IT in IB: Enterprise Integration in Electronic Execution

Georgy Bolyuba
Cash Equities IT, Barclays

Who I am and what I am talking about

Working with Java for 10 years, last 5 of them in Czech Republic

- Monster
- IDC
- Barclays

Here to talk about

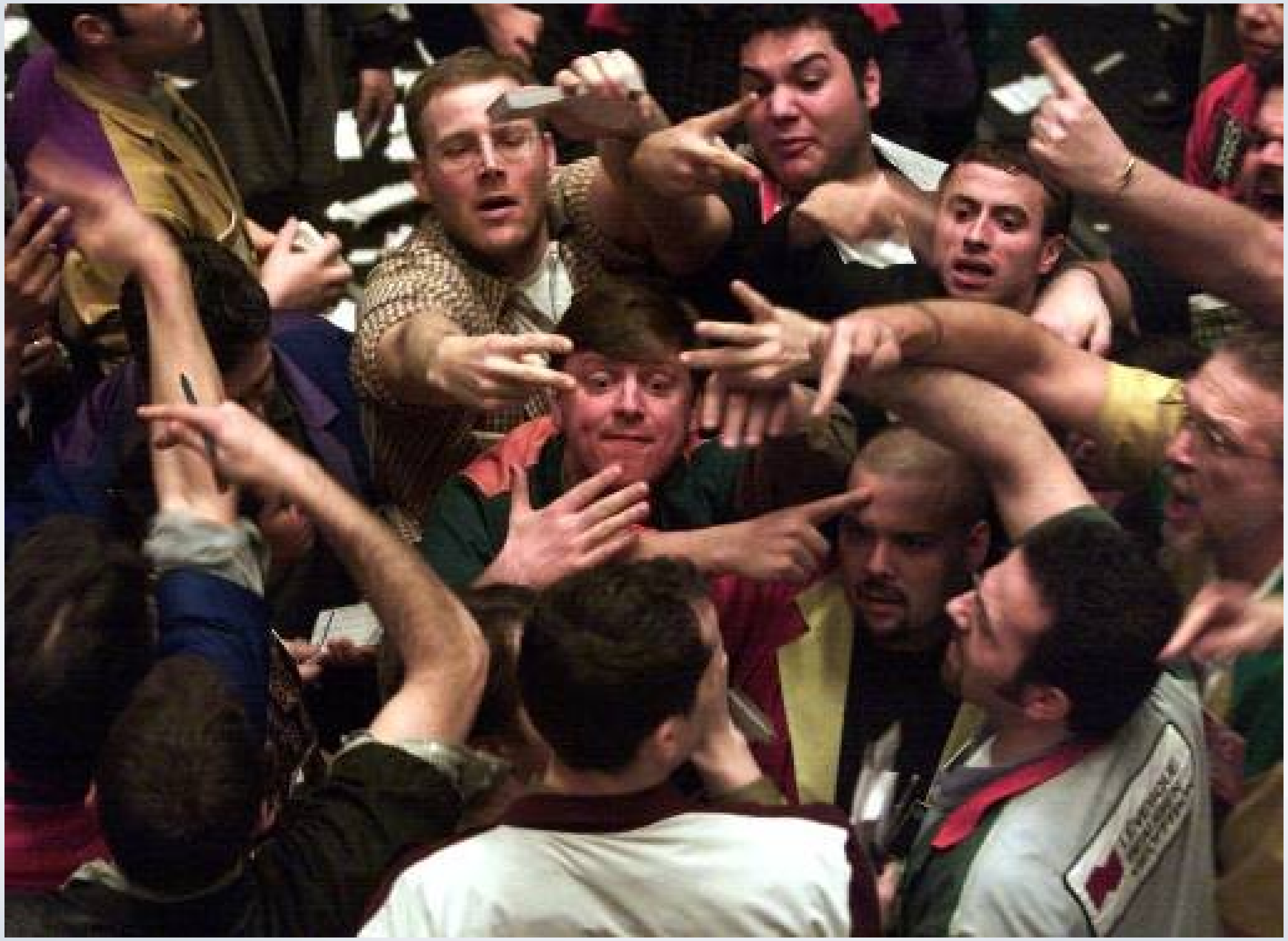
- Enterprise Integration in Electronic Execution
- Enterprise Integration Patterns (EIP)
- Apache Camel and Spring Integration

