
Java is a trademark of Sun Microsystems, Inc.

# JavaOne℠

## Enterprise Integration Patterns in Practice

Andreas Egloff
Sun Microsystems

&

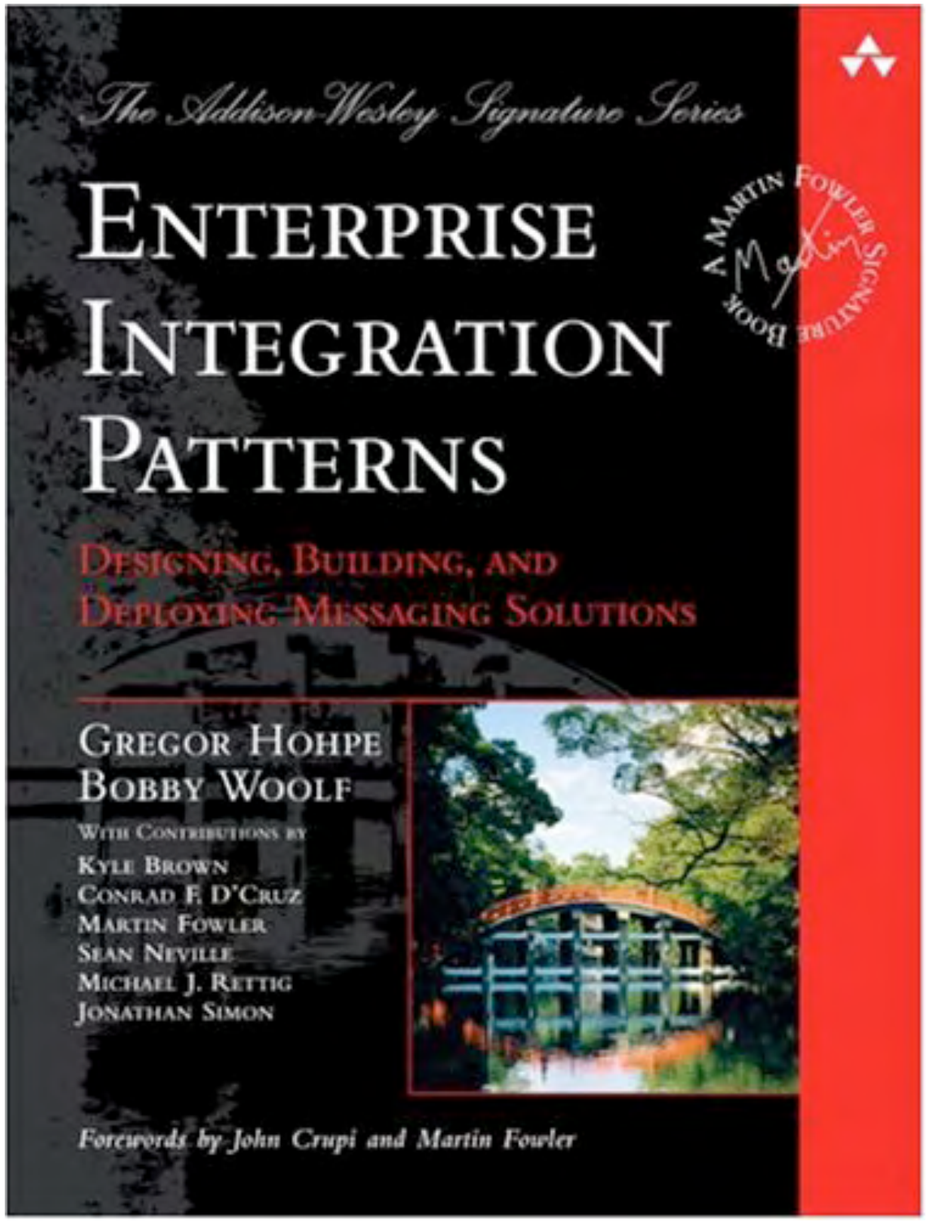
Bruce Snyder
SpringSource

# What Are Design Patterns?

*A design pattern is a formal way of documenting a solution to a design problem in a particular field of expertise.*

(Wikipedia)

# Got EIP?

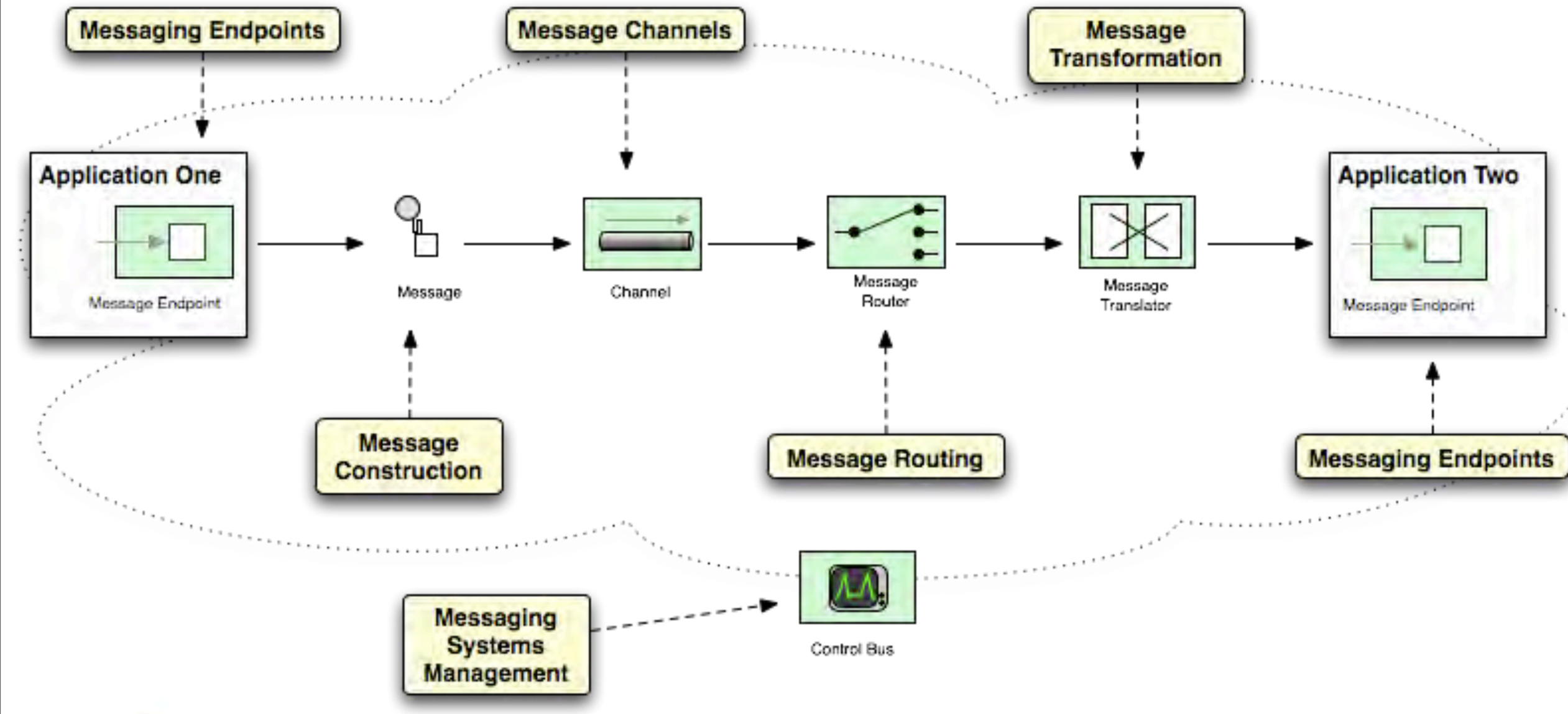

The Addison-Wesley Signature Series
Enterprise Integration Patterns
Designing, Building, and Deploying Messaging Solutions
A Martin Fowler Signature Book
Gregor Hohpe
Bobby Woolf
With Contributions by
Kyle Brown
Conrad F. D'Cruz
Martin Fowler
Sean Neville
Michael J. Rettig
Jonathan Simon
Forewords by John Crupi and Martin Fowler

# Core Principles of EIP

- Patterns using asynchronous messaging as a style of integration

- **Pros**
  - Scaling
  - Decoupling
- **Cons**
  - Latency vs. throughput
  - Decoupling not always appropriate

