15.1.2015

Ari Ayvazyan, Helmuth Brunner

5AHIT

LoadBalancer

AUFGABENDURCHFÜHRUNG

https://github.com/aayvazyan-tgm/CamelLoadBalancer

Inhaltsverzeichnis

## TOC \t "Header & Footer, 1,heading 1, 2,heading 2, 3"

Aufgabenstellung PAGEREF \_Toc \h 2

Design PAGEREF \_Toc1 \h 3

Durchführung PAGEREF \_Toc2 \h 5

Integrierte Patterns PAGEREF \_Toc3 \h 6

Technologie Beschreibung PAGEREF \_Toc4 \h 8

Testdurchführung PAGEREF \_Toc5 \h 9

Starten des LoadBalancers PAGEREF \_Toc6 \h 9

Starten der Webservelts PAGEREF \_Toc7 \h 9

Custom Load Balacing Methode PAGEREF \_Toc8 \h 9

Round Robin Methode PAGEREF \_Toc9 \h 9

Weighted Round-Round Methode PAGEREF \_Toc10 \h 10

Quellen PAGEREF \_Toc11 \h 11

## Aufgabenstellung

Es soll ein Load Balancer mit mindestens 2 unterschiedlichen Load-Balancing Methoden (jeweils 7 Punkte) implementiert werden (ähnlich dem PI Beispiel [1]; Lösung zum Teil veraltet [2]). Eine Kombination von mehreren Methoden ist möglich. Die Berechnung bzw. das Service ist frei wählbar!

Folgende Load Balancing Methoden stehen zur Auswahl:

◦ Weighted Round-Round

◦ Least Connection

◦ Least Connected Slow- Start Time

◦ Weighted Least Connection

◦ Agent Based Adaptive Balancing / Server Probes

Um die Komplexität zu steigern, soll zusätzlich eine "Session Persistence" (2 Punkte) implementiert werden.

**Auslastung**

Es sollen die einzelnen Server-Instanzen in folgenden Punkten belastet werden können:

◦ Memory (RAM)

◦ CPU Cycles

◦ I/O Zugriff (Harddisk)

Bedenken Sie dabei, dass die einzelnen Load Balancing Methoden unterschiedlich auf diese Auslastung reagieren werden. Dokumentieren Sie dabei aufkommenden Probleme ausführlich.

**Tests**

Die Tests sollen so aufgebaut sein, dass in der Gruppe jedes Mitglied mehrere Server fahren und ein Gruppenmitglied mehrere Anfragen an den Load Balancer stellen. Für die Abnahme wird empfohlen, dass jeder Server eine Ausgabe mit entsprechenden Informationen ausgibt, damit die Verteilung der Anfragen demonstriert werden kann.

