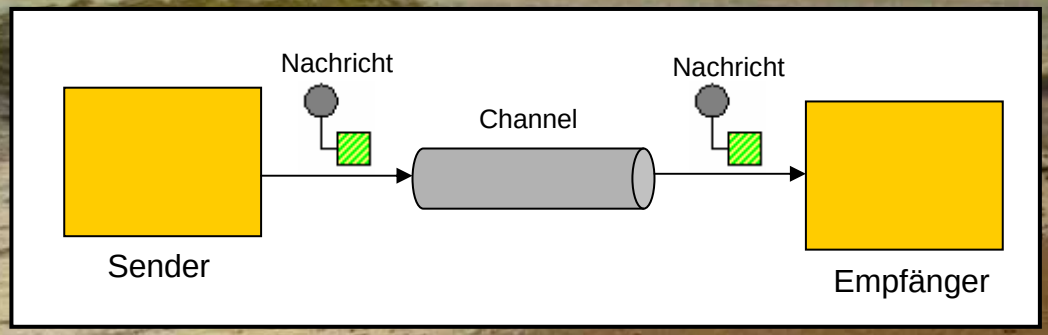
Enterprise Integration Patterns sind bewährte Lösungsschablonen zur Integration von Geschäftsfunktionen und Anbindung von verschiedenen Anwendungen auf unterschiedlichen Plattformen.

Buchempfehlung



Titel: Enterprise Integration Patterns
Autor: Gregor Hohpe und Bobby Woolf
Verlag: Addison-Wesley
ISBN: 0321200683

Messaging



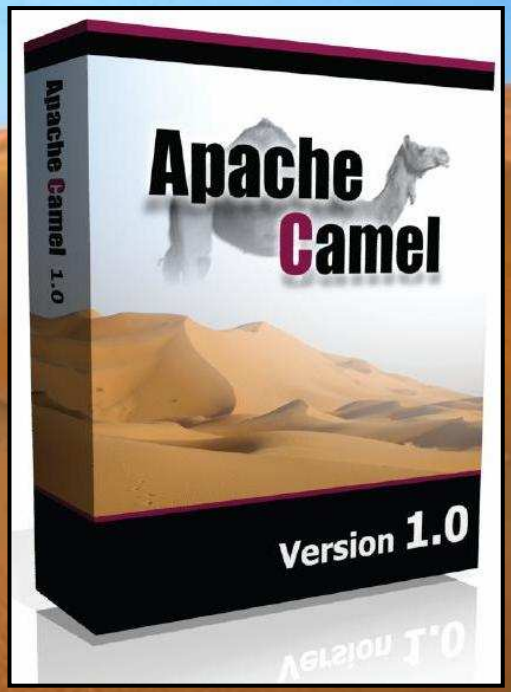
location dependencies

data format dependencies

temporal dependencies

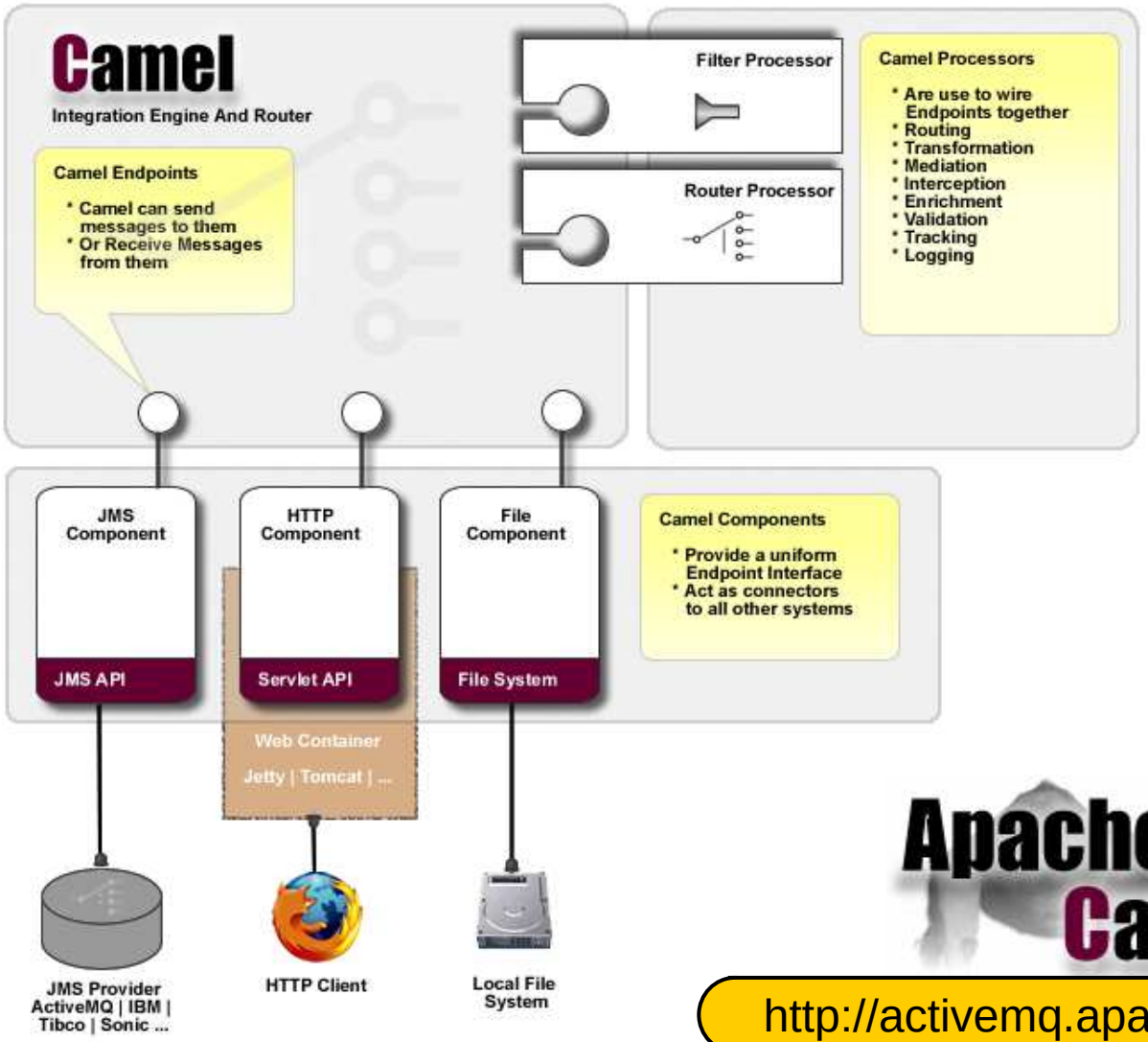
Was ist Apache Camel?

- Open-Source Java Framework
- Implementiert ca. 35 Enterprise Integration Patterns
- Domain Specific Language (Fluent API)



http://activemq.apache.org/camel

Apache Camel Architecture



Apache Camel

http://activemq.apache.org/camel

Es war einmal…

…vor nicht allzulanger Zeit

…an einem wunderschönen Tag

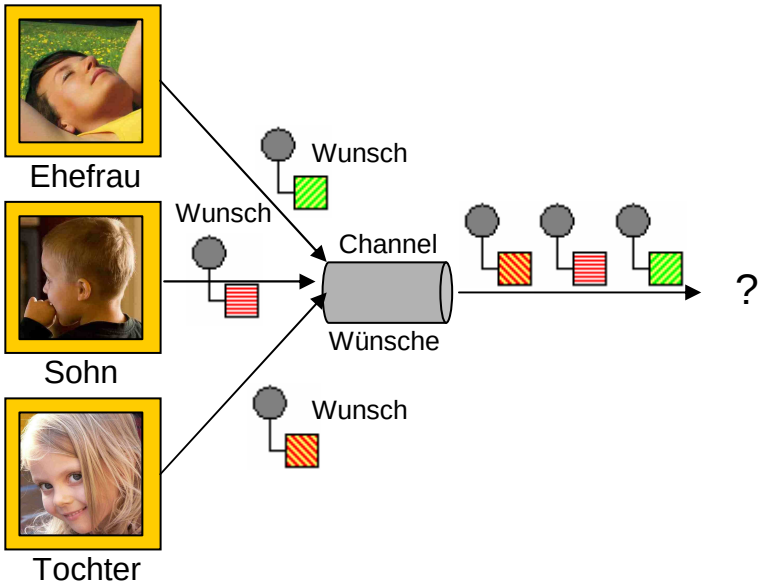
…ein IT-Architekt, der seinen **Einkauf** plante.

logica


Kinder


Einkaufsliste
· Käse
· Tomaten
· Brot
· Malstifte
· Kinderüberraschung
· Teddybär

Einkaufsliste mit Messaging…

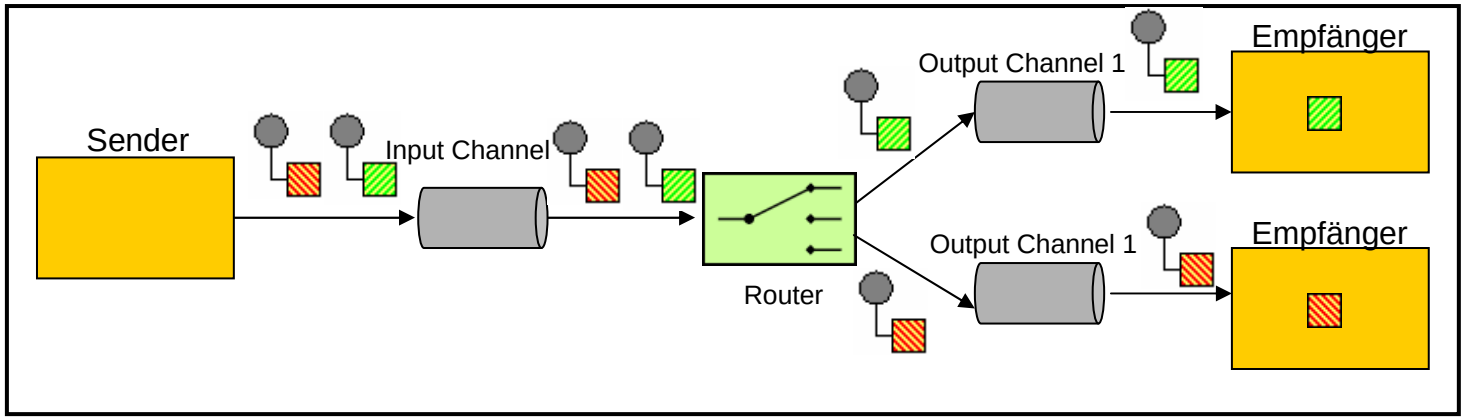


Einkaufsliste mit Apache Camel…

```java
public static void main(String args[]) throws Exception {
    CamelContext context = new DefaultCamelContext();
    TibjmsConnectionFactory connectionFactory = new TibjmsConnectionFactory();
    connectionFactory.setServerUrl("tcp://localhost:7222");
    connectionFactory.setUserName("admin");
    connectionFactory.setUserPassword("");
    context.addComponent("jms", JmsComponent.jmsComponentAutoAcknowledge(connectionFactory));
    context.addRoutes(new RouteBuilder() {
        public void configure() {
            from("jms://wishes")
            .to("file://shoppinglists");
        }
    });
    context.sta
    System.in.re
    context.stop
}
```

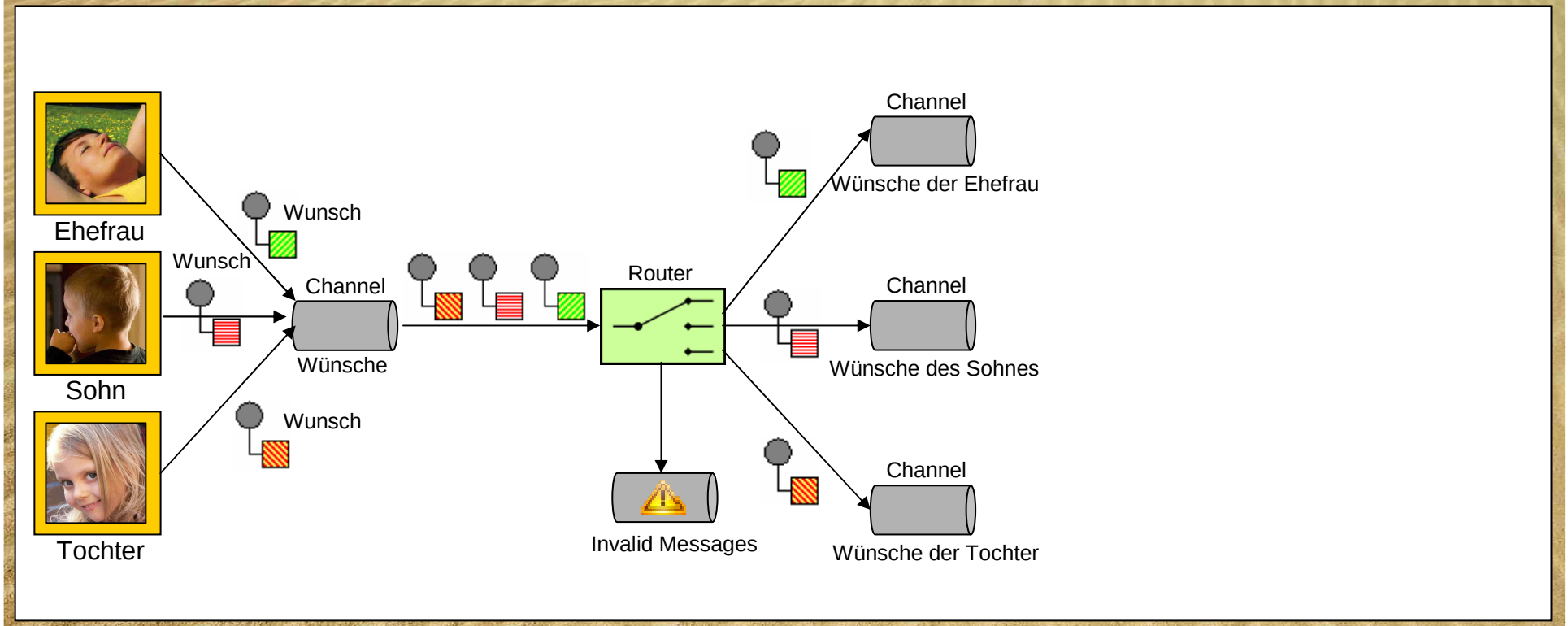
```xml
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <camelContext id="camel"
       xmlns="http://activemq.apache.org/camel/schema/spring">
       <route>
           <from uri="jms://wishes" />
           <to uri="file://shoppinglists" />
       </route>
   </camelContext>
   <bean id="jmsComponent" class="org.apache.camel.component.jms.JmsComponent">
       <property name="connectionFactory" ref="jmsConnectionFactory"/>
   </bean>
   <bean id="jmsConnectionFactory" class="com.tibco.tibjms.TibjmsConnectionFactory">
       <property name="serverUrl"><value>tcp://localhost:7222</value></property>
       <property name="userName"><value>admin</value></property>
       <property name="userPassword"><value></value></property>
   </bean>
</beans>
```

Pattern: Router



- Emfängt eine Nachricht von einem Channel und leitet diese abhängig von bestimmten Kriterien an einen anderen Channel weiter.
- Verändert nur das Ziel der Nachricht und nicht den Inhalt der Nachricht.