# Pattern Overview



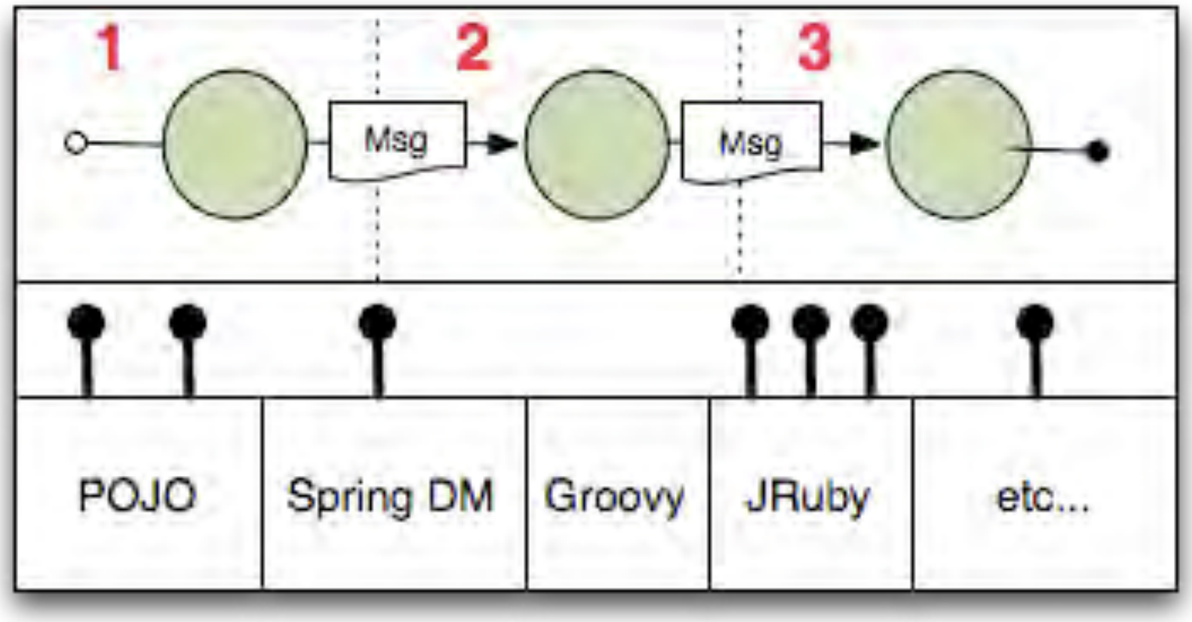
# Identifying and Expressing Patterns

# Layers of Functionality

*Message Flow / Service Composition*

*Message Endpoint / Services*

*Code Composition*




- Not an either/or choice, combine layers as appropriate

# Message Flow / Service Composition

- Strengths and uses
  - De-coupling and re-using Services
    - Easier to evolve
  - Scaling, Throughput
    - Asynchronous nature
  - Message routing, processing, transformation
  - Easy to mediate

# Message Endpoints / Services

- Message Endpoints / Services
  - Message based interface
  - Expose coarse grained services
  - Typically stateless
  - Anticipate re-use
  - Visibility scope examples
    - Within application
    - Within ESB
    - Anyone with access to JMS server
    - External

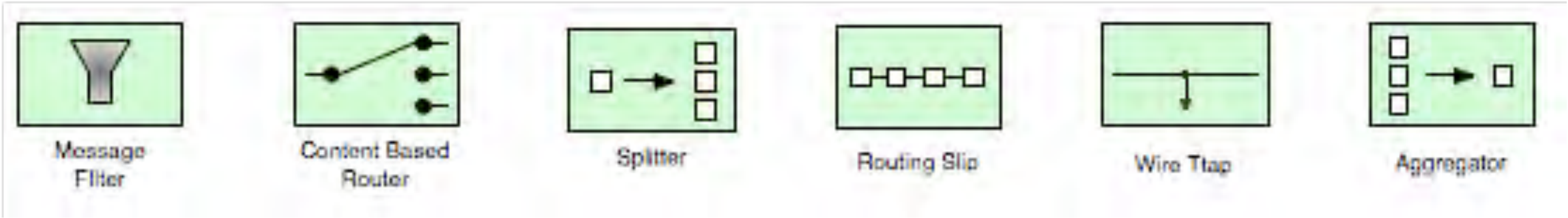
# Code Composition

- Strengths and uses
  - Fine grained API access
  - Stateful, fine grained interactions
  - Low latency
- Challenge
  - Interface tighter coupled

# Implicit and Explicit Patterns

- Some patterns inherent in technology / framework
  - Not every pattern a "keyword"
    - e.g. JMS publish/subscribe ...
- Patterns realized in user code
  - e.g. Scatter-Gather realized in Java
- Platform has pre-built constructs
  - e.g. Unix pipe symbol "|" : pipes-and-filters
  - Route in Camel or Fuji: pipes-and-filters

# Visualizing EIPs




- Stencils exist for Visio and OmniGraffle
  - http://www.eaipatterns.com/downloads.html

# Patterns in Practice

# Gotcha - Flexibility vs. Productivity

- **Issue:** Too little explicit support out-of-the-box
  - Too much roll your own

- **Issue:** Explicit support out-of-the-box is too rigid
  - Does not exactly fit the use case
    - Forced to work around or again roll your own

- *What you want is out-of-the-box productivity with explicit constructs AND flexibility to customize the constructs*

# Intro to Frameworks

- Apache Camel
  - http://camel.apache.org/

- Project Fuji / OpenESB v3
  - http://fuji.dev.java.net/

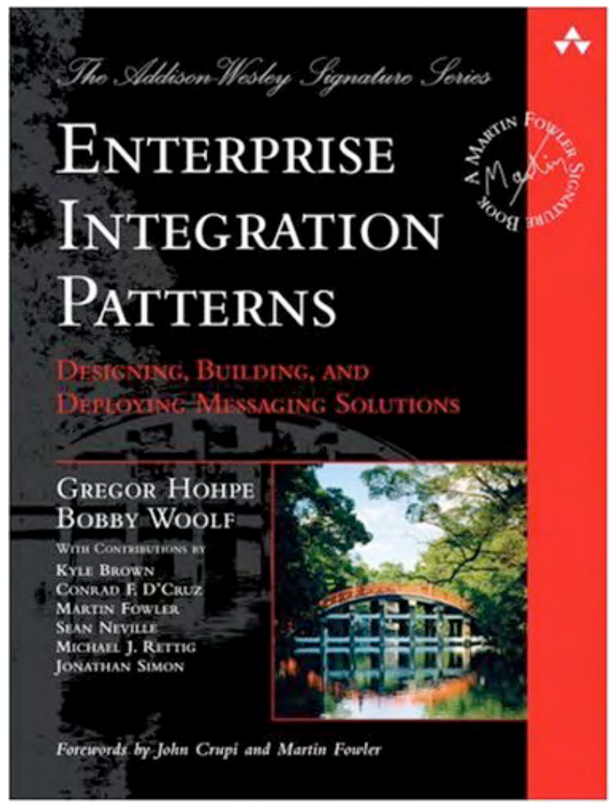
# What is Apache Camel?


http://camel.apache.org/

*A framework for simplifying integration through the use of the Enterprise Integration Patterns for message mediation, processing, routing and transformation*