**Modalitäten**

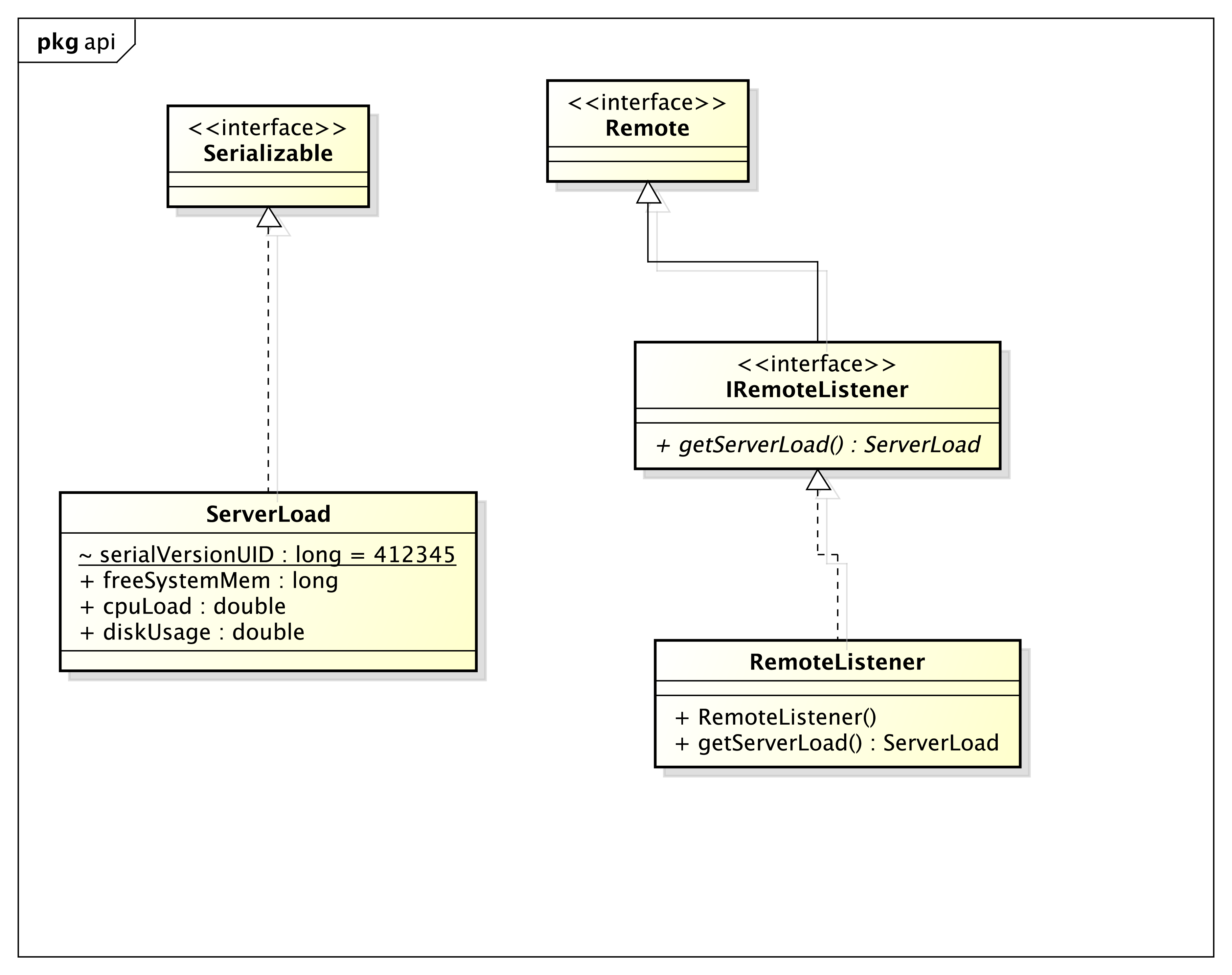
Gruppenarbeit: 2 Personen

Abgabe: Protokoll mit Designüberlegungen / Umsetzung / Testszenarien, Sourcecode (mit allen notwendigen Bibliotheken), Java-Doc, Jar

Viel Erfolg!

## Design

Load Blancer:

RMI Server:

## Durchführung

Im ersten Schritt haben wir ein bisschen recherchiert und uns verschiede Framework, vorzüglich für Java angeschaut. Dabei ist uns nur ein Tool ins Auge gesprungen und dieses war Apache Camel und somit haben wir uns für dieses Framework entschieden.

Wir verwenden Apache Camel um zwischen den einzelnen Servern zu routen, der Server ist direkt in die Applikation Eingebung und mit kann ganz einfach mit dem Build-Tool Gradle erstellt werden. Infolge wird auch das Servlet am Server deployed.

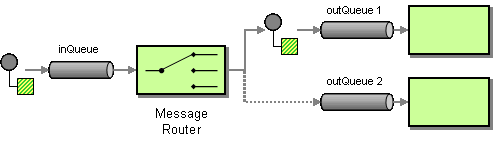
Als erstes haben wir die Balancing Methode Round Robing implementiert. In weiterer Folge sind dann CustomLoadBalancing und Weight Round Robing hinzugeführt worden.

Für den CustomLoadBalancer haben wir mittels RMI ein Client-Server Architektur implementiert, mittels dieser Implementierung ist es uns möglich Auslastungen sowie andere Systemwerte vom Server auszulesen und an den Load Balancer weiterleiten.

## Integrierte Patterns

Folgende Patterns wurden integriert:

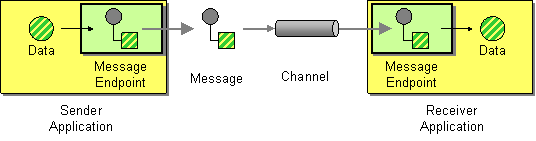
* Message Router



[3]

Beim Message Router Pattern wird von einer Input-Destination die richtige Output-Destination gefunden.

* Message Endpoint



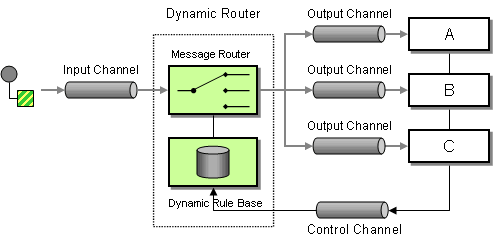
[4]

Bei diesem Pattern werden Nachrichten von einem Sender an eine Empfänger-Application weitergeleitet.

Code Snippet:

|  |
| --- |
| @Override **public void** configure() **throws** Exception {  from(**"jetty:http://0.0.0.0:8081/rr?matchOnUriPrefix=true"**)  .routeId(**"LOADBALANCER"**).loadBalance()  .roundRobin()  .to(**destinations**)  ; } |

* Dynamic Routing



[5]

Das Dynamic Routing Pattern ist das Herz eines LoadBalancers. Die Nachrichten werden unter festgelegten Regeln an die einzelnen Destination verteilt.

## Technologie Beschreibung

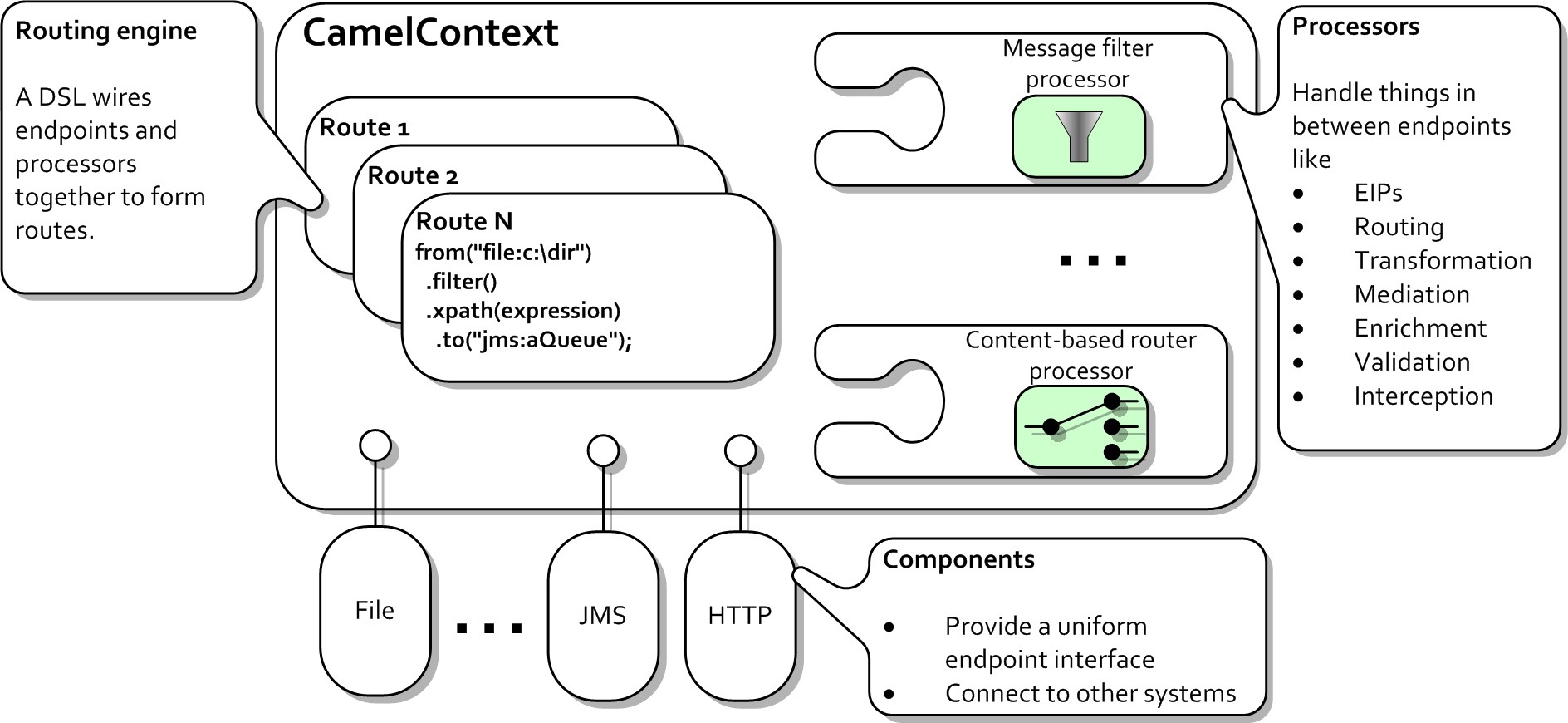
Apache Camel ist eine regelbasierte Routingengine, mit der Regeln definiert werden können.

Weil Apache Camel mit URI (Uniform Resource Identifiers) arbeitet kann es ein viel Zahl von Protokollen ansprechen, wie zum Beispiel HTTP, JMS oder AMQP, weiters kann Apache Camel mit JBI, SCA, Apache ActiveMQ, RabbitMQ, Apache MINA oder Apache CXF zusammenarbeiten.

Weiters unterstütz Apache Camel Bean Binding und bietet die Möglichkeit populäre Frameworks wie Spring, Blueprint oder Guice einzubinden.

Folgende Grafik veranschaulicht das Prinzip von Apache Camel.

Es gibt mehrere Routes an die die Messages weiter geleitet werden, sowie die Routing Methode kann auch definiert werden.

[7]

[6]

## Testdurchführung

Wir verwenden Gradle als Build-Tool mit diesem kann die Applikation gestartet werden.

Es wird einmal der Loadbalancer gestartet der zwischen den Server wechselt und die Server mit den Webapplikationen.

# Starten des LoadBalancers

./gradlew :run

# Starten der Webservelts

./gradlew :LoadBalancerClientServlet:run

Nun ist der Server unter “localhost:8081” erreichbar.

|  |
| --- |
| localhost:8081/c |

|  |
| --- |
| localhost:8081/rr |

Die Auswahl der Load Balancing Methoden kann wie folgt getroffen werden:

# Custom Load Balacing Methode

Bei dieser Methode werden die Auslastungen von den Server herangezogen und so die Anfragen der Benutzer verteilt.

Aufruf durch folgende URL:

# Round Robin Methode

Die Anfragen der Clients werden immer zwischen den einzelnen Servern verteilt.

Aufruf durch folgende URL:

# Weighted Round-Round Methode

Hier bei wird der Load Balancer so einstellt das zum Beispiel Server 1, 3 Anfragen verarbeitet und der Server 2 nur 1 Anfrage verarbeitet. Weil der Server 1 mehr Last aufnehmen kann wir diesem auf auch mehrere Anfragen zugeteilt.

|  |
| --- |
| localhost:8081/wrr |

Aufruf durch folgende URL:

## Quellen

[1] "Praktische Arbeit 2 zur Vorlesung 'Verteilte Systeme' ETH Zürich, SS 2002", Prof.Dr.B.Plattner, übernommen von Prof.Dr.F.Mattern (<http://www.tik.ee.ethz.ch/tik/education/lectures/VS/SS02/Praktikum/aufgabe2.pdf>)

[2] <http://www.tik.ee.ethz.ch/education/lectures/VS/SS02/Praktikum/loesung2.zip>

[3] <http://www.eaipatterns.com/MessageRouter.html>

[4] <http://www.eaipatterns.com/MessageEndpoint.html>

[5] <http://www.eaipatterns.com/DynamicRouter.html>

[6] <http://java.dzone.com/sites/all/files/figure1_1.jpg>

[7] <http://camel.apache.org>