Einsatz von Router

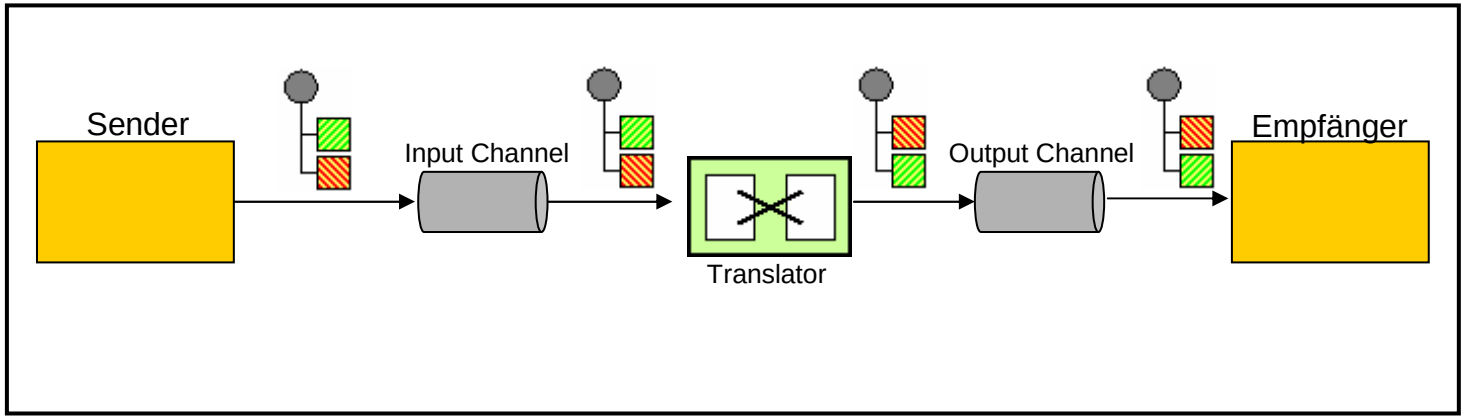


Router mit Apache Camel

```
public static void main(String args[]) throws Exception {
    CamelContext context = new DefaultCamelContext();
    TibjmsConnectionFactory connectionFactory = new TibjmsConnectionFactory();
    connectionFactory.setServerUrl("tcp://localhost:7222");
    connectionFactory.setUserName("admin");
    connectionFactory.setUserPassword("");
    context.addComponent("jms", JmsComponent.jmsComponentAutoAcknowledge(connectionFactory));
    context.addRoutes(new RouteBuilder() {
        public void configure() {
            from("jms://wishes")
            .choice()
                    .when(header("from").co
                        .to("direct:wishes-
                    .when(header("from").co
                        .to("direct:wishes-
                    .when(header("from").co
                        .to("direct:wishes-
                    .otherwise()
                        .to("log:invalid-wi
        }
    });
    context.start();
    System.in.read();
    context.stop();
}
```

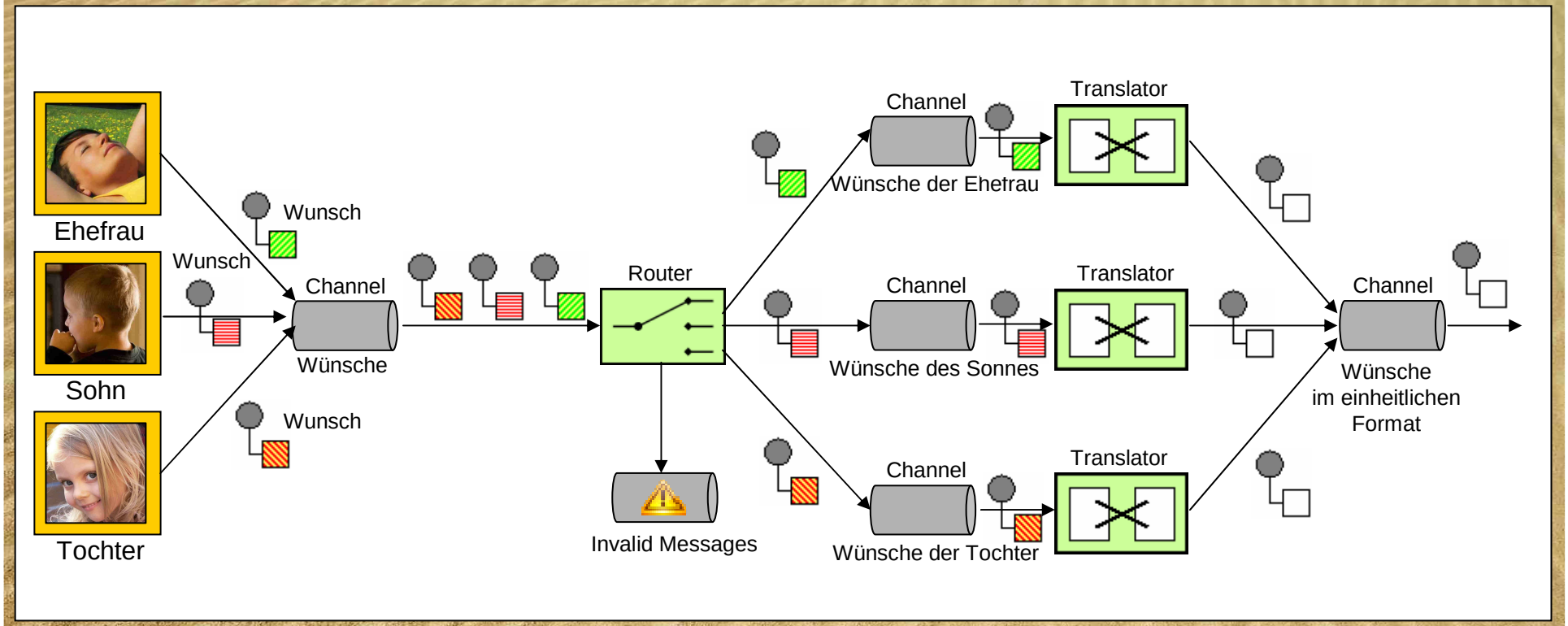
```
<camelContext id="camel"
    xmlns="http://activemq.apache.org/camel/schema/spring">
    <route>
        <from uri="jms://wishes" />
        <choice>
            <when>
                <xpath>$from = 'wife'</xpath>
                <to uri="direct:wishes-wife" />
            </when>
            <when>
                <xpath>$from = 'son'</xpath>
                <to uri="direct:wishes-son" />
            </when>
            <when>
                <xpath>$from = 'daughter'</xpath>
                <to uri="direct:wishes-daughter" />
            </when>
            <otherwise>
                <to uri="log:invalid-wishes" />
            </otherwise>
        </choice>
    </route>
</camelContext>
```

Pattern: Message Translator



- Transformiert von einem Datenformat in ein anderes Datenformat.
- Äquivalent zum Adapter Pattern für Nachrichten.

Einsatz von Message Translator



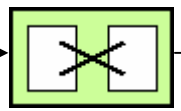
Transformation der Nachrichten (Ehefrau)

```
public static void main(String args[]) throws Exception {
    JndiContext jndiContext = new JndiContext();
    jndiContext.bind("wishTransformator", new WishTransformator());
    CamelContext context = new DefaultCamelContext(jndiContext);
    // ...
    final JaxbDataFormat jaxb = new JaxbDataFormat();
    jaxb.setContextPath("com.logica.examples.camel.model");
    context.addRoutes(new RouteBuilder() {
        public void configure() {
        // ...
        from("direct:wishes-wife")
            .unmarshal(new CsvDataFormat())
            .setHeader("from", "wife")
            .beanRef("wishTransformator", "transformFromCSV")
            .marshal(jaxb)
        .to("direct:normalized-wishes");
        // ...
        }
    });
    context.start();
    System.in.read();
    context.stop();
}
```

```
public class WishTransformator {
   public Wish transformFromCSV(
            List<String> csvWish,
            @Header(name="from") String requester) {
       String productName = csvWish.get(0);
       String quantity = csvWish.get(1);
       Wish wish = new Wish(productName,
               Integer.parseInt(quantity));
       wish.setRequester(requester);
       return wish;
   }
}
```

Wunsch der Ehefrau:

```
// CSV: Produkt,Anzahl
Milch,2
```



Allgemeine Darstellung:

```
<wish>
  <productName>Milch</productName>
  <quantity>2</quantity>
</wish>
```

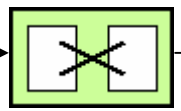
Transformation der Nachrichten (Sohn)

```
public static void main(String args[]) throws Exception {
    JndiContext jndiContext = new JndiContext();
    jndiContext.bind("wishTransformator", new WishTransformator());
    CamelContext context = new DefaultCamelContext(jndiContext);
    // ...
    final JaxbDataFormat jaxb = new JaxbDataFormat();
    jaxb.setContextPath("com.logica.examples.camel.model");
    context.addRoutes(new RouteBuilder() {
        public void configure() {
        // ...
        from("direct:wishes-son")
            .unmarshal(new SerializationDataFormat())
            .setHeader("from", "son")
            .beanRef("wishTransformator", "transformFromWishOfSon")
            .marshal(jaxb)
        .to("direct:normalized-wishes");
        // ...
        }
    });
    context.start();
    System.in.read();
    context.stop();
}
```

```
public class WishTransformator {
   public Wish transformFromWishOfSon(
            WishOfSon wishOfSon,
            @Header(name="from") String requester) {
        Wish wish = new Wish(wishOfSon.getProductName(),
                wishOfSon.getQuantity());
        wish.setRequester(requester);
        return wish;
   }
}
```

Wunsch des Sohns:

```
// Java Serialization
[binäre Daten]
```



Allgemeine Darstellung:

```
<wish>
  <productName>Kinderei</productName>
  <quantity>2</quantity>
</wish>
```

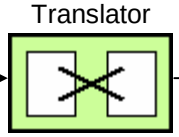
Transformation der Nachrichten (Tochter)

```java
public static void main(String args[]) throws Exception {
    // ...
    CamelContext context = new DefaultCamelContext();
    // ...
    context.addRoutes(new RouteBuilder() {
        public void configure() {
        // ...
        from("direct:wishes-daughter")
            .to("xslt://wunsch2wish.xslt")
        .to("direct:normalized-wishes");
        }
    });
    context.start();
    System.in.read();
    context.stop();
}
```

```xml
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns="http://www.logica.com/examples/camel">
   <xsl:template match="wunsch">
       <wish>
           <productName><xsl:value-of select="produkt"/></productName>
           <quantity><xsl:value-of select="anzahl"/></quantity>
       </wish>
   </xsl:template>
</xsl:stylesheet>
```

Wunsch der Tochter:

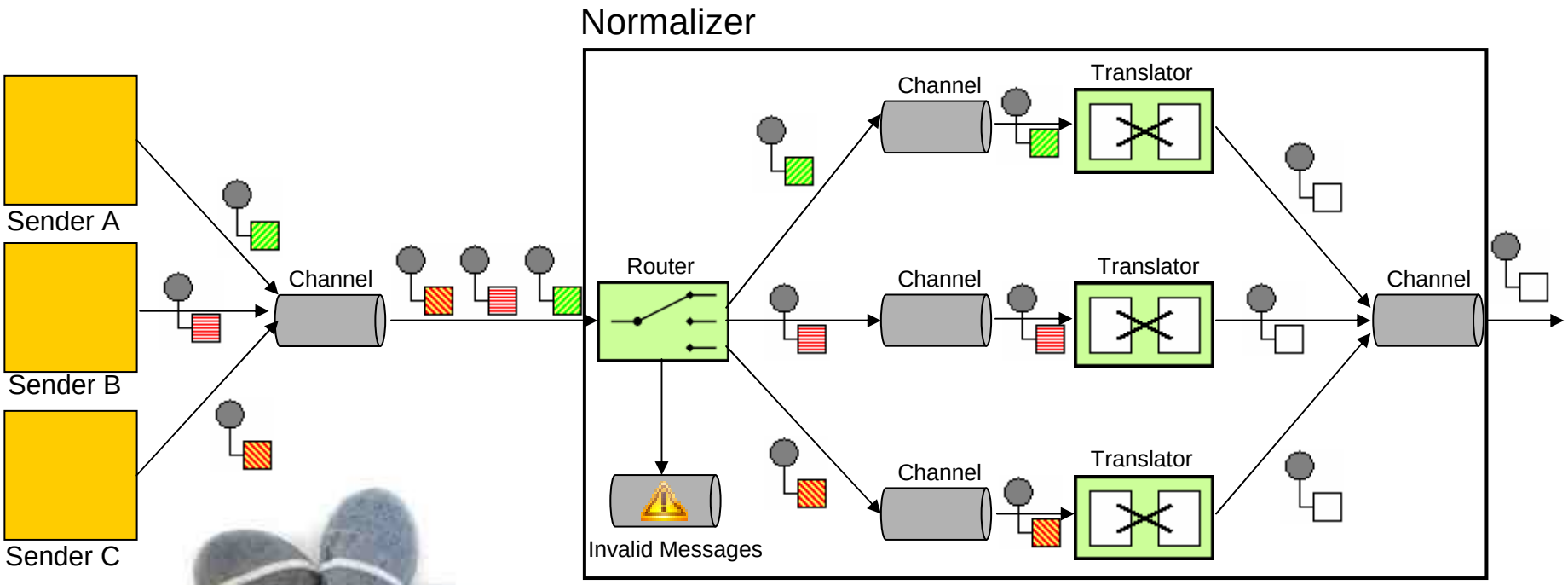
```xml
<wunsch>
  <produkt>Teddybaer</produkt>
  <anzahl>1</anzahl>
</wunsch>
```



Allgemeine Darstellung:

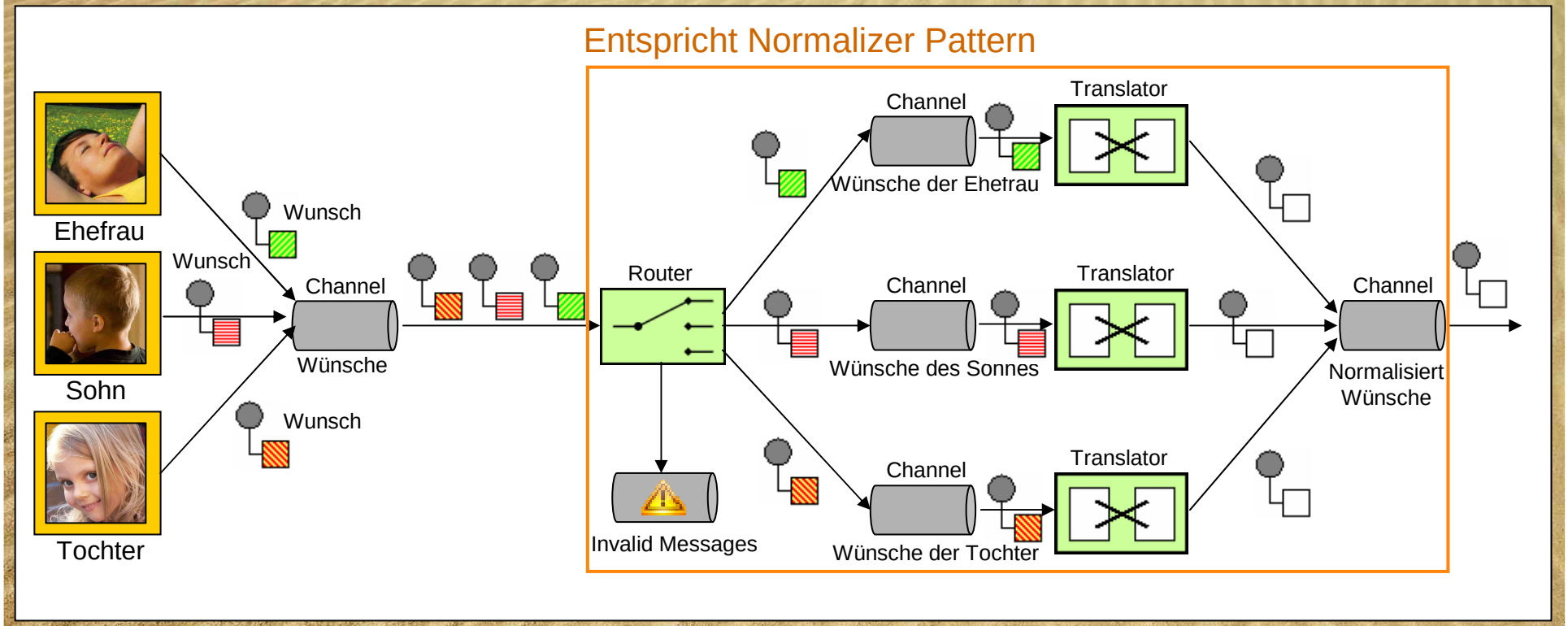
```xml
<wish>
  <productName>Teddybaer</productName>
  <quantity>1</quantity>
</wish>
```

Pattern: Normalizer

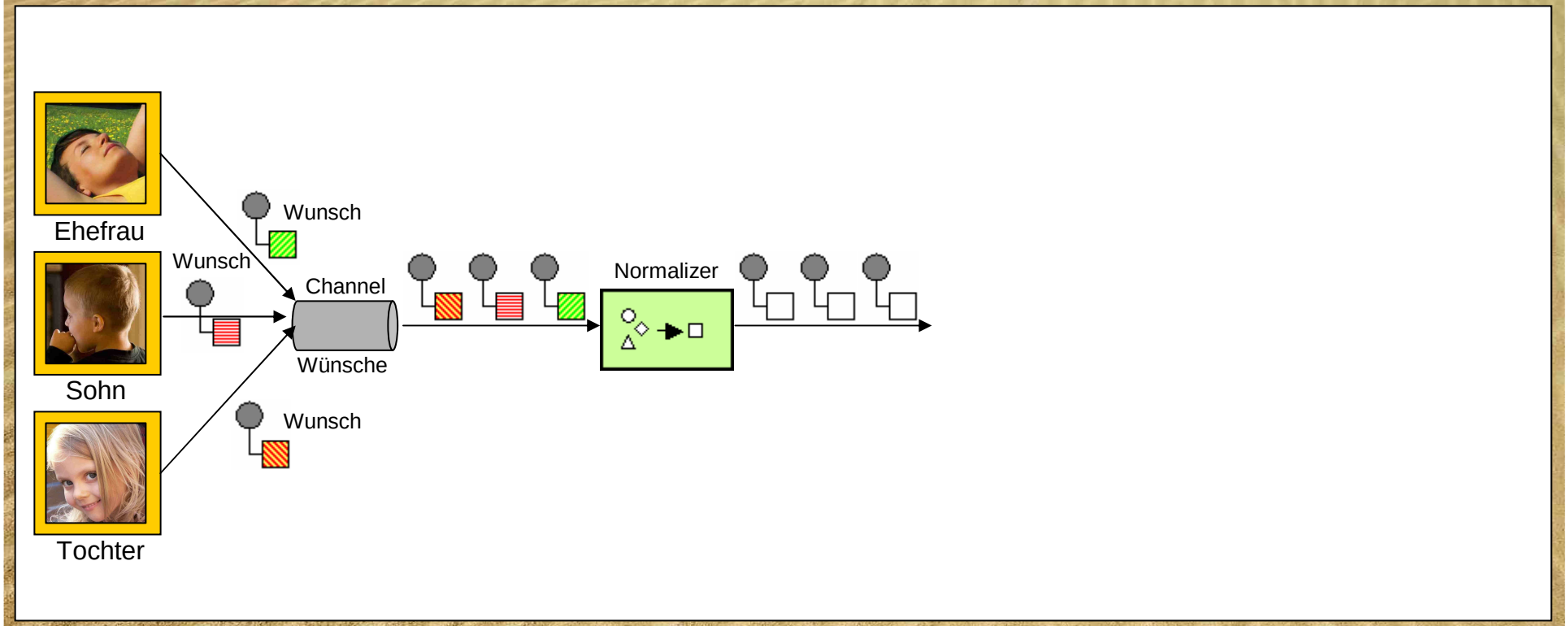


- Transformiert Nachrichten in unterschiedlichen Datenformaten in ein einheitliches Datenformat.
- Verwendet intern einen Message Router und mehrere Message Translators.

Einsatz von Normalizer

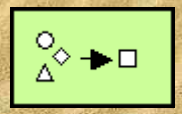


Einsatz von Normalizer



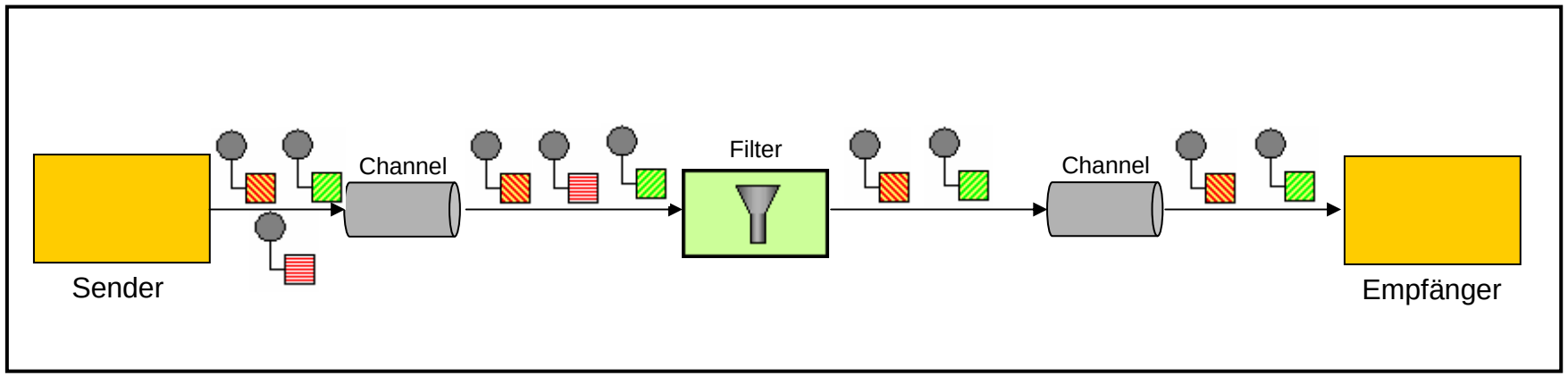
Normalizer mit Apache Camel

```
public static void main(String args[]) throws Exception {
    JndiContext jndiContext = new JndiContext();
    jndiContext.bind("wishTransformator", new WishTransformator());
    CamelContext context = new DefaultCamelContext(jndiContext);
    // ...
    context.addRoutes(new WishNormalizer());
    context.start();
    System.in.read();
    context.stop();
}
```



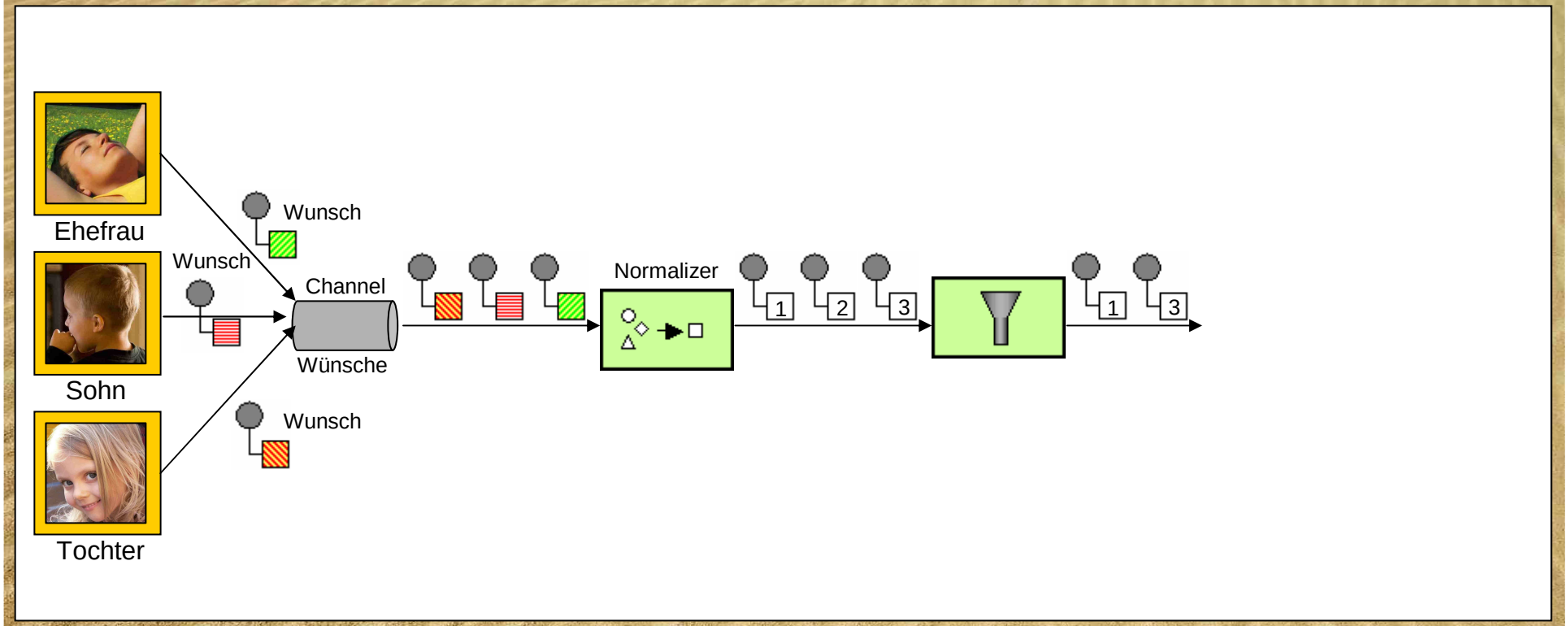
```
public class WishNormalizer extends RouteBuilder {
    public void configure() {
        from("jms:wishes")
            .choice()
                .when(header("from").contains("wife"))
                    .to("direct:wishes-wife")
                .when(header("from").contains("son"))
                    .to("direct:wishes-son")
                .when(header("from").contains("daughter"))
                    .to("direct:wishes-daughter")
                .otherwise()
                    .to("log:unknown");
        from("direct:wishes-wife")
            .unmarshal(new CsvDataFormat())
            .setHeader("from", "wife")
            .beanRef("wishTransformator", "transformFromCSV")
            .marshal(jaxb)
        .to("direct:normalized-wishes");
        from("direct:wishes-son")
            .unmarshal(new SerializationDataFormat())
            .setHeader("from", "son")
            .beanRef("wishTransformator", "transformFromWishOfSon")
            .marshal(jaxb)
        .to("direct:normalized-wishes");
        from("direct:wishes-daughter")
            .to("xslt://wunsch2wish.xslt")
        .to("direct:normalized-wishes");
    }
}
```

Pattern: Filter



- spezielle Art von einem Router
- leitet nur Nachrichten weiter, die bestimmten Kriterien entsprechen
- hat im Gegensatz zu einem Router nur einen Output-Channel

Einsatz von Filter

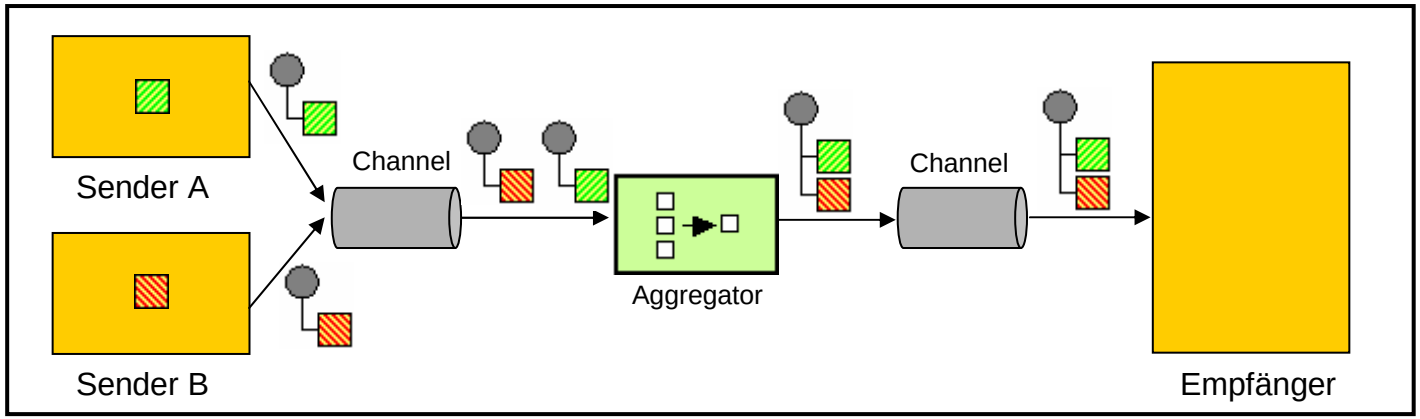


Filter mit Apache Camel

```
public static void main(String args[]) throws Exception {
    // ...
    CamelContext context = new DefaultCamelContext();
    // ...
    final Namespaces ns = new Namespaces("logica",
            "http://www.logica.com/examples/camel");
    context.addRoutes(new WishNormalizer());
    context.addRoutes(new RouteBuilder() {
        public void configure() {
            from("direct:normalized-wishes")
                .beanRef("wishValidator", "validate")
                .filter(ns.xpath("//logica:wish/resonable='true'"))
            .to("direct:filtered-wishes");
            // ...
        }
    });
    context.start();
    System.in.read();
    context.stop();
}
```

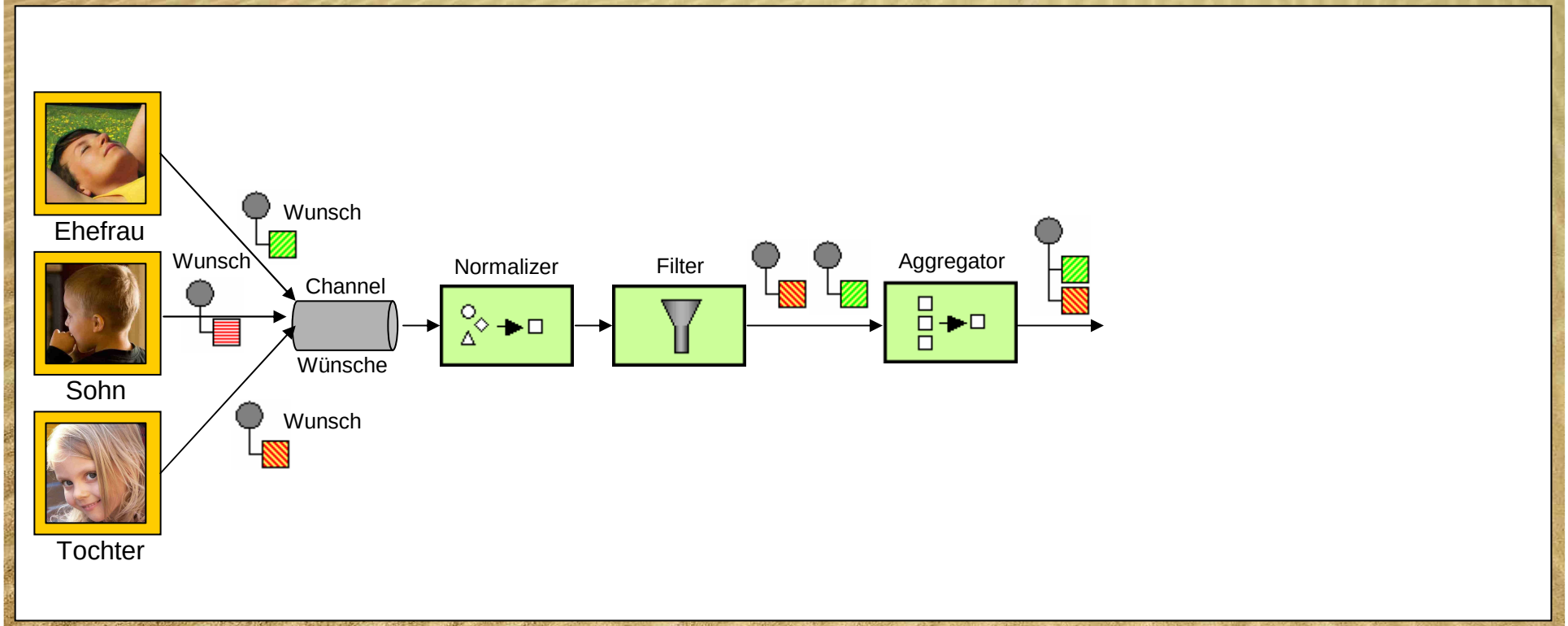
```
public class WishValidator {
    public Wish validate(Wish wish){
        // TODO: Integrate with the business rule engine!
        wish.setResonable(true);
        return wish;
    }
}
```

Pattern: Aggregator



- spezieller, zustandsbehafteter Filter
- identifiziert Nachrichten, die zueinander gehören
- sendet die aggregierten Informationen in einer eigenständigen Nachricht weiter

Einsatz von Aggregator



Aggregator in Apache Camel

```
public static void main(String args[]) throws Exception {
    CamelContext context = new DefaultCamelContext();
    // ...
    context.addRoutes(new WishNormalizer());
    context.addRoutes(new RouteBuilder() {
        public void configure() {
            // ..
            from("direct:filtered-wishes")
               .aggregator(constant(true), new XmlBodyInAggregatingStrategy())
               .completedPredicate(header("aggregated").isEqualTo(5))
               .process(new Processor(){
                   public void process(Exchange exchange)
                       throws Exception {
                       String inBody = exchange.getIn().getBody(String.class);
                       String outBody = "<wishes>" + inBody + "<wishes>";
                       exchange.getOut().setBody(outBody);
               }})
            .to("direct:aggregated-wishes");
            // ...
        }
    });
    context.start();
    System.in.read();
    context.stop();
}
```

Einkaufsliste (sortiert)

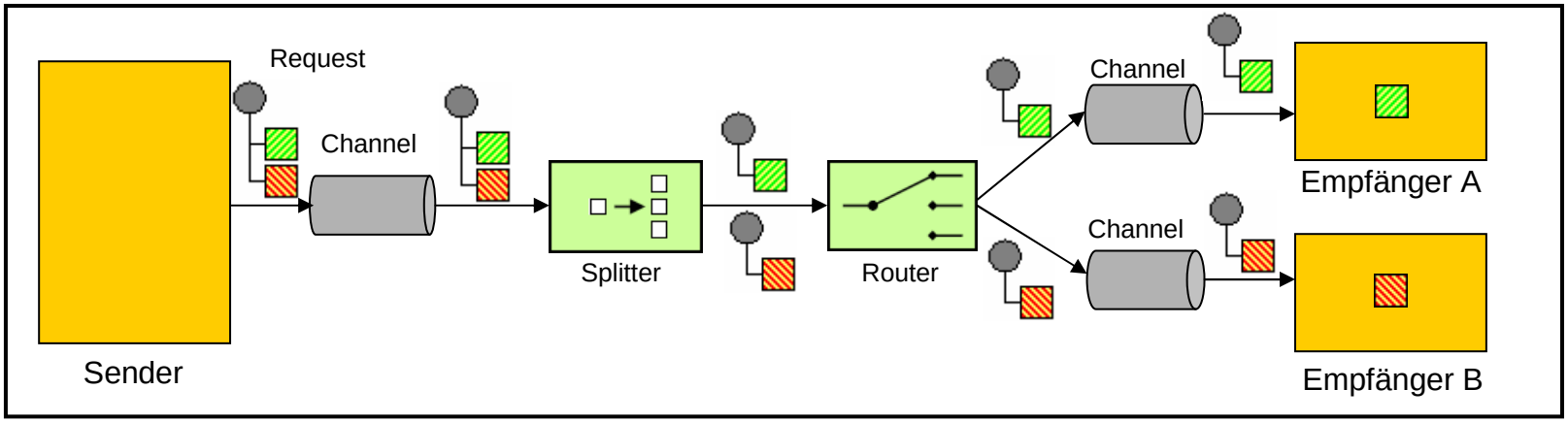
Bäckerei
- Brot

Lebensmittel-discounter
- Käse
- Tomaten
- Kinderüber-raschung

Schreibwaren
- Malstifte

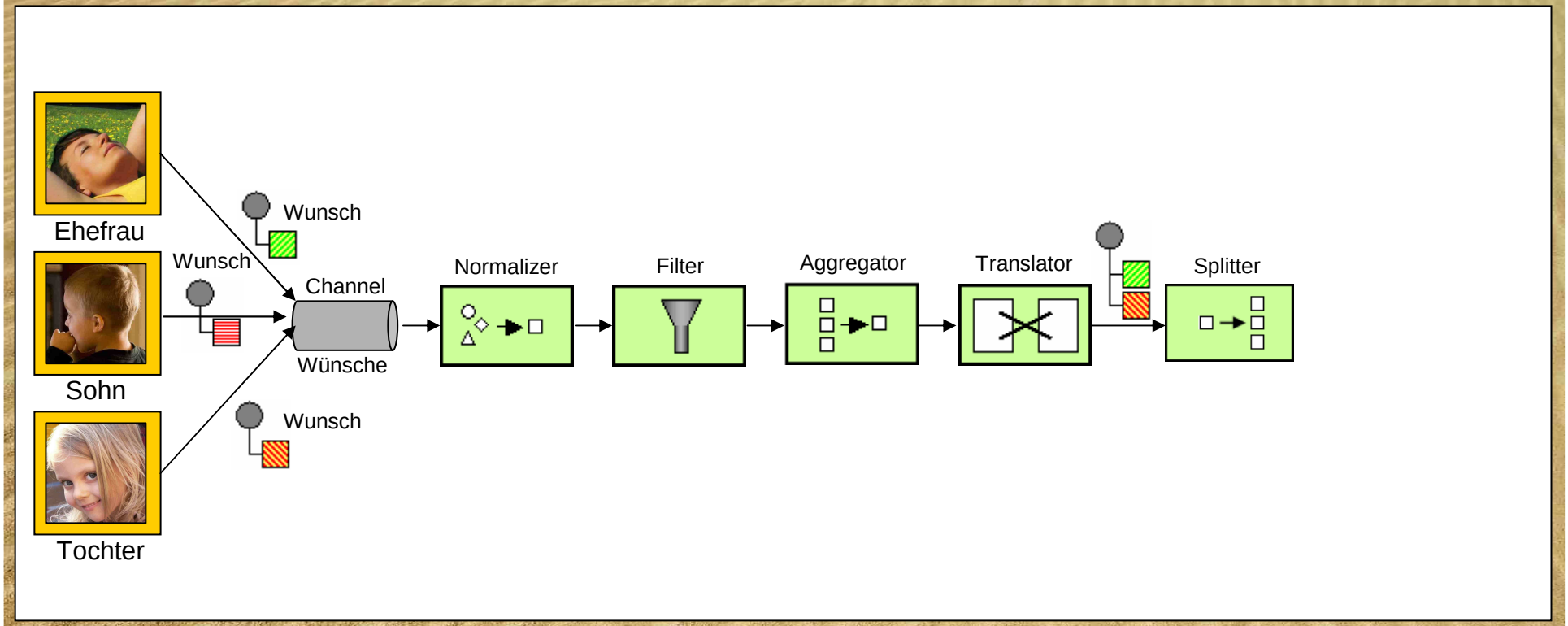
Spielzeug
- Teddybär

Pattern: Splitter



- Teilt eine zusammengesetzte Nachricht in eine Reihe einzelner Nachrichten auf.
- Wird in der Regel mit einem Router eingesetzt.

Einsatz von Splitter



Wo kann Apache Camel eingesetzt werden?

Messaging System
(z.B. Active MQ)

Service Runtime
(z.B. Apache CXF)

Enterprise Service Bus
(z.B. ServiceMix)

Ihre eigenen Java Anwendung
(mit oder ohne Spring Framework)

Testen mit Apache Camel

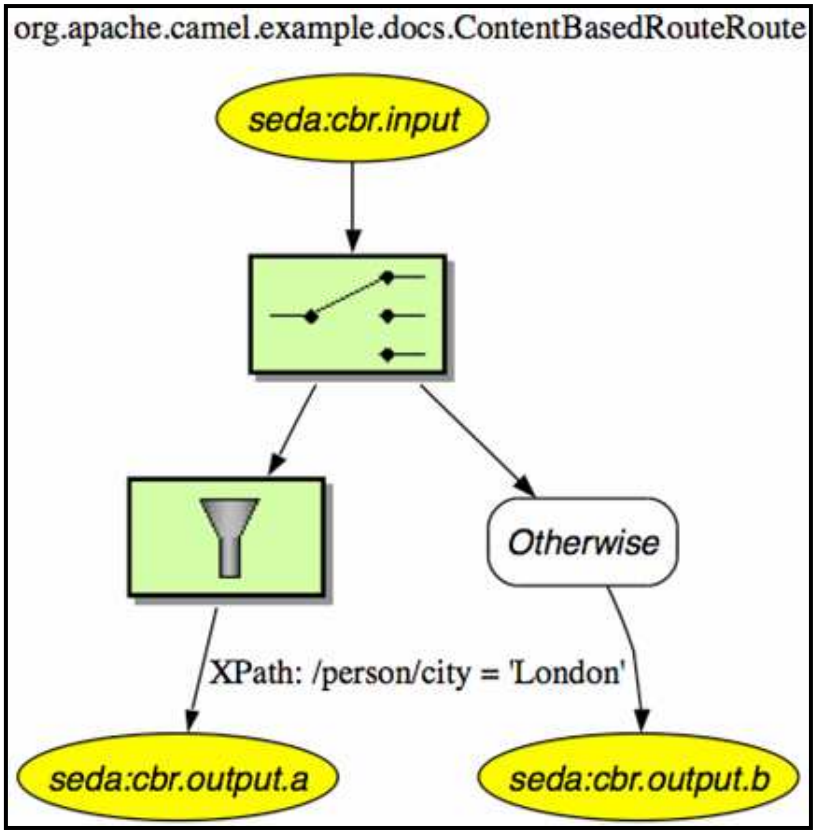
```
public static void main(String args[]) throws Exception {
    CamelContext context = new DefaultCamelContext();
    // create routes
    MockEndpoint resultEndpoint = context.getEndpoint("mock:foo", MockEndpoint.class);
    resultEndpoint.expectedMessageCount(2);
    resultEndpoint.message(0).header("foo").isEqualTo("bar");
    // now lets assert that the mock:foo endpoint received 2 messages
    resultEndpoint.assertIsSatisfied();
}
```

Mock Endpoint ermöglichen das Testen ob:

- korrekter Anzahl von Nachrichten emfangen wurde
- korrekter Payload, in richtiger Reihenfolge emfangen wurde
- die Nachrichten bestimmten Bedinungen (XPath) entsprechen

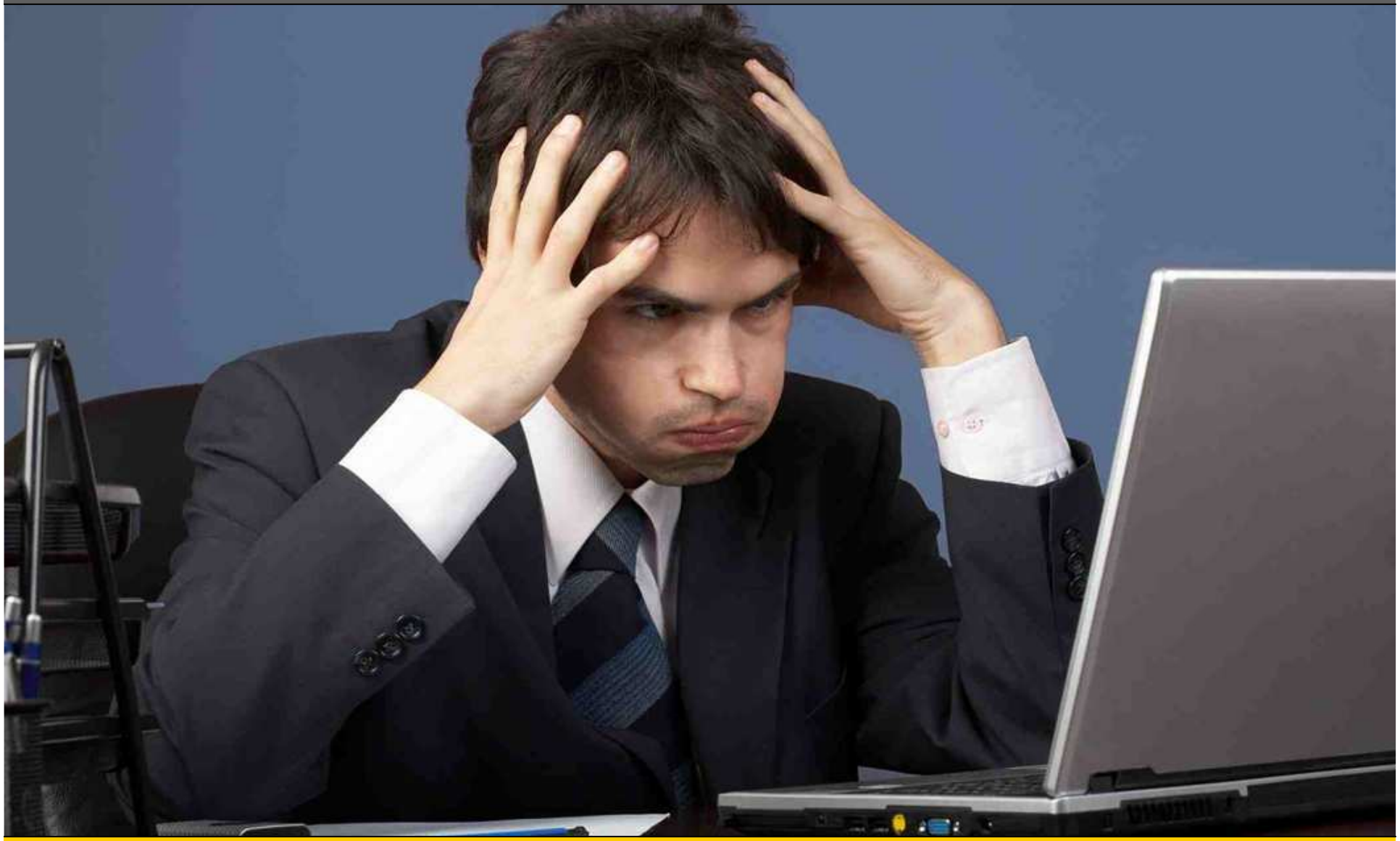
Dokumentation mit Apache Camel

- Generierung von Dokumentation mit einem Maven Plugin
- Visualisierung des Nachrichtenflusses mit GraphViz
- unterstützt HTML, PNG, SVG

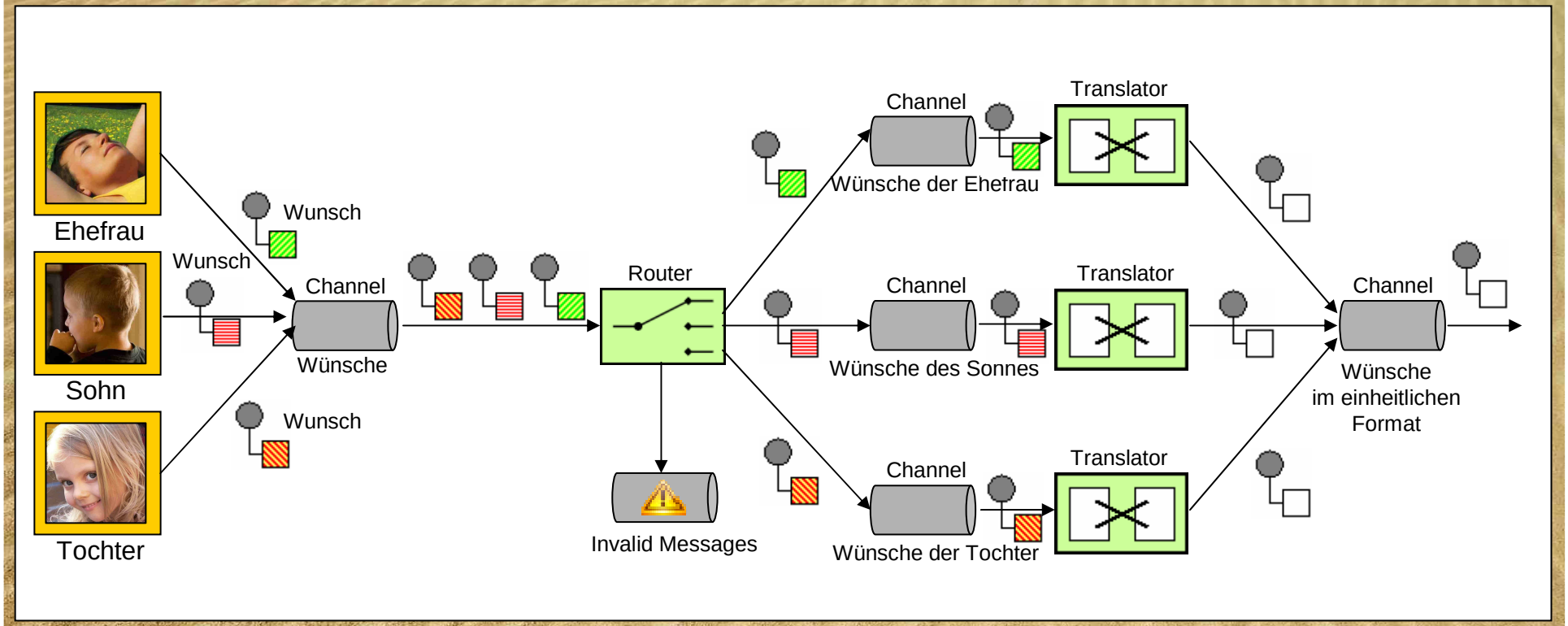


Warum EIP und Apache Camel?

Denn dies führt in der Regel dazu:



Setzen Sie statt dessen Enterprise Integration Patterns ein…



…und dokumentieren Sie Ihre Architektur!

Das spart Ihnen…

Denken Sie lieber über die wirklich wichtigen Dinge nach!

logica


Kostenlos!
Besuchen Sie uns an unserem Stand…
und sichern Sie sich ein EIP-Poster!
Nächster Kunde Bitte

Fazit

1. Enterprise Integration ist komplex!
2. Enterprise Integration Patterns sind bewährte Lösungs-Schablonen. Nutzen Sie diese!
3. Apache Camel ist ein hervorragendes Integration Framework. Testen Sie es!
4. Lassen Sie sich von Logica beraten!

Enjoy your day!


THANK YOU!

Eduard Hildebrandt

IT Consultant

+49 (0711) 72846627

+49 (0160) 8870983

eduard.hildebrandt@logica.com

www.logica.com/de

Releasing your potential

Logica

Logica is a leading IT and business services company, employing 39,000 people across 36 countries. It provides business consulting, systems integration, and IT and business process outsourcing services. Logica works closely with its customers to release their potential – enabling change that increases their efficiency, accelerates growth and manages risk. It applies its deep industry knowledge, technical excellence and global delivery expertise to help its customers build leadership positions in their markets. Logica is listed on both the London Stock Exchange and Euronext (Amsterdam) (LSE: LOG; Euronext: LOG). More information is available at www.logica.com.