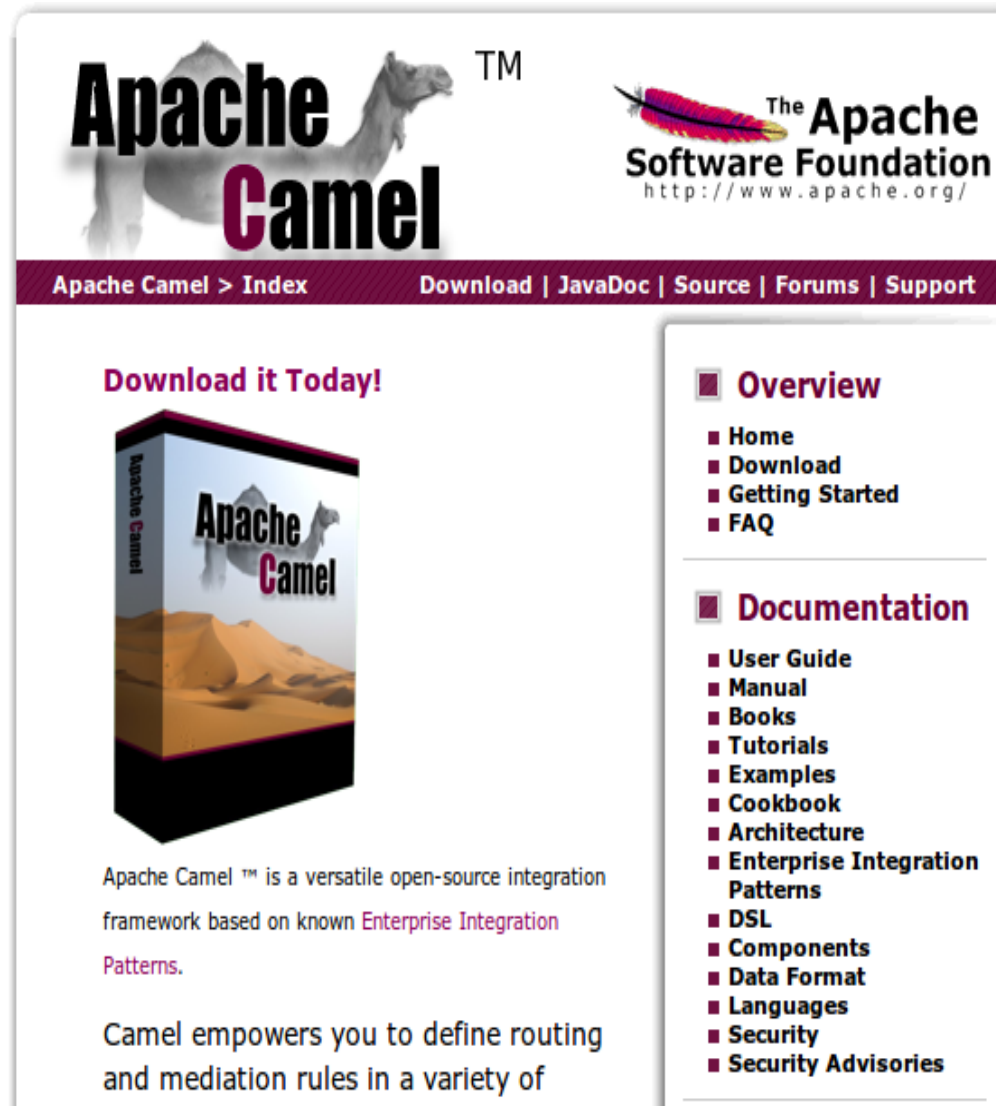
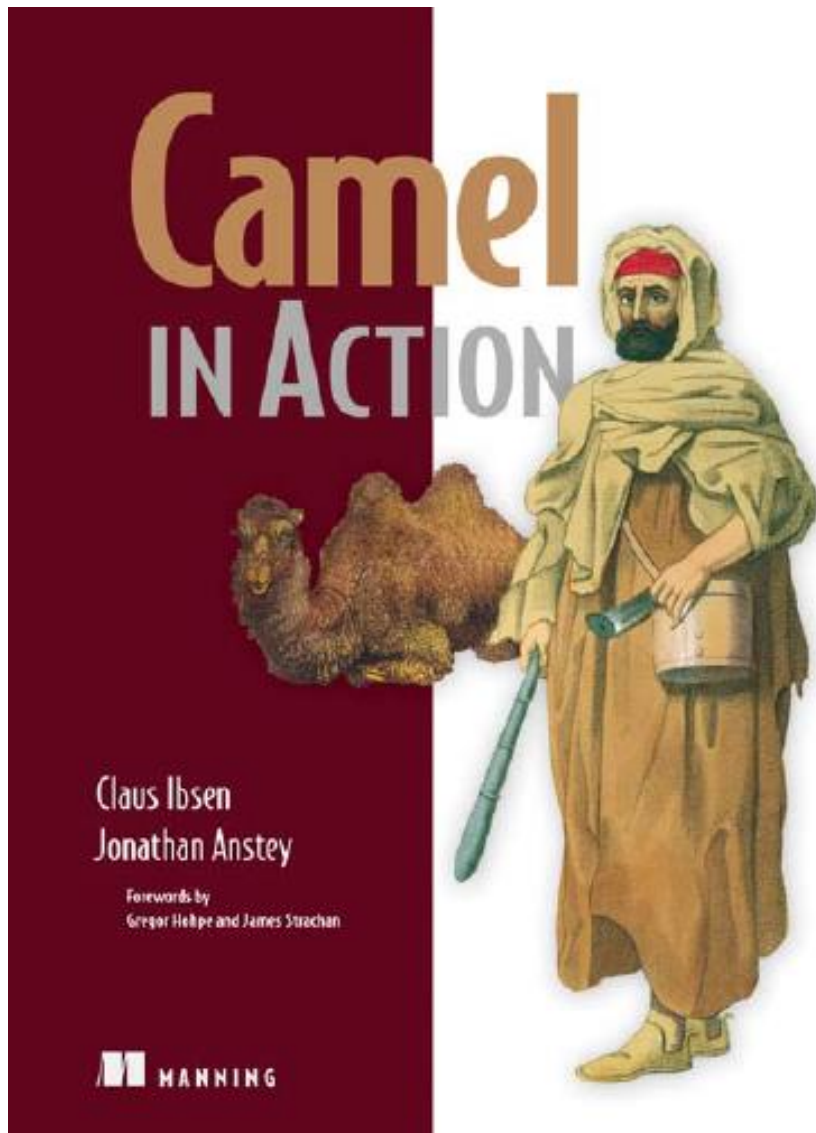




integrata
cegos

Apache Camel

Eine Java-basierte Integrationsplattform



The screenshot shows the Apache Camel website. At the top, there's a header with the Apache Camel logo (a camel) and the text 'Apache Camel' with a trademark symbol. To the right is the Apache Software Foundation logo and the URL 'http://www.apache.org/'. Below the header is a navigation bar with links: 'Apache Camel > Index', 'Download | JavaDoc | Source | Forums | Support'. The main content area has a section titled 'Download it Today!' with a 3D box icon labeled 'Apache Camel'. Below this is a paragraph: 'Apache Camel™ is a versatile open-source integration framework based on known Enterprise Integration Patterns.' followed by 'Patterns.' and another paragraph: 'Camel empowers you to define routing and mediation rules in a variety of'. On the right side, there are two sections: 'Overview' with links to Home, Download, Getting Started, and FAQ; and 'Documentation' with links to User Guide, Manual, Books, Tutorials, Examples, Cookbook, Architecture, Enterprise Integration Patterns, DSL, Components, Data Format, Languages, Security, and Security Advisories.

Apache Camel™

The Apache Software Foundation
<http://www.apache.org/>

Apache Camel > Index Download | JavaDoc | Source | Forums | Support

Download it Today!

Apache Camel

Apache Camel™ is a versatile open-source integration framework based on known Enterprise Integration Patterns.

Patterns.

Camel empowers you to define routing and mediation rules in a variety of

Overview

- Home
- Download
- Getting Started
- FAQ

Documentation

- User Guide
- Manual
- Books
- Tutorials
- Examples
- Cookbook
- Architecture
- Enterprise Integration Patterns
- DSL
- Components
- Data Format
- Languages
- Security
- Security Advisories

- Die in diesem Seminar verwendete Werkzeuge und Frameworks sind Open Source
 - LPGL Lizenzmodell
- Dies ist ein Programmier-Seminar
 - Damit werden die Inhalte durch Übungen vertieft und verinnerlicht
 - Musterbeispiele werden zur Verfügung gestellt
 - Diese können am Ende des Seminars als ZIP-Datei kopiert werden
 - USB-Stick oder ähnliches
- Dokumentation und Ressourcen stehen auch im Internet zur Verfügung
- Konventionen
 - Befehle werden in `Courier-Schriftart` dargestellt
 - Dateinamen werden in *`kursiver Courier-Schriftart`* dargestellt
 - Links werden in `unterstrichener Courier-Schriftart` dargestellt
 - Zitate werden in *"Anführungszeichen kursiv"* formatiert, die Quellenangabe steht eingerückt darunter

© Javacream

Javacream

Dr. Rainer Sawitzki

Alois-Gilg-Weg 6

81373 München

Alle Rechte, einschließlich derjenigen des auszugsweisen Abdrucks, der fotomechanischen und elektronischen Wiedergabe vorbehalten.

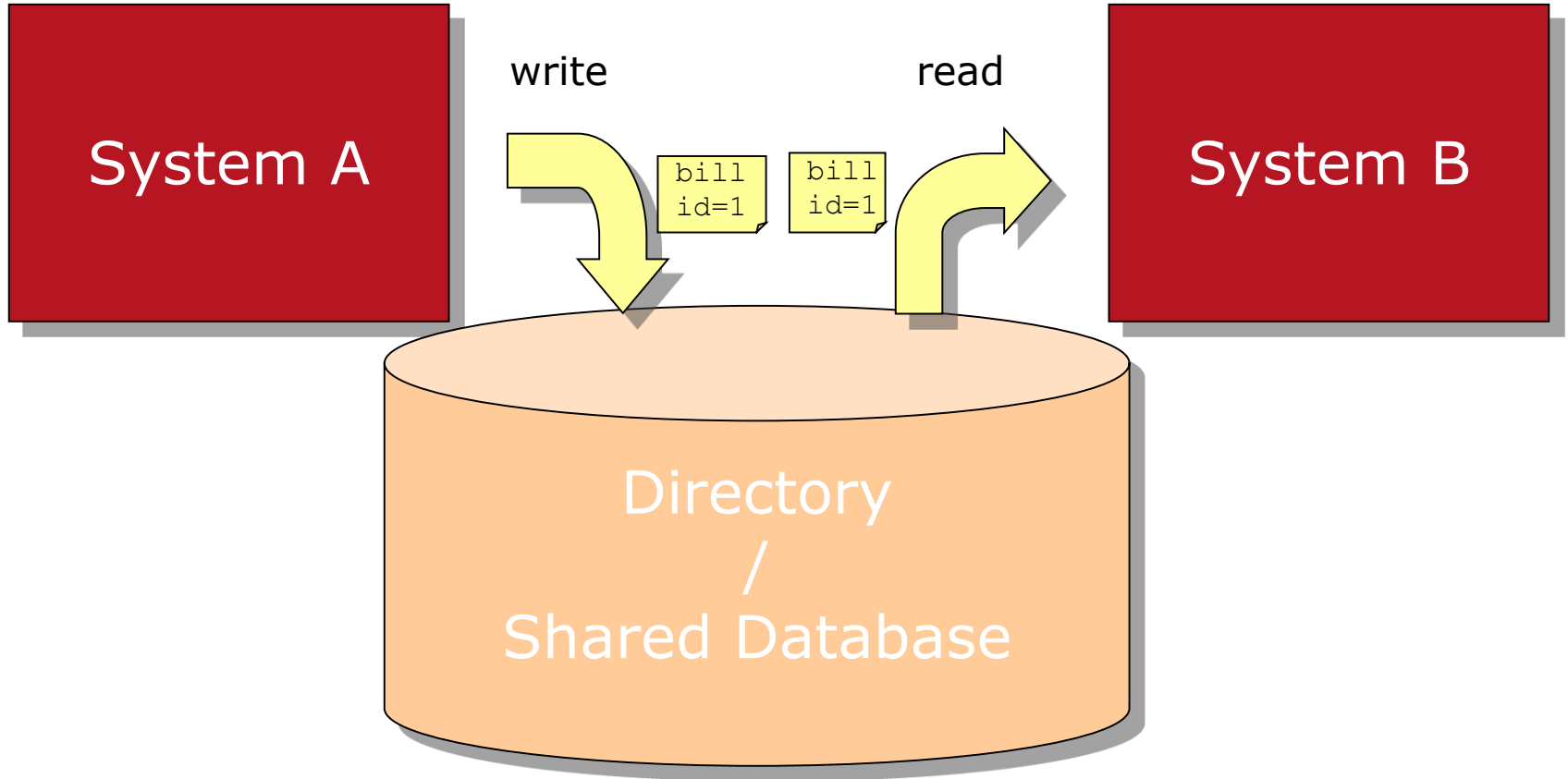
Einführung	6
Apache Camel	40
Programmierung	68
Spezielle Processors	92
Enterprise Integration Patterns	114