Investment Banking

- Risks: investment banks help clients deal with it
- Equity (Stock) Markets: heard about them?
- Equities Business
  - Buying/Selling shares on orders from clients
  - Heavily regulated business
  - Very competitive environment

Stock Market

Used to look like this:



Stock Market

Now it looks like this:

Electronic trading

- Dominates Equities Markets globally
- Makes IB happy: computers do not make mistakes and they are faster than people
- Makes regulators happy: lots of data available for analysis
- Makes customers happy: markups are tiny, competition is fierce

Better execution, better reporting, better risk management. Technology is the answer to all of these. Lets roll out sleeves and get to work...

Electronic trading

Soon enough you are dealing with hundreds of applications and systems

- Order Management
- Execution (Best execution, Algos)
- Booking and allocation
- Reporting and compliance
- Risk management
- Settlement

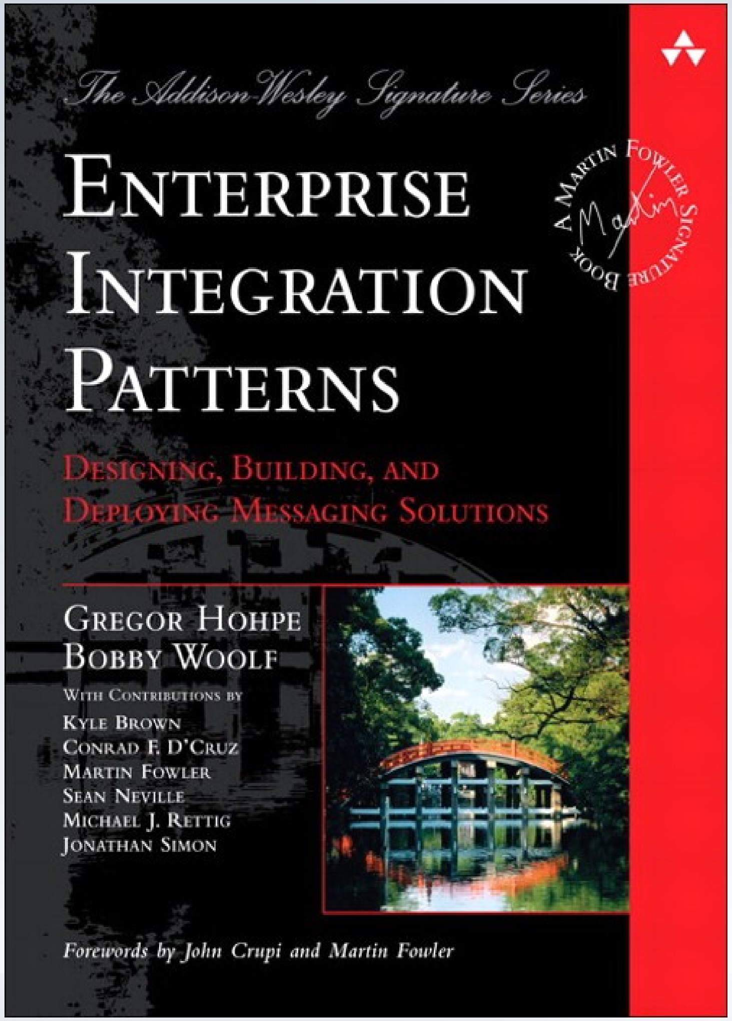
Add to that the fact that IT has to change together with the business and you have a pretty complicated integration problem to solve

Electronic trading

Messaging seems to be a natural fit

- Message usually self descriptive (has all the info you need on it)
- Shared domain model (Order, Execution, etc)
- Processing is event driven and can usually be async
- Great degree of decoupling is achievable (virtually zero assumptions about up stream or downstream systems are required)

EIP


The Addison-Wesley Signature Series

ENTERPRISE INTEGRATION PATTERNS

DESIGNING, BUILDING, AND DEPLOYING MESSAGING SOLUTIONS

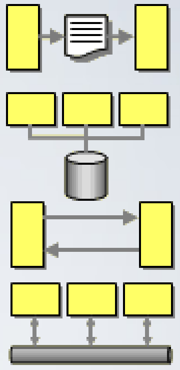
A MARTIN FOWLER SIGNATURE BOOK

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

- More than one solution
  - File Transfer
  - Shared Storage
  - RPC
  - Messaging



“Messaging is most difficult style, yet the most promising”

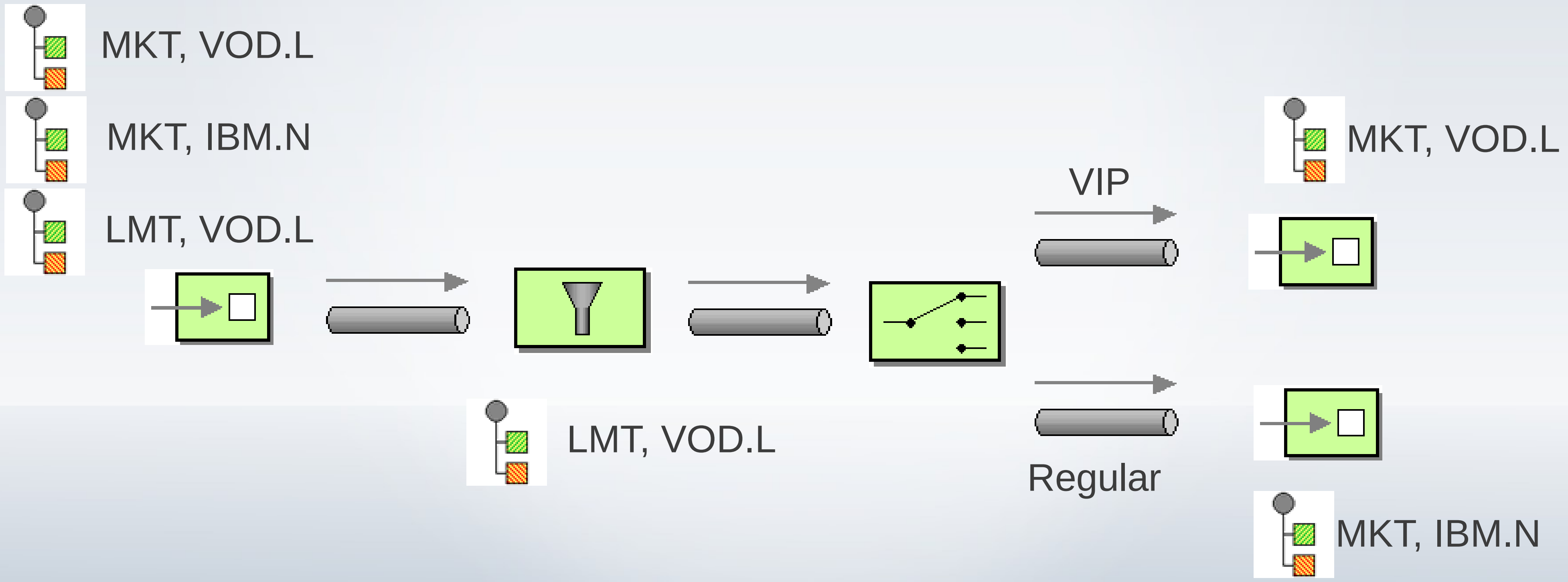
EIP: Basic concepts

- Message Channel
- Message
- Pipes and Filters
- Message Endpoint
- Message Translator
- Message Router

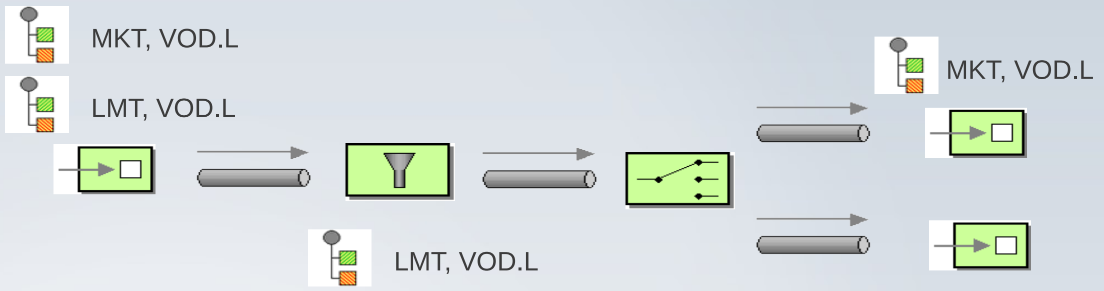
EIP: Simple example

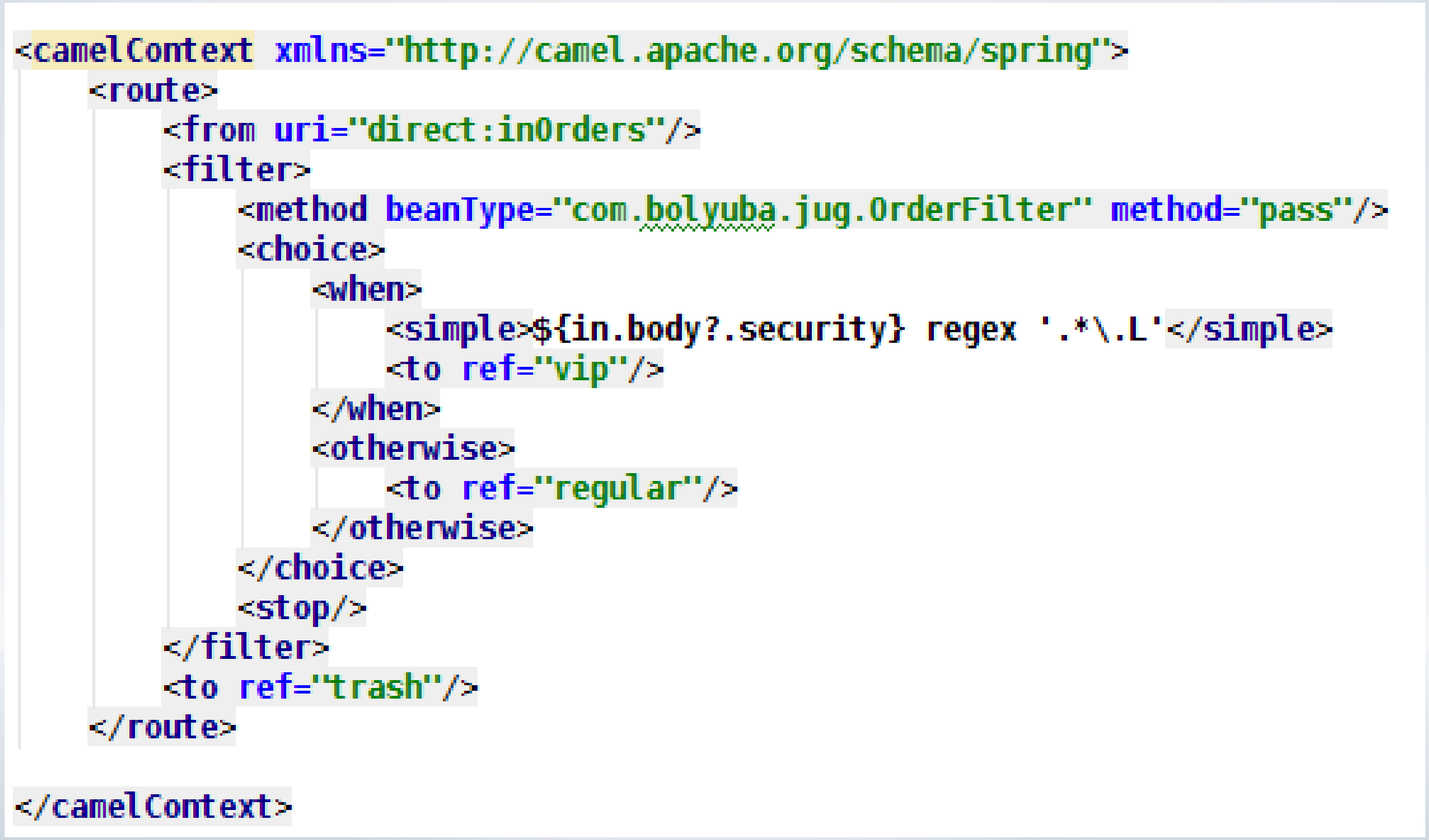
For every order that comes into the application do the following:

- Check order type (Accept only Market orders)
- Route orders based on symbol (*.L goes to a VIP channel)



Camel: Simple example



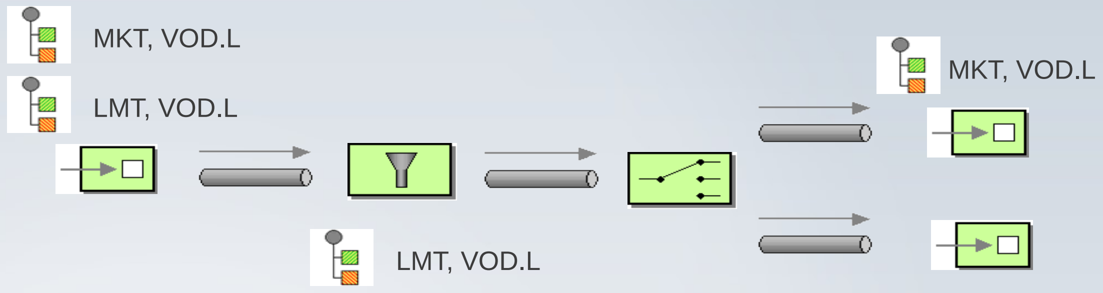


```
<camelContext xmlns="http://camel.apache.org/schema/spring">
    <route>
        <from uri="direct:inOrders"/>
        <filter>
            <method beanType="com.bolyuba.jug.OrderFilter" method="pass"/>
            <choice>
                <when>
                    <simple>${in.body?.security} regex '.*\.L'</simple>
                    <to ref="vip"/>
                </when>
                <otherwise>
                    <to ref="regular"/>
                </otherwise>
            </choice>
            <stop/>
        </filter>
        <to ref="trash"/>
    </route>

</camelContext>
```

Camel: Simple example

- Routes and endpoints everywhere
- Route takes care of connections
- Endpoints provide producers and consumers
  - Producer sends message to the messaging infrastructure
  - Consumer handles messages coming from messaging infrastructure
- URI is used to resolve enpoints
- Everything is synchronous by default

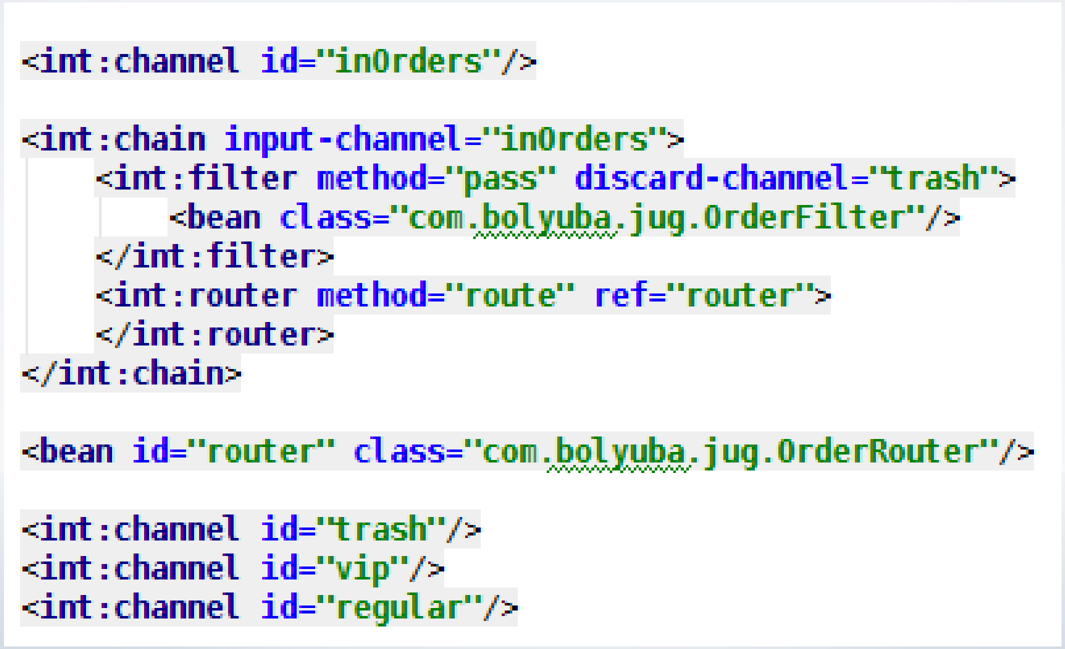
Spring Integration: Simple example



```
<int:channel id="inOrders"/>

<int:filter input-channel="inOrders" method="pass" output-channel="ordersForProcessing" discard-channel="trash">
	<bean class="com.bolyuba.jug.OrderFilter"/>
</int:filter>

<int:channel id="trash"/>

<int:channel id="ordersForProcessing"/>

<int:router input-channel="ordersForProcessing" method="route">
	<bean class="com.bolyuba.jug.OrderRouter"/>
</int:router>

<int:channel id="vip"/>
<int:channel id="regular"/>
```

Spring Integration: Simple example

- Channels everywhere
- No clear concept of route
- More of the “Pipes and Filters” flavor
- Everything is synchronous by default
- Alternative (closer to Camel):



```xml
<int:channel id="inOrders"/>

<int:chain input-channel="inOrders">
    <int:filter method="pass" discard-channel="trash">
        <bean class="com.bolyuba.jug.OrderFilter"/>
    </int:filter>
    <int:router method="route" ref="router">
    </int:router>
</int:chain>

<bean id="router" class="com.bolyuba.jug.OrderRouter"/>

<int:channel id="trash"/>
<int:channel id="vip"/>
<int:channel id="regular"/>
```

Messaging solution

- Spring integration
  - Takes EIP seriously (think Pipes & Filters)
  - Goal is to designing messaging bus using channels
  - Easy to understand, comes naturally (Channels)
  - Changing configuration is redesigning the channels network
  - Config might be a bit confusing once it starts to grow
- Camel
  - Embraces Endpoint pattern (more on this later)
  - Routes are more restrictive than channels (think WireTap in SI)
  - Localizes configuration by having everything related to the given route in one place
  - Config might be a bit …confusing (things like <stop/>, <to> inside <filter>, etc)

Messaging solution

- Both application deliver similar functionality when it comes to lightweight messaging solution
- If messaging is all you need, both will deliver
- Worth thinking about:
  - Camel uses Exchange pattern for Request-Reply. If you do not need that, might be a waist (Think low latency)
  - Both have concept of converters, Camel has a concept of pluggable DataFormat's. About 20 of supported out of the box
  - SI uses SpEl mostly (there some support for groovy and other scripting languages), Camel supports number of them out of the box (think filtering expressions)

Integration

Second part to the problem is actually the first part of the problem: application integration

- Both frameworks provide adapter base integration
  - SI: via ChannelAdapter and Gateways
  - Camel: via Component and ultimately Endpoint
- Both come with implementations
  - SI: ~ 20 available in 2.1.0
  - Camel: ~ 60 and counting
- Both are extensible (obviously) in a spirit of EIP

- Camel's two layer abstraction and URI based resolution gives extensions a nice framework while SI goes with lightweight “do what you want” type solution

Summary

- Both frame works deliver on both promises: messaging framework and integration
- Camel is a bit more mature (old?) and as the result has a wide range of components to choose from
- If unsure, flip a coin

Topics we did not cover

- Messages Persistence
  - SI: Storage is available on channel level
  - Camel: Nothing, delegates it to endpoint
- Parallel/Async processing: Both support that (via Task Executors, Polling Consumers, etc). Who will come up on top here? (Exchange Emulator, Disruptor)
- Low latency messaging applications: is there a place for EIP framework there?
- Complex Event Processing: Can you map queries onto EIP patterns? (Testing Framework)
- Tooling (Fuse IDE for Camel)
- DSLs. Camel has lots of them :)

Questions?