# Apache Camel is Focused on EIP

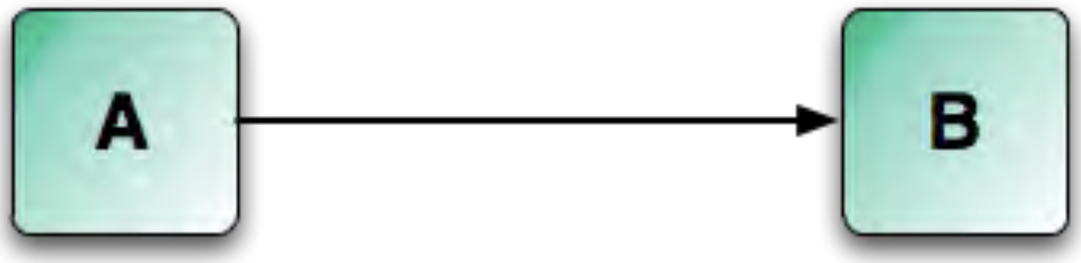

=>




http://camel.apache.org/

http://eaipatterns.com/

# Message Routing

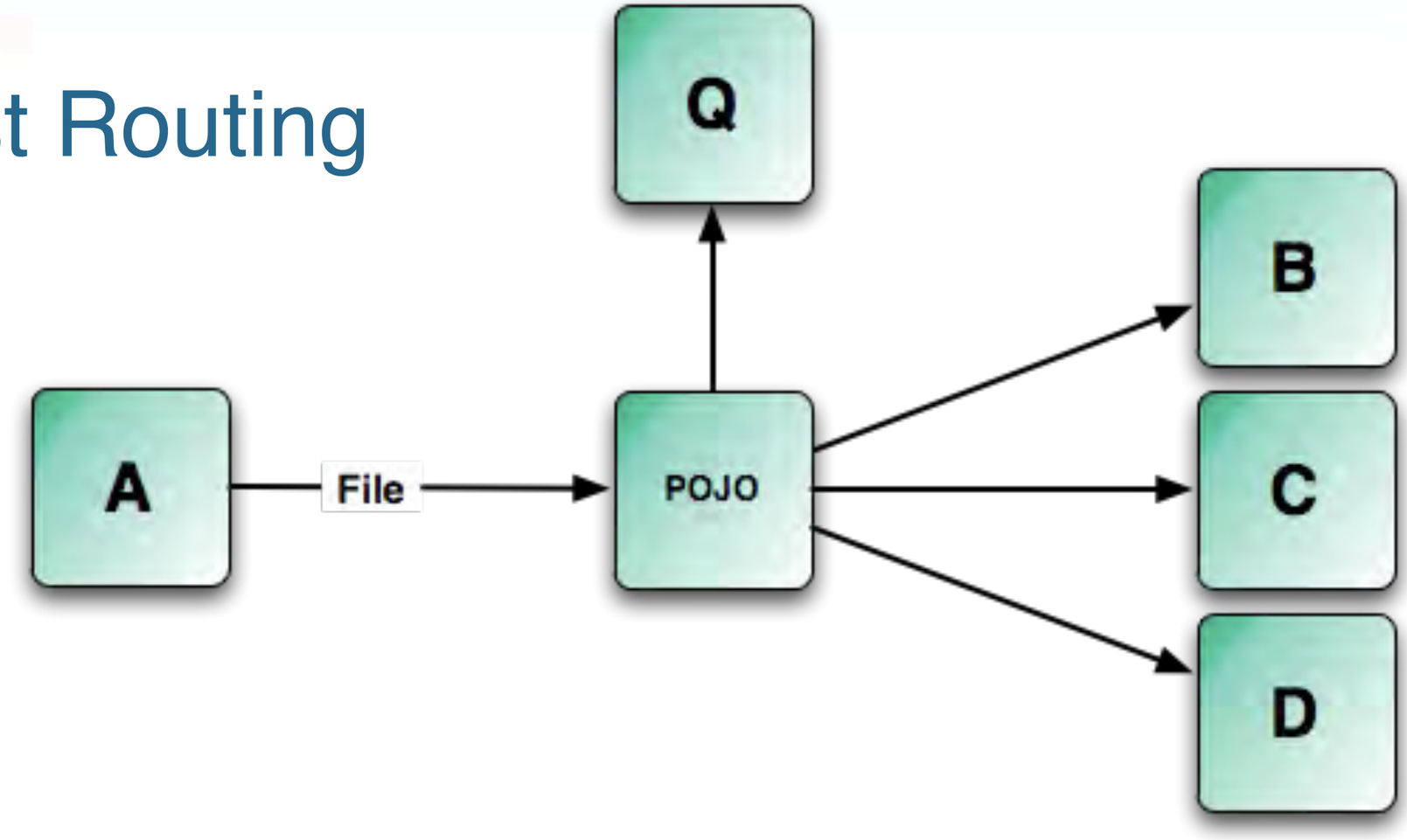



```
from("A").to("B");
```

# Slightly More Complex Routing

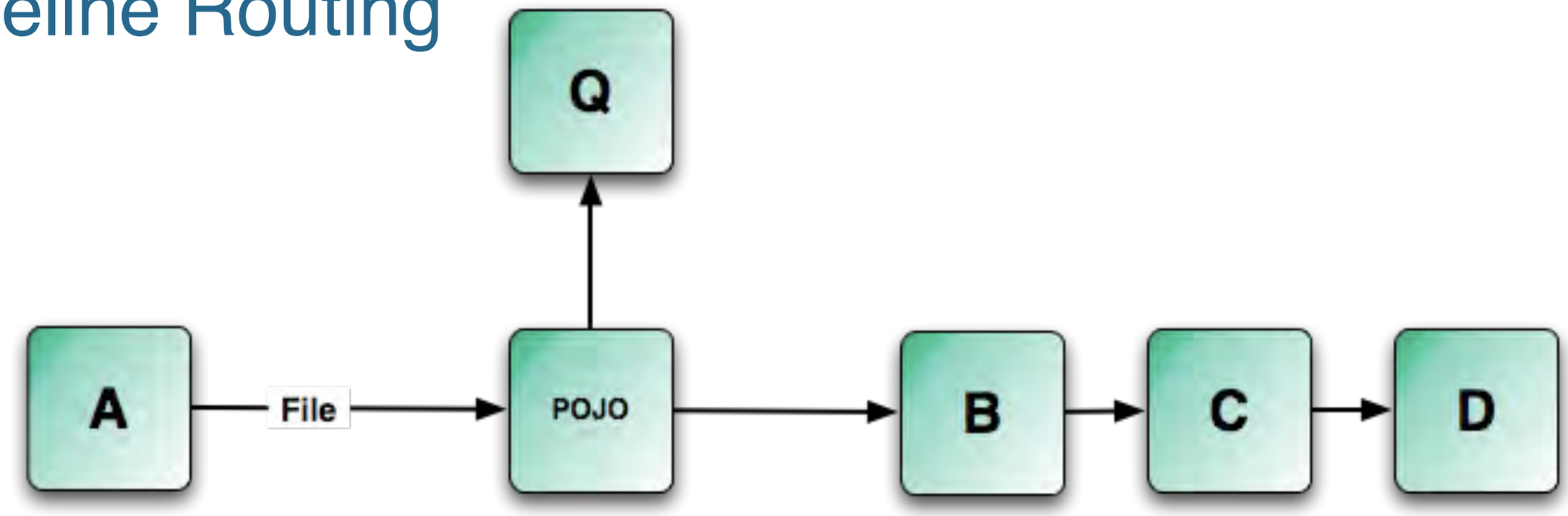

```
from("file:///tmp/myFile.txt").
to("bean:MyBean?method=handleMessage").
to("jms:TEST.Q");
```

# Multicast Routing




```
from("file:///tmp/myFile.txt").
choice().when().
  method("MyBean", "matches").
  to("Q").
end().
multicast("B", "C", "D");
```

# Pipeline Routing




```
from("file:///tmp/myFile.txt").
choice().when().
  method("MyBean", "matches").
  to("Q").
end().
pipeline("B", "C", "D");
```

# Camel Components

- 70+ components supported

| | | | | | | |
|---|---|---|---|---|---|---|
| ActiveMQ | Esper | IBatis | JPA | Multicast | SEDA | UDP |
| ActiveMQ Journal | Event | IMAP | JT/400 | POP | SFTP | Validation |
| AMQP | File | IRC | LDAP | Quartz | Smooks | Velocity |
| Atom | FIX | JavaSpaces | List | Queue | SMTP | VM |
| Bean | Flatpack | JBI | Log | Ref | Spring Integration | XMPP |
| Browse | Freemarker | JCR | Mail | Restlet | SQL | XQuery |
| Cometd | FTP | JDBC | MINA | RMI | Stream | XSLT |
| Apache CXF | Hibernate | JDBC | Mock | RNC | String Template | |
| DataSet | HL7 | Jetty | MSMQ | RNG | TCP | |
| Direct | HTTP | JMS | MSV | RSS | Test | |

# What is Project Fuji?

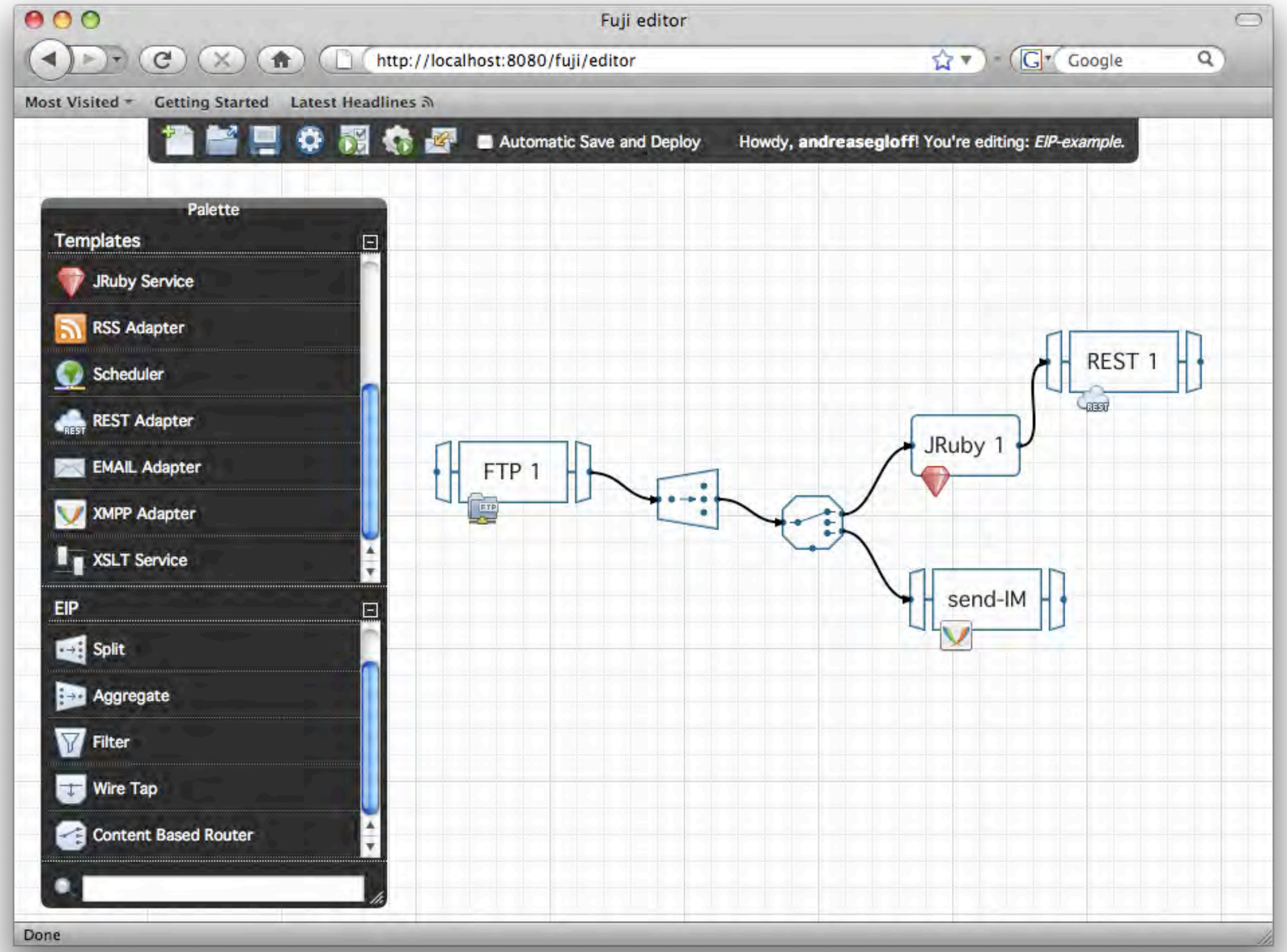

http://fuji.dev.java.net/

- Basis for OpenESB v3
- Fuji Goals
  - Agility + Flexibility + Ease of Use = Productivity
- Service Platform to realize
  - Service Oriented Architecture (SOA)
  - Light weight SOA (aka Web Oriented Architecture)
  - Event Driven Architecture (EDA)
  - ... last but not least MOM style applications
- > 40 Components in OpenESB community today

# Interesting Properties of Fuji

- Convention, Configuration, Code... in that order
- Light weight, OSGi based
- Not JMS centric but Service Centric
  - In-VM service composition option
  - Choice of transport when going across VMs
    - E.g. load balancing via http or federating using JXTA
- Visual and textual (DSL) editing for EIPs
  - Camel component is another option

# Web Tooling Option
# Service Composition with EIPs in a Browser



# Web UI Gives the User...

- All-in-one interface for service composition
  - Service Definition
  - Routing, EIPs
  - Configuration
  - Deployment
- Layered on textual representation (DSL)
  - Check out from version control, edit in IDE
- Tooling option for different preferences and skills
  - e.g. casual technologist vs, Developer
- Extensible

# Composition in an IDE / Text Editor
# A textual representation of EIPs

- Goals
  - Simple, concise syntax
    - Easy to type and read
  - Top-down design
    - Generate service templates
  - Concept
    - used to declare and configure services, routing
      - Sets up services and routing at deployment time; NOT interpreted on the fly at runtime

"Hello World 1" - simple routing

```
rss "finance-feed"
file "archive"

route do
  from "finance-feed"
  to "archive"
end
```

# Basic Concepts of the DSL Integration Flow Language

"Hello World 2" - pipes-and-filters

```
rss "finance-feed"
jruby "tranform"
file "archive"

route do
    from "finance-feed"
    to "tranform"
    to "archive"
end
```

Output of one filter/stage flows to input of next stage similar to Unix Pipes

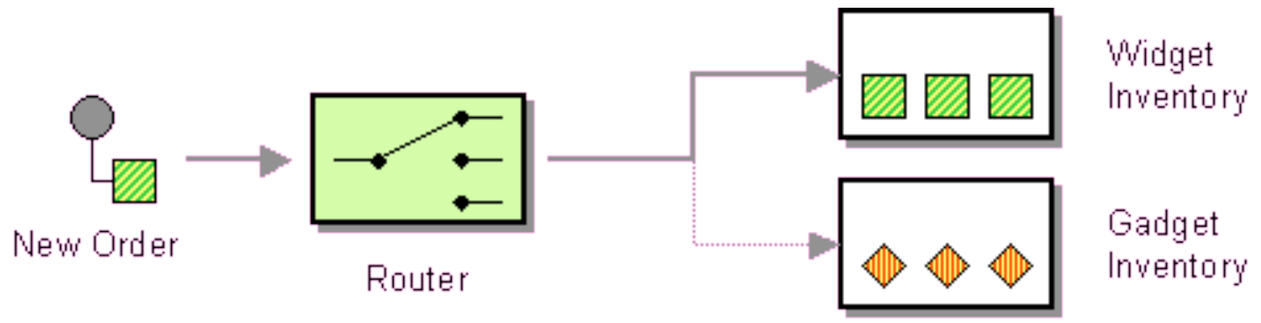
# Basic Concepts of the DSL
# Explicit EIP Constructs

"Hello World 3" -
adding EIP constructs

```
rss "finance-feed"
jruby "tranform"
file "archive"

route do
    from "finance-feed"
    to "tranform"
    broadcast do
        route to "archive"
        route to "notify"
    end
end

xmpp "im"

route "notify" do
    filter regex("Buffet")
    to "im"
end
```

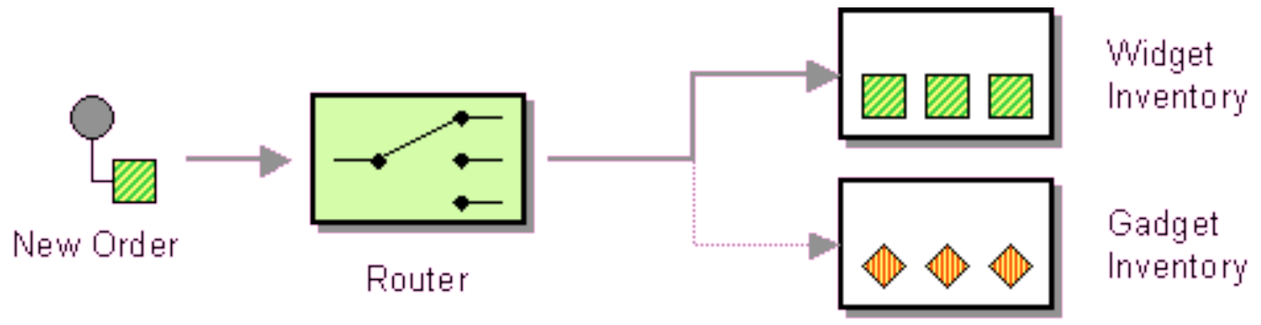
# Example Pattern Implementations

# Message Routing: Content-Based Router



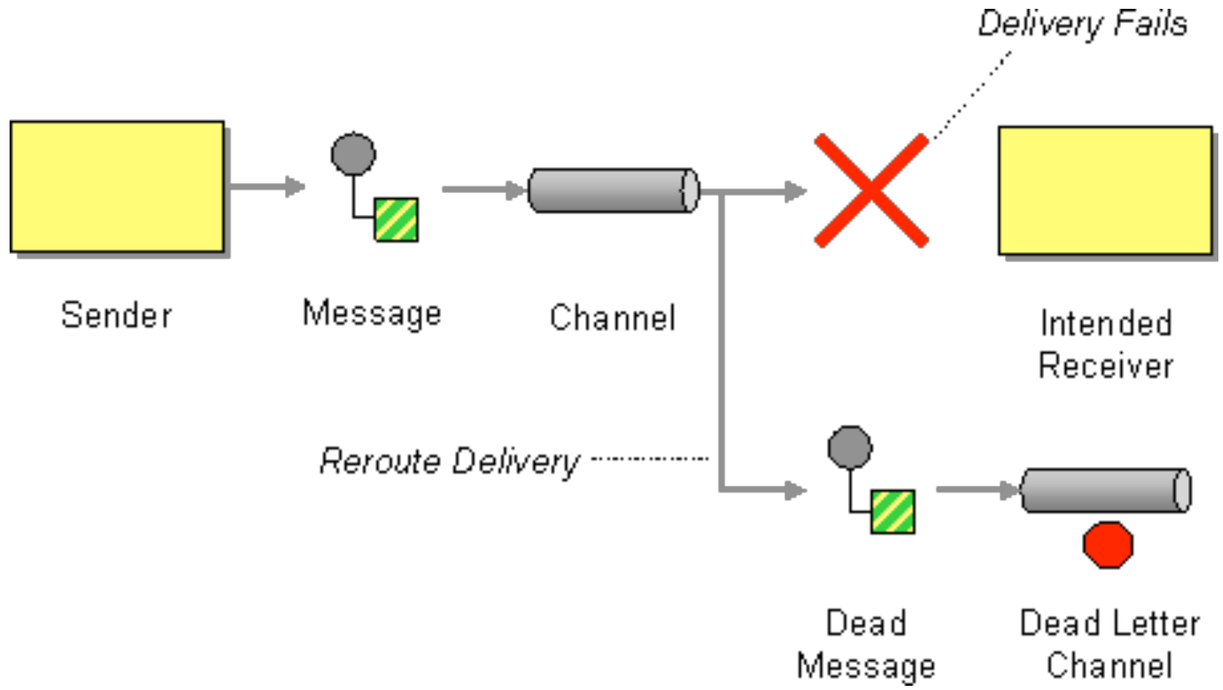

```
RouteBuilder builder = new RouteBuilder() {
    public void configure() {
      from("activemq:NewOrders").
      choice().
         when(header("product").
            isEqualTo("widget")).
            to("http://remotehost:8888/someApp/").
            to("activemq:Orders.Widgets").
         when(header("product").
            isEqualTo("gadget")).
            to("ftp://bsnyder@remotehost/private/reports/").
            to("activemq:Orders.Gadgets").
         otherwise().
            to("activemq:ERRORS");
      }
   };
}
```

# Message Routing: Content-Based Router



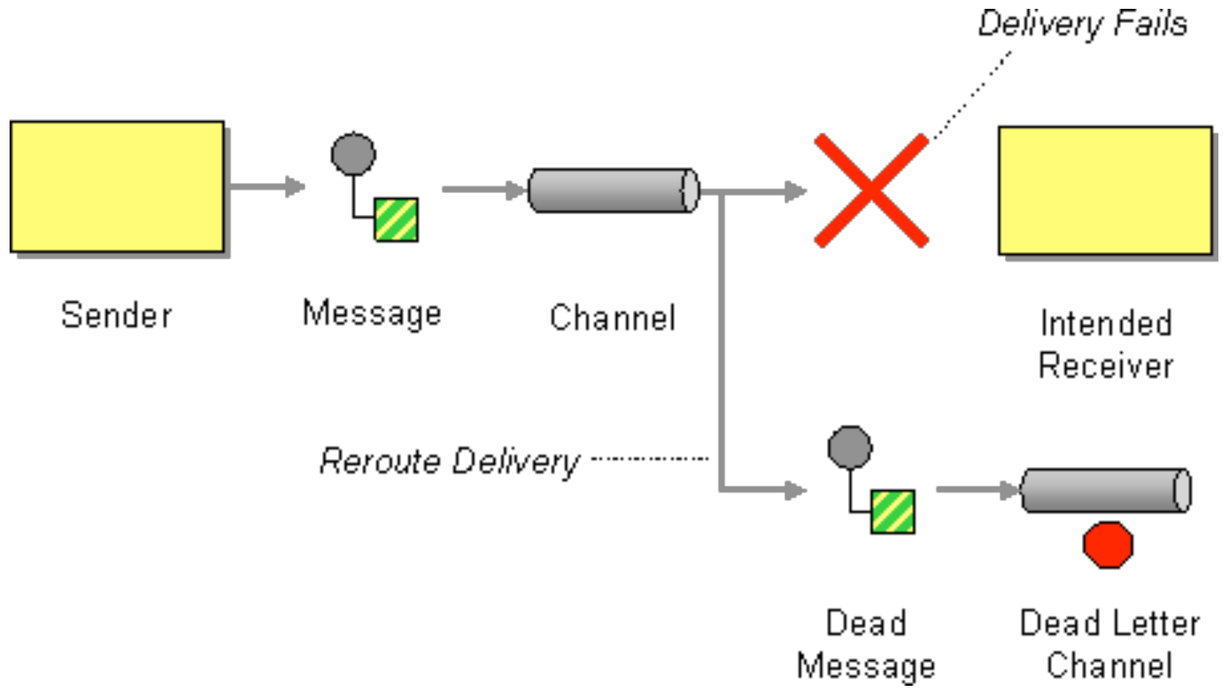

```
<route>
  <from uri="activemq:NewOrders"/>
  <choice>
    <when>
      <xpath>/order/product = 'widget'</xpath>
      <to uri="activemq:Orders.Widgets"/>
    </when>
    <when>
      <xpath>/order/product = 'gadget'</xpath>
      <to uri="activemq:Orders.Gadgets"/>
    </when>
    <otherwise>
      <to uri="activemq:ERRORS"/>
    </otherwise>
  </choice>
</route>
```

# Message Channels: Dead Letter Channel



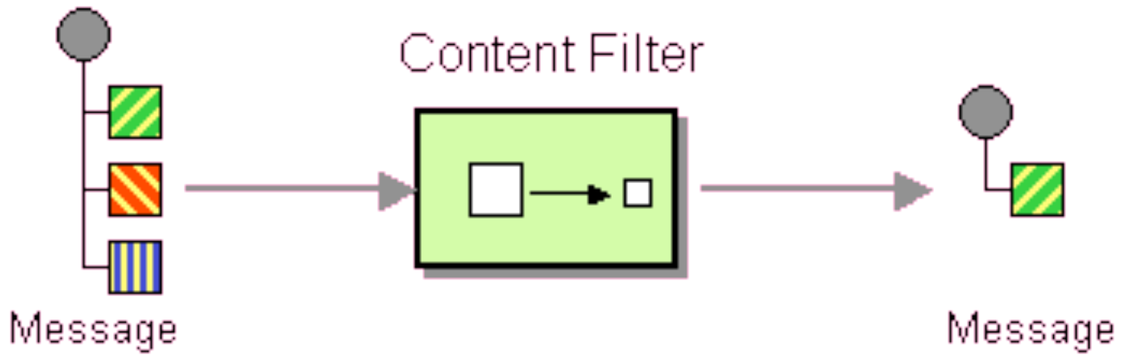

```
RouteBuilder builder = new RouteBuilder() {
    public void configure() {
        errorHandler(deadLetterChannel("activemq:ERRORS.QUEUE"));

        from("file:///tmp/MyFile.txt?delay=2000").
        to("activemq:TEST.Q");
        }
    };
}
```

# Message Channels: Dead Letter Channel



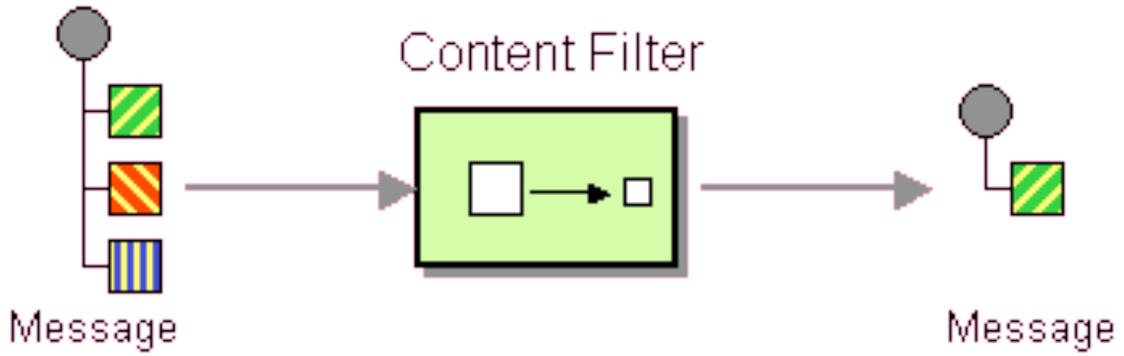

```
<bean id="errorHandler" class="org.apache.camel.builder.DeadLetterChannelBuilder"
  p:deadLetterUri="smtp://mail.springsource.com:25" />
<camelContext id="camel" errorHandlerRef="errorHandler"
  xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="seda:a" />
    <to uri="seda:b" />
  </route>
</camelContext>
```

# Message Transformation: Content Filter




```
RouteBuilder builder = new RouteBuilder() {
    public void configure() {
        from("activemq:THIS.QUEUE").
            process(new Processor() {
                public void process(Exchange e) {
                    Message in = exchange.getIn();
                    in.setBody(in.getBody(String.class) + " Ride the Camel!");
                }
            }).
        to("activemq:THAT.QUEUE");
        }
    };
}
```

# Message Transformation: Content Filter




```
<bean id="transformerBean" class="com.mycompany.orders.transform.MyTransformerBean" />
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <from uri="activemq:THIS.QUEUE" />
  <bean ref="transformerBean" method="transformMessage" />
  <to uri="activemq:THAT.QUEUE" />
</camelContext>
```

# Camel Pattern: Throttler

- Limit the number of messages to be sent in a given time window

```
public class MyRouteBuilder extends RouteBuilder {
    public void configure() {
      from("activemq:TEST.QUEUE”).
        throttler(3).timePeriodMillis(10000).
        to("http://remotehost:8888/meticProcessingService");
    }
}
```

(only send three messages every 10 seconds)

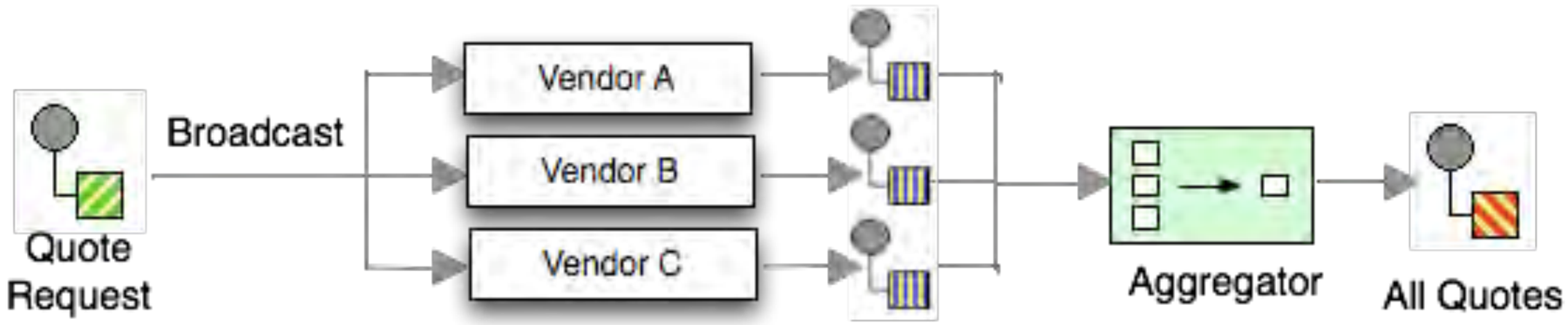
# Camel Pattern: Delayer

- Impose a simple delay on messages before being sent along

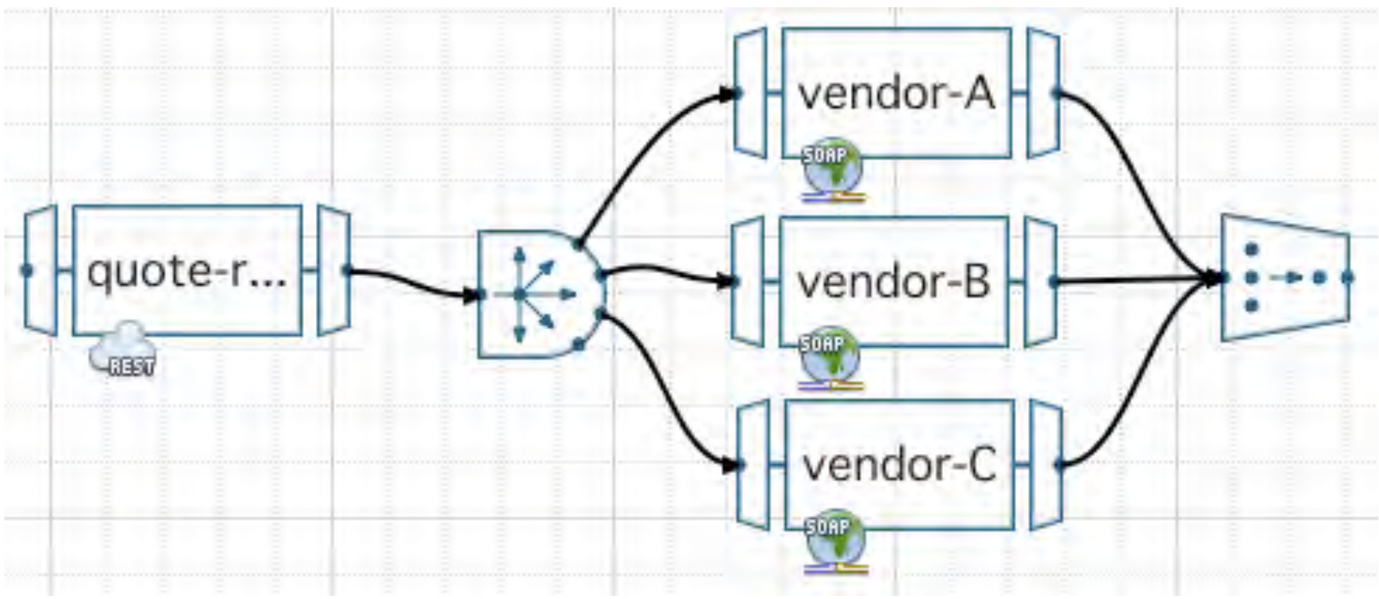
```
public class MyRouteBuilder extends RouteBuilder {
    public void configure() {
      from("activemq:TEST.QUEUE").
        delayer(header("JMSTimestamp", 3000).
        to("http://remotehost:8888/meticProcessingService");
    }
}
```

(delay messages for a duration of JMSTimestamp value plus three seconds)

# Message Routing: Scatter-Gather ("all")

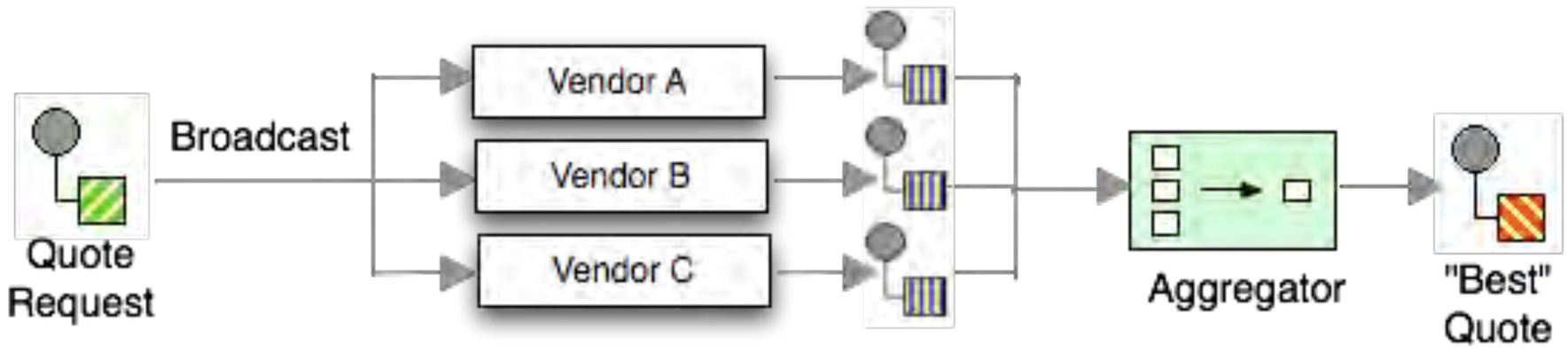



```
rest "quote-request"
soap "vendor-A"
soap "vendor-B"
soap "vendor-C"

route do
    from "quote-request"
    broadcast do
        route to "vendor-A"
        route to "vendor-B"
        route to "vendor-C"
    end
    aggregate set "all-quotes"
end
```

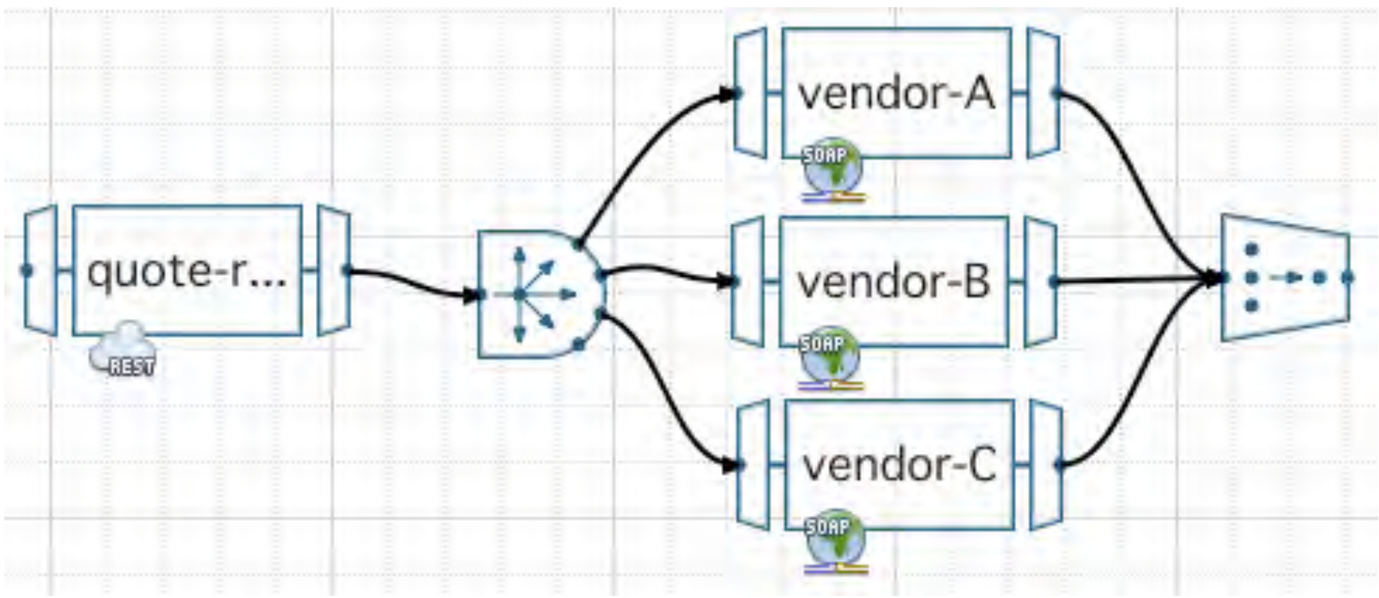



## Fuji - DSL and Web UI

# Message Routing: Scatter-Gather ("best")

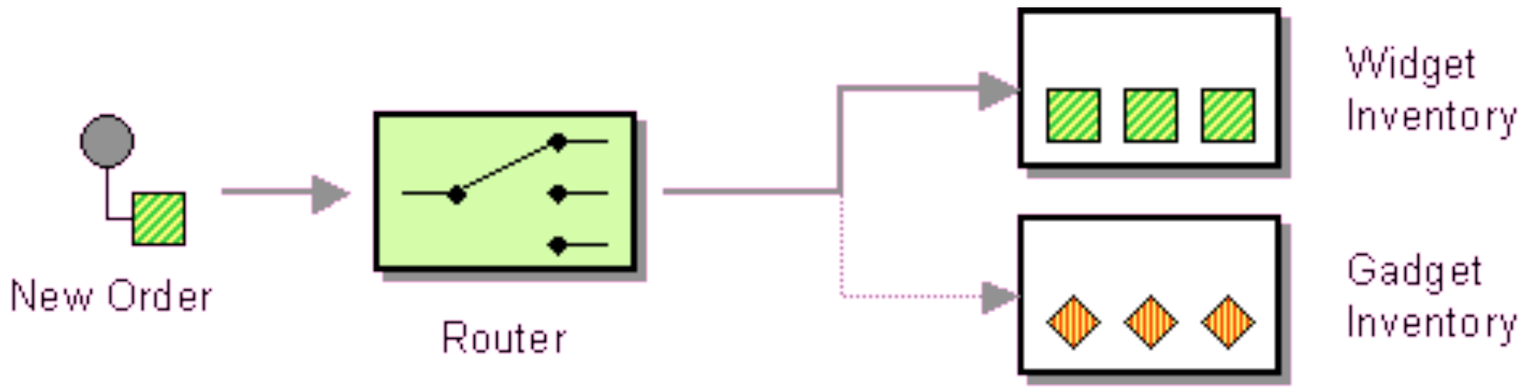



```
rest "quote-request"
soap "vendor-A"
soap "vendor-B"
soap "vendor-C"

route do
    from "quote-request"
    broadcast do
        route to "vendor-A"
        route to "vendor-B"
        route to "vendor-C"
    end
    aggregate java "best-quote"
end
```

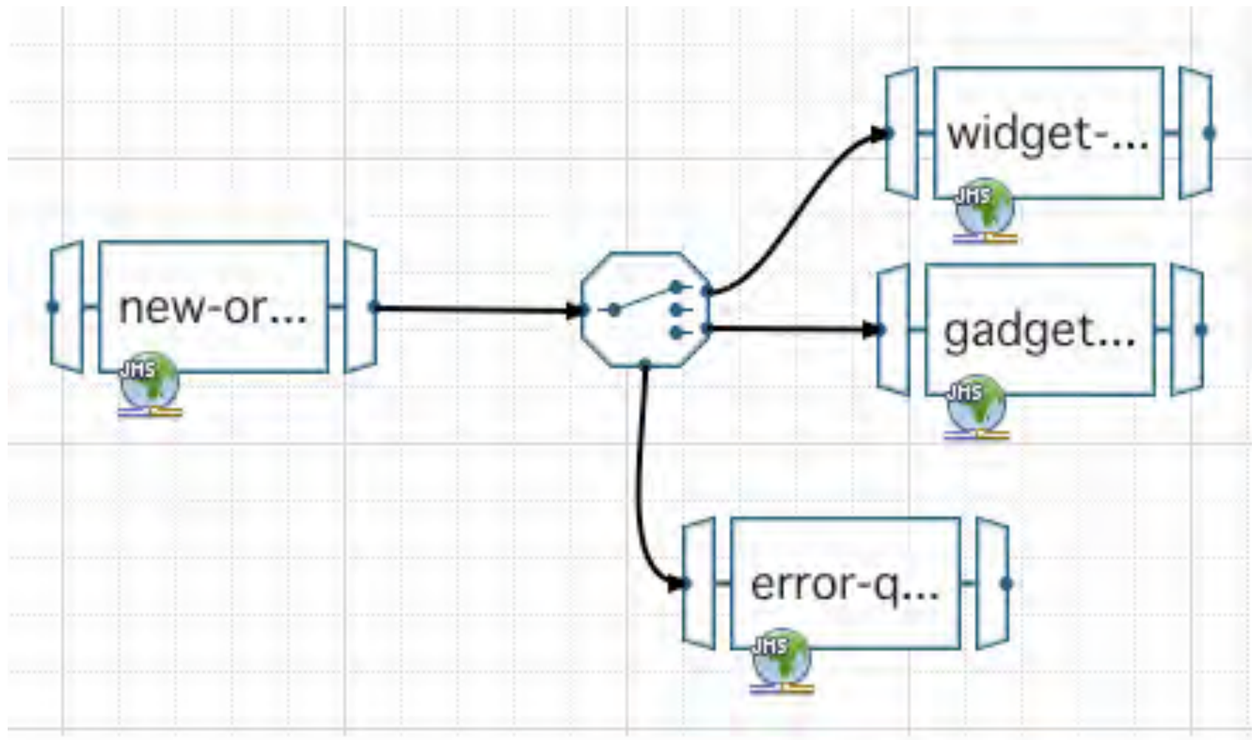



## Fuji - DSL and Web UI

# Message Routing: Content-Based Router

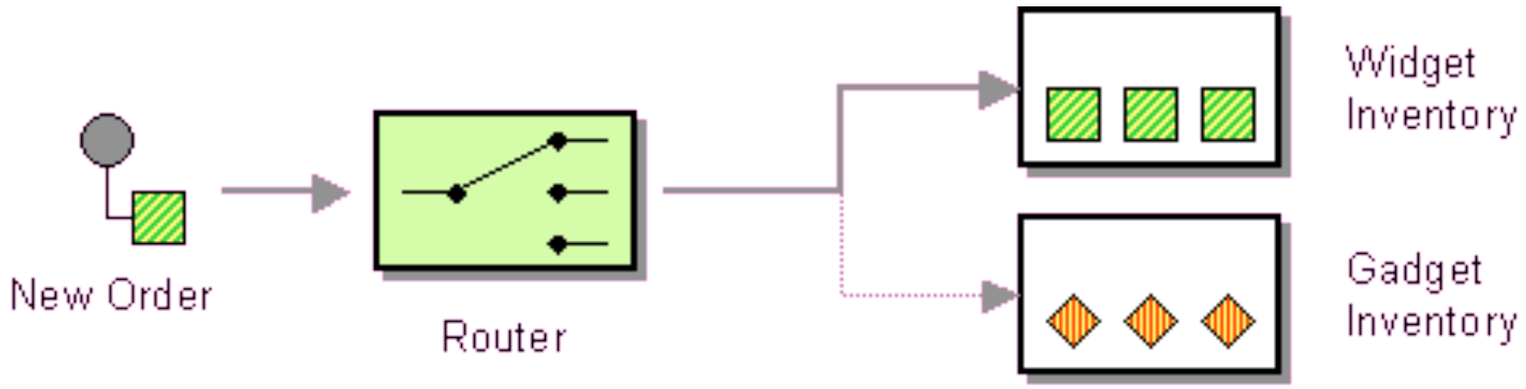



```
jms "new-orders"
jms "widget-orders"
jms "gadget-orders"
jms "error-queue"

route do
    from "new-orders"
    select xpath("/product") do
        when "widget" to "widget-orders"
        when "gadget" to "gadget-orders"
        else to "error-queue"
    end
end
```

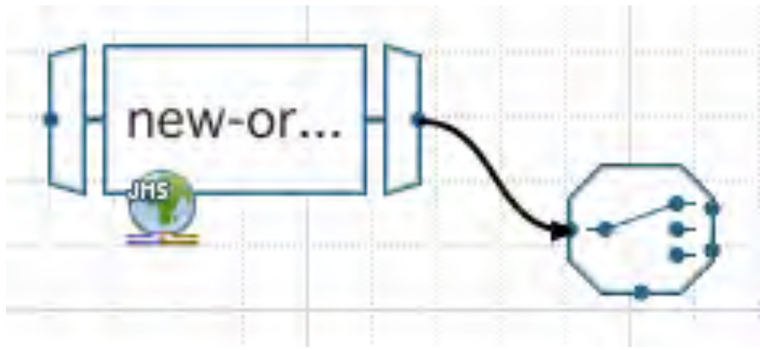



Fuji - DSL and Web UI

# Message Routing: Content-Based Router (dynamic)



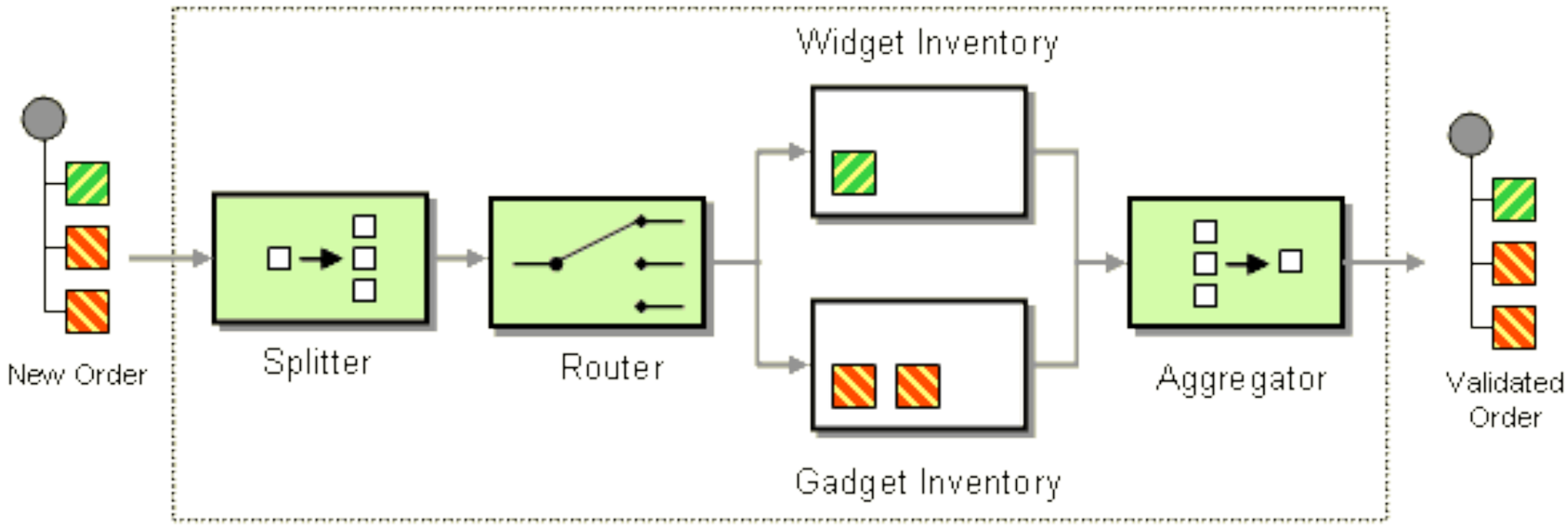

```
jms "new-orders"

route do
    from "new-orders"
    select xpath "my-dynamic-CBR"
end
```



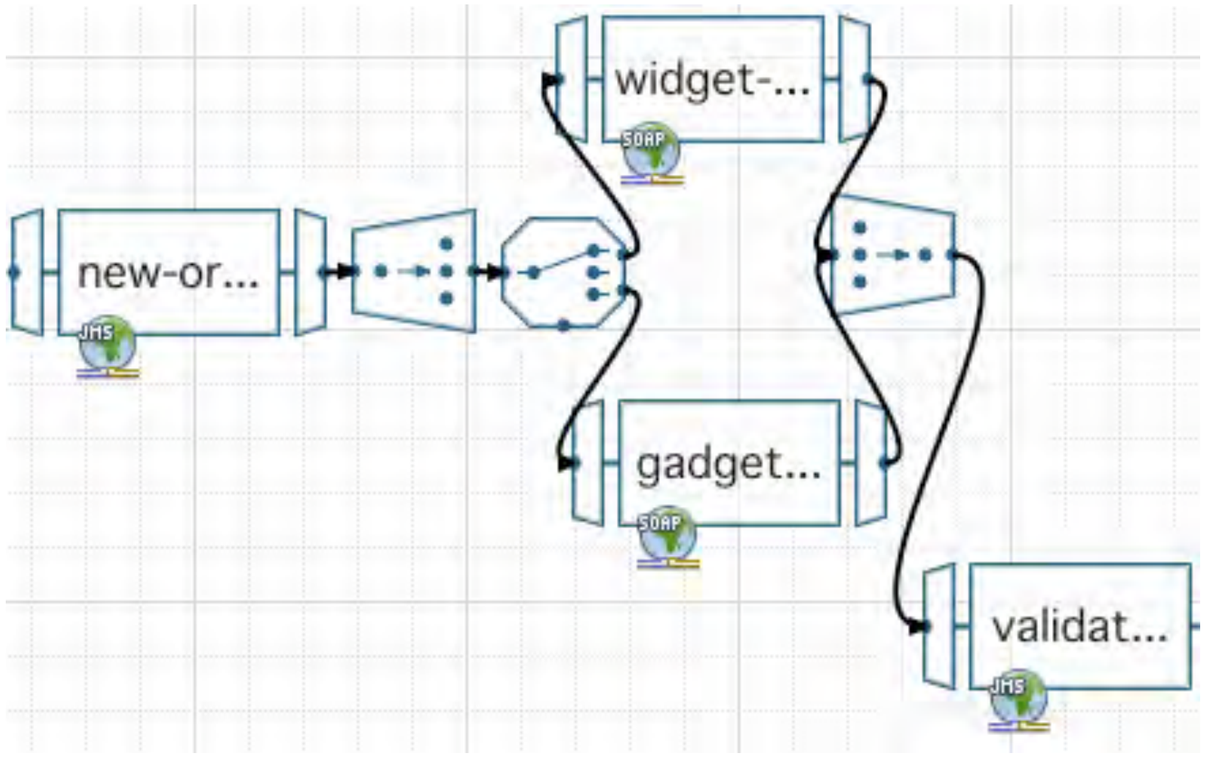

- CBR rules configured at deployment/runtime

Fuji - DSL and Web UI

# Message Routing: Composed Message Processor




```
jms "new-order"
soap "widget-inventory"
soap "gadget-inventory"
jms "error-queue"
jms "validated-order"

route do
    from "new-order"
    split xpath("/item")
    select xpath("/product") do
        when "widget" to "widget-inventory"
        when "gadget" to "gadget-inventory"
        else to "error-queue"
    end
    aggregate set "all-orders"
    to "validated-order"
end
```




Fuji - DSL and Web UI

# EIP Product Demo

- Apache Camel
- Project Fuji


# JavaOne℠

# Thank You

Andreas Egloff
andreas.egloff@sun.com

Bruce Snyder
bruce.snyder@springsource.com