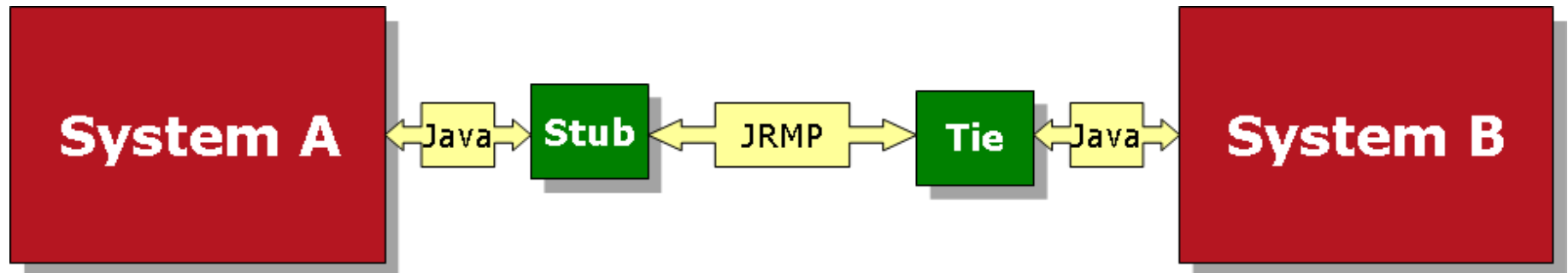
1

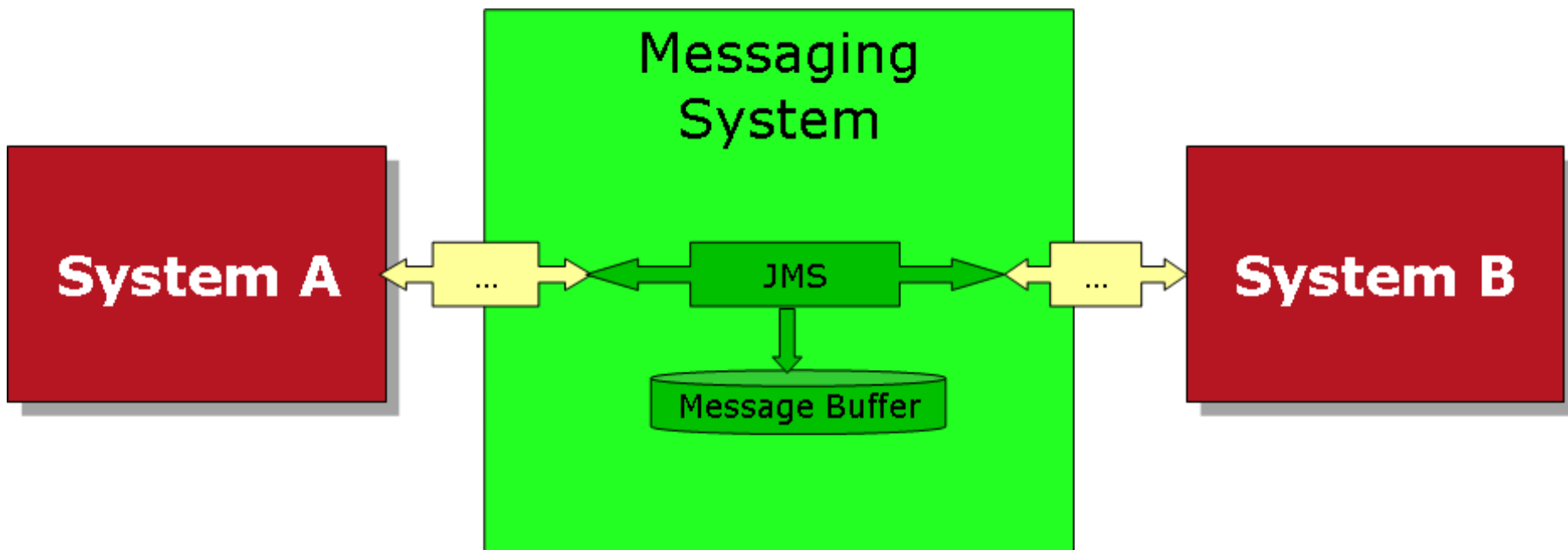
EINFÜHRUNG

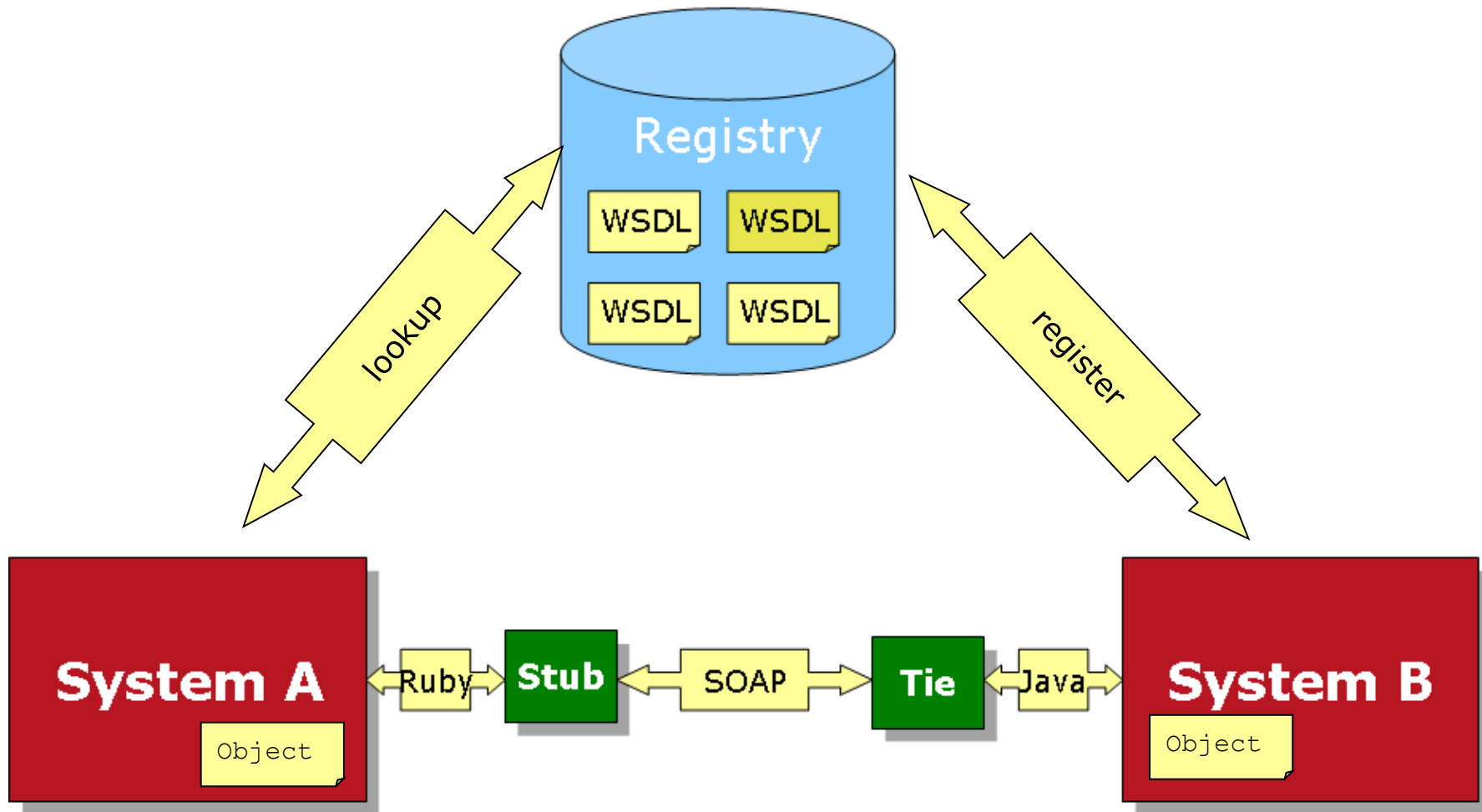
1.1

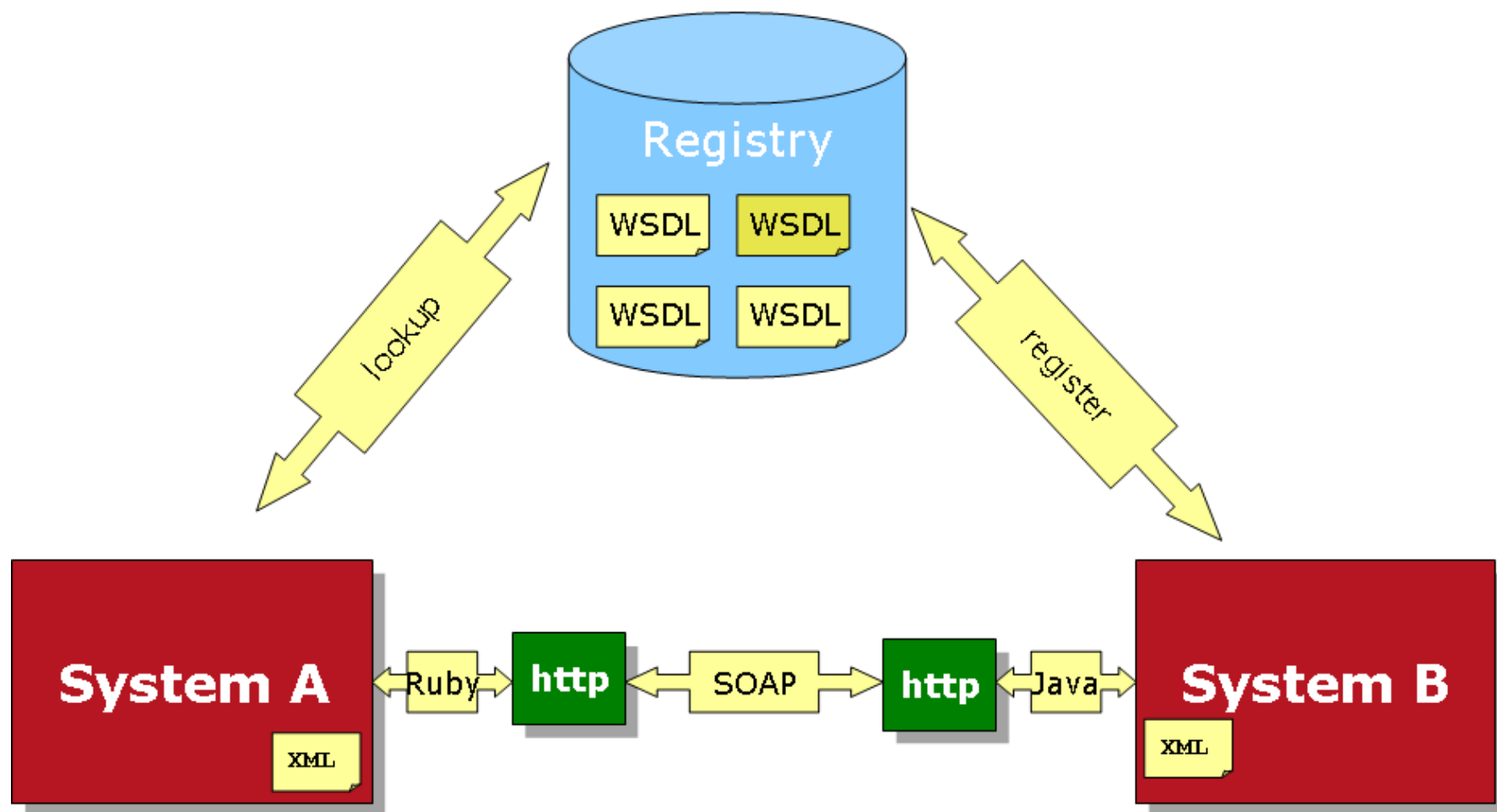
VON DER VERTEILTEN ANWENDUNG ZUM ESB

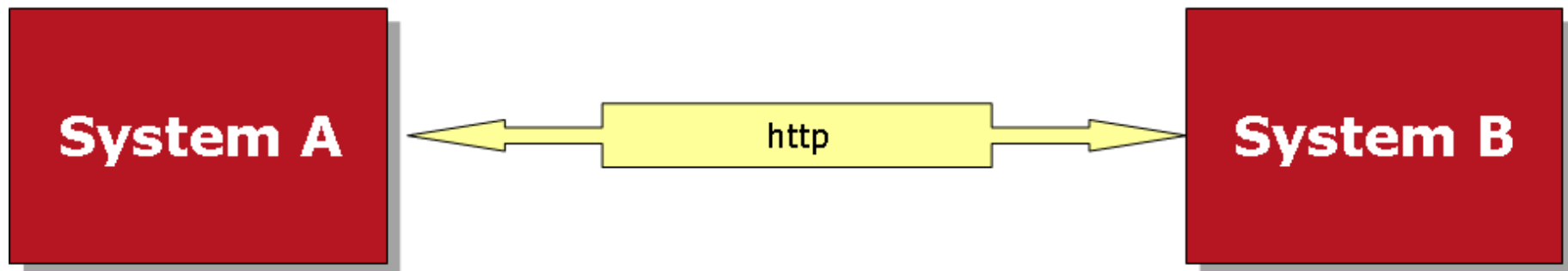


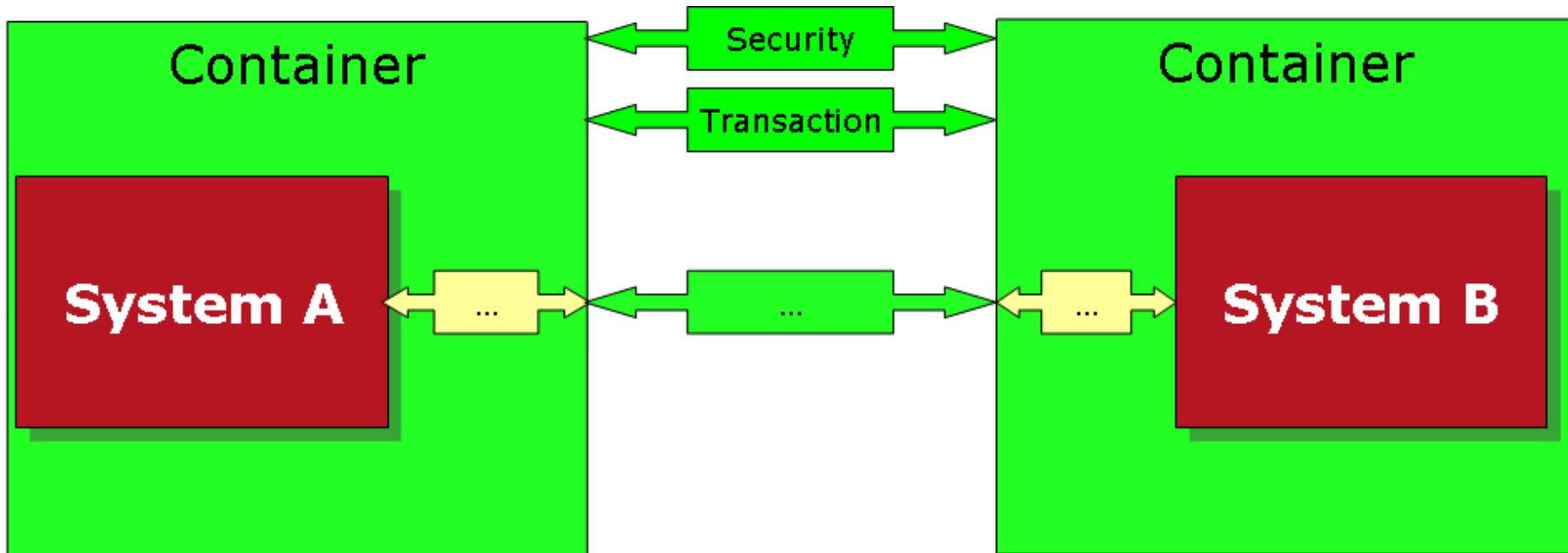


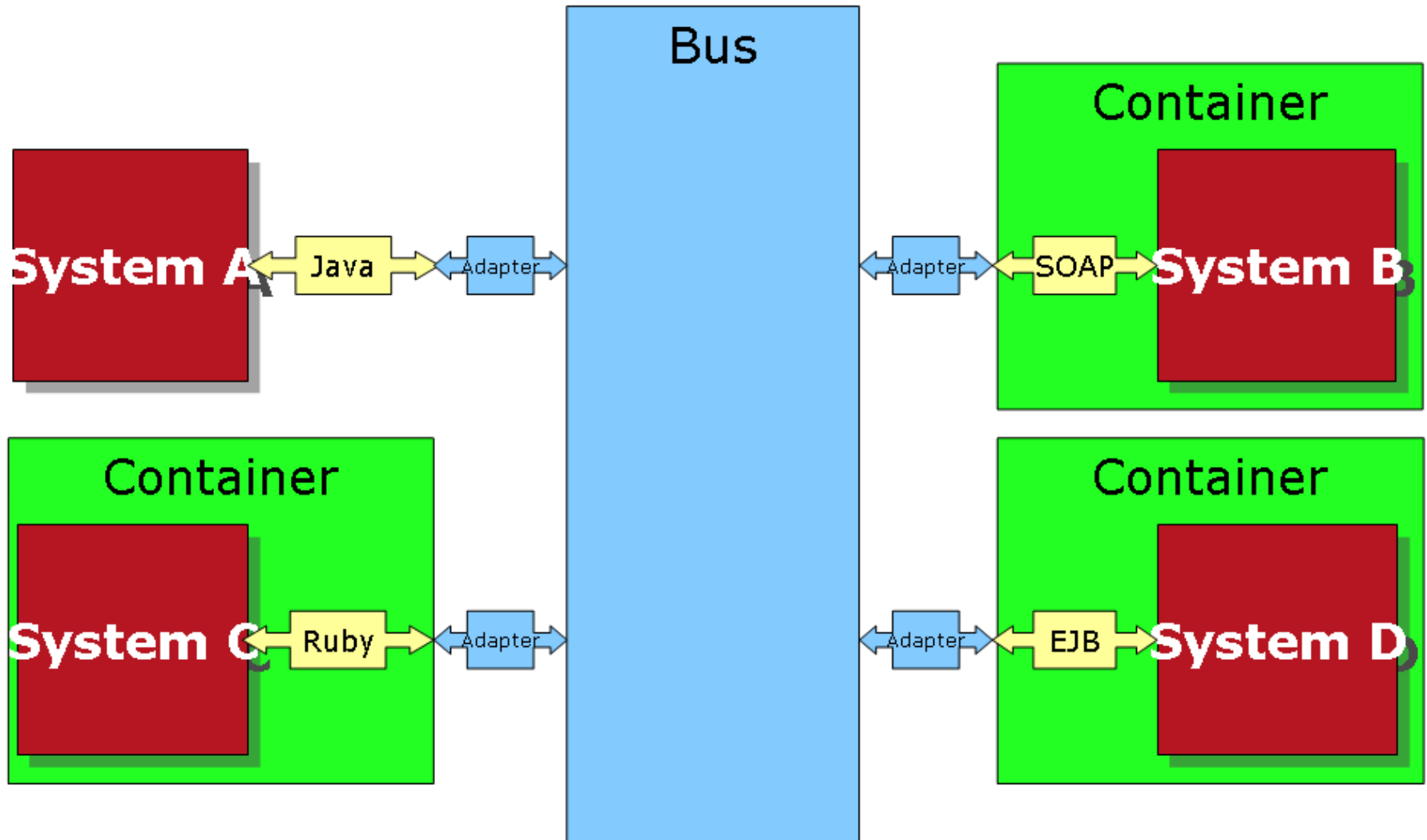


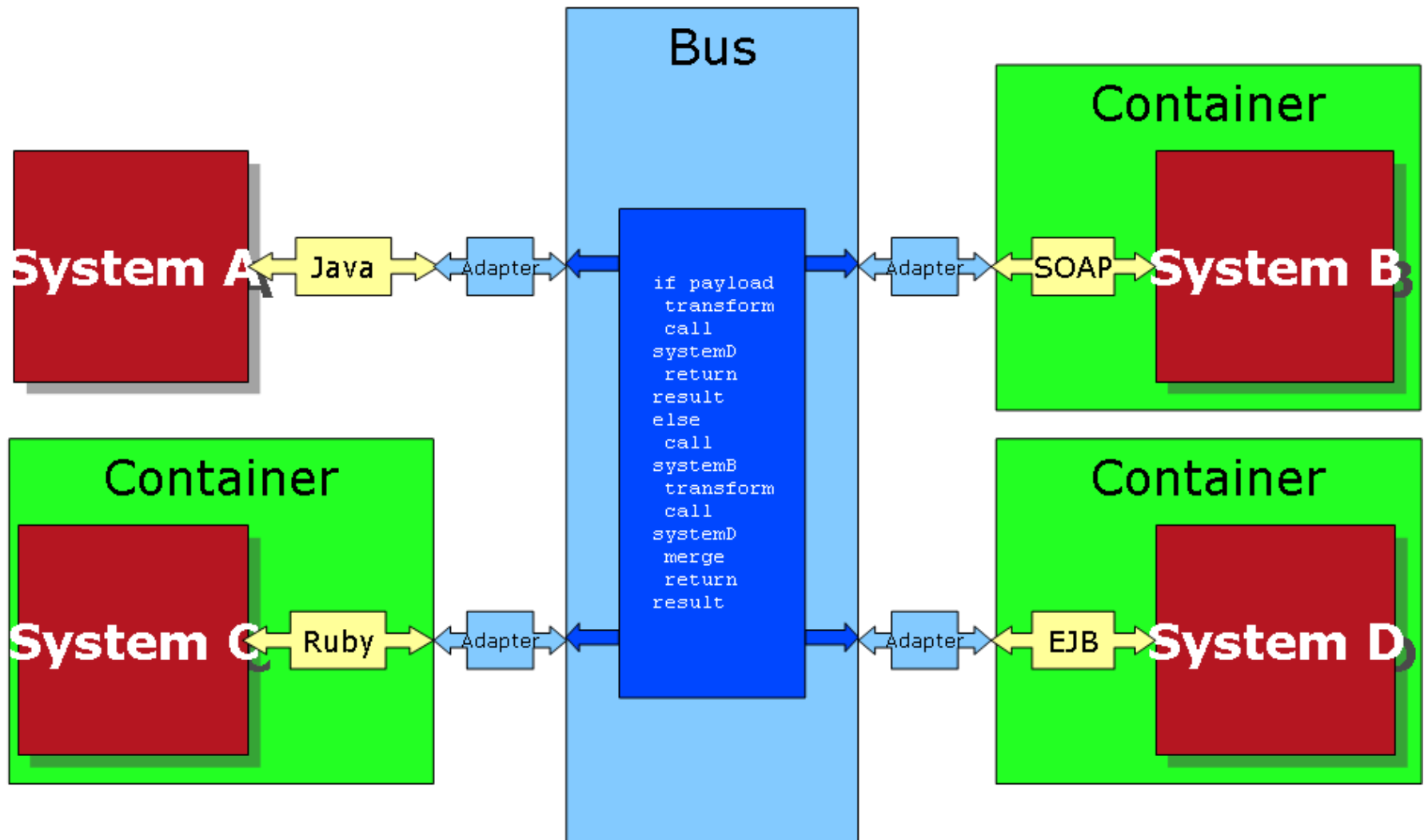


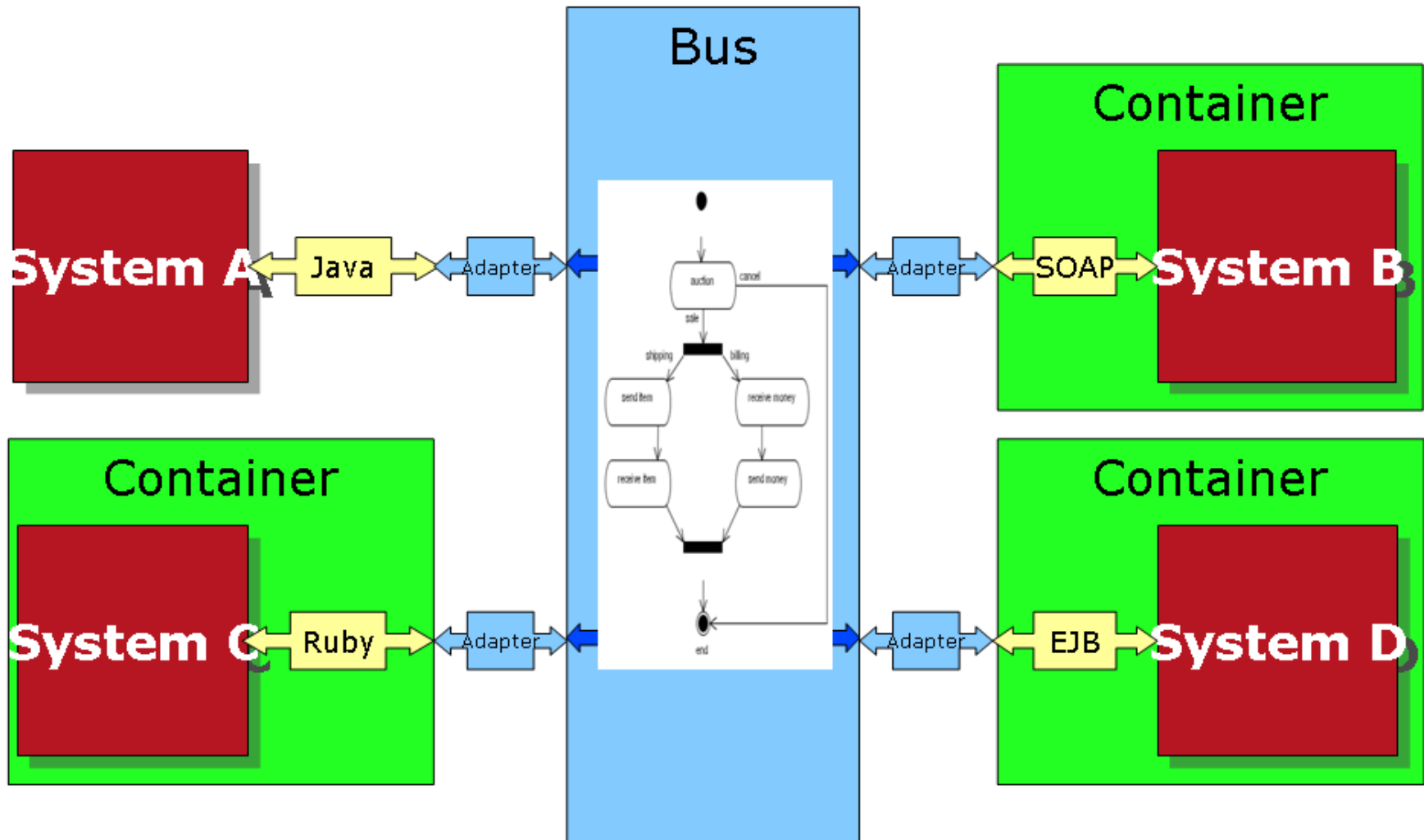










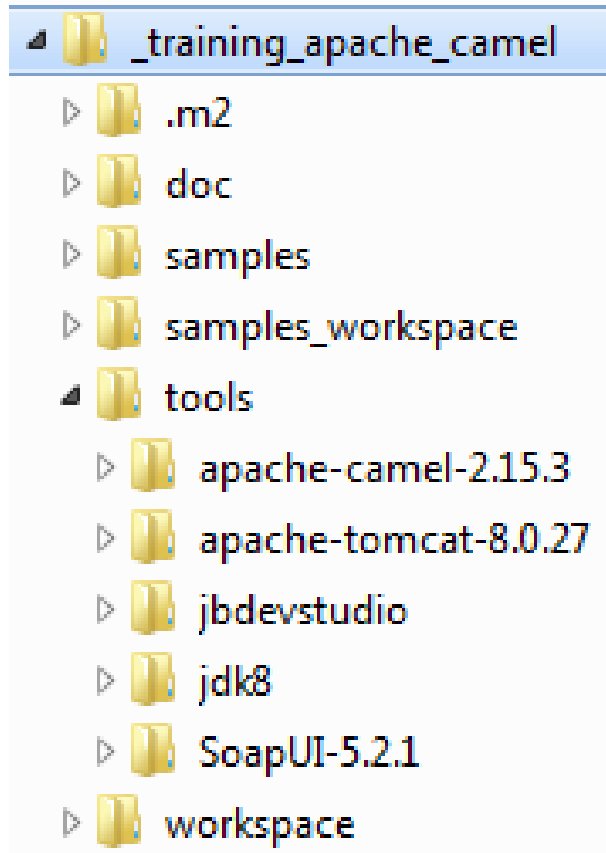


1.2

INSTALLATION UND WERKZEUGE

- Eclipse mit JBoss Tools und Fuse-Unterstützung
 - Freier Download mit JBoss Developer Lizenz
- Apache Maven
 - Dependency Management und Build-Werkzeug
 - Frei erhältlich von Apache.org
- Spring
 - Frei verfügbares Context and Dependency Injection Framework
 - Camel erweitert Spring um eine Domain Specific Language zur Routen-Definition
- Java
 - Programmiersprache zur Realisierung komplexer Geschäftslogik während der Routen-Ausführung
- JUnit
 - Ein Java-basiertes Testwerkzeug

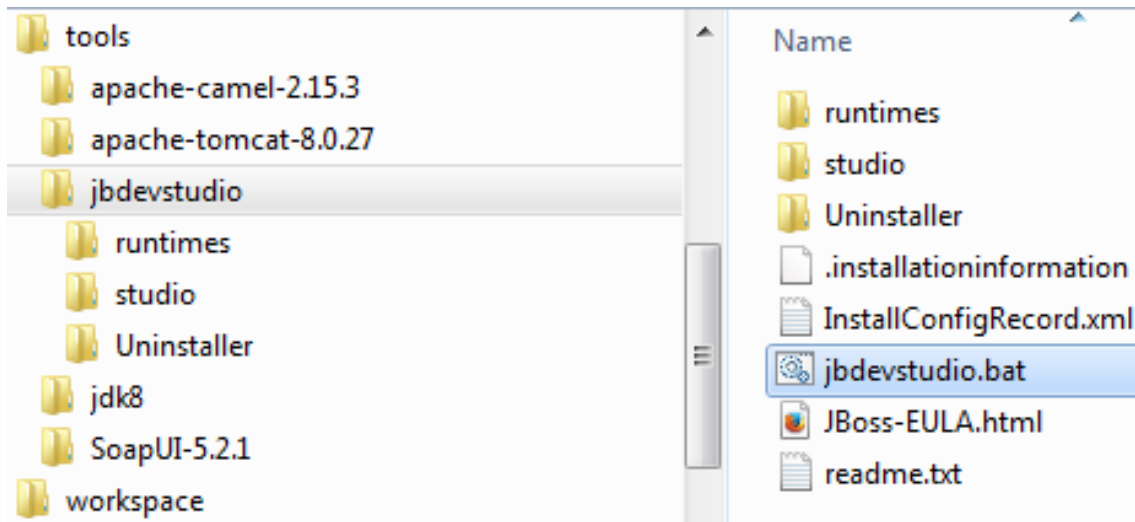
- Entpacken der zur Verfügung gestellten ZIP-Datei enthält alle notwendigen Werkzeuge

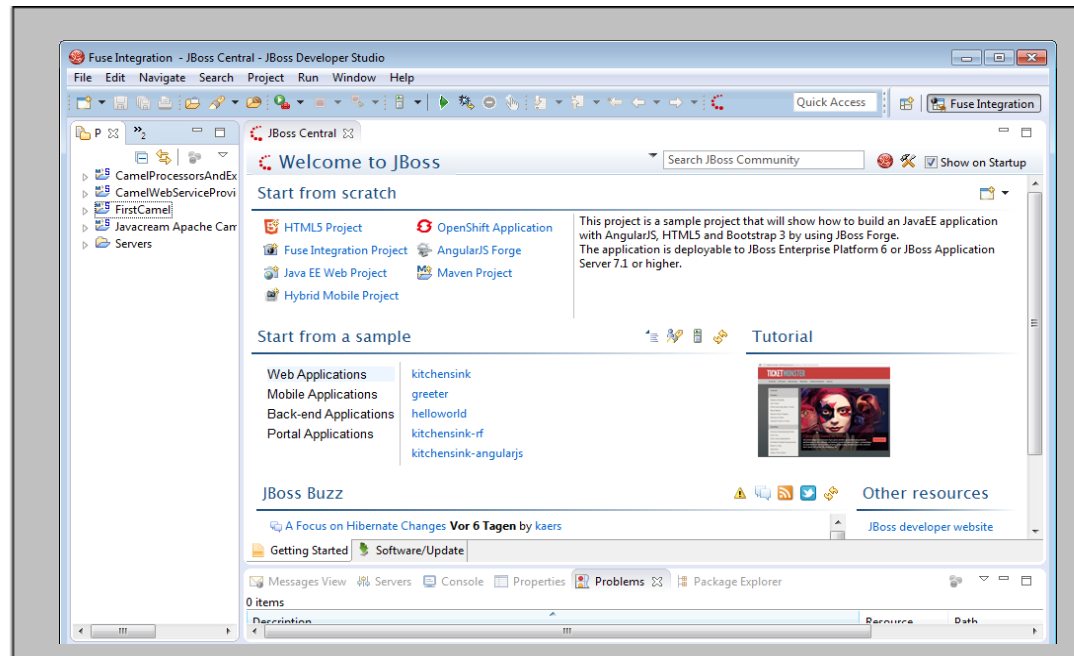


1.3

JBOSS FUSE TOOLS

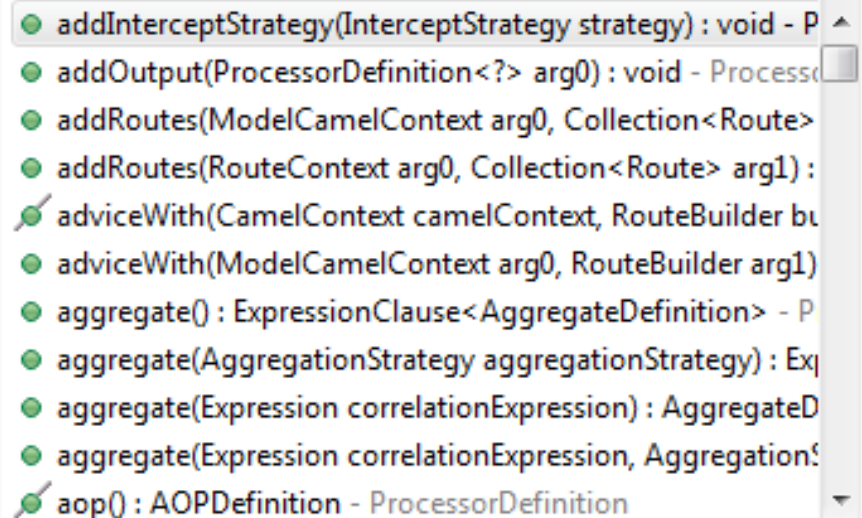
- `jbossdevstudio.bat`





- Eclipse-basierte IDE mit allen Java-relevanten Features
 - Code Assist
 - Compiler
 - Refactoring
 - ...

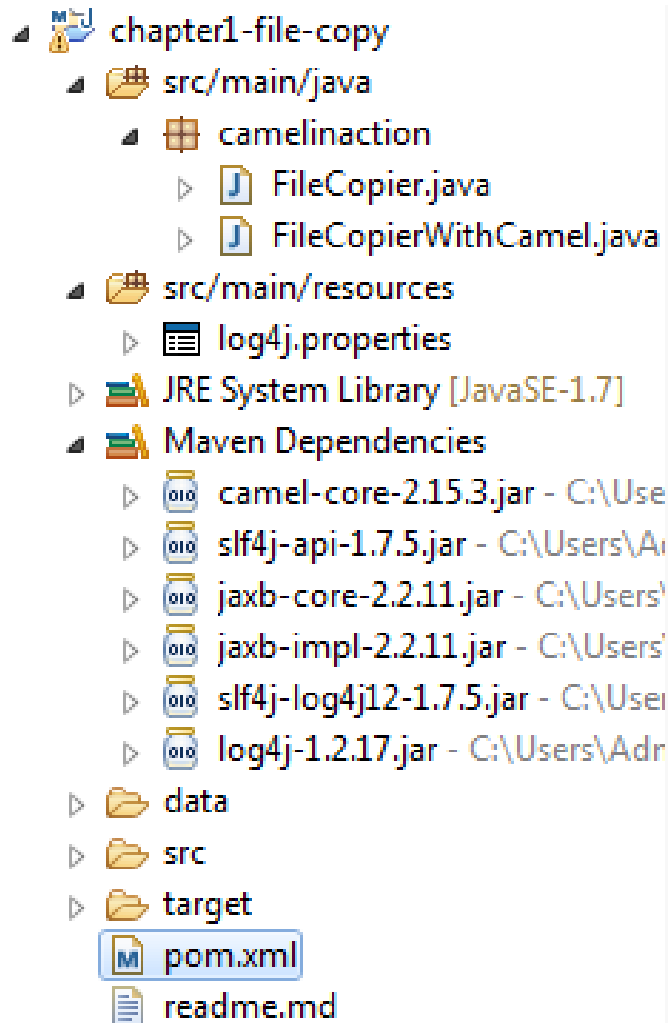

```
public class FileCopierWithCamel {  
  
    public static void main(String args[]) throws Exception {  
        // create CamelContext  
        CamelContext context = new DefaultCamelContext();  
  
        // add our route to the CamelContext  
        context.addRoutes(new RouteBuilder() {  
            public void configure() {  
                from("file:data/inbox?noop=true").to("file:data/outbox");  
            }  
        });  
  
        // start the route and let it do its work  
        context.start();  
        Thread.sleep(10000);  
  
        // stop the CamelContext  
        context.stop();  
    }  
}
```



- addInterceptStrategy(InterceptStrategy strategy) : void - P
- addOutput(ProcessorDefinition<?> arg0) : void - Process
- addRoutes(ModelCamelContext arg0, Collection<Route>
- addRoutes(RouteContext arg0, Collection<Route> arg1) :
- ✓ adviceWith(CamelContext camelContext, RouteBuilder bu
- adviceWith(ModelCamelContext arg0, RouteBuilder arg1)
- aggregate() : ExpressionClause<AggregateDefinition> - P
- aggregate(AggregationStrategy aggregationStrategy) : Ex
- aggregate(Expression correlationExpression) : AggregateD
- aggregate(Expression correlationExpression, AggregationS
- ✓ aop() : AOPDefinition - ProcessorDefinition

Press 'Ctrl+Space' to show Template Proposals

- Die notwendigen abhängigen Bibliotheken
 - werden in einem Maven-POM definiert
 - von einem Build-Prozess automatisch geladen
 - und mit dem Klassenpfad des Projekts synchronisiert



Dependencies

Filter:

Dependencies

camel-core (managed:2.15.3)
slf4j-log4j12 (managed:1.7.5)

Add...

Remove

Properties...

Manage...

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Configures the Camel Context-->

<beans xmlns="http://www.springframework.org/schema/beans"
        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">

    <choice>
        <alias>
            <bean>
            <beans>
            <description>
            <import>
            <!-- Bean - Inserts a bean tag
            # comment - xml comment
            <!-- Web Flow Executor - Inserts a Web Flow Executor tag
            <!-- Web Flow Registry - Inserts a Web Flow Registry tag
            # XSL processing instruction - XSL processing instruction
        </choice>
    </route>
</camelContext>
```

Element : alias
Defines an alias for a bean (which can reside in a different definition resource).

```
<beans xmlns="http://www.springframework.org/schema/beans"
  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">
    <!-- here is a sample which processes the input files
         (leaving them in place - see the 'noop' flag)
         then performs content based routing on the message using XPath -->
```

<> consumerTemplate

<> contextScan

<> dataFormats

<> endpoint

<> errorHandler

<> export

<> intercept

<> interceptFrom

<> interceptSendToEndpoint

<> jmxAgent

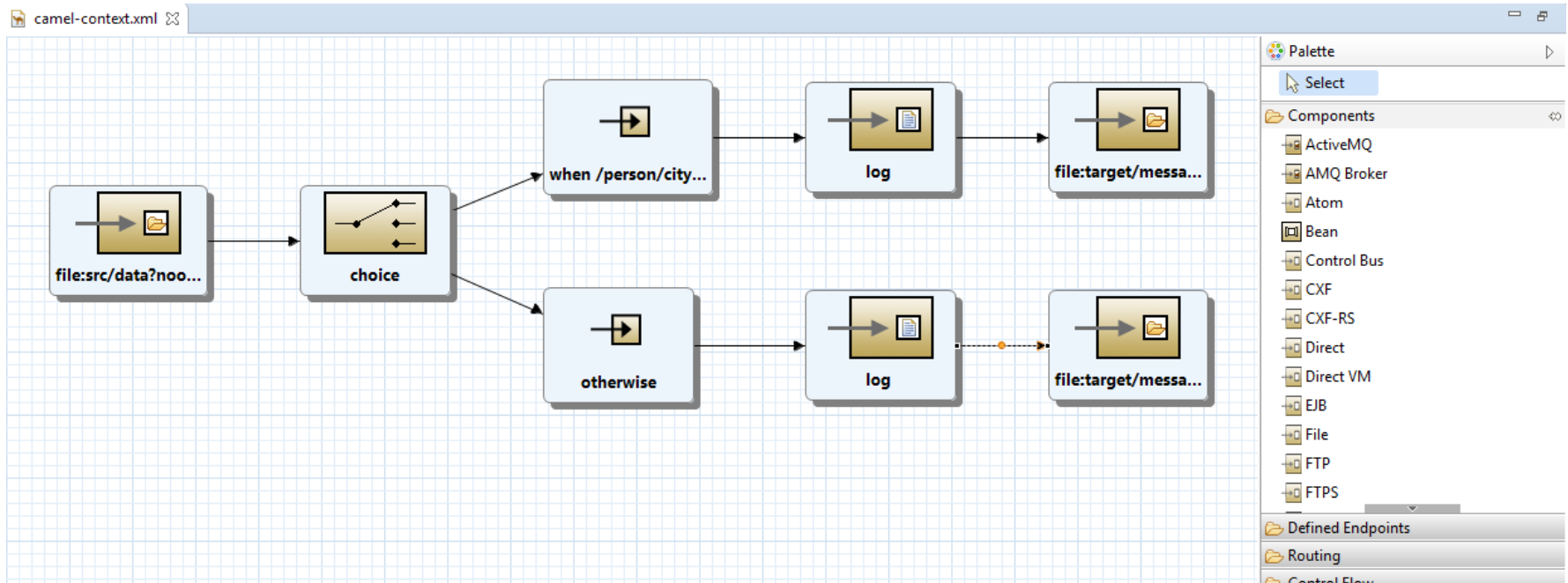
<> onCompletion

<> onException

Element : consumerTemplate

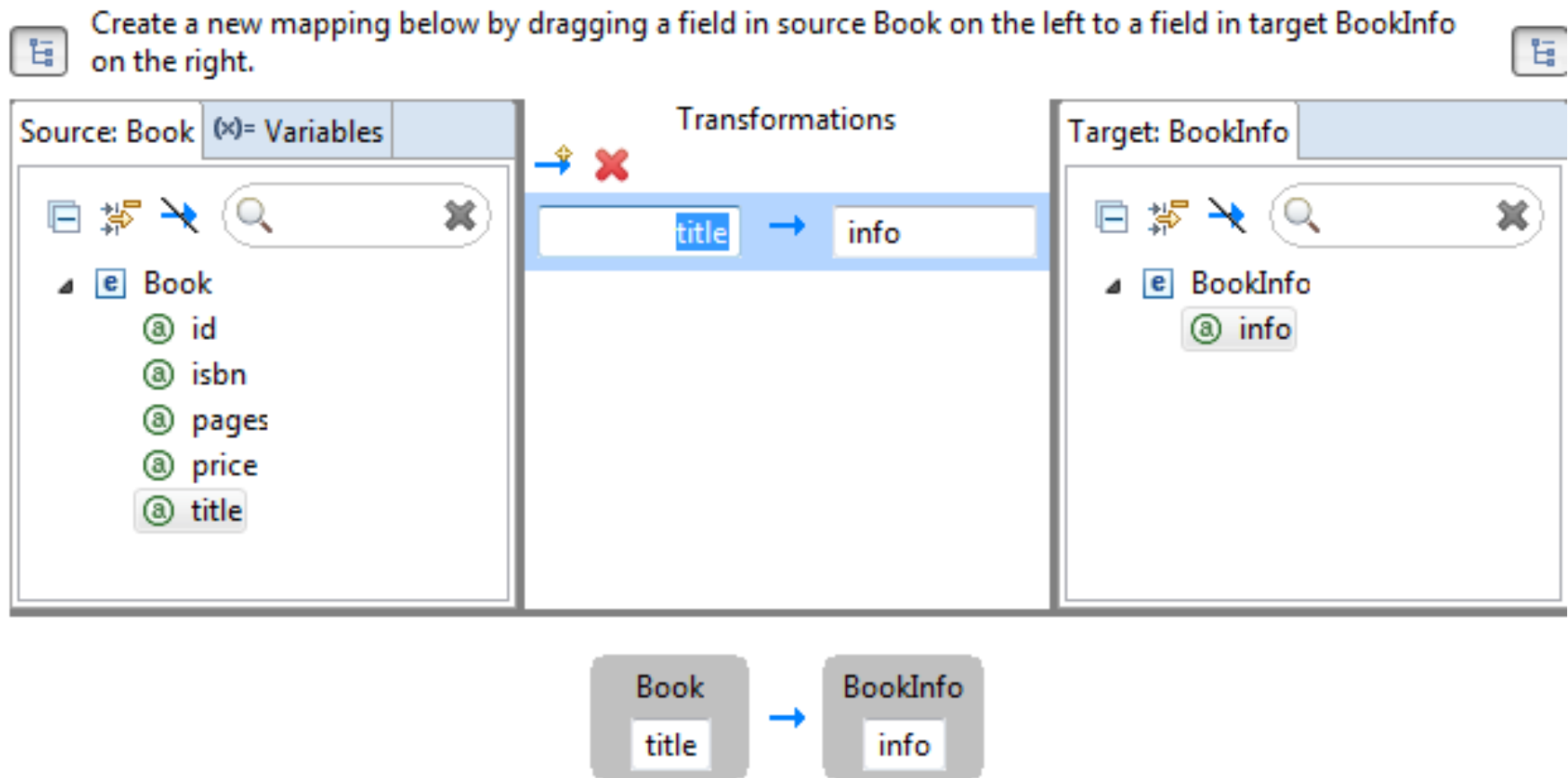
Content Model : ()

</beans>



- Transformation von Datenstrukturen
 - Java
 - XML
 - JSON
- Intern wird das Dozer-Mapping-Framework benutzt

Fuse Transformer in Eclipse



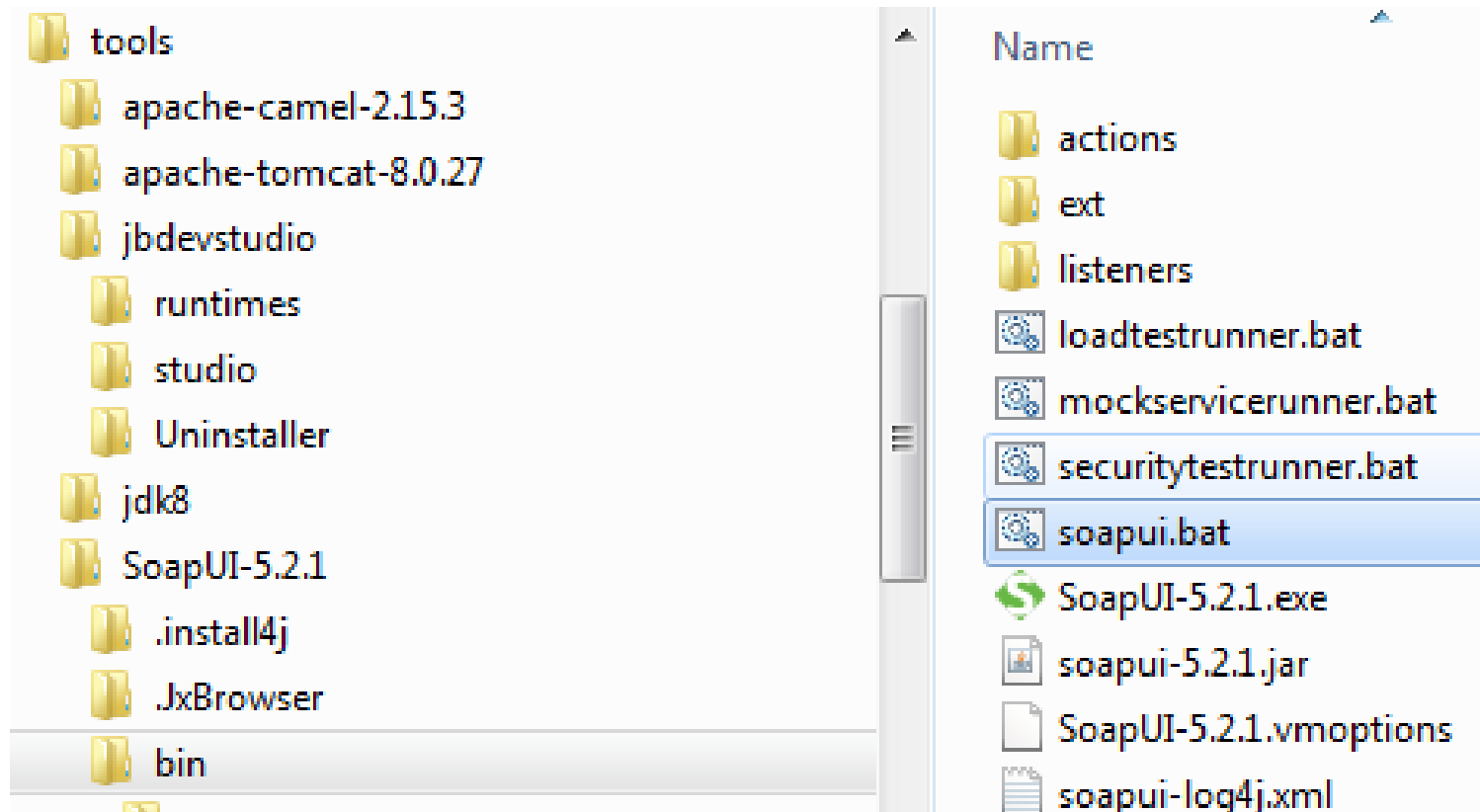
- Ein JMX-Client
 - Darstellung aller Informationen, die via JMX zur Verfügung gestellt werden
 - Aufruf von Management-Operationen
- Camel stellt selbst eine Vielzahl von Informationen und Operationen via JMX zur Verfügung
 - Routen
 - Überwachung
 - Administration
 - Komponenten
 - Konfiguration

1.4

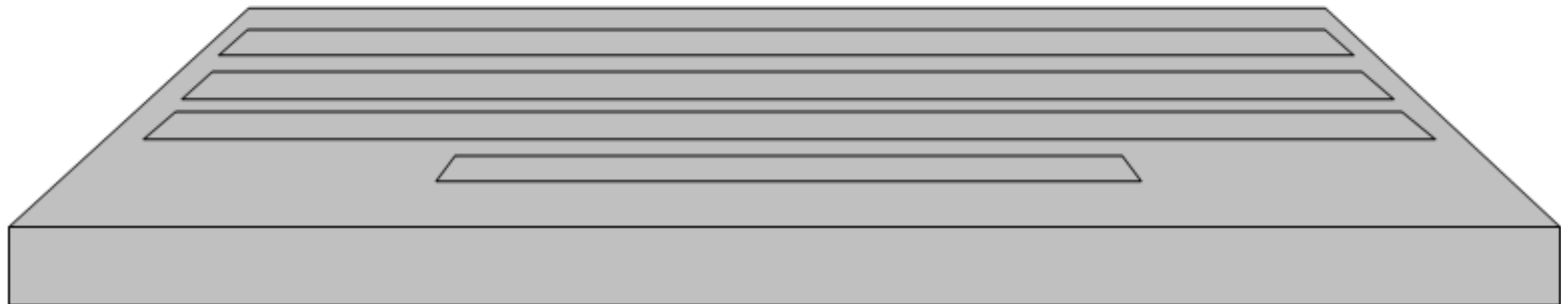
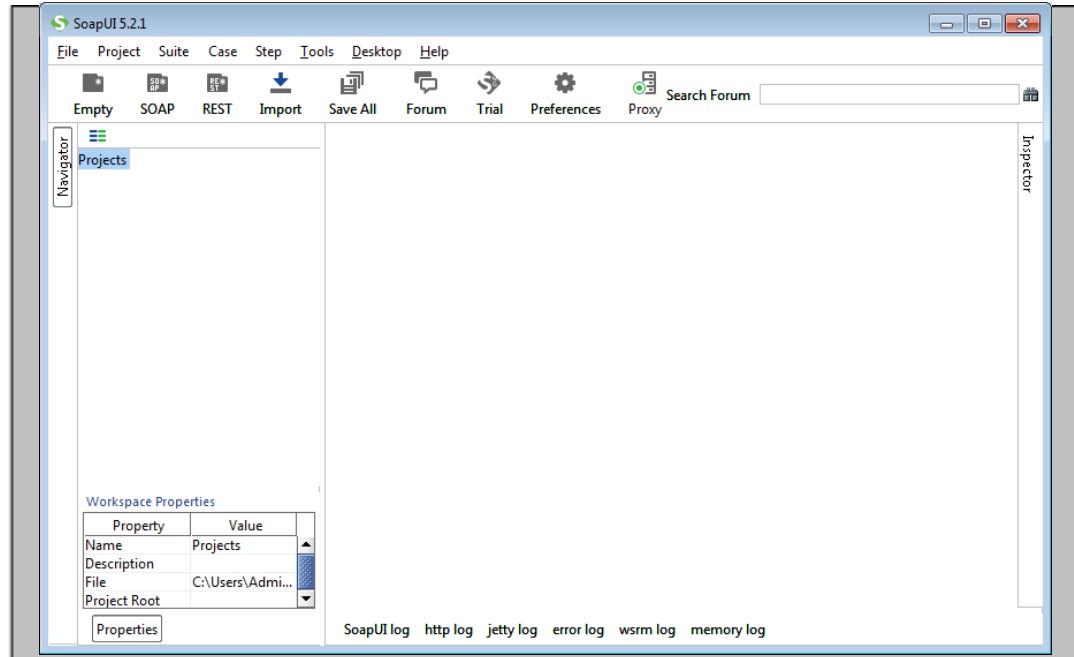
SOAP UI

- Testclient für Web Services
 - Freie Version
 - Kommerzielle Version mit einigen netten Erweiterungen
- Die SOAP-UI ist wie Eclipse in Projekte organisiert
- Jedes Projekt kann eine beliebige Menge von Web Services enthalten

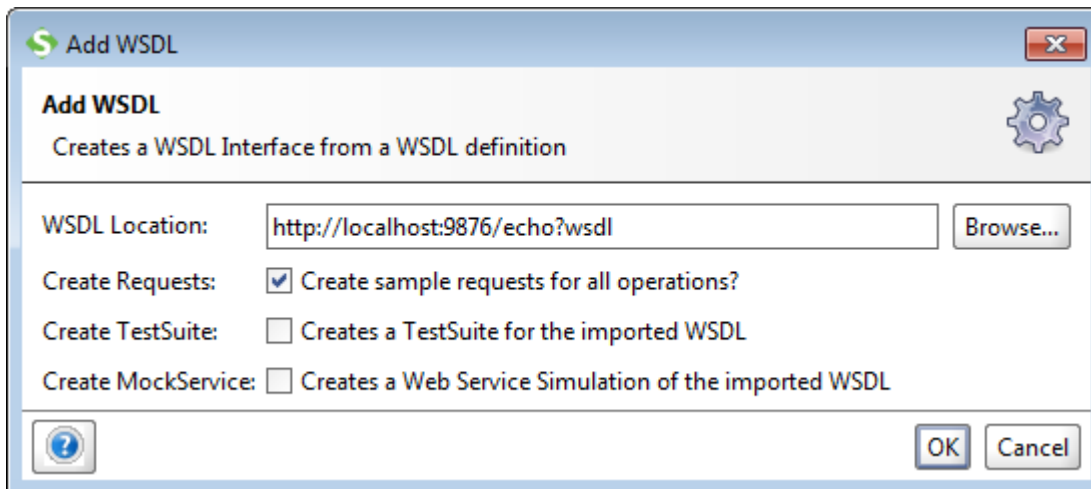
- `soapui.bat`



Die SOAP UI



- Anlegen eines Projekts
 - File - Create Empty Project
- Hinzufügen einer Web Service WSDL
 - Im Kontextmenü des angelegten Projekts: Add Wsdl
 - Abfrage der WSDL-Location
 - Dateisystem
 - http-Adresse
 - Im Beispiel: `http://HOST:9876/echo?WSDL`

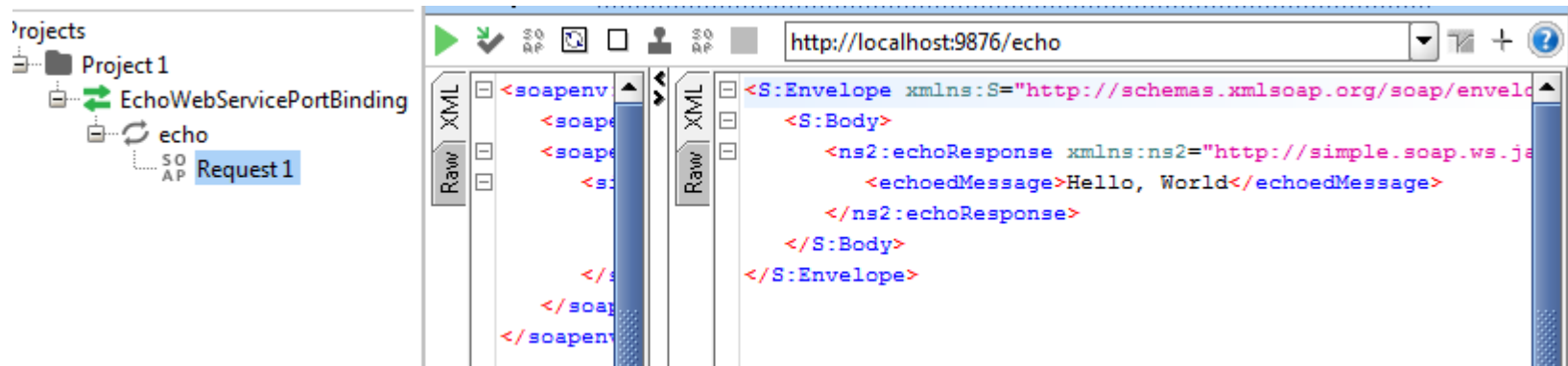


Requests und Response



The screenshot shows the Apache Camel IDE interface. On the left, the 'Projects' tree displays 'Project 1' with an 'EchoWebServicePortBinding' and an 'echo' endpoint. The 'Request 1' tab is selected. The main window shows the raw XML of the SOAP request sent to 'http://localhost:9876/echo'.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <sim:echo>
      <!--Optional:-->
      <echoMessage>Hello, World</echoMessage>
    </sim:echo>
  </soapenv:Body>
</soapenv:Envelope>
```



The screenshot shows the Apache Camel IDE interface. On the left, the 'Projects' tree displays 'Project 1' with an 'EchoWebServicePortBinding' and an 'echo' endpoint. The 'Request 1' tab is selected. The main window shows the raw XML of the SOAP response received from 'http://localhost:9876/echo'.

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:echoResponse xmlns:ns2="http://simple.soap.ws.java" >
      <echoedMessage>Hello, World</echoedMessage>
    </ns2:echoResponse>
  </S:Body>
</S:Envelope>
```

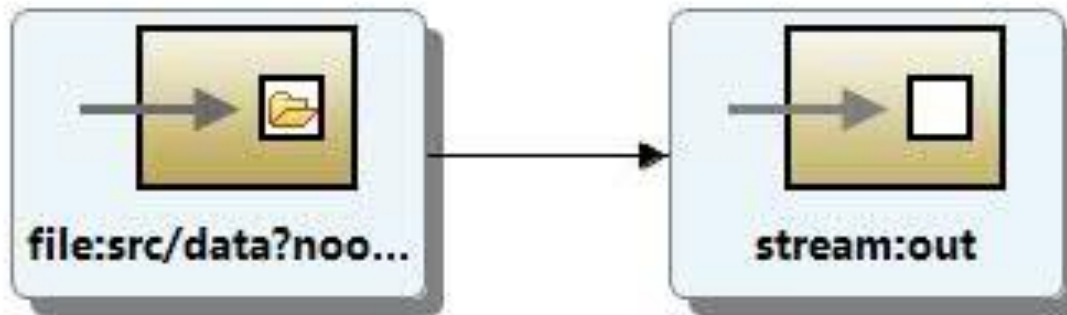
2

APACHE CAMEL

2.1

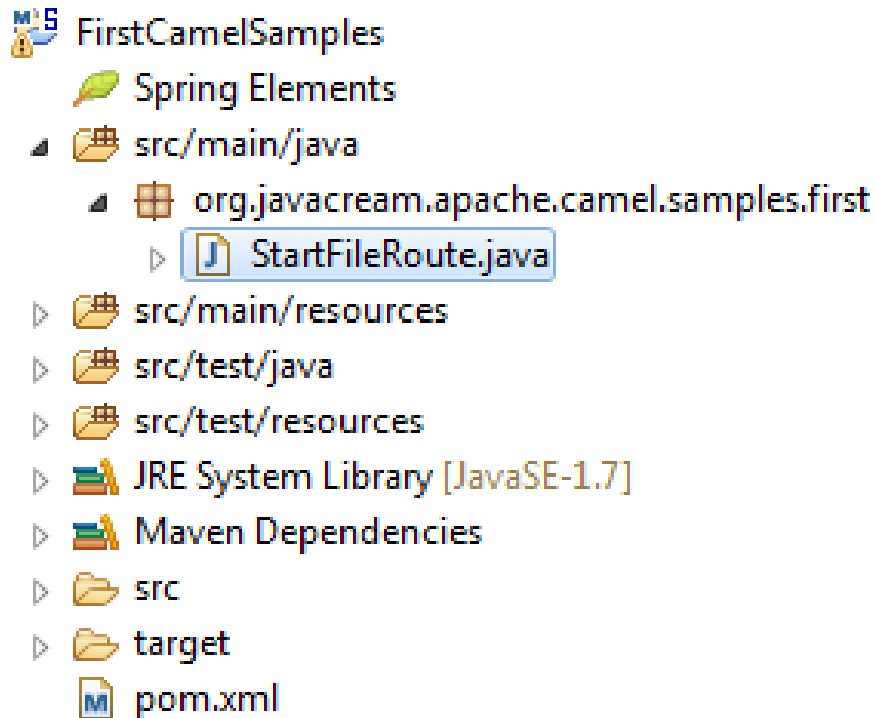
EIN ERSTES BEISPIEL













- Ausgabe eines Dateiinhalts auf die Konsole



```
<camelContext
xmlns="http://camel.apache.org/schema/spring">
    <route>
        <from uri="file:src/data?noop=true" />
        <to uri="stream:out" />
    </route>
</camelContext>
```

- StartFileRoute - Run As - Java Application



- ▲  org.javacream.camel.first.SpringRouteStarter [4016][Connected]
- ▶  MBeans
- ▲  Camel
 - ▲  camel-1
 - ▲  Endpoints
 - ▲  file
 -  src/data?noop=true
 - ▶  stream
 -  out
 - ▲  Routes
 - ▲  route1
 - ▶  file:src/data?noop=true

```
[main] SpringCamelContext INFO Apache Camel 2.
[main] ManagedManagementStrategy INFO JMX is enabled
[main] DefaultTypeConverter INFO Loaded 183 type
[main] SpringCamelContext INFO AllowUseOrigina
[main] SpringCamelContext INFO StreamCaching :
[main] FileEndpoint INFO Endpoint is cor
[main] FileEndpoint INFO Using default r
[main] SpringCamelContext INFO Route: route1 :
[main] SpringCamelContext INFO Total 1 routes,
[main] SpringCamelContext INFO Apache Camel 2.

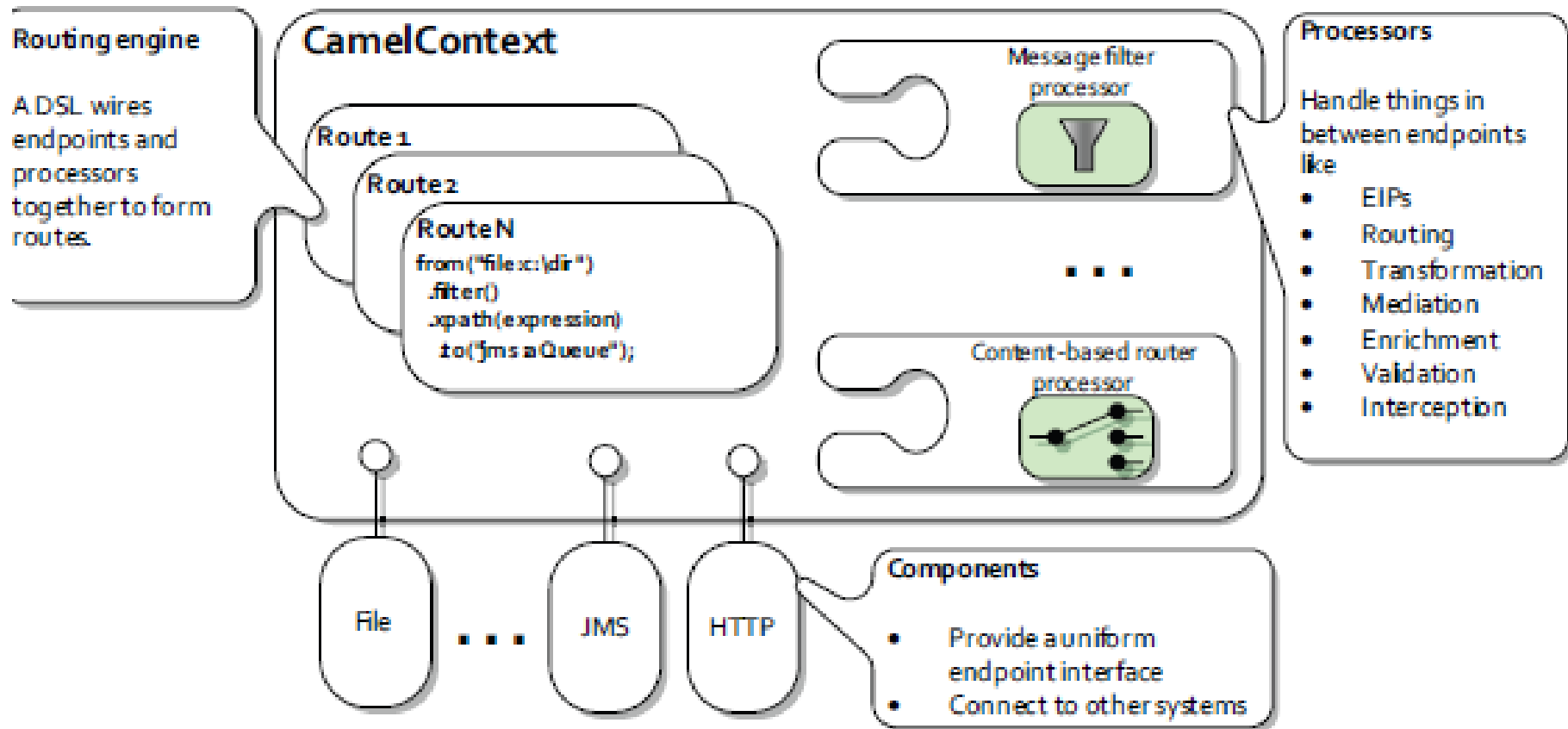
<?xml version="1.0" encoding="UTF-8"?>
<person user="james">
  <firstName>James</firstName>
  <lastName>Strachan</lastName>
  <city>London</city>
</person>
<?xml version="1.0" encoding="UTF-8"?>
<person user="hiram">
  <firstName>Hiram</firstName>
  <lastName>Chirino</lastName>
  <city>Tampa</city>
</person>
```

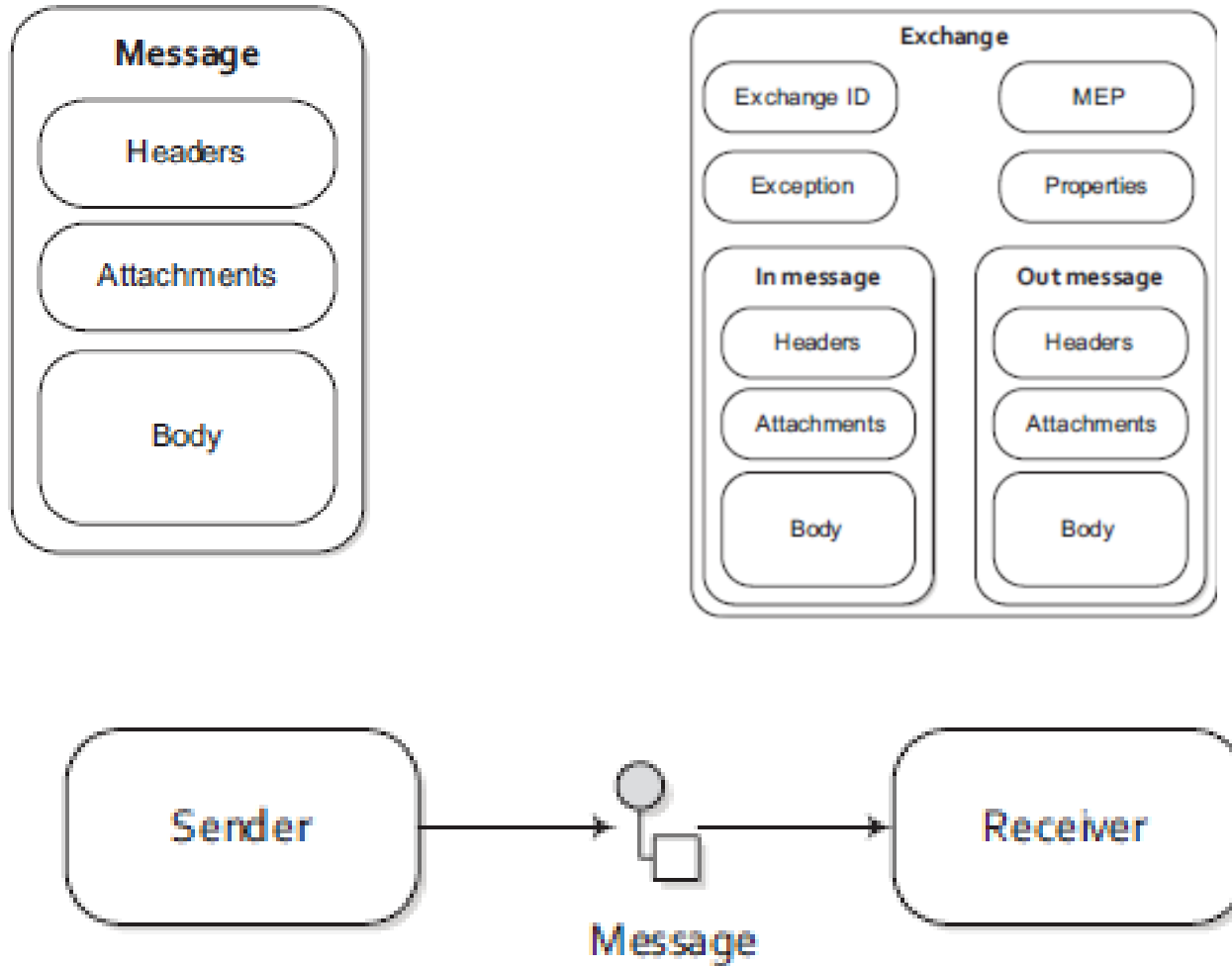
2.2

PRODUKTÜBERSICHT UND ARCHITEKTUR

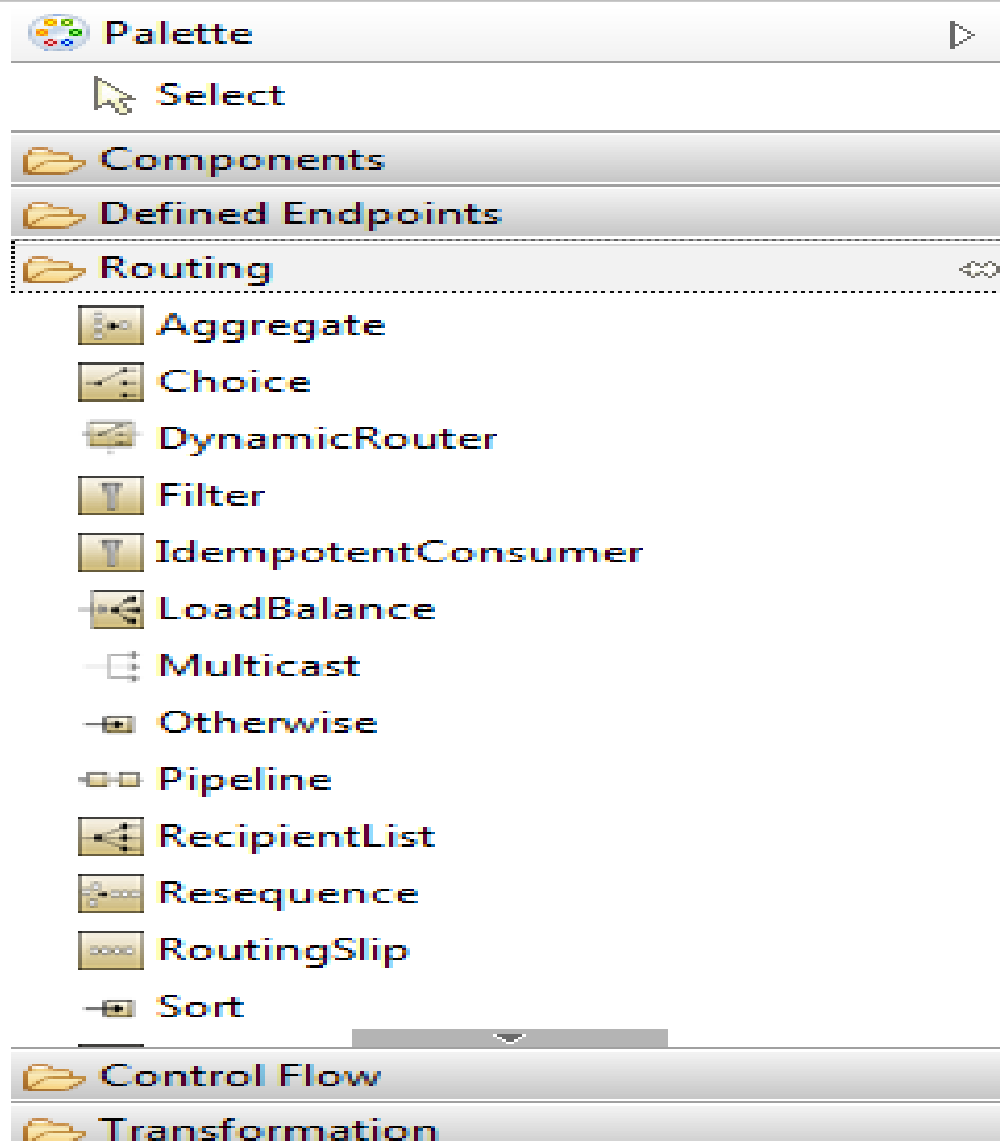
- „ein Open Source Integrations-Framework“
 - Offizielle Beschreibung
- kein vollständiges ESB-Produkt
 - enthält aber die notwendigen Elemente, um einen ESB zu realisieren
- eine Implementierung der Patterns der Enterprise Application Integration
- keine Business Process Engine
- ein Produkt, das von anderen Produkten benutzt wird
 - Apache ServiceMix
 - Apache MQ

- Routing und Mediation
- Domain-Specific Languages (DSLs)
 - Java ist zur Definition von Routen nicht gesetzt
- Implementierung der Enterprise Integration Patterns (EIPs)
 - z. B. Payload-abhängiges Routing
- Modulare Architektur mit leichtgewichtigem Kern
 - Umfangreiche Komponentenbibliothek ist in der Standard-Distribution enthalten
- Konsistentes Programmiermodell
 - POJO-Unterstützung (Plain Old Java Objects)
 - Einfache Konfiguration
- Test Werkzeuge
- Kompetente, aktive Community
 - Kommerzieller Support über Fuse/RedHat

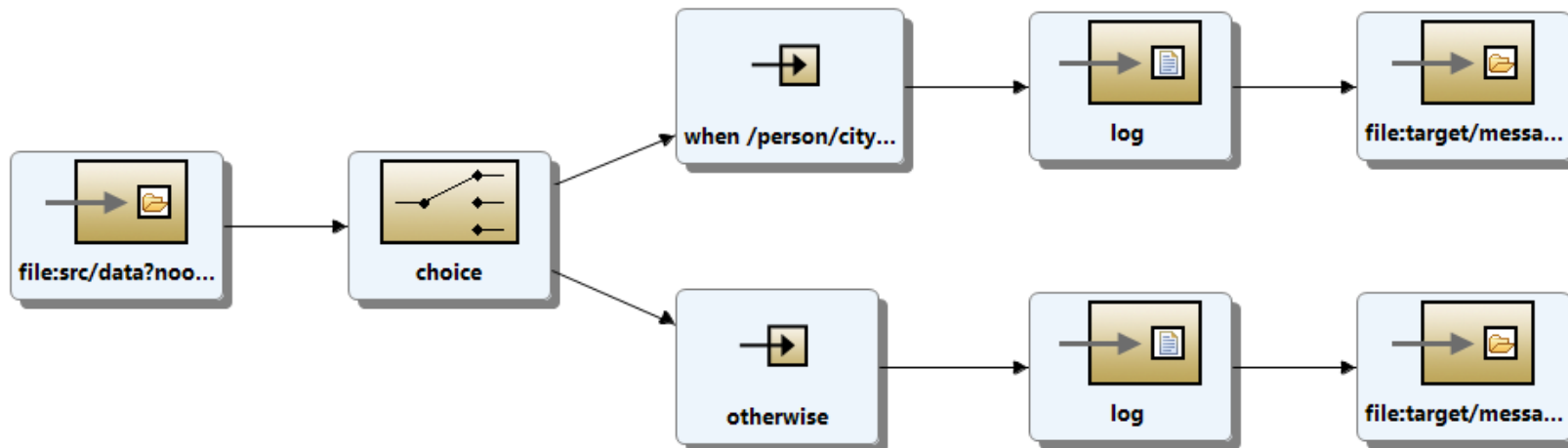




- Eine Sammlung gängiger „Patterns“
 - Eigentlich eine Sammlung von Symbolen mit semantischer Bedeutung
<http://www.enterpriseintegrationpatterns.com>
- IDE-Unterstützung
 - Palette mit Symbolen für Drag&Drop



- Die Engine arbeitet einen gerichteten Grafen ab
- Dieser definiert den Ablauf der Nachrichtenverarbeitung



- Camel unterstützt gängige Programmiersprachen zur Routen-Definition
 - Java
 - Groovy
 - Spring XML
 - Scala
- Die Prinzipielle Arbeitsweise ist für alle Sprachen identisch
 - Definition eines Inbound-Endpoints zum Einstieg in die Route
 - Angabe des zu verwendenden Protokolls
 - Weitere Konfigurationseinstellungen
 - Definition der Routen-Logik
 - Kann für erste Beispiele entfallen
 - Definition eines Outbound-Endpoints zum Ausstieg aus der Route
 - Angabe des zu verwendenden Protokolls
 - Weitere Konfigurationseinstellungen

- **Java DSL**

```
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"
```


2.3

CAMEL RUNTIME

- Standalone
 - Der Camel-Context wird als Java-Prozess innerhalb einer eigenen Main-Anwendung gestartet
- Embedded
 - Camel wird in einer beliebigen Java-Anwendung benutzt
- Als Anwendung innerhalb eines Applikationsservers
 - Typischerweise als WAR-Datei
 - Vorsicht: Camel öffnet je nach Endpoints eigene Socketverbindungen

- Der CamelContext wird in einer main-Methode erzeugt

```
import org.apache.camel.main.Main;
public class JavaRouteStarter {
    public static void main(String[] args) throws Exception
    {
        Main main = new Main();
        main.enableHangupSupport();
        main.addRouteBuilder(new RouteBuilder() {
            @Override
            public void configure() throws Exception {
                //Routen-Definition
            }
        });
        main.run(args);
    }
}
```

- Der `CamelContext` wird in einer Spring-Konfiguration definiert und durch Erzeugen des Spring-Kontextes erzeugt

```
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class SpringRouteStarter {
    public static void main(String[] args) throws Exception {
        new ClassPathXmlApplicationContext(
            "/META-INF/spring/camel-context.xml");
    }
}
```

- Hier wird der Camel-Kontext von einem Servlet-Container hochgefahren
- Meistens wird hier ein Servlet-Listener von Spring benutzt, der eine Spring-Konfiguration einlädt
 - Konfiguration in der web.xml
- Zusätzliche Servlets/Listener können beliebig ergänzt werden
 - Beispielsweise das Apache CXF-Servlet für Web Services

```
<web-app ...>
<!-- location of spring xml files -->
<context-param>
<param-name>contextConfigLocation</param-name>
    <param-value>classpath:camel-
config.xml</param-value>
</context-param>

<!-- the listener that kick-starts Spring -->
<listener>
<listener-
class>org.springframework.web.context.ContextLoaderLi
s
t
ener</listener-class>
</listener>

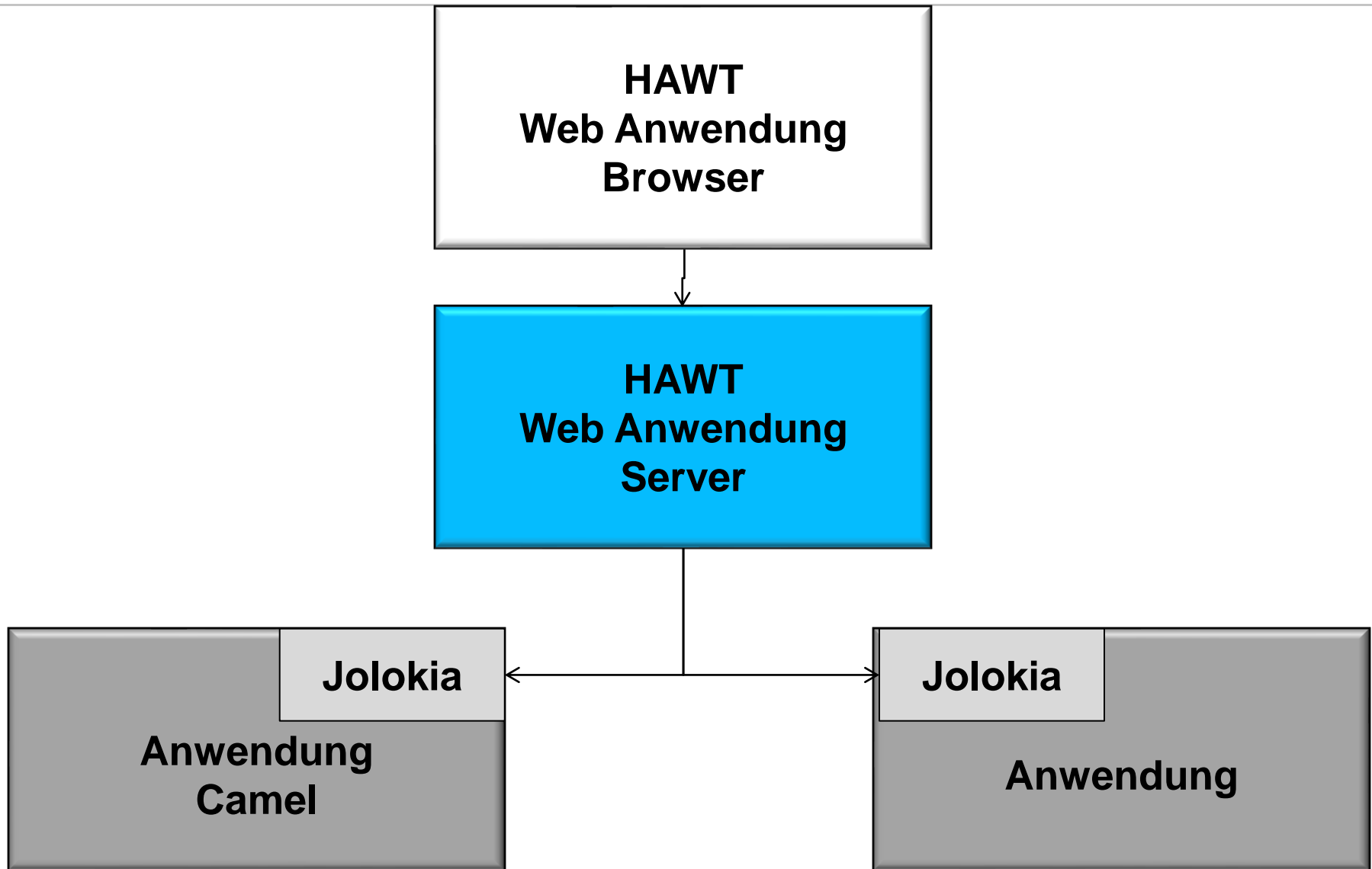
...
```

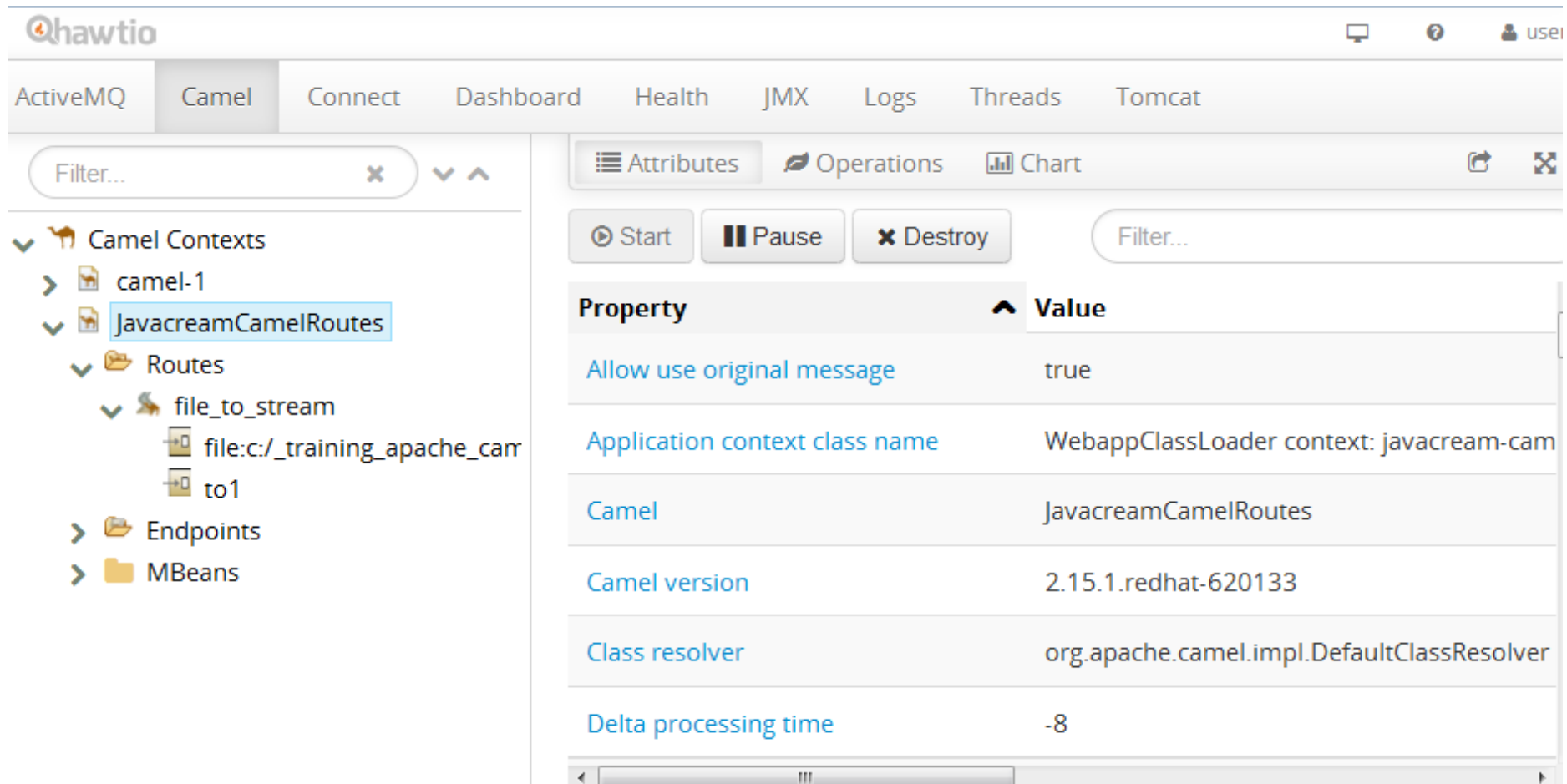
2.4

ÜBERWACHUNG UND ADMINISTRATION

- Camel ist auf JMX ausgerichtet
 - Administration
 - Zugriff auf Konfigurationseinstellungen
 - Aufruf steuernder Operationen
 - Überwachung
 - Interne Erfassung von Routen-spezifischen Metriken
- Camel ist keine „Integration Suite“
 - Komfortable Web-Konsolen etc. sind nicht (mehr) Bestandteil des Projekts
 - Diese werden jedoch von anderen Herstellern angeboten
 - kommerziell
 - aber auch Open Source

- Projekt der JBoss –Community
 - Java-basiert
- Allgemeines Überwachungswerkzeug mit PlugIns für verschiedene Produkte
 - unter anderem natürlich Camel
- Installation als Web Archiv in einem Web Server
- Der Zugriff auf Metrik-Informationen erfolgt über Jolokia
 - Jolokia selber ist eine von Camel völlig unabhängige Anwendung, die JMX-Informationen über eine REST-Schnittstelle zur Verfügung stellt





The screenshot shows the Hawtio web console interface. The top navigation bar includes tabs for ActiveMQ, Camel (selected), Connect, Dashboard, Health, JMX, Logs, Threads, and Tomcat. Below the navigation bar, there is a filter input field and a tree view of Camel contexts. The tree view shows the following structure:

- Camel Contexts
 - camel-1
 - JavacreamCamelRoutes (selected)
 - Routes
 - file_to_stream
 - file:c:/_training_apache_carr
 - to1
 - Endpoints
 - MBeans

The right pane displays the configuration for the selected context, JavacreamCamelRoutes. It includes tabs for Attributes (selected), Operations, and Chart. Below the tabs are buttons for Start, Pause, and Destroy, followed by another filter input field. The main area shows a table of properties:

Property	Value
Allow use original message	true
Application context class name	WebappClassLoader context: javacream-cam
Camel	JavacreamCamelRoutes
Camel version	2.15.1.redhat-620133
Class resolver	org.apache.camel.impl.DefaultClassResolver
Delta processing time	-8

3

PROGRAMMIERUNG

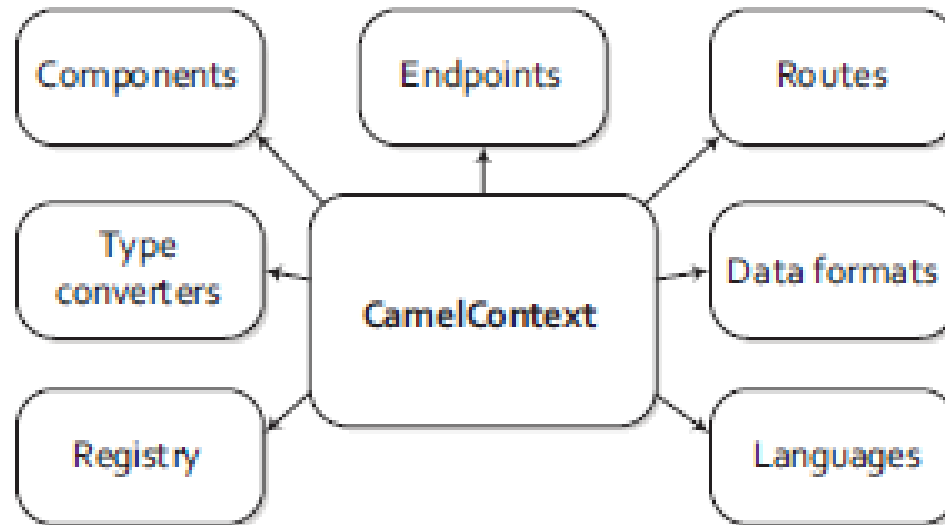
3.1

ELEMENTE

- Endpoints
 - Definieren die Endpunkte einer Route, über die Nachrichten ausgetauscht werden
- Processors
 - Eine Route besteht aus einzelnen Processors
 - Die untereinander Nachrichten austauschen
 - Processor ist ein abstraktes Konzept
- Komponenten
 - Komponenten definieren die konkrete Ausprägung eines Verhaltens
 - Endpoints
 - Processor-Implementierungen
 - Nachrichtenverarbeitung
 - Ablaufsteuerung
 - Transformationen
- Routen
 - Definieren einen Ablauf einer Nachrichtenverarbeitung

- Routen
 - verbinden Endpoints
 - Ein eingehender Endpoint wird mit einem Ausgang verknüpft
 - definieren den Pfad, den ein Exchange während der Ausführung der Route durchläuft
 - sind konfigurierbar
 - definieren einen Ausführungskontext
 - werden von einem Entwickler konkret programmiert
 - Routen sind nicht nur einfache „Palette-Drag&Drop“-Diagramme!
 - Damit sind für Routen alle Richtlinien der Software-Qualität zu berücksichtigen
 - Versionierung
 - Dokumentation
 - DRY & KISS
 - „Don't Repeat Yourself“, „Keep it simple“

- Dieser hält alles zusammen



- Realisierung
 - Simple HashMap
 - Spring Context
 - OSGi Service Registry

- **Elementare Typen sind**
 - `CamelContext` und `DefaultCamelContext`
 - `RouteBuilder`
 - `Processor-Interface`
 - Ein `Processor` verarbeitet Nachrichten
 - `Predicate-Interface`
 - Ein `Predicate` prüft Bedingungen und steuert damit den Ablauf der Route
 - `Component-Implementierungen`
 - Konkrete Endpoints
 - Transformer
 - EIP-Implementierungen wie Splitter und Aggregator
- Der `RouteBuilder` stellt ein fluentes API zur Verfügung, um programmatisch den Graphen der Route zu definieren

```
CamelContext context = new DefaultCamelContext();
ConnectionFactory connectionFactory =
    new
    ActiveMQConnectionFactory("vm://localhost");
context.addComponent("jms",
    JmsComponent.jmsComponentAutoAcknowledge(connectionFactor
y));
context.addRoutes(new RouteBuilder() {
    public void configure() {

        from("ftp://orders.com/?username=u&password=pwd").
        to("jms:incomingOrders");
    }
});
context.start();
```

- Erweiterung der Spring-Konfiguration um den camel-Namespace

```
<beans xmlns="http://www.springframework.org/schema/beans"
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-
3.0.xsd
http://camel.apache.org/schema/spring
http://camel.apache.org/schema/spring/camel-spring.xsd">
...
<camelContext
xmlns="http://camel.apache.org/schema/spring"/>
</beans>
```

Die Route wird innerhalb des `CamelContexts` als hierarchischer Graph definiert

```
<camelContext
xmlns="http://camel.apache.org/schema/spring">
<route>
<from
uri="ftp://orders.com/?username=u&password=pwd"/>
<to uri="jms:incomingOrders"/>
</route>
</camelContext>
```

- RouteBuilder-Implementierungen werden innerhalb eines Spring-Contextes definiert

- **Statisch**

```
<camelContext  
  xmlns="http://camel.apache.org/schema/spring">  
  <routeBuilder ref="ftpToJmsRoute"/>  
</camelContext>
```

- **Dynamisch**

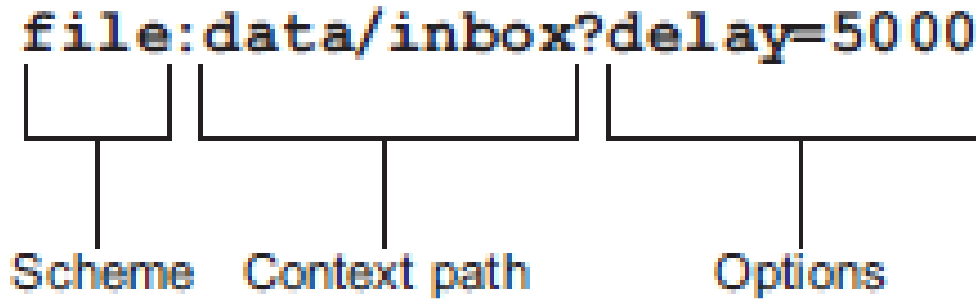
```
<camelContext  
  xmlns="http://camel.apache.org/schema/spring">  
  <packageScan>  
    <package>org.javacream.routes</package>  
  </packageScan>  
</camelContext>
```

3.2

ENDPOINTS

- Apache Camel enthält einen ganzen Satz von fertigen Endpoints
 - Eigene Komponenten können bei Bedarf implementiert werden
- Endpoints definieren die Einstiegs- und Ausstiegspunkte der Routen
- Jede Komponenten-Implementierung definiert einen Satz von Eigenschaften
 - Aus der Sicht eines Programmierers definieren die Komponenten ein Programmier-API
 - Dieses wird mit den von Camel zur Verfügung gestellten Sprachen benutzt
 - Convention over Configuration
 - Nicht speziell definierte Werte werden mit einem „Reasonable Default“ vorbelegt
- Endpoints erzeugen oder konsumieren Exchanges
 - Dabei werden Endpoint-spezifische Header gesetzt
 - file-Endpoint mit Property „CamelFileName“
 - Namen der Properties sind der Endpoint-Dokumentation zu entnehmen

- Eine Übersicht gebräuchlicher Camel-Komponenten umfasst:
 - file
 - ftp/sftp
 - jms
 - jdbc
 - Web Service
 - soap
 - rest
 - bean
 - Aufruf einer Spring-Bean-Methode
 - direct
 - Aufruf einer Route innerhalb des selben Kontextes
- Aktuell Liste Bestandteil der Online-Dokumentation



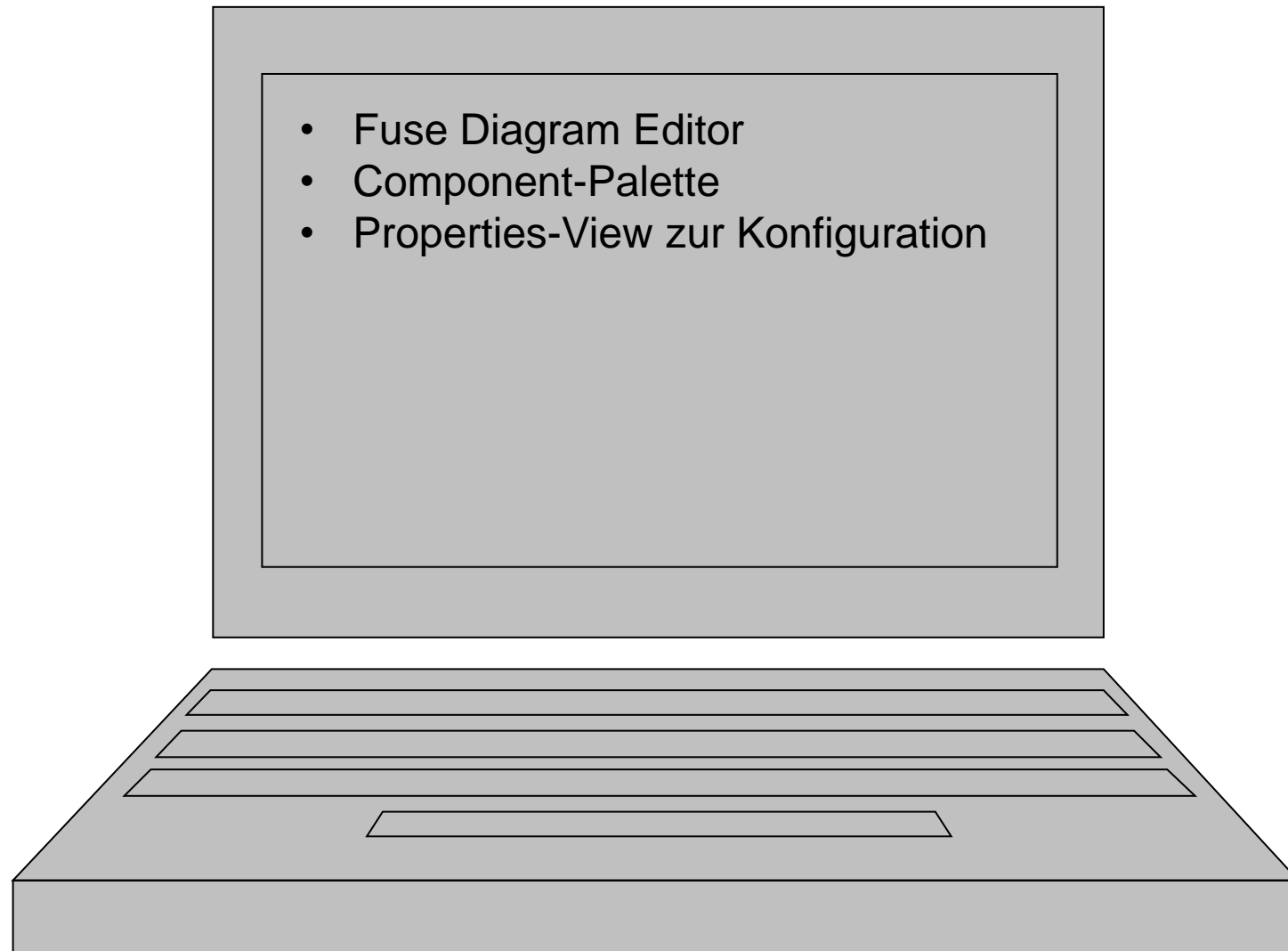
- Das Schema identifiziert die zu verwendende Komponente
 - Diese wird über konfiguriert über
 - den Context Path und
 - die Optionen

```
//Erzeugen einer ConnectionFactory
ConnectionFactory connectionFactory =
    new
    ActiveMQConnectionFactory("vm://localhost");
context.addComponent("jms",

//Registrieren unter der jms-URI
JmsComponent.jmsComponentAutoAcknowledge(connectionFac
tory));
```

- Zur Bereitstellung von WebServices-Endpoints benutzt Camel die CXF-Komponente
 - Apache CXF ist ein von Camel unabhängiges Projekt, das eine Web Services Runtime implementiert
- Dabei implementiert die Route den Web Service in verschiedenen Ausprägungen
 - Der Outbound Endpoint wird als Web Service zur Verfügung gestellt
 - Camel realisiert damit Web Services Fassaden
 - Bridge zwischen Web Services
 - Proxy-Server aus Firewall-Gründen
 - Realisierung zentraler Dienste wie Logging, Auditing, Authentifizierung
 - Die Route implementiert durch Filter/Transformationen eine echte Geschäftslogik
 - Vorsicht: Dies ist sicherlich nicht der eigentliche Sinn einer Route!

- ein `Producer` erzeugt einen `Exchange`
- ein `Consumer` konsumiert einen `Exchange`
 - Event Driven
 - Polling
- Endpoints sind häufig in beiden Ausprägungen vorhanden
 - Der File Endpoint als Consumer liest Dateien
 - Der File Endpoint als Producer schreibt Dateien



3.3

PROCESSORS

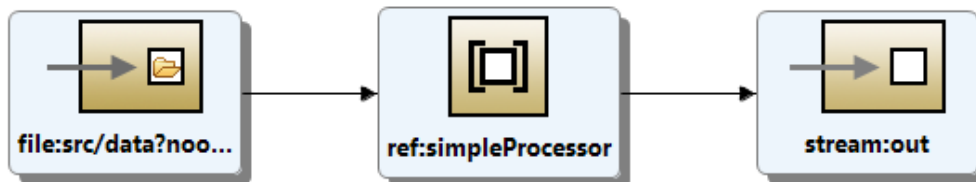
- Ein eigener Prozessor implementiert die Camel_Schnittstelle
`Processor`
- Der Prozessor hat dabei Zugriff auf das `Exchange`-Objekt
 - `Properties`
 - `CamelContext`
 - In- und Out-Message
 - Out-Message nur für synchrone Routen!
 - Messages haben
 - `Header`
 - `Body`
 - `Attachment`
- Prozessoren
 - werden erzeugt
 - der Routen-Definition eingetragen
 - können beliebig verkettet werden

■ Definition in XML

```
<camelContext id="JavacreamCamelRoutes"
  xmlns="http://camel.apache.org/schema/spring">
  <route id="file_to_stream">
    <from uri="file:src/data?noop=true" />
    <process ref="simpleProcessor" />
    <to uri="stream:out" />
  </route>
</camelContext>
```

■ Definition in Java

```
from("file:src/data?noop=true") .process(new
SimpleProcessor()) .to("stream:out");
```




```
public class SimpleProcessor implements Processor {

    @Override
    public void process(Exchange exchange) throws Exception {
        System.out.println(exchange.getProperties());
        System.out.println(exchange.getExchangeId());
        System.out.println(exchange.getFromRouteId());
        System.out.println(exchange.getFromEndpoint());
        System.out.println(exchange.getPattern());
        System.out.println(exchange.getIn().getHeaders());
        System.out.println(exchange.getIn().getBody());
        System.out.println(exchange.getOut().getHeaders());
        System.out.println(exchange.getOut().getBody());
    }
}
```

- Prozessoren können jederzeit den aktuellen Exchange ändern
 - und damit den Ablauf der Route beeinflussen

```
@Override
public void process(Exchange exchange) throws Exception {
    exchange.setProperty("NEW HEADER", "Header-Value");
    Message message = new DefaultMessage();
    message.setBody("NEW BODY");
    exchange.setIn(message);
}
```

- Für Standard-Aufgaben zur Exchange-Manipulation können bereits fertige Implementierungen der Camel-Distribution benutzt werden
 - Details der Arbeitsweise sind in der Camel-Dokumentation beschrieben

Transformation

 ConvertBody

 Enrich

 InOnly

 InOut

 Marshal

 PollEnrich


 RemoveHeader

 RemoveHeaders

 RemoveProperties

 RemoveProperty

 SetBody

 SetExchangePattern

 SetFaultBody

 SetHeader

 SetOutHeader

 SetProperty

 Transform

 Unmarshal

4

SPEZIELLE PROCESSORS

4.1

EXPRESSIONS

- Für die Implementierung von Logik in den Routen stellt Camel praktisch alle Sprachen zur Verfügung, die irgendwie in einer Java Runtime unterstützt werden können
 - Java selber
 - XML-Sprachen wie XPath, XQuery
 - Skript-Sprachen
- Allen Sprachen sind bestimmte Sprachfeatures gemeinsam
 - Zugriff auf das Exchange-Objekt innerhalb einer Route
 - Java-Methoden `body()` oder `header()`
 - Zugriff auf den `CamelContext`
 - und damit auf die gesamte Laufzeitumgebung
 - Modifikation des Contexts zur Laufzeit
 - Übersicht
 - <http://camel.apache.org/simple.html>

- Bean Language
 - Zugriff auf Java-Objekte und Methoden
- Constant
- Unified Expression Language
- Header
- JSonPath
- JXPath
- Mvel
- OGNL
- Ref Language
- ExchangeProperty / Property

- Skript-Sprachen wie
 - BeanShell
 - JavaScript
 - Groovy
 - Python
 - PHP
 - Ruby
- Simple
 - File Language
- Spring Expression Language
- SQL
- Tokenizer
- XPath
- XQuery
- VTD-XML

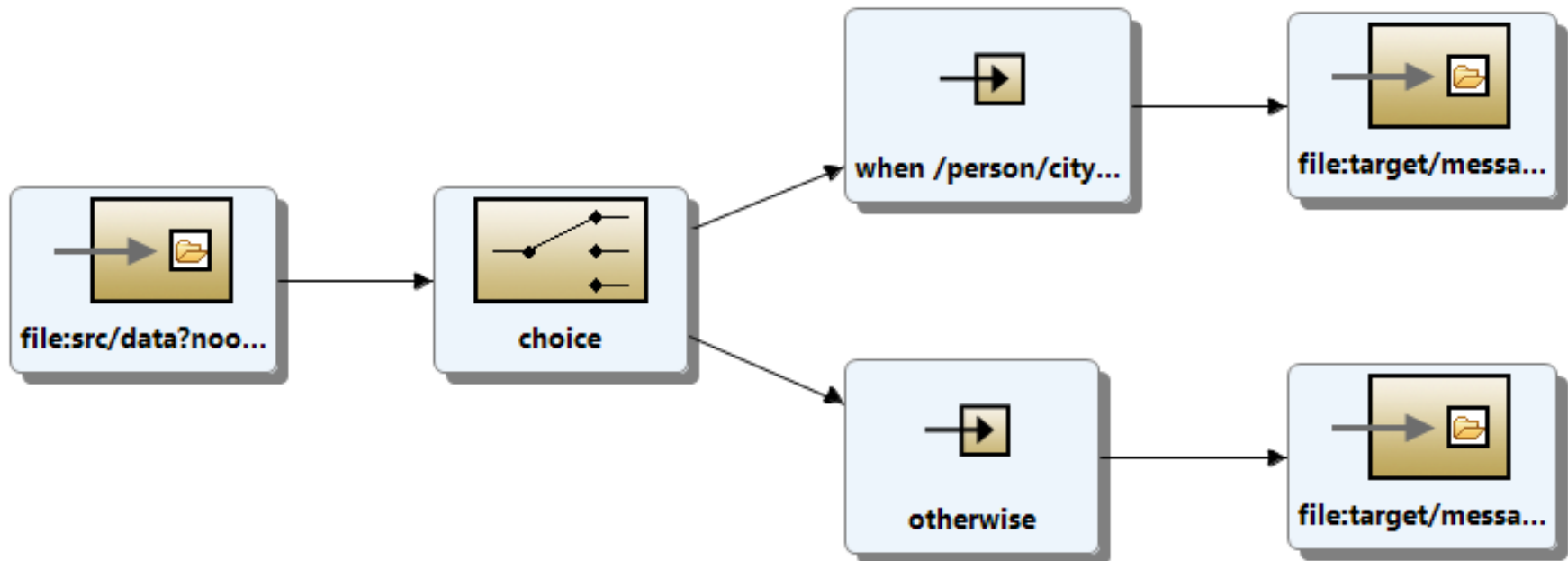
- Eine (ursprünglich) einfache Skript-Sprache
- ANT-ähnliche Syntax mit Ausdrücken der Form
 - `${bean.property}`
 - `${bean.method}`
- `simple` unterstützt
 - Literale
 - Arithmetische und logische Operatoren
 - String-Operationen
 - Array-Operationen
 - Einige vordefinierte Funktionen

- Mit `bean` wird eine Methode einer Spring-Bean aufgerufen

```
<bean ref="simpleSpringBean"
      method="executeMyLogic"></bean>
```
- Diese Methode kann bestimmte Parametertypen deklarieren, die von Camel übergeben werden
 - `org.apache.camel.Exchange`
 - `org.apache.camel.Message`
 - `org.apache.camel.CamelContext`
 - `org.apache.camel.TypeConverter`
 - `org.apache.camel.spi.Registry`
 - `java.lang.Exception`
- Der erste Parameter ist stets der Body der Message

4.2

CHOICE UND PREDICATES



```
<camelContext
xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="file:src/data?noop=true" />
    <choice>
      <when>
        <xpath>/person/city = 'London'</xpath>
        <to uri="file:target/messages/uk" />
      </when>
      <otherwise>
        <to uri="file:target/messages/others" />
      </otherwise>
    </choice>
  </route>
</camelContext>
```

- Ein Predicate bestimmt ein logisches Ergebnis zur Verwendung
 - `choice`
 - `filter`

- **interface** `Predicate` mit
 - **Methode** `boolean matches(Exchange exchange)`

- **Implementierungen**
 - `el`
 - `groovy`
 - `javaScript`
 - `...`
 - `xpath`

4.3

TRANSFORMER

- Transformationen erfolgen
 - innerhalb einer Route
 - innerhalb einer Komponente
 - Automatisch durch registrierte Type-Converter
- Mechanismen
 - Data Formats
 - Templates
- Transformer ermöglichen eine Datenumwandlung in die von Camel unterstützten Datentypen
- Befehle
 - `transformer`
 - `marshal`
 - `unmarshal`

- Standard Java
 - Serialization
 - String
- Object Marshalling
 - Avro
 - Boon
 - JSON
 - Protobuf
- Object/XML Marshalling
 - Castor
 - JAXB
 - XmlBeans
 - XStream
 - JiBX

- Web Services
 - SOAP
- JSON / XML
 - XmlJson
- Flache Datenstrukturen
 - BeanIO
 - Bindy
 - CSV
 - EDI
 - Flatpack
 - uniVocity
- Domain specific marshalling
 - HL7

- Kompression
 - GZip data format
 - Zip DataFormat
 - Zip File DataFormat
- Security
 - Crypto
 - PGP
 - XMLSecurity DataFormat
- Verschiedenes
 - Base64
 - Custom DataFormat RSS
 - TidyMarkup
 - Syslog
 - ICal
 - Barcode

4.4

FEHLERBEHANDLUNG

- Globale Error Handlers
 - Logging
 - Dead Letter
 - Default
- `onException`
 - Fängt einen Ausnahme-Typen
- try-catch-Blöcke
 - `doTry`
 - `doCatch`
 - `doFinally`
 - `throwException`
- Kompensation
 - Hinterlegen von Completion-Handlern, die je nach Erfolg der Routenausführung aufgerufen werden
 - Damit können verschachtelte try-catch-Blöcke vermieden werden

- Definition innerhalb des CamelContext
 - `<errorHandler level="DEBUG" type="LoggingErrorHandler" logName="ErrorLog"></errorHandler>`
 - `<errorHandler level="DEBUG" type="DeadLetterChannel" deadLetterUri="direct:errors"></errorHandler>`
- Error Handler definieren optional einen Retry-Mechanismus
 - Redelivery
 - Benutzung eines Predicate-Ausdrucks in `retryWhile`
`<errorHandler id="myRouteSpecificErrorHandler" type="DefaultErrorHandler">`
`<redeliveryPolicy maximumRedeliveries="2"/>`
`</errorHandler>`

- Definition innerhalb des CamelContexts

```
public class ExceptionRouteBuilder extends RouteBuilder {  
    @Override  
    public void configure() throws Exception {  
        onException(MyException.class, AnotherException.class)  
        .to("direct:error");  
        from("...") //...  
    }  
}
```

- Hier werden analog dem Java-Exception-Mechanismus innerhalb der Routen-Definition try-catch-finally-Bereiche definiert

```
<route>
  <from uri="..."/><to uri="..."/>
  <doTry>...
    <doCatch>
      <exception>MyException</exception>
      <to uri="..."/>
    </doCatch>
    <doFinally>
      <to uri="..."/>
    </doFinally>
  </doTry>
  <to uri="..."/>
</route>
```


- `onCompletion`
- `onException`

5

ENTERPRISE INTEGRATION PATTERNS


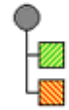

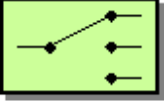
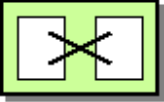
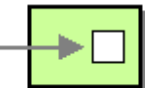
5.1

ÜBERSICHT





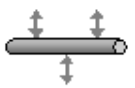
- Teile der Enterprise Integration Patterns sind bereits im Abschnitt über die Programmierung beschrieben worden
 - Auch ein Choice ist ein Pattern!
- In diesem Kapitel werden eher die System-relevanten Patterns beschrieben

5.2



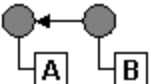

KATALOG

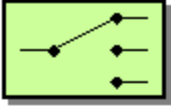

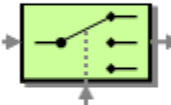
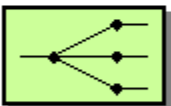
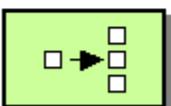

	Message Channel	How does one application communicate with another using messaging?
	Message	How can two applications connected by a message channel exchange a piece of information?
	Pipes and Filters	How can we perform complex processing on a message while maintaining independence and flexibility?
	Message Router	How can you decouple individual processing steps so that messages can be passed to different filters depending on a set of conditions?
	Message Translator	How can systems using different data formats communicate with each other using messaging?
	Message Endpoint	How does an application connect to a messaging channel to send and receive messages?

Messaging Channels

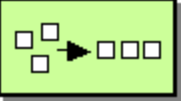
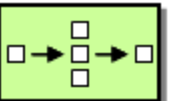
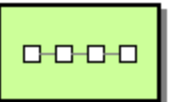
	Point to Point Channel	How can the caller be sure that exactly one receiver will receive the document or perform the call?
	Publish Subscribe Channel	How can the sender broadcast an event to all interested receivers?
	Dead Letter Channel	What will the messaging system do with a message it cannot deliver?
	Guaranteed Delivery	How can the sender make sure that a message will be delivered, even if the messaging system fails?
	Message Bus	What is an architecture that enables separate applications to work together, but in a de-coupled fashion such that applications can be easily added or removed without affecting the others?

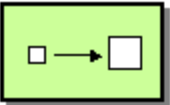
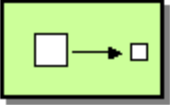
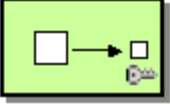

Message Construction

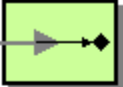
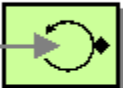
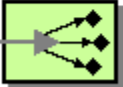
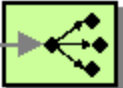


	Event Message	How can messaging be used to transmit events from one application to another?
	Request Reply	When an application sends a message, how can it get a response from the receiver?
	Correlation Identifier	How does a requestor that has received a reply know which request this is the reply for?
	Return Address	How does a replier know where to send the reply?

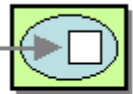
	Content Based Router	How do we handle a situation where the implementation of a single logical function (e.g., inventory check) is spread across multiple physical systems?
	Message Filter	How can a component avoid receiving uninteresting messages?
	Dynamic Router	How can you avoid the dependency of the router on all possible destinations while maintaining its efficiency?
	Recipient List	How do we route a message to a list of (static or dynamically) specified recipients?
	Splitter	How can we process a message if it contains multiple elements, each of which may have to be processed in a different way?
	Aggregator	How do we combine the results of individual, but related messages so that they can be processed as a whole?

Message Routing II

	Resequencer	How can we get a stream of related but out-of-sequence messages back into the correct order?
	Composed Message Processor	How can you maintain the overall message flow when processing a message consisting of multiple elements, each of which may require different processing?
	Scatter-Gather	How do you maintain the overall message flow when a message needs to be sent to multiple recipients, each of which may send a reply?
	Routing Slip	How do we route a message consecutively through a series of processing steps when the sequence of steps is not known at design-time and may vary for each message?
	Throttler	How can I throttle messages to ensure that a specific endpoint does not get overloaded, or we don't exceed an agreed SLA with some external service?
	Sampling	How can I sample one message out of many in a given period to avoid downstream route does not get overloaded?
	Delayer	How can I delay the sending of a message?
	Load Balancer	How can I balance load across a number of endpoints?
	Multicast	How can I route a message to a number of endpoints at the same time?

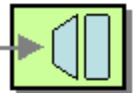
	Content Enricher	How do we communicate with another system if the message originator does not have all the required data items available?
	Content Filter	How do you simplify dealing with a large message, when you are interested only in a few data items?
	Claim Check	How can we reduce the data volume of message sent across the system without sacrificing information content?
	Normalizer	How do you process messages that are semantically equivalent, but arrive in a different format?
	Sort	How can I sort the body of a message?
	Script	How do I execute a script which may not change the message?
	Validate	How can I validate a message?

	Event Driven Consumer	How can an application automatically consume messages as they become available?
	Polling Consumer	How can an application consume a message when the application is ready?
	Competing Consumers	How can a messaging client process multiple messages concurrently?
	Message Dispatcher	How can multiple consumers on a single channel coordinate their message processing?
	Selective Consumer	How can a message consumer select which messages it wishes to receive?
	Durable Subscriber	How can a subscriber avoid missing messages while it's not listening for them?
	Idempotent Consumer	How can a message receiver deal with duplicate messages?



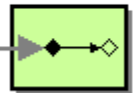
Transactional Client

How can a client control its transactions with the messaging system?



Messaging Gateway

How do you encapsulate access to the messaging system from the rest of the application?



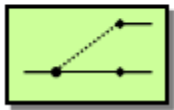
Service Activator

How can an application design a service to be invoked both via various messaging technologies and via non-messaging techniques?



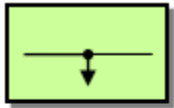
ControlBus

How can we effectively administer a messaging system that is distributed across multiple platforms and a wide geographic area?



Detour

How can you route a message through intermediate steps to perform validation, testing or debugging functions?



Wire Tap

How do you inspect messages that travel on a point-to-point channel?

Message History

How can we effectively analyze and debug the flow of messages in a loosely coupled system?

Log

How can I log processing a message?

- Der Nachrichtenaustausch erfolgt an Hand von Austauschmustern
 - IN
 - OUT
 - INOUT
- Verschiedene Endpoints unterstützen mehrere Patterns
 - JMS mit oder ohne Listener zum Empfang einer Response-Message
- Für manche Endpoints kann das Standard-Verhalten geändert werden
 - One-way message für einen INOUT-Endpoint
 - One-way route mit Request-Response

5.3

REALISIERUNG MIT APACHE CAMEL

- Content Based Routing
 - Steuerung des Routen-Ablaufs in Abhängigkeit vom Inhalt Exchange
- Filtering
- Wire Tap
 - Ein Exchange wird in eine zusätzliche Route gesendet
 - Sinnvoll beispielsweise für Logging/Auditing
- Multicast
 - Senden der Nachricht an mehrere Empfänger
- Recipient List
 - Nachrichtenversand an eine Liste von Empfängern
 - Dynamische Verwaltung der Liste möglich

- Throttler
 - Beschränkt die Anzahl der Nachrichten, die an einen Endpoint gesendet werden
- Dynamic Routing
 - Route-Auswahl als Ergebnis eines Programmaufrufs
- Load balancing
 - über mehrere Endpoints

- Die Apache-Distribution enthält einen reichhaltigen Satz von Beispielprogrammen
 - Aggregation
 - Splitting
 - Parallelisierung

- Transaktionelle Routen
 - Endpoints können transaktionell sein
 - Die Nachrichten-Verarbeitung kann mit commit/rollback kontrolliert werden
- Testen von Routen
 - Die Camel-Distribution enthält JUnit-Erweiterungen
 - Ebenso existieren „Mock-Endpoints“
 - Inbound: Erzeugen von vordefinierten Messages
 - Outbound: Prüfen, ob und welche Nachrichten eingegangen sind

© Integrata Cegos GmbH

Integrata Cegos GmbH
Zettachring 4
70567 Stuttgart

Alle Rechte, einschließlich derjenigen des auszugsweisen Abdrucks, der fotomechanischen und elektronischen Wiedergabe vorbehalten.