padrões de **integração** de sistemas com spring integration 4.0

helderda**rocha**

helder@argonavis.com.br

Conteúdo

1. Padrões de Integração de Sistemas (EIP)

   Overview e principais componentes

2. Um problema de integração

   Solução teórica do problema usando EIP

3. Spring Integration

   Solução prática do problema com Spring Integration

4. Alternativas

Quem sou? Who am I? Кто я?

Helder da Rocha

tecnologia + ciência + arte

Java desde 1995

Web, XML, Design

+ de 2500 alunos

+ de 8000 horas de aula

argonavis.com.br

helderdarocha.com.br

argonavis.com.br

Por que integrar?

- Aplicações interessantes raramente vivem **isoladas**
  - **Sincronizar** emails, calendários, etc.
  - **Vincular** portal a um aplicativo de controle de estoque
  - **Integrar** com serviços e dados na nuvem
- Evitar reinventar a roda
  - Reusar serviços que funcionam bem
  - Pontos de integração existem
- Com integração aplicações podem ficar **melhores**

Desafios das soluções de integração

Redes **não são confiáveis**

Redes são **lentas**

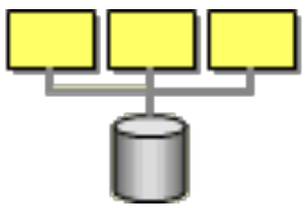
Aplicações são **diferentes**

Aplicações **mudam**

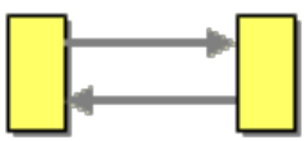
Soluções são **complexas**

Estratégias (estilos) de **integração**

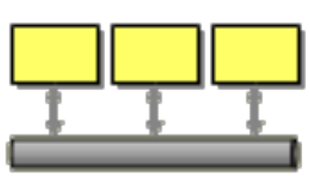
Transferência e Arquivos (1)



Banco de Dados Compartilhado (2)



RPC (3)



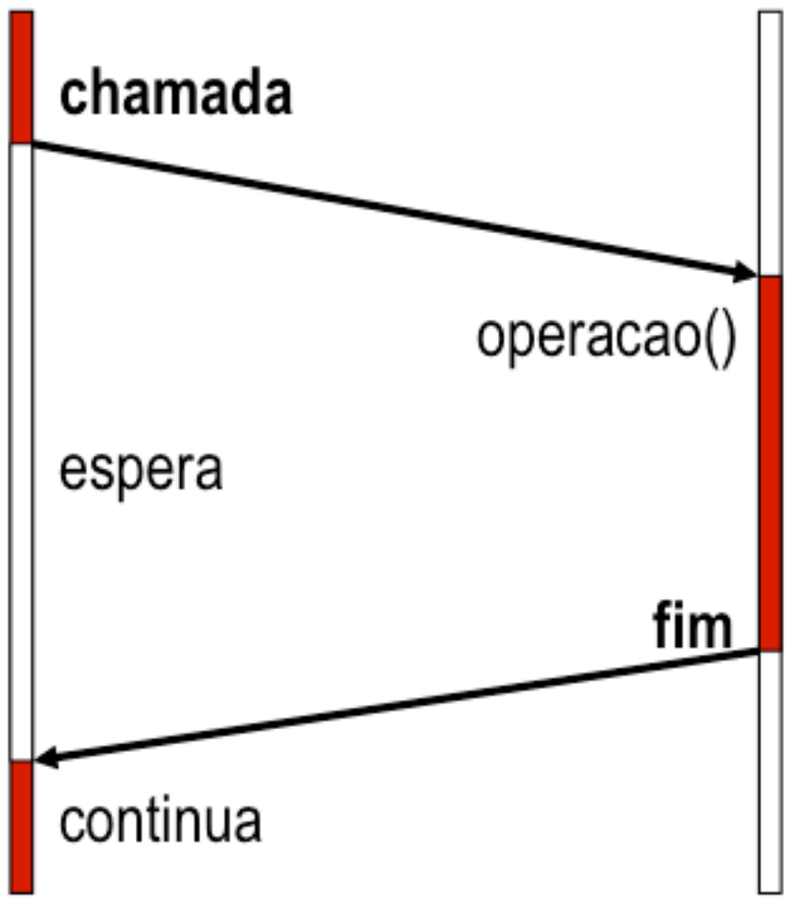
Mensageria (4)

Métodos de comunicação

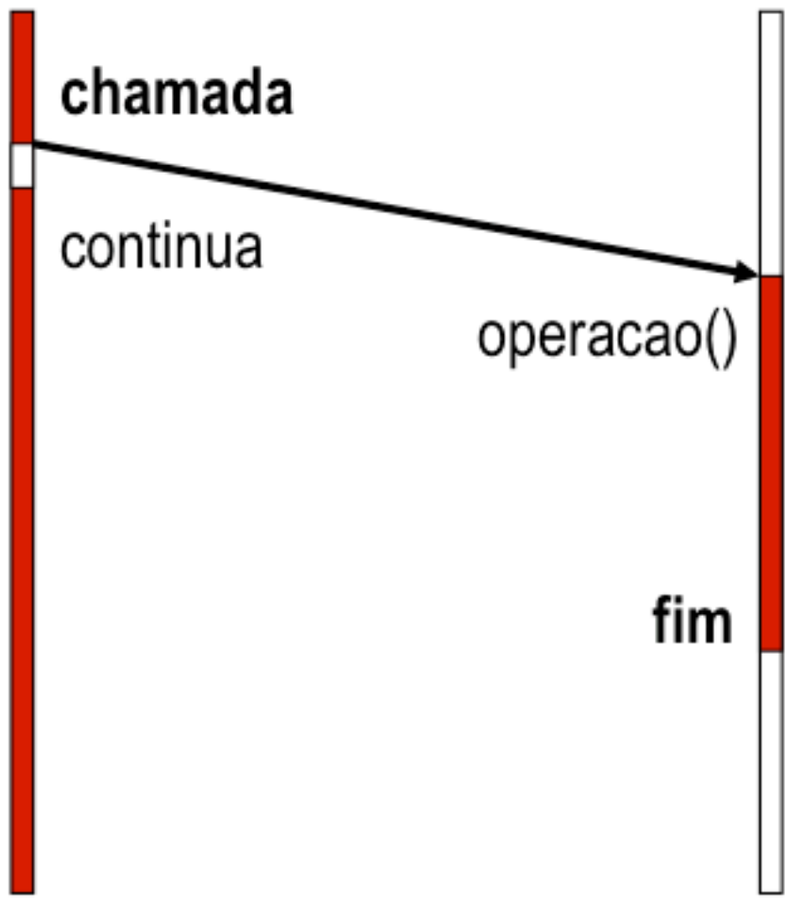
Síncrono: cliente chama o serviço e espera pelo fim da execução da operação

Assíncrono: cliente chama o serviço e não espera a operação terminar

RPC



Mensageria



O que é **mensageria**?

- Comunicação entre máquinas
  - Através de **mensagens** enviados em **canais** (filas) compartilhados entre as máquinas
  - Remetente, **produtor** de mensagens
  - Destinatário, **consumidor** de mensagens
- Mensagem
  - Estrutura de dados (objeto, string, tipo, bytes)
  - **Cabeçalho**, metadados
  - Corpo, **payload**

O que é um **sistema de mensageria**?

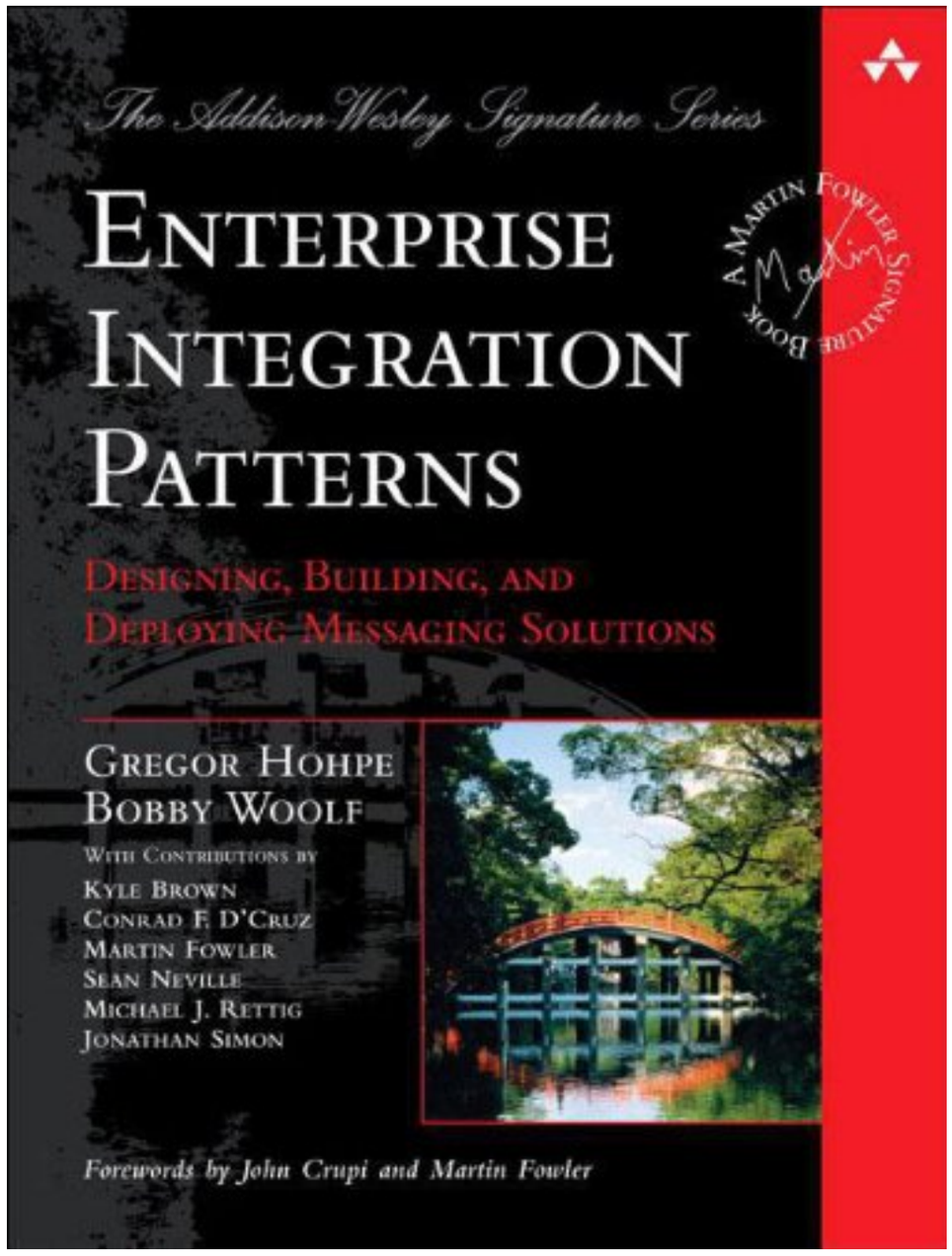
- Message Oriented Middleware
  - **Mediator** pattern (GoF) entre consumidores e produtores
  - Administra o sistema (**canais** e **conexões**)
- Etapas
  - **Criar** – produtor cria a mensagem e preenche com dados
  - **Enviar** – produtor adiciona mensagem a um canal
  - **Entregar** – sistema de mensageria transfere a mensagem de uma máquina para a outra tornando-a disponível ao consumidor
  - **Receber** – consumidor lê a mensagem do canal
  - **Processar** – consumidor extrai os dados da mensagem

Padrões de design

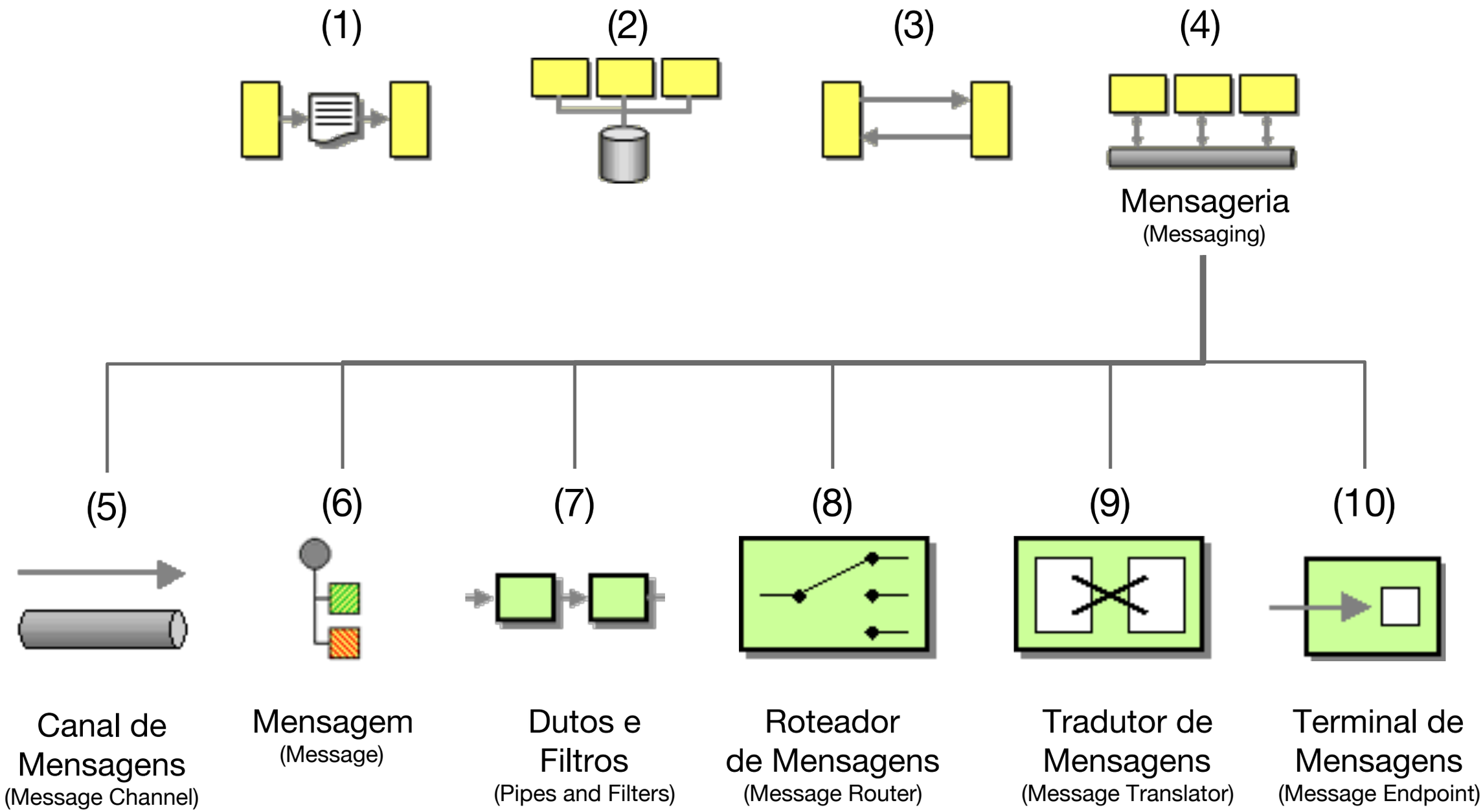
- **Abstração** de alto nível
  - Nome / Icone
  - Contexto
  - Problema / Solução / Exemplo
  - Vocabulário
  - Notação
  - Padrões relacionados
  - Conseqüências da aplicação do padrão

Padrões de integração

- 65 padrões de integração de sistemas
  - Aplicações em Java, .NET e outras plataformas
- 62 focam em soluções de mensageria
- Foco: minimizar o **acoplamento** entre componentes

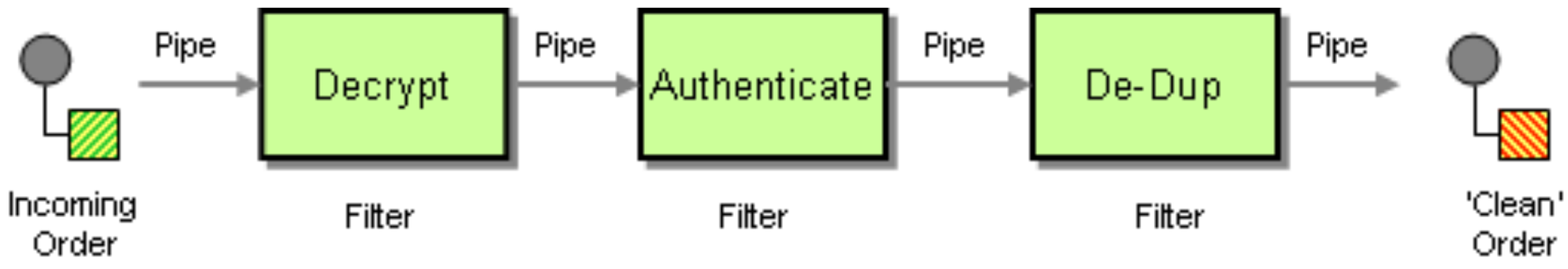


Padrões de Mensageria (4)



Arquitetura **dutos e filtros**

- Padrão de arquitetura (PEAA*)
  - Representa a **conexão** de **componentes** (filtros), em série, através de dutos (**canais**) que os conectam
  - Decorator (GoF**) e Intercepting Filter (PEAA)
  - Em vez de enviar a mensagem diretamente ao destinatário, pode-se **interceptá-la** em **etapas** intermediárias para validação do seu conteúdo, roteamento, transformação de dados, etc.
- Cada **componente** age como filtro; cada **canal** age como duto.
- Processamento **pipeline**; paralelismo

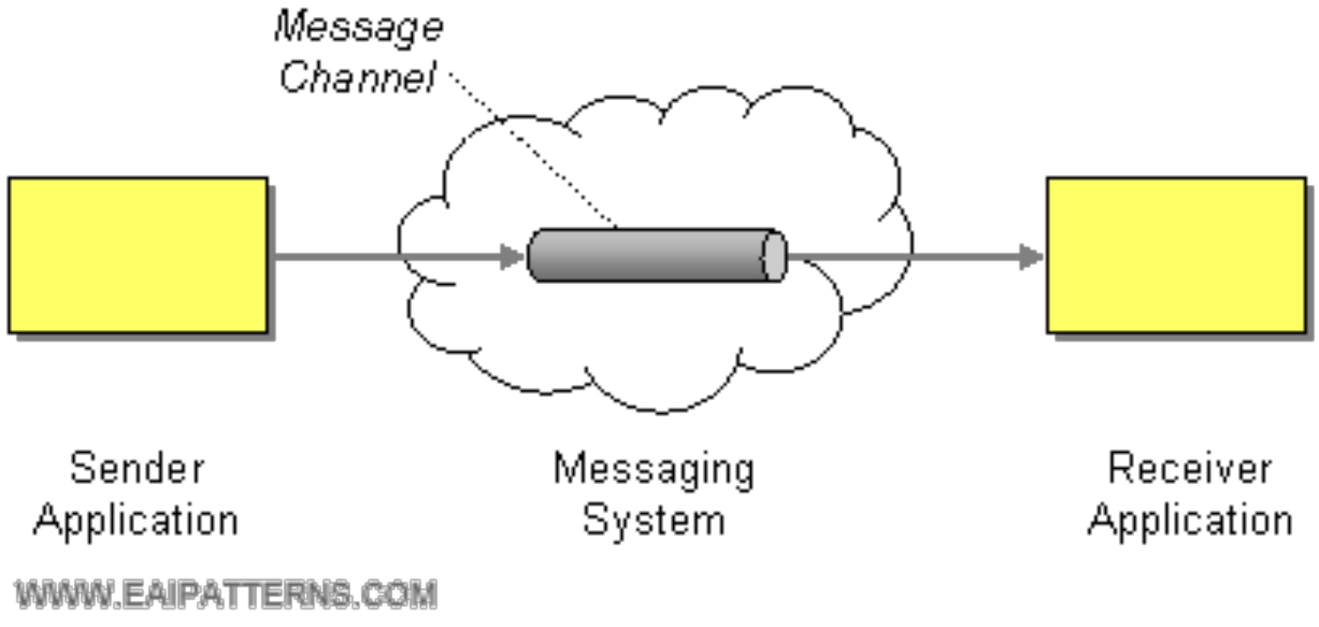


Fonte: eaipatterns.com

Message Channel

PROBLEMA “Como pode uma aplicação comunicar-se com outra aplicação usando mensageria?”

SOLUÇÃO “Conectar as aplicações usando um **Canal de Mensagens (Message Channel)**, onde uma aplicação grava informação no canal e a outra lê do canal.”

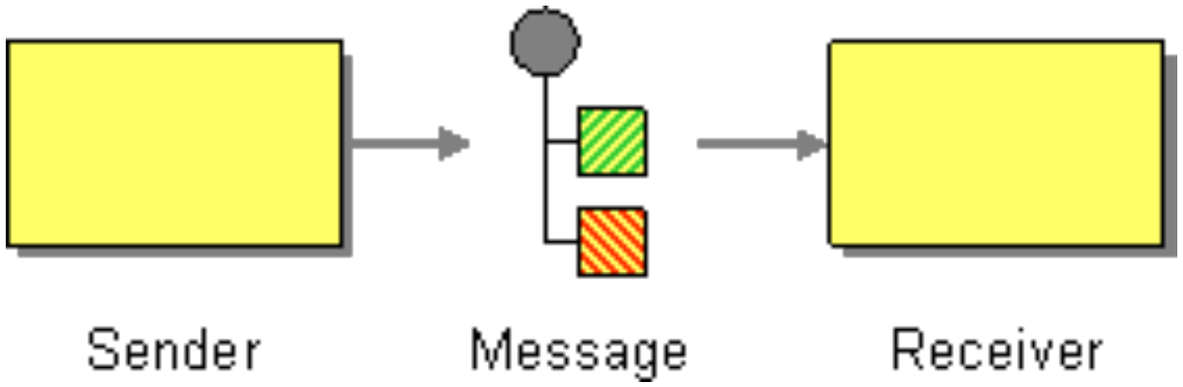


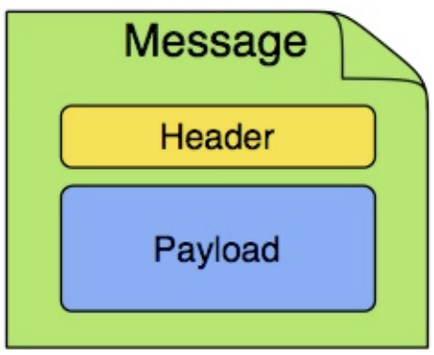
Fonte: eaipatterns.com

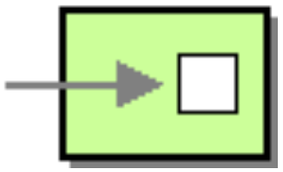
Message

PROBLEMA “Como é que duas aplicações conectadas por um canal de mensagens podem trocar informação?”

SOLUÇÃO “Empacote a informação dentro de uma **Mensagem (Message)**, um registro de dados que o sistema de mensageria pode transmitir através de um canal de mensagens.”



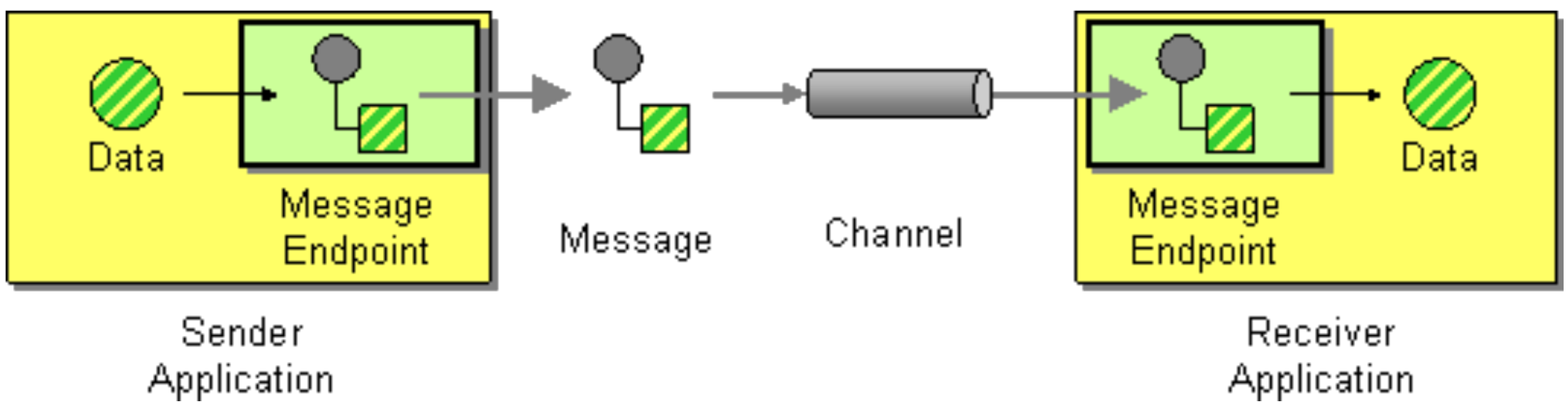




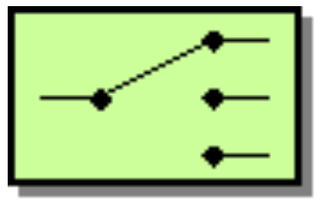
Message Endpoint

PROBLEMA “Como pode uma aplicação conectar-se a um canal de mensageria para enviar e receber mensagens?”

SOLUÇÃO “Conecte uma aplicaçao a um canal de mensageria usando um **Terminal de Mensagens (Message Endpoint)**, um cliente do sistema de mensageria que a aplicação pode usar para enviar ou receber mensagens.”



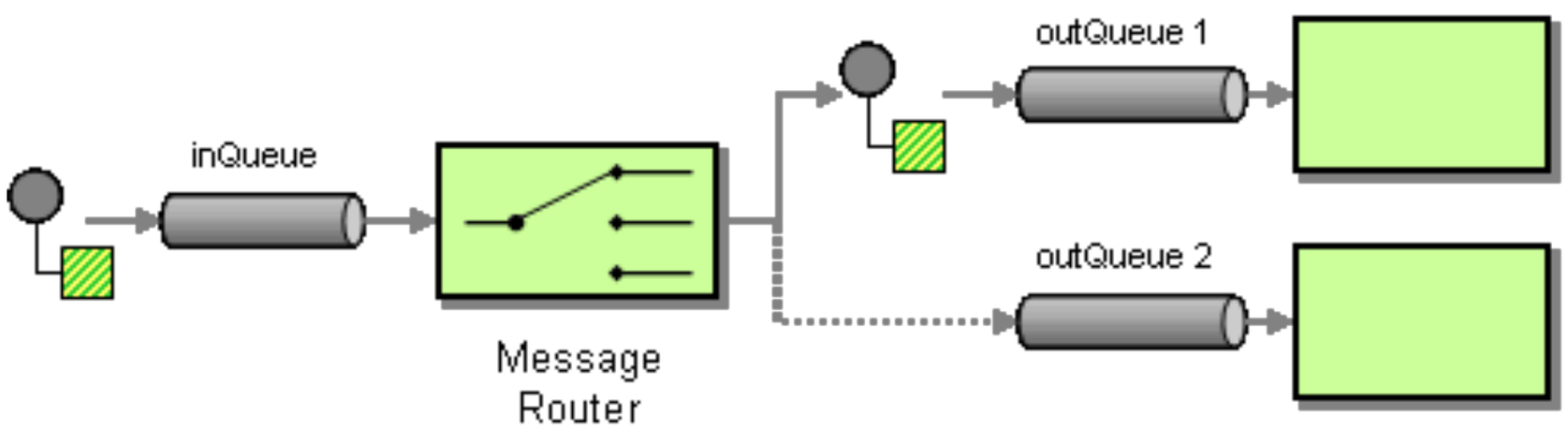
Imagens: eaipatterns.com



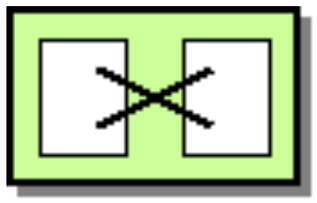
Message Router

PROBLEMA “Como desacoplar passos individuais de processamento de forma que mensagens possam ser passadas para diferentes filtros dependendo de uma série de condições?”

SOLUÇÃO “Use um filtro especial, um **Roteador de Mensagens (Message Router)** que consome uma Mensagem de um Canal de Mensagens **e** publique-a em outro Canal de Mensagens, dependendo de uma série de condições.”



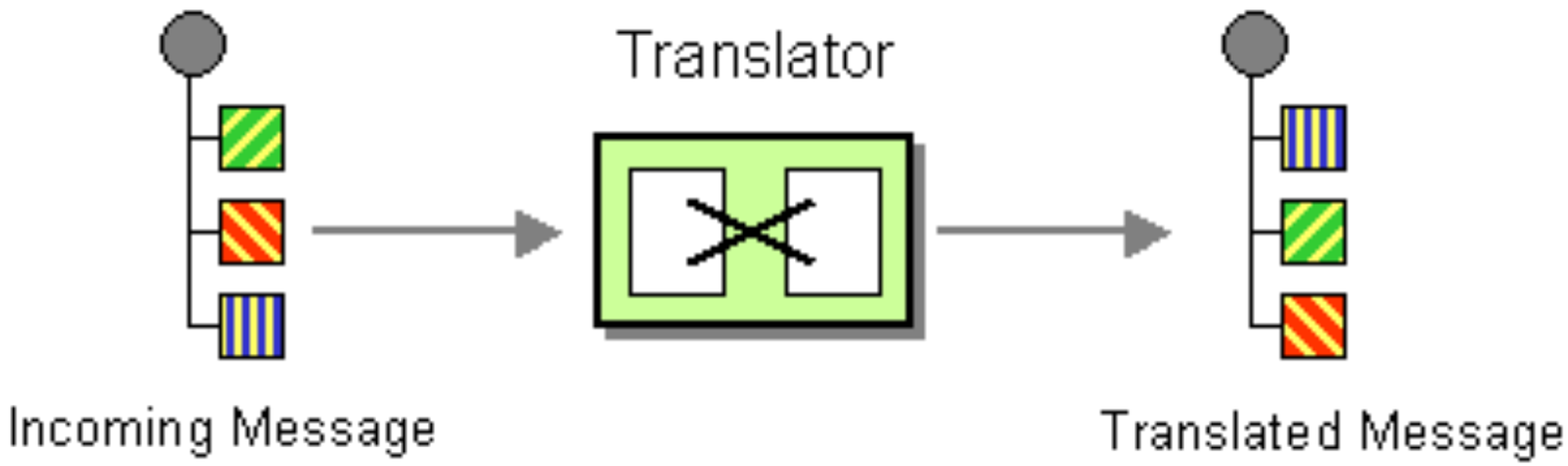
Imagens: eaipatterns.com



Message Translator

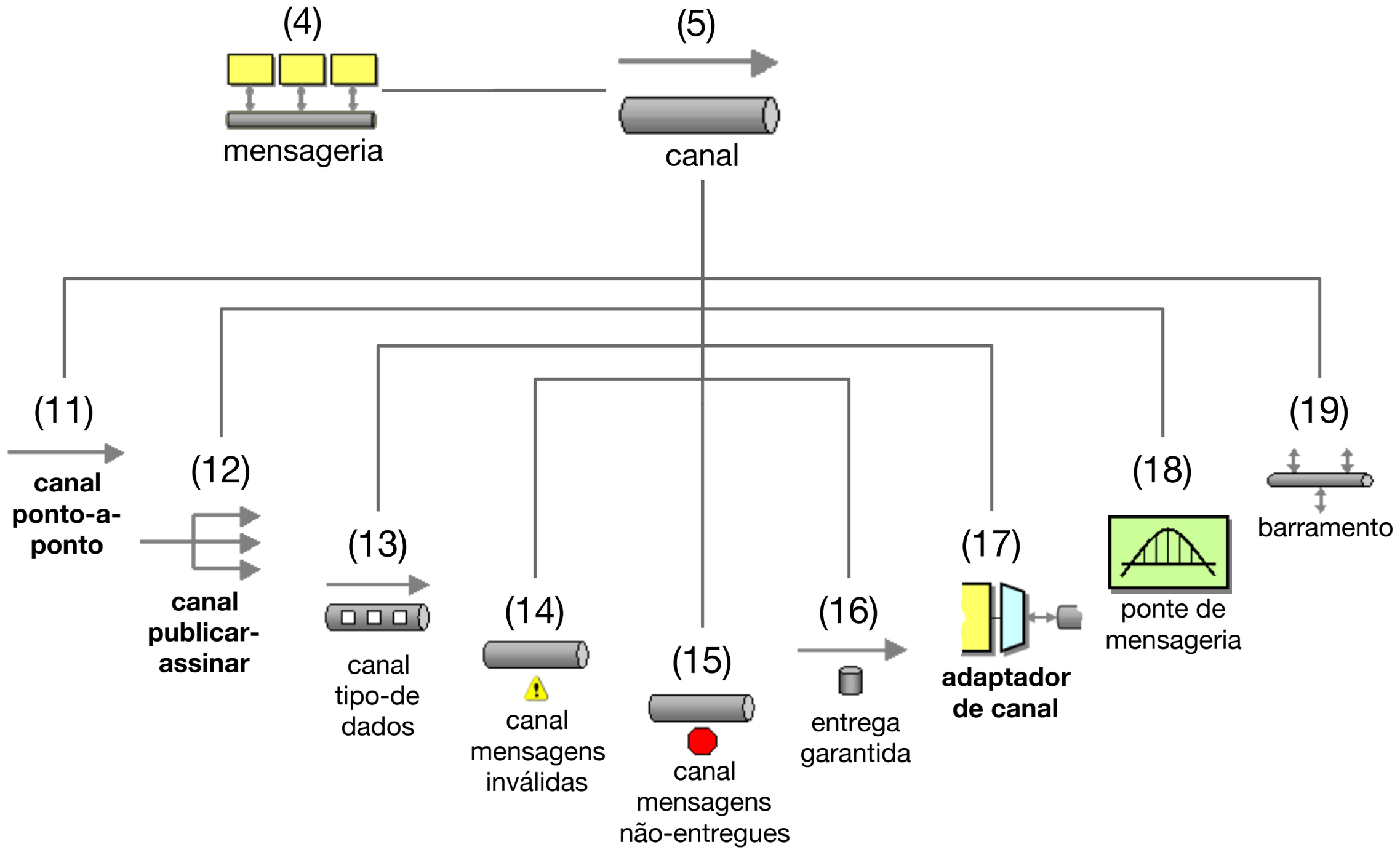
PROBLEMA “Como é possível realizar a comunicação usando mensageria, entre sistemas que usam formatos de dados diferentes?”

SOLUÇÃO “Use um filtro especial, um **Tradutor de Mensagens (Message Translator),** entre outros filtros ou aplicações para traduzir de um formato de dados para outro.”



Imagens: eaipatterns.com

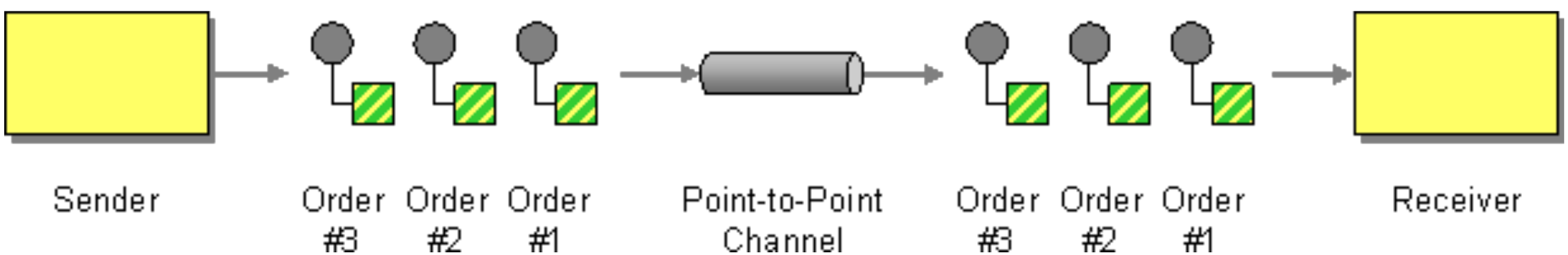
Padrões de Canal de Mensagens (5)



Point-to-point channel

PROBLEMA “Como o remetente pode ter certeza que apenas um destinatário irá receber a mensagem ou executar um comando?”

SOLUÇÃO “Envie a mensagem através de um **Canal Ponto-a-Ponto**, que garante que apenas um destinatário irá receber a mensagem.”

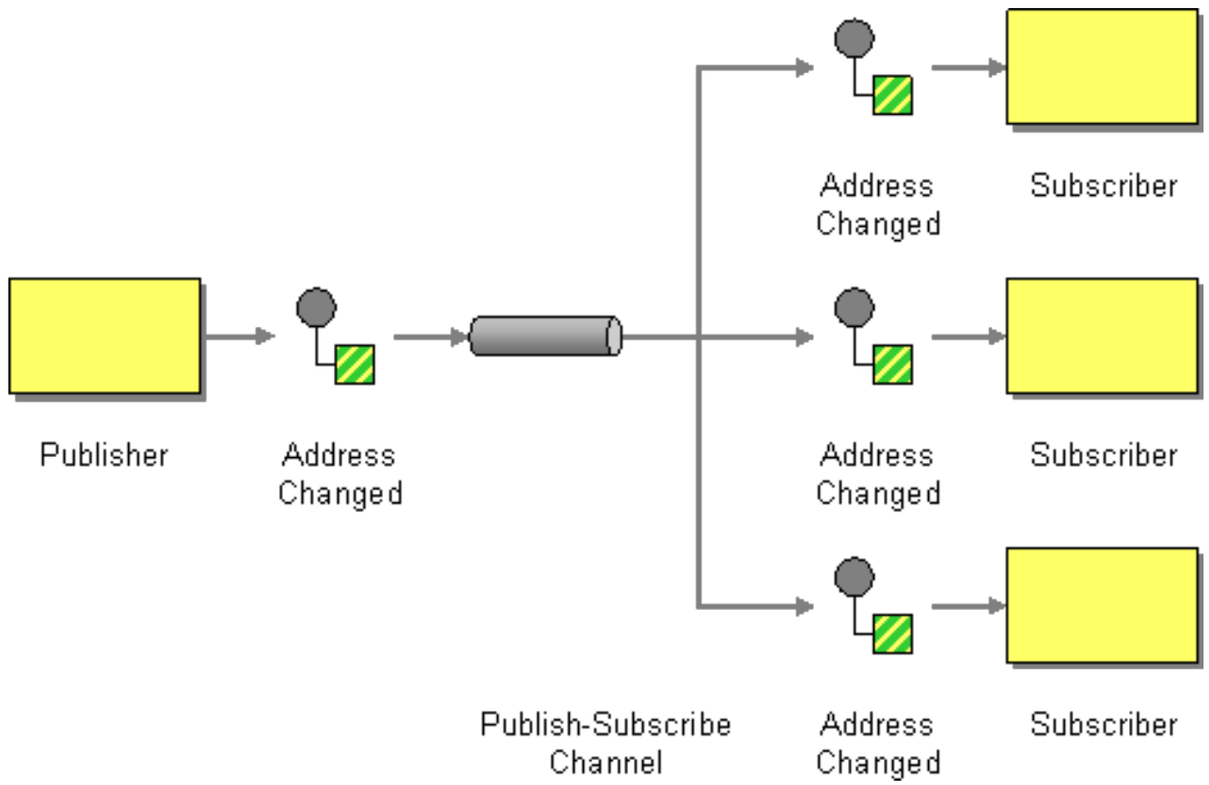


Imagens: eaipatterns.com

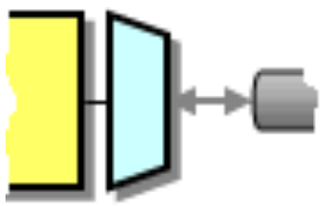
Publish-subscribe channel

PROBLEMA “Como pode um remetente transmitir um evento a todos os destinatários interessados?”

SOLUÇÃO “Envie o evento para um Canal Publicar-Inscrever, que entrega uma cópia do evento a cada destinatário.”



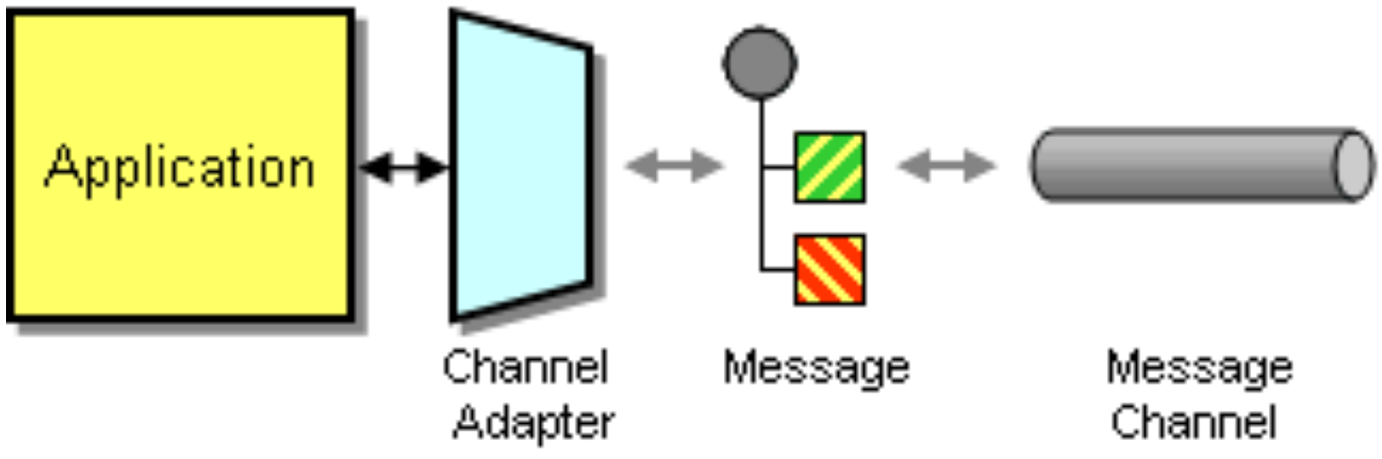
Imagens: eaipatterns.com



Channel Adapter

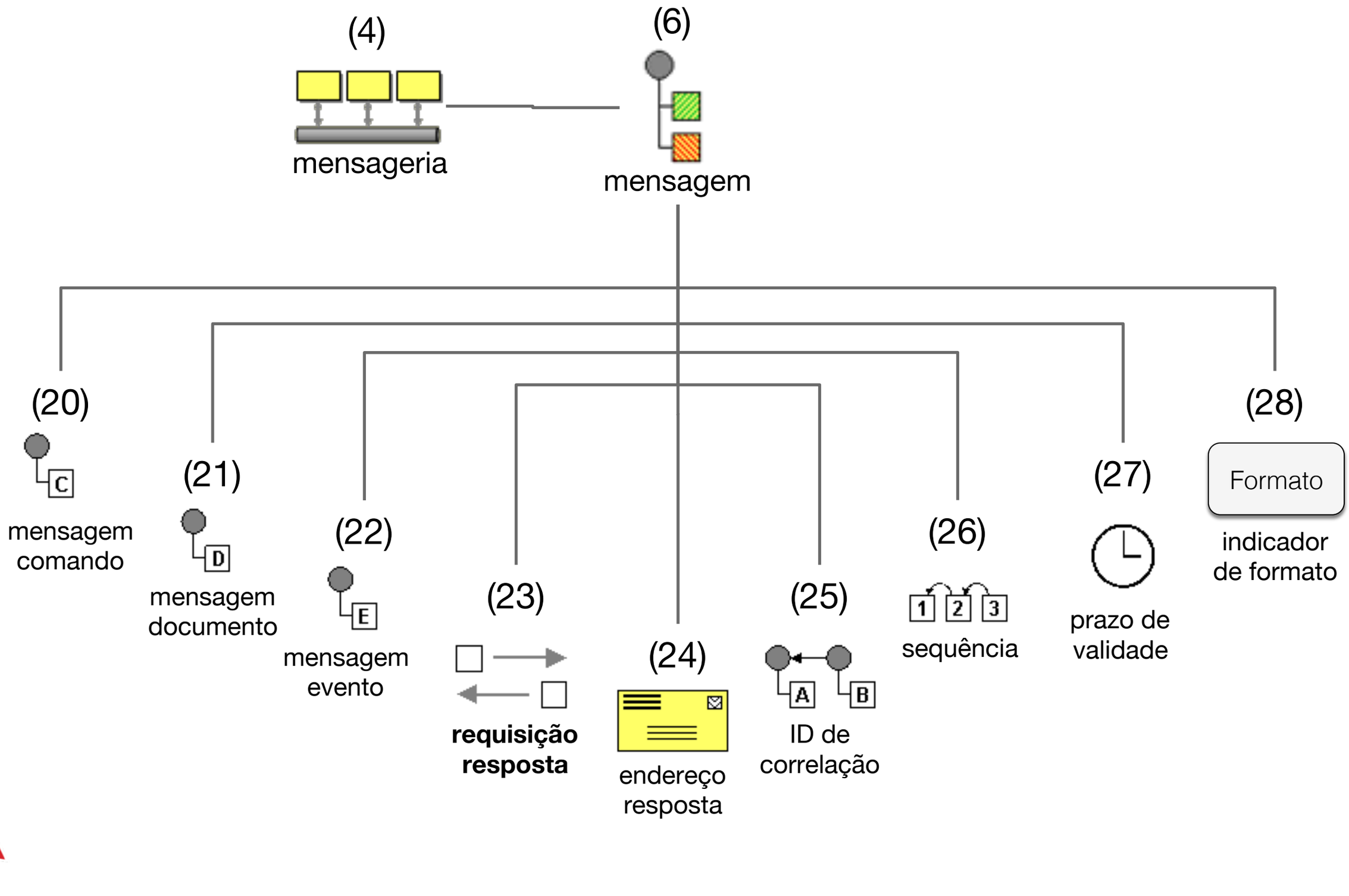
PROBLEMA “Como conectar uma aplicação ao sistema de messaging para que ela possa enviar e receber mensagens?”

SOLUÇÃO “Use um Channel Adapter que pode acessar a API da aplicação ou seus dados, e publique mensagens em um canal baseado nesses dados, e que possa receber mensagens e chamar funcionalidades dentro da aplicação”



Imagens: eaipatterns.com

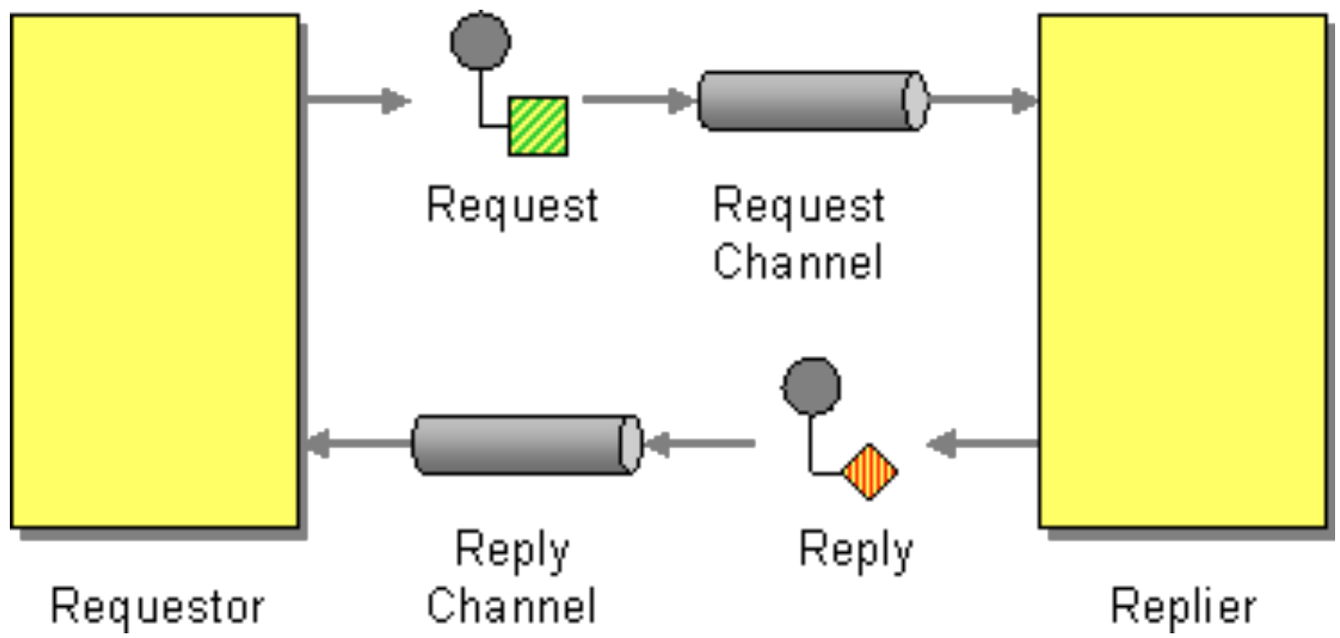
Padrões de Construção de Mensagens (6)



Request-Reply

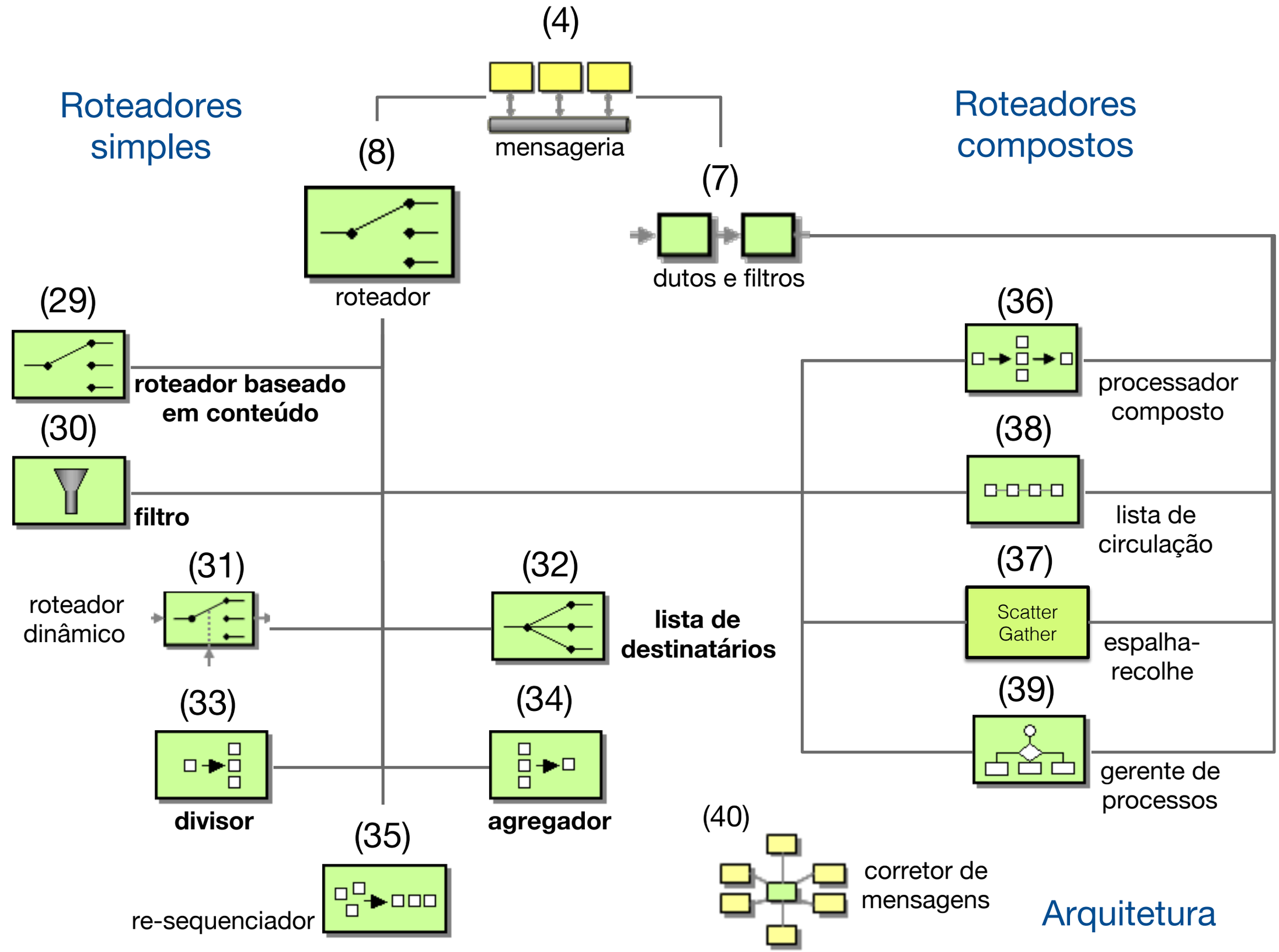
PROBLEMA “Quando uma aplicação envia uma mensagem, como ela pode obter uma resposta do servidor?”

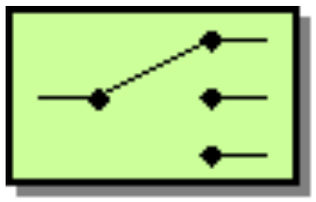
SOLUÇÃO “Envie um **par de mensagens** Requisição-Resposta, cada uma no seu próprio canal”



Imagens: eaipatterns.com

Padrões de Roteamento de Mensagens (8)

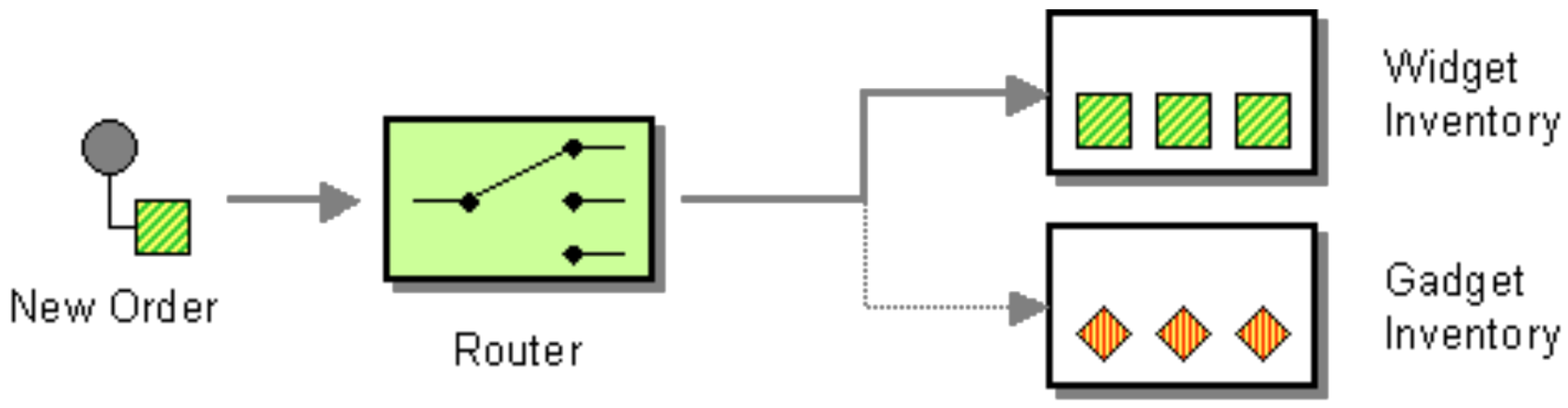




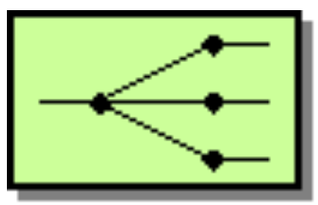
Content-Based Router

PROBLEMA “Como lidar com a situação onde a implementação de uma única função lógica está espalhada por múltiplos sistemas físicos?”

SOLUÇÃO “Use um **Roteador Baseado em Conteúdo** para rotear cada mensagem para o receptor correto baseado no conteúdo da mensagem”



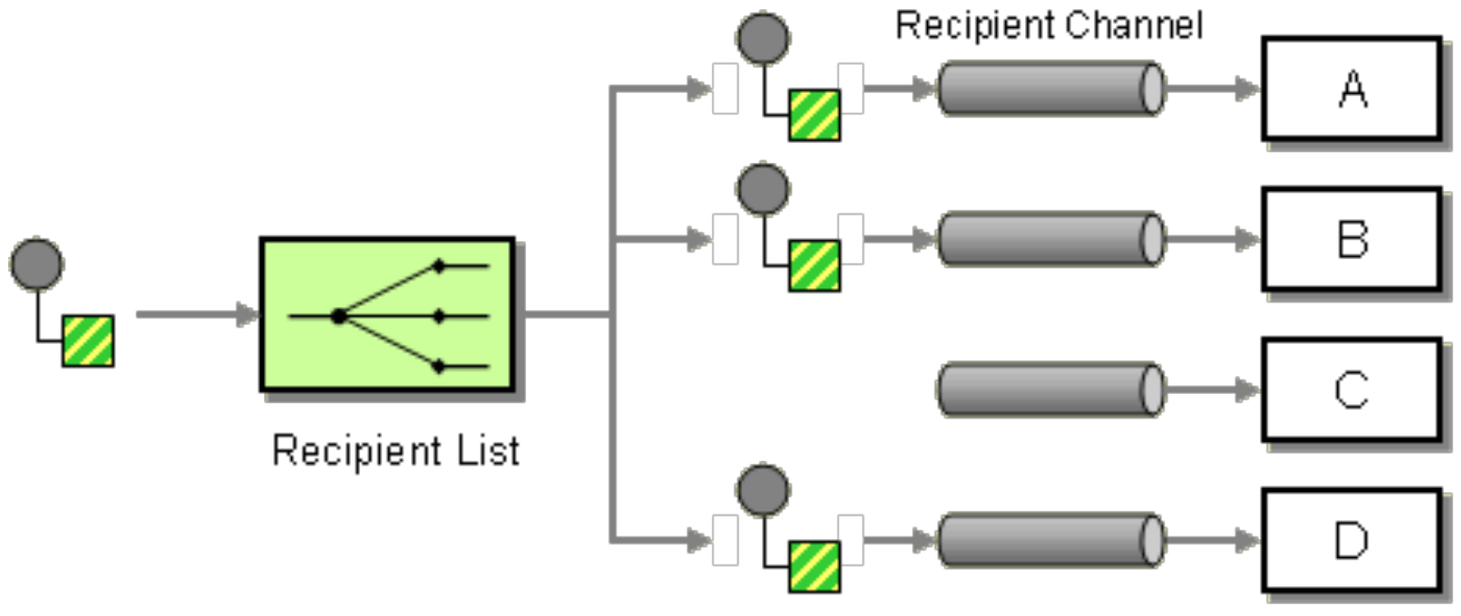
Imagens: eaipatterns.com



Recipient List

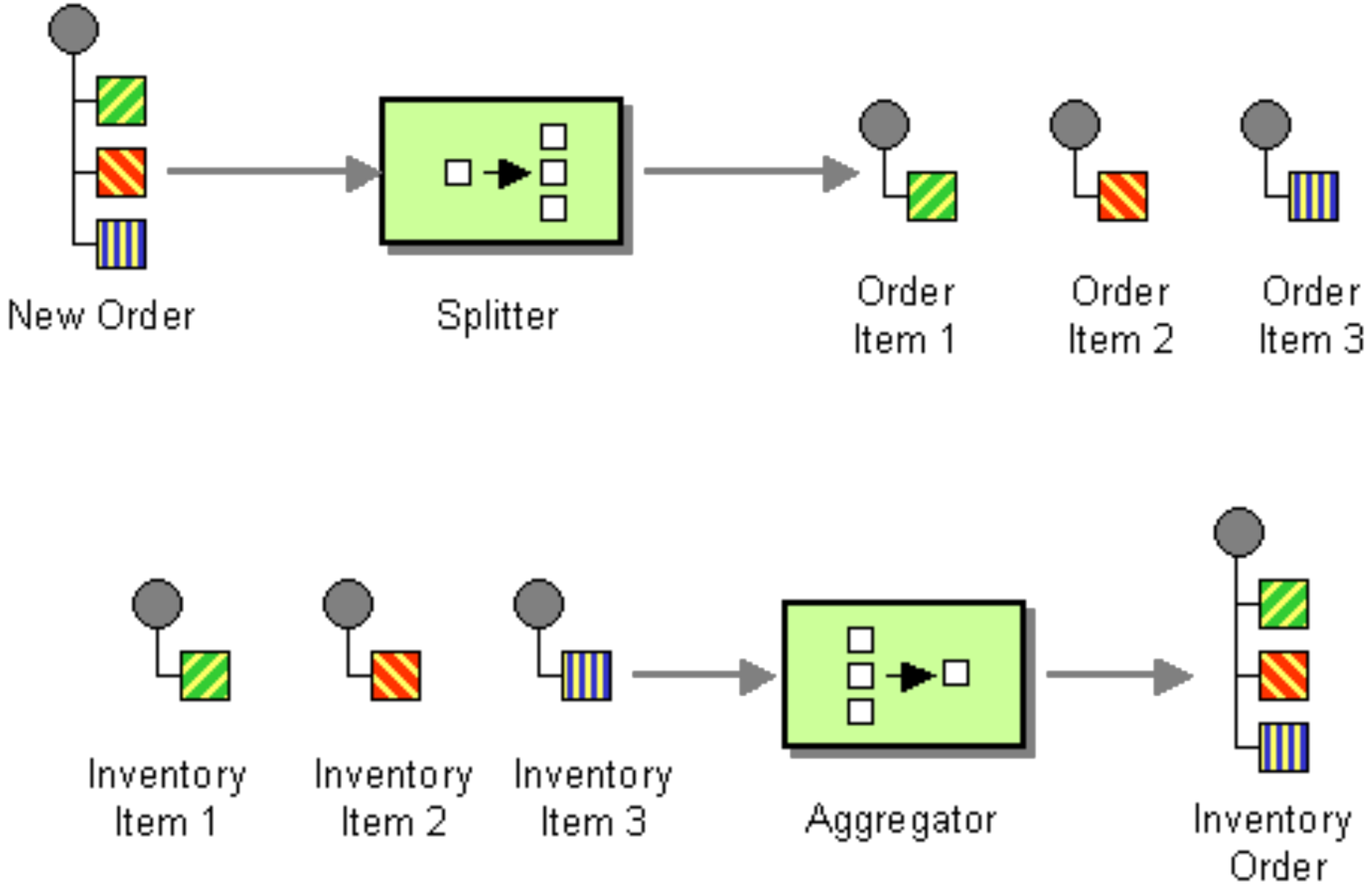
PROBLEMA “Como rotear uma mensagem para uma lista de receptores especificados dinamicamente?”

SOLUÇÃO “Defina um canal para cada receptor. Depois use uma **Lista de Destinatários** para inspecionar uma mensagem entrante, determine a lista de receptores desejados, e repasse a mensagem para todos os canais associados com os receptores da lista.”

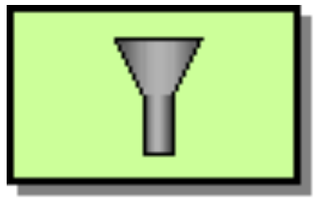


Imagens: eaipatterns.com

Splitter e Aggregator



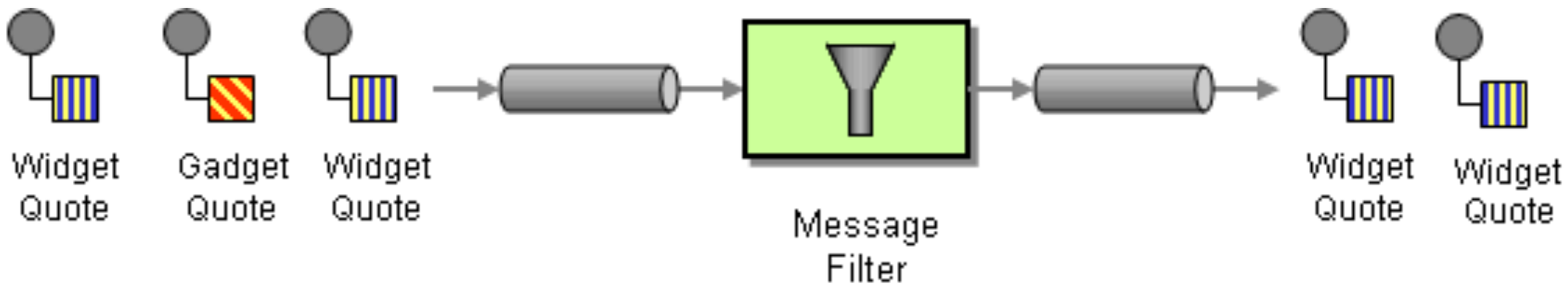
Imagens: eaipatterns.com



Message Filter

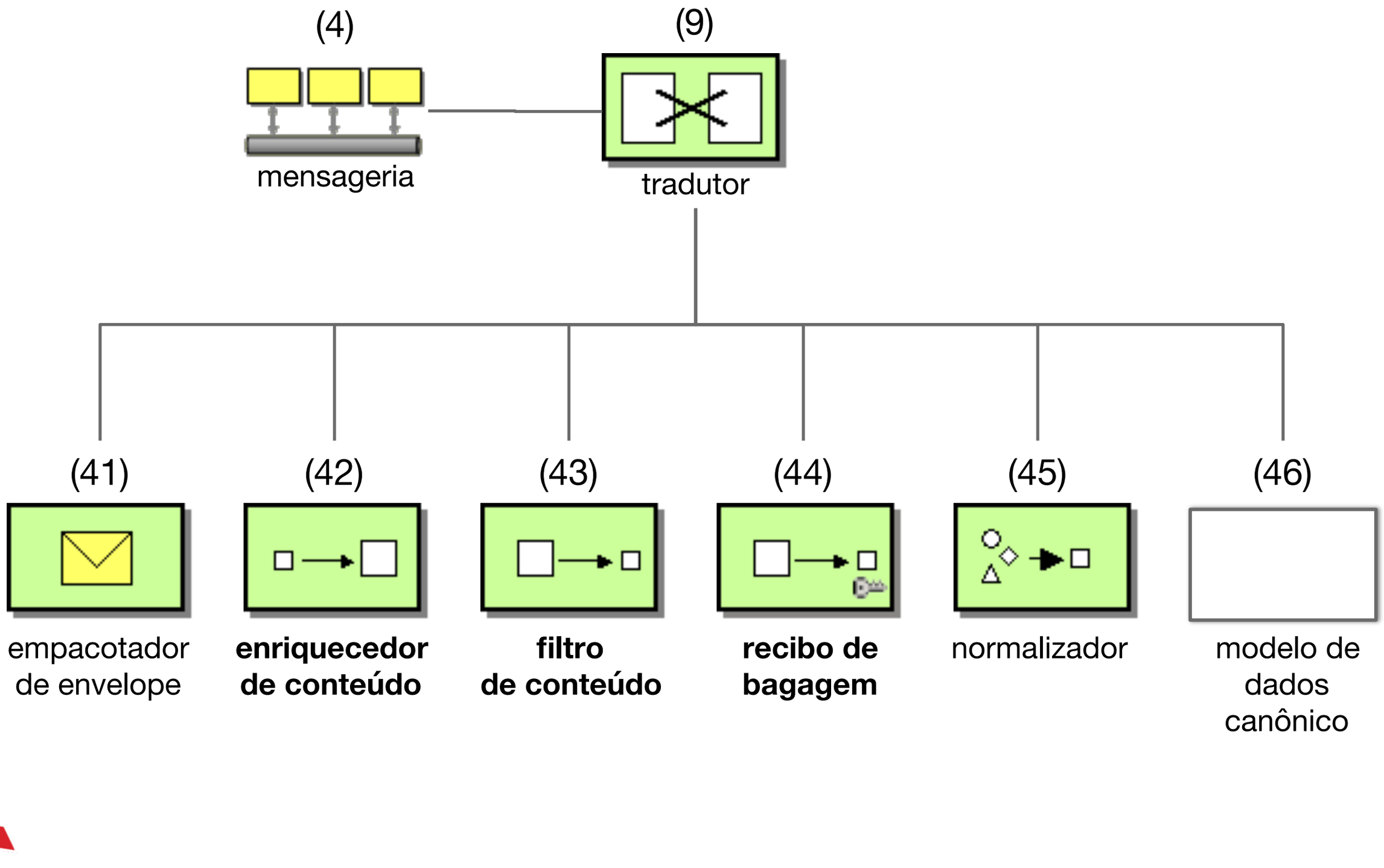
PROBLEMA “Como um componente pode evitar o recebimento de mensagens que não interessam?”

SOLUÇÃO “Use um tipo especial de roteador, um **Filtro de Mensagens**, para eliminar mensagens indesejadas de um canal com base em um conjunto de critérios.”

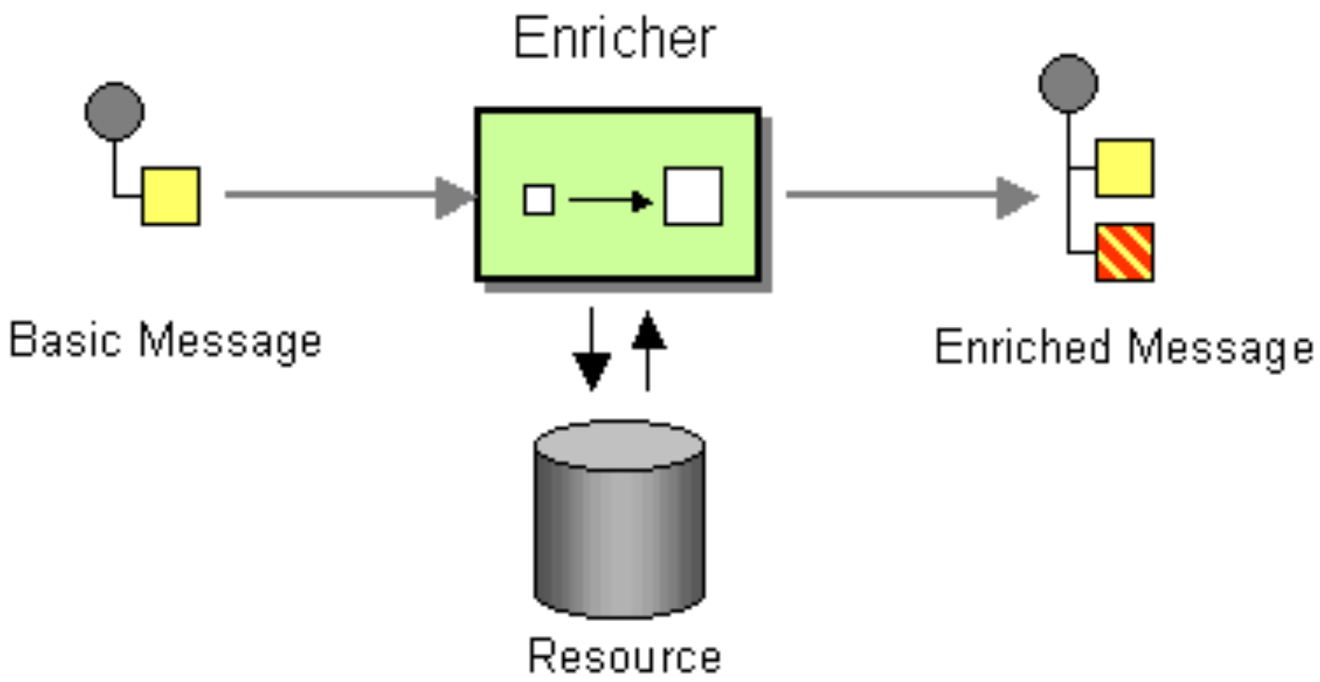


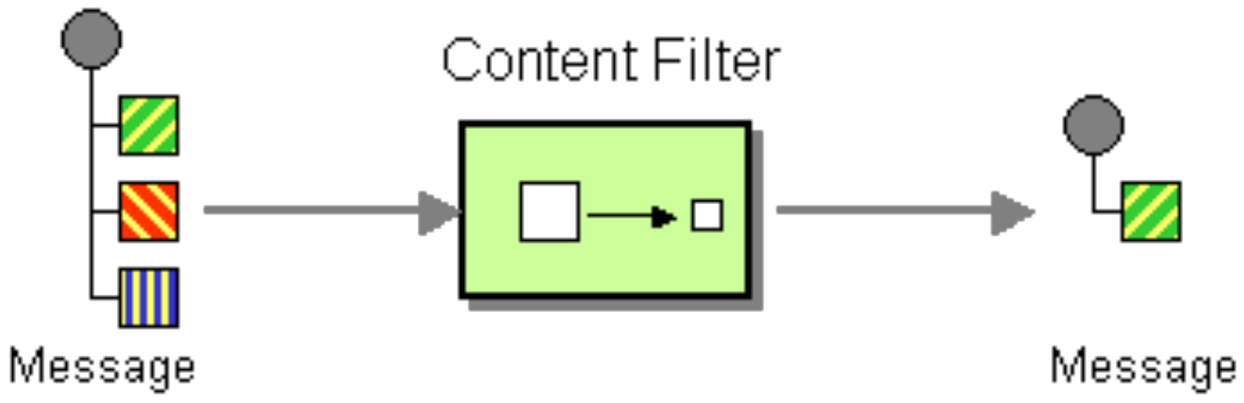
Imagens: eaipatterns.com

Padrões relacionados a Message Translator (9)

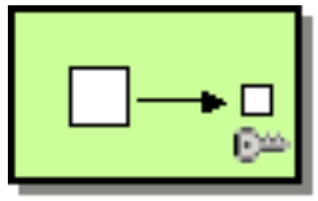


Content Enricher / Filter





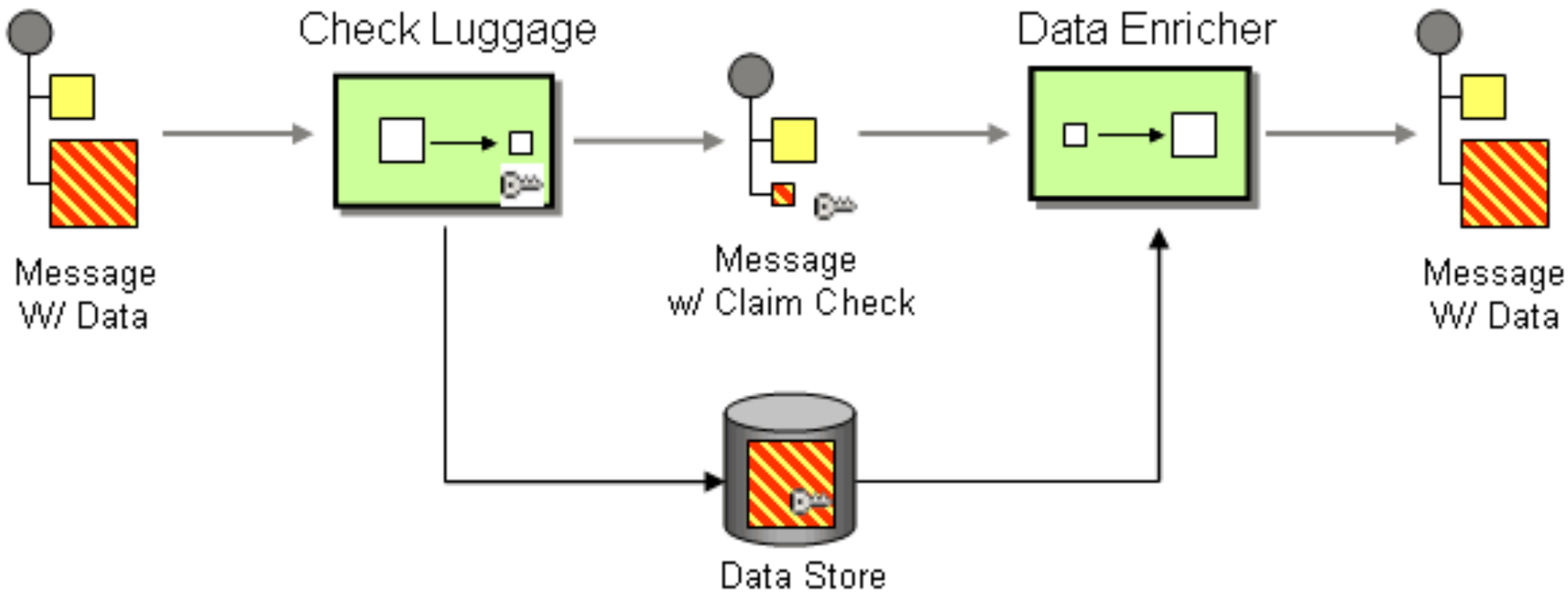
Imagens: eaipatterns.com



Claim Check

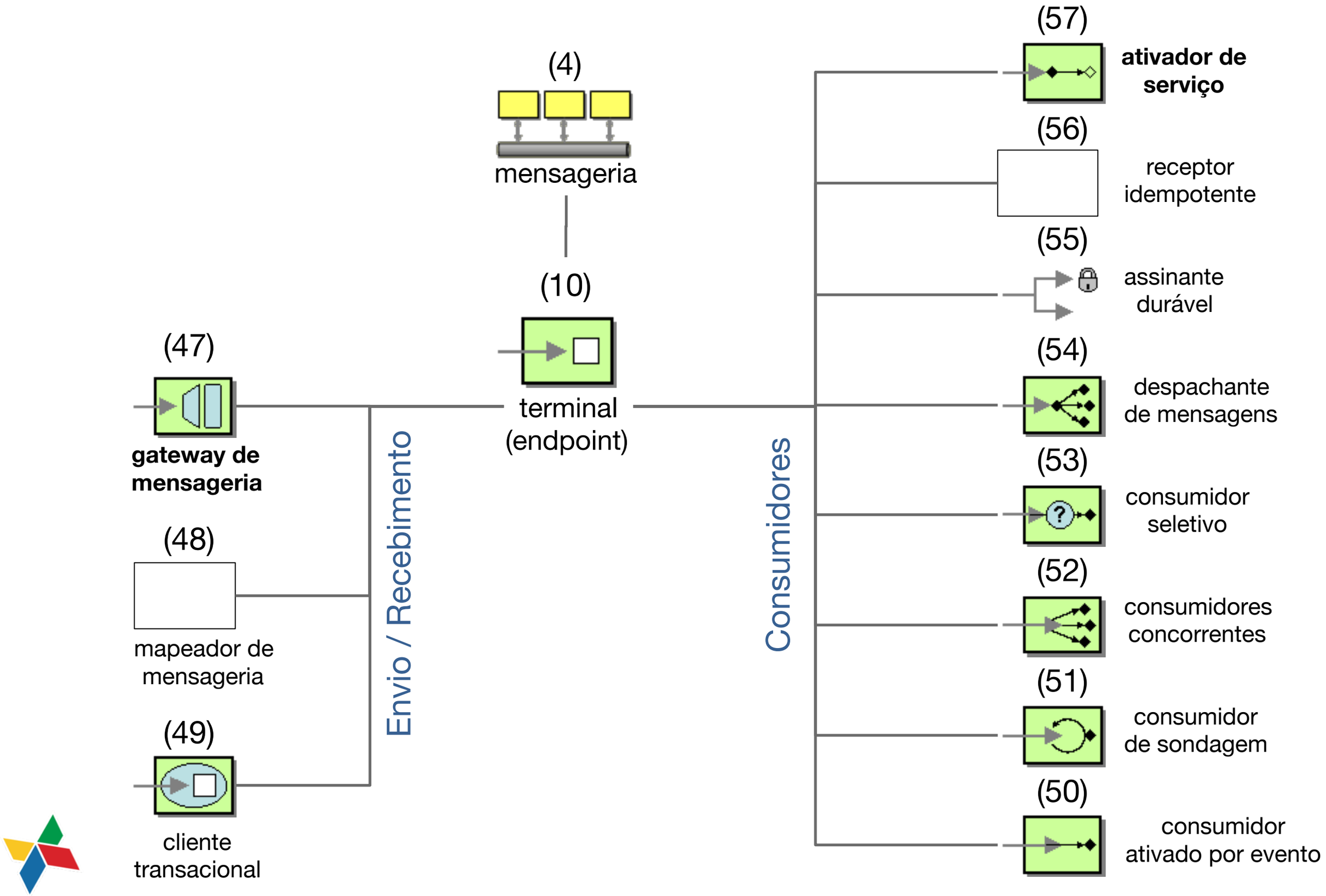
PROBLEMA “Como podemos reduzir o volume de dados de uma mensagem enviada através do sistema sem sacrificar o conteúdo da informação?”

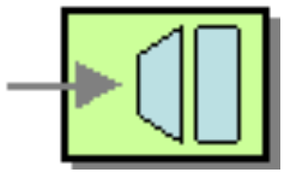
SOLUÇÃO “Guarde os dados da mensagem em um repositório persistente e passe um **Recibo de Bagagem** para os componentes seguintes**.** Esses componentes poderão usar o Recibo para recuperar a informação armazenada”



Imagens: eaipatterns.com

Padrões para **Terminais** de Mensageria (10)

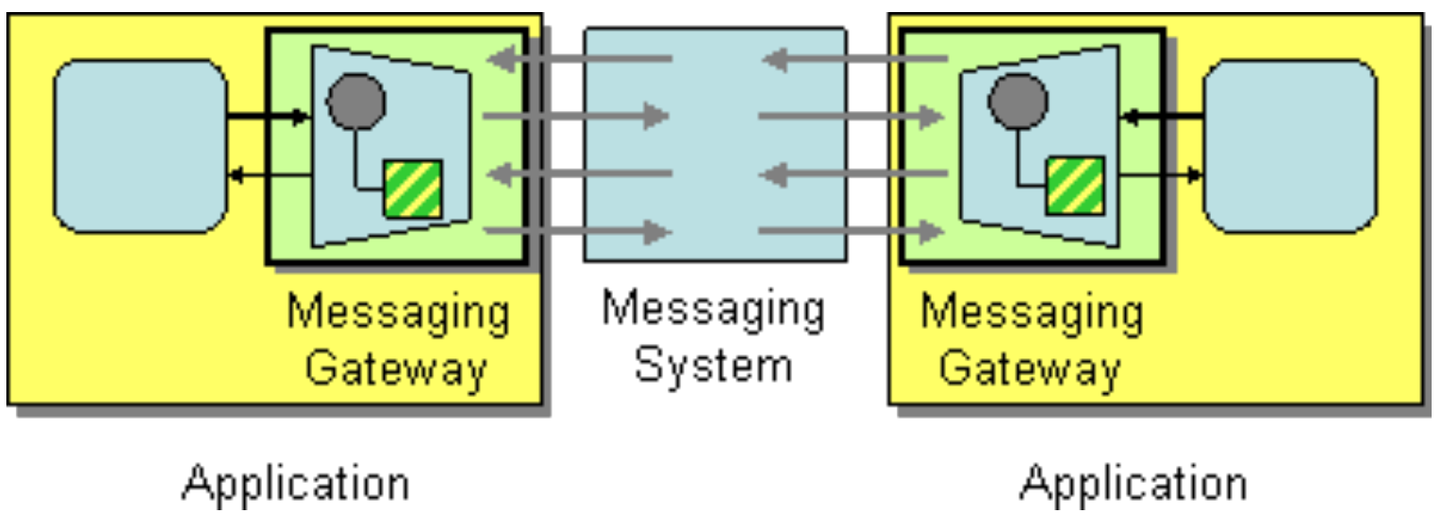




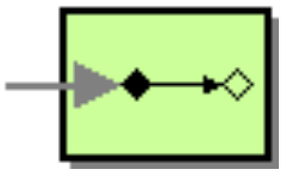
Messaging Gateway

PROBLEMA “Como isolar o acesso ao sistema de mensageria do restante da aplicação?”

SOLUÇÃO “Use um **Gateway de Mensageria**, uma classe que encapsula chamadas especificas ao sistema de mensageria e expõe uma interface com métodos específicos ao domínio da aplicação”



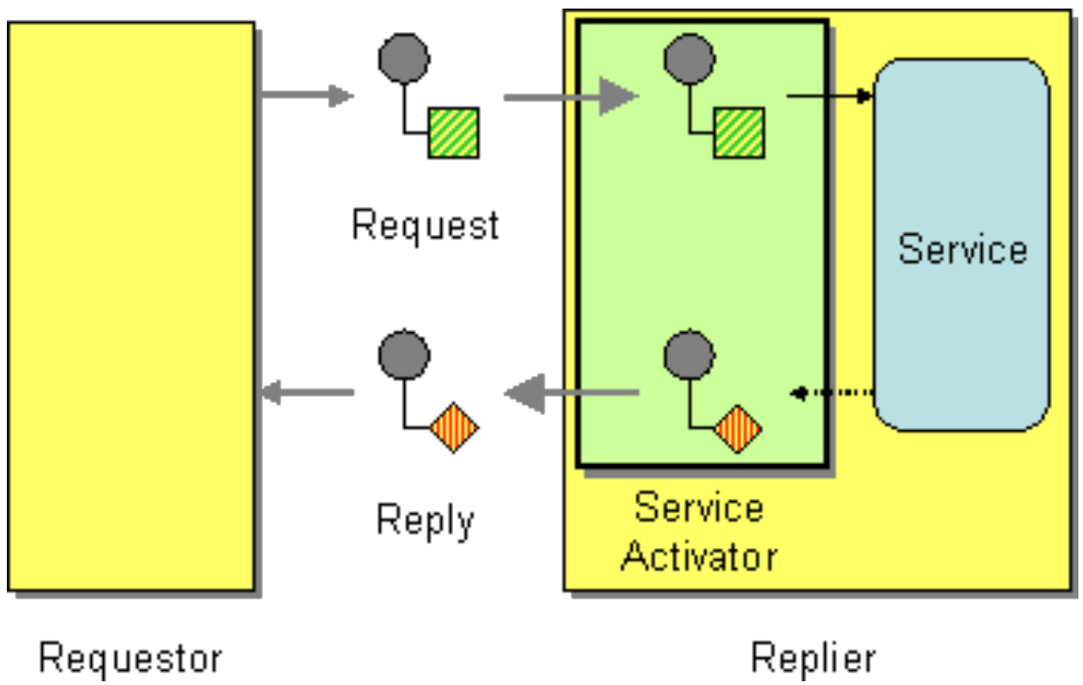
Imagens: eaipatterns.com



Service Activator

PROBLEMA “Como projetar um serviço que possa ser chamado de forma síncrona (sem usar mensageria) ou de forma assíncrona (usando tecnologias de mensageria)?”

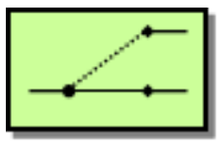
SOLUÇÃO “Projete um **Ativador de Serviços** que conecte as mensagens do canal ao serviço”



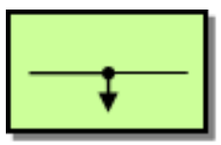
Imagens: eaipatterns.com

Padrões de Gerenciamento do Sistema

Control Bus (Barramento de Controle) (58)



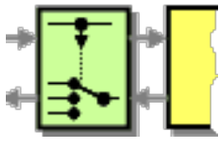
Detour (Desvio) (59)



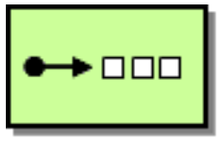
Wire Tap (Escuta) (60)

Message History (Histórico de Mensagens) (61)

Message Store (Repositório de Mensagens) (62)



Smart Proxy (Proxy Inteligente) (63)



Test Message (Mensagem de Teste) (64)

Channel Purger (Purificador de Canal) (65)

Como usar os padrões?

- Os padrões são **abstrações de alto nível** que permitem descrever a solução de um problema de integração
- Use os padrões para descrever a arquitetura de **rotas** de uma solução de integração
- Não há solução única para cada tipo de problema
  - Mesmos resultados podem ser alcançados com arquiteturas diferentes (+ conseqüências diferentes)
- Exemplo: duas formas de filtrar mensagens
  - Predictive routing: Message Router com rotas pré-definidas
  - Reactive filtering: Canal Pub-Sub com Message Filter

Um **problema** de integração

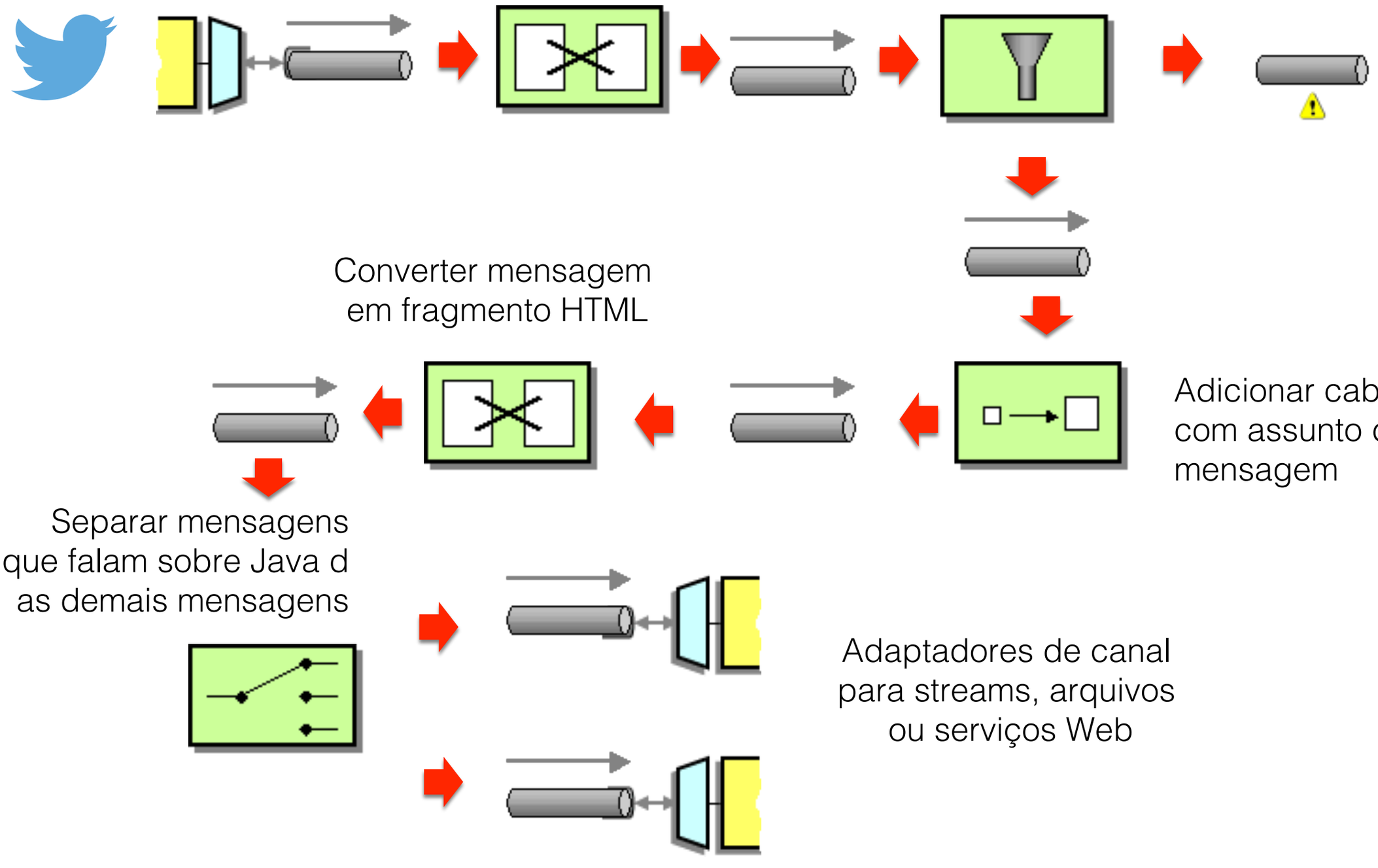
- Obter todas as mensagens com hashtag **#TheDevConf** ou **#TDC2015** no Twitter, periodicamente (polling)
- **Processar** apenas mensagens que tiverem **links**
- **Estruturar** o conteúdo das mensagens identificando **hashtags**, **usuários**, **links** e **remetentes**
- **Separar** mensagens que tratam de “**Java**” das demais mensagens
- **Disponibilizar** (arquivo, Web, etc.) **duas** listas de mensagens (Java e outras)

Uma **solução** usando padrões

Channel
Adapter de entrada
para Twitter

Extrair dados do Tweet
e guardar em uma
mensagem simples

Filtro para jogar fora
mensagens que não
têm links



Converter mensagem
em fragmento HTML

Adicionar cabeçalho
com assunto da
mensagem

Separar mensagens
que falam sobre Java d
as demais mensagens

Adaptadores de canal
para streams, arquivos
ou serviços Web

Como implementar?

- Uma **linguagem** de programação (ex: Java) + um **serviço de mensageria** (ex: JMS + ActiveMQ)
  - Desvantagem: é preciso ter cuidado para componentes não ficarem excessivamente acoplados
- Um **framework** que implemente padrões de integração de sistemas
  - Mule ESB
  - Apache Camel
  - Spring Integration

Spring Integration

- Solução do **Spring Framework** / Spring IO
  - Integrado ao ecossistema do Spring
- Configuração através de **XML** ou **anotações**
- Rotas em **XML ou Java DSL**

Spring Framework

- Spring é uma **plataforma Java** construída sobre conceitos **injeção de dependências** (DI) e aspectos
- Componentes Spring (beans) são **POJOs**, vivem em um **container** e são interligados pelo framework usando **DI**
  - O container inicializa um **contexto** que instancia e realiza as ligações (**wiring**) entre componentes
  - A configuração é feita declarativamente usando defaults (wiring automático), XML e anotações
- Exemplo de inicialização de contexto configurado em XML

```
ApplicationContext ctx = new
                ClassPathXmlApplicationContext("beans.xml");
```

Exemplo de wiring em XML

```
<bean id="produtosDao"
        class="loja.ProdutosDao">
    <property name="hibernateTemplate"
                ref="hibernateTemplate"/>
    </property> ...
</bean>
```

```
<bean id="hibernateTemplate"
        class="org.springframework.orm.
                hibernate.HibernateTemplate">
    <property name="sessionFactory”
                ref="sessionFactory"/>
    </property>
</bean>
```

```
<bean id="sessionFactory"
        class="org.springframework.orm.
                hibernate.LocalSessionFactoryBean">
                ...
</bean>
```

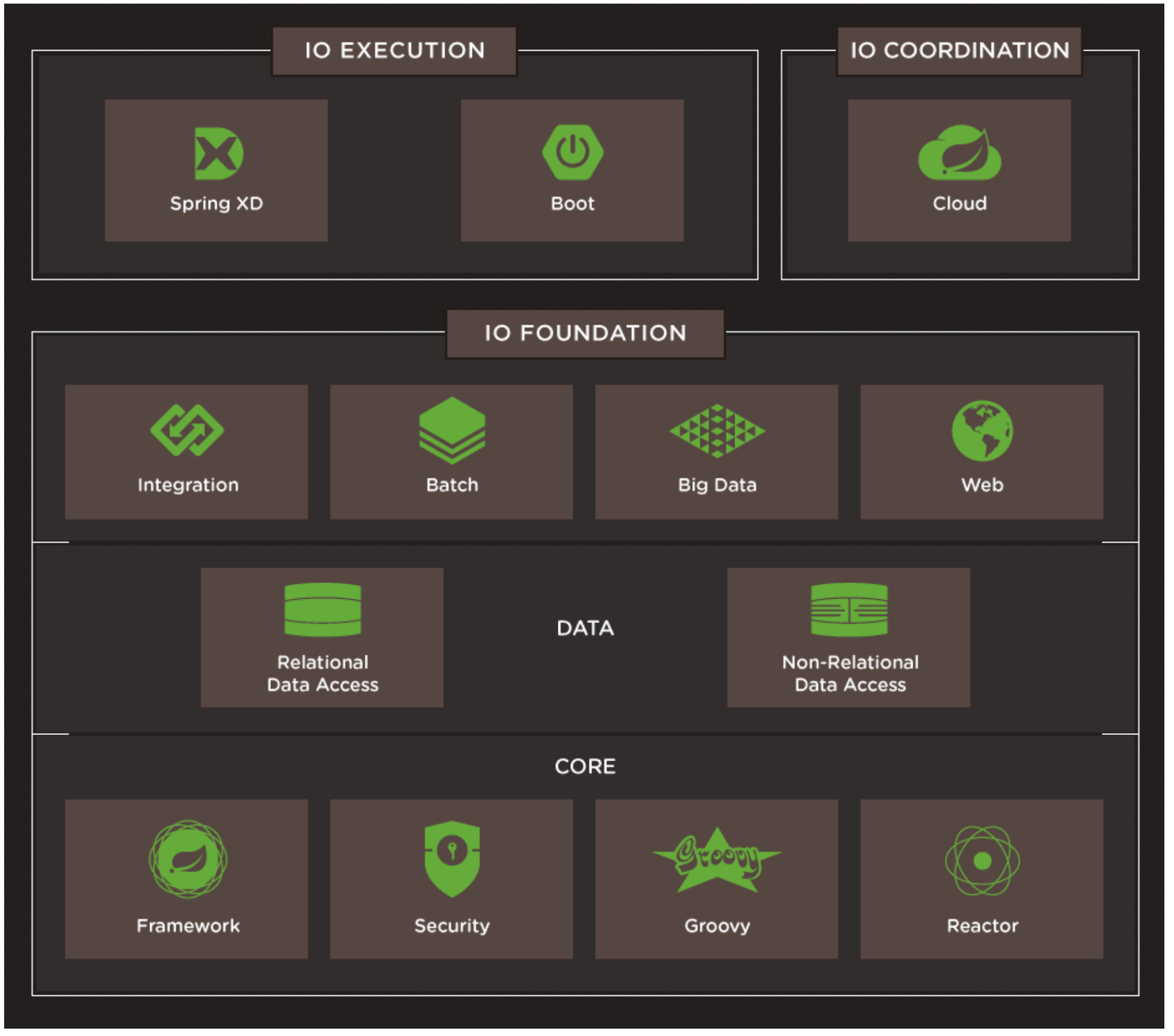
dataSource

ApplicationContext

- Através do **contexto** pode-se localizar e usar instâncias de beans (se for necessário)

```
ApplicationContext ctx =
    new ClassPathXmlApplicationContext("beans.xml");

ProdutosDao bean = (ProdutosDao)
                        ctx.getBean("produtosDao");

Produto p = bean.findProduto(123);
BidDecimal preco = p.getPreco();
```

Ecossistema Spring



Maven

- Os módulos Spring geralmente são incluídos em um projeto através do **Maven** ou Gradle

```
<dependencies>
    <dependency>
        <groupId>org.springframework.integration</groupId>
        <artifactId>spring-integration</artifactId>
        <version>4.0.4.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.integration</groupId>
        <artifactId>spring-integration-twitter</artifactId>
        <version>4.0.4.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.social</groupId>
        <artifactId>spring-social-twitter</artifactId>
        <version>1.1.0.RELEASE</version>
    </dependency>
</dependencies>
```

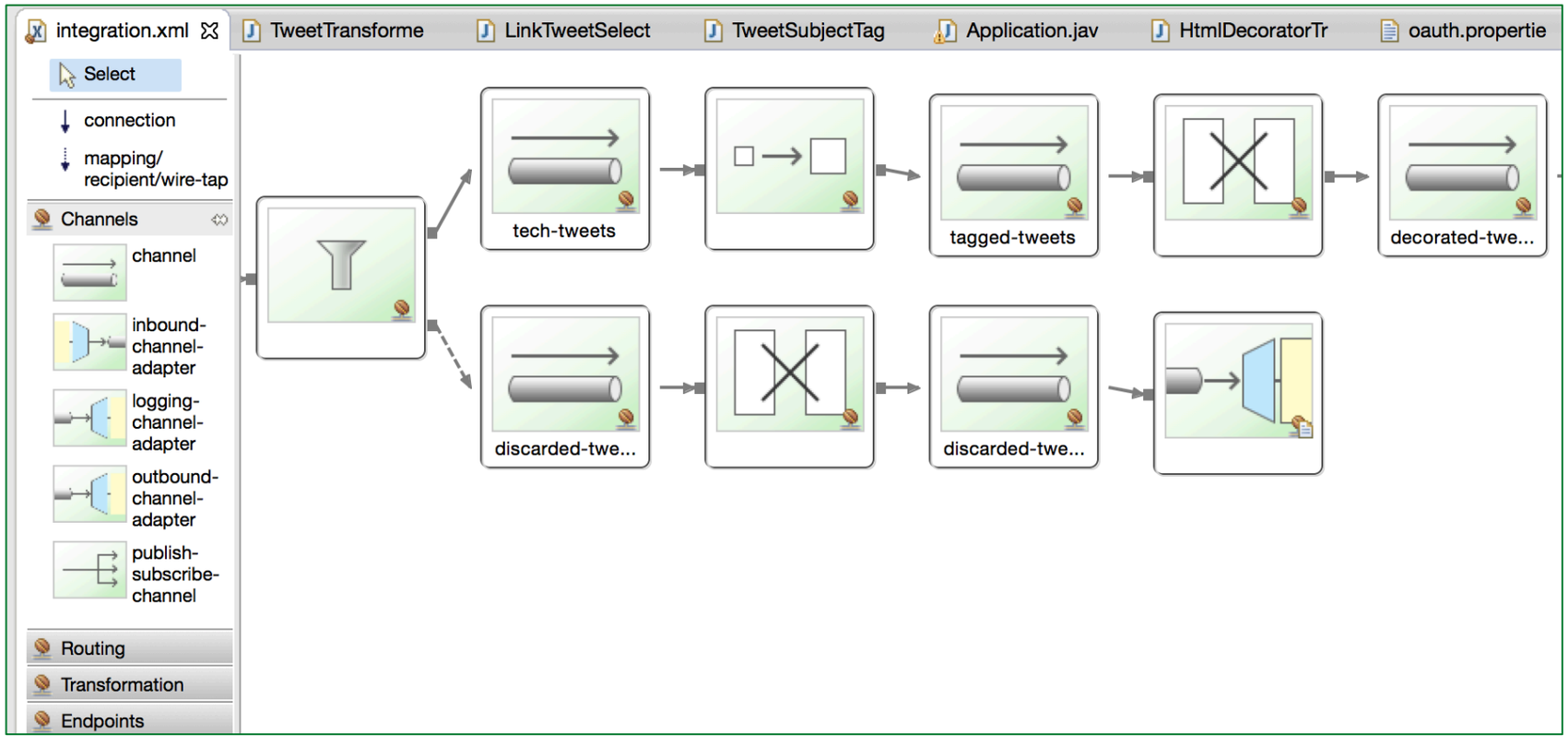
Configuração do contexto

- Para usar configuração declarativa XML também é necessário **declarar XSD** de componentes usados

```
<beans xmlns="http://www.springframework.org/schema/beans”
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:int="http://www.springframework.org/schema/integration"
       xsi:schemaLocation="
            http://www.springframework.org/schema/integration
            http://www.springframework.org/schema/integration/
                                              spring-integration-4.0.xsd">

    <bean id="teste" class="teste.Hello" />
    <int:channel id="tweets" />
</beans>
```

Ferramenta **STS**

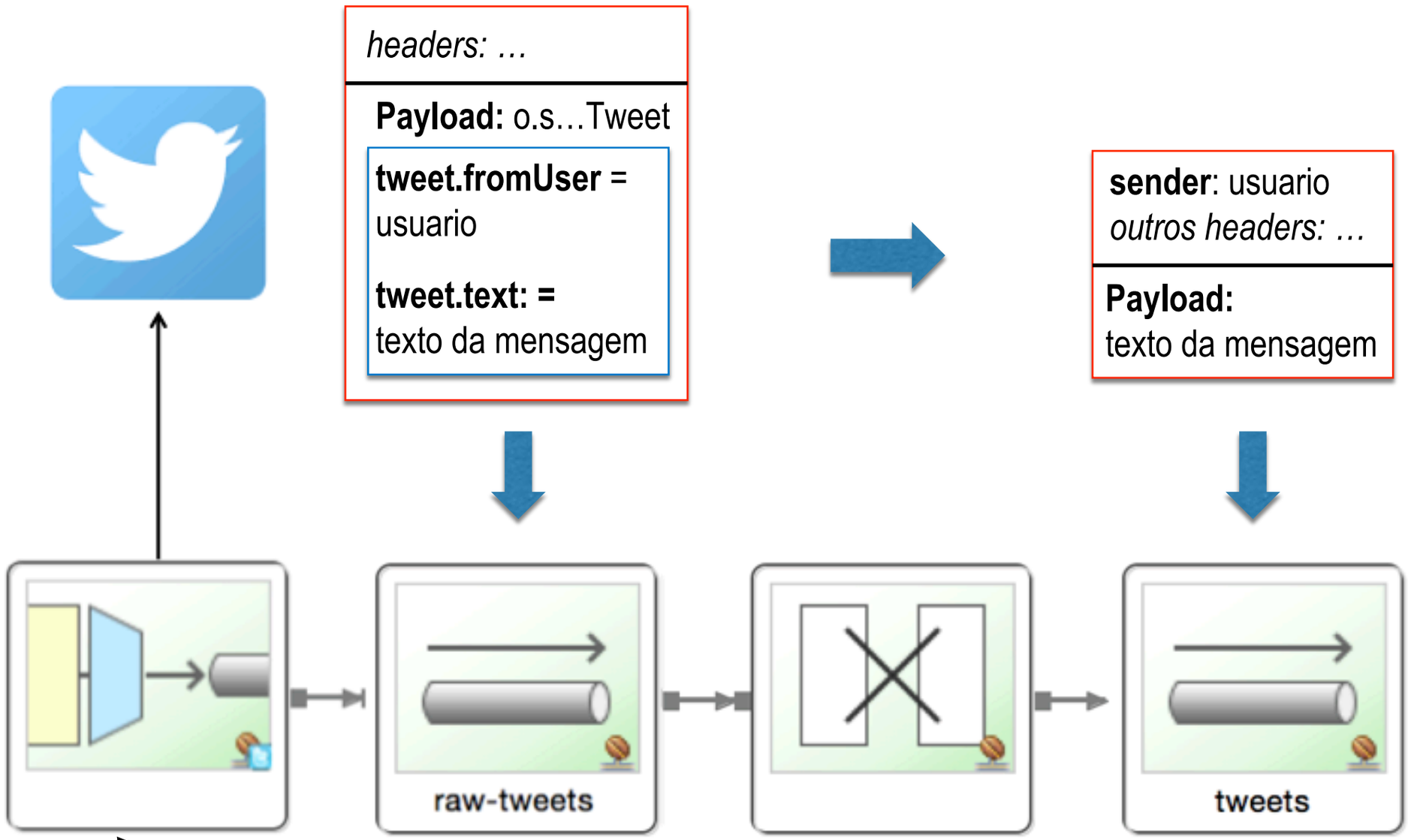
- Spring Tool Suite (Eclipse)
  - Facilita desenvolvimento (ex: automaticamente cria arquivos de configuração e adiciona namespaces)



Solução usando SI

- Para **implementar** a solução apresentada usando padrões EIP em Spring Integration usaremos
  - Um **Inbound** Channel Adapter para Twitter (spring-integration-twitter + spring-social-twitter)
  - Dois **Outbound** Channel Adapters para Arquivos (spring-integration-file)
  - Vários Message Channel **<channel>**
  - Dois Message Translators **<transformer>**
  - Um Content-Based Router **<header-value-router>**
  - Um Content Enricher **<header-enricher>**
  - Um Message Filter **<filter>**

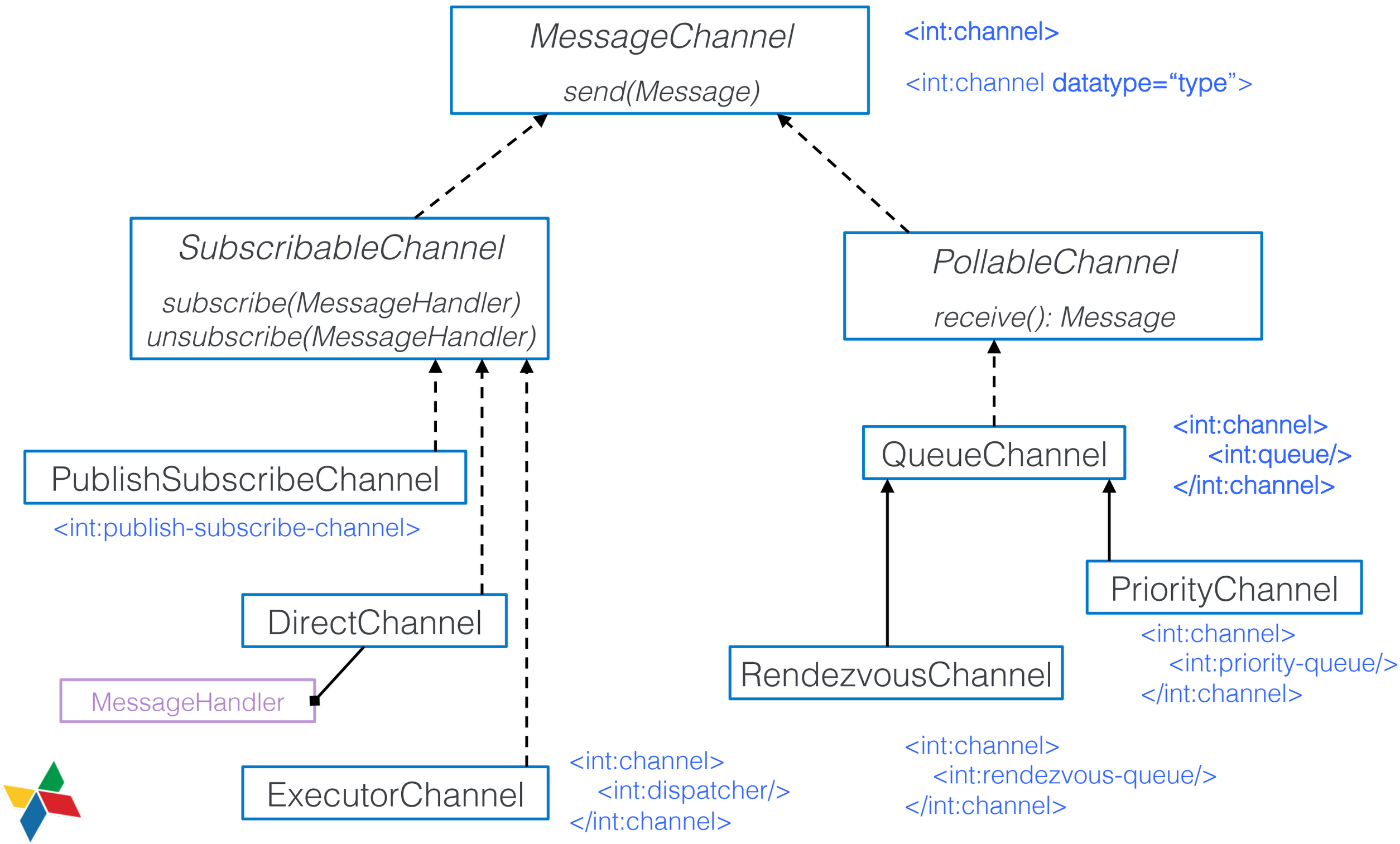
Inbound **Channel Adapter**



Faz query no **Twitter**:
“`#TheDevConf OR #tdc2015`”
Põe mensagens no canal **raw-tweets**

Translator: Extrai dados de objeto **Tweet** e cria novo objeto **Message**

Canais do Spring Integration



Direct channels

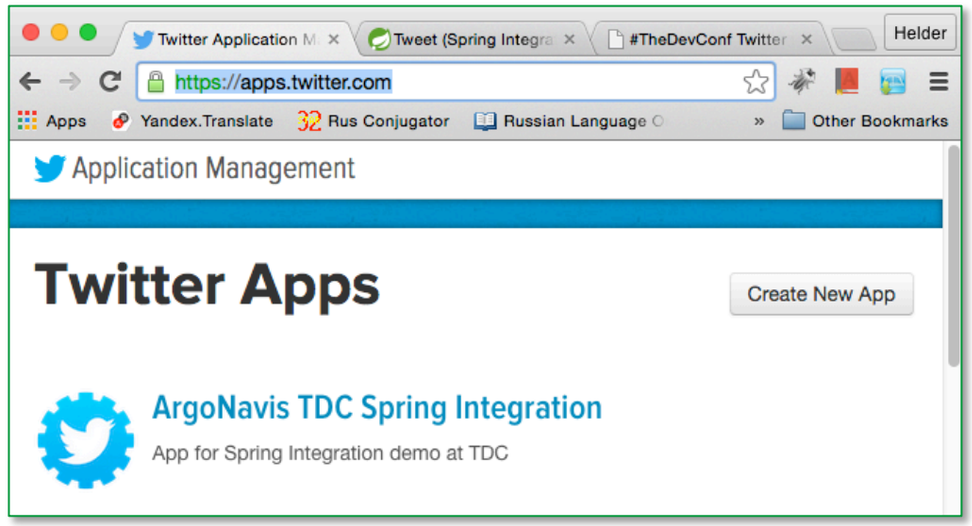
- Canal **default** do Spring Integration
  - Podem ser configurados em XML ou via anotações

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:int="…"
       …
       outros xmlns …>

    <bean id="componente" class="com.pacote.Classe" />
     …
    <int:channel id="tweets" />
    <int:channel id="tech-tweets" />
    <int:channel id="tagged-tweets" />
    <int:channel id="decorated-tweets" />
    <int:channel id="java-tweets" />
    <int:channel id="other-tweets" />
…
</beans>
```

Arquivo de configuração de contexto (beans)

Twitter template

- Da API **Spring Social**
  - Necessário para poder fazer queries na API do Twitter



Crie app e obtenha as chaves

```
twitter.oauth.consumerKey=kSJz4Iy6kcPcLw9wdUaP
twitter.oauth.consumerSecret=kSJz4Iy6kcPcLw9wdUa
twitter.oauth.accessToken=kSJz4Iy6kcPcLw9wdUaPcc
twitter.oauth.accessTokenSecret=dSJz4Iy6kc9wdUaP
```

oauth.properties

```
<context:property-placeholder location="classpath:oauth.properties" />

<bean id="twitterTemplate"
        class="org.springframework.social.twitter.api.impl.TwitterTemplate">
    <constructor-arg value="${twitter.oauth.consumerKey}" />
    <constructor-arg value="${twitter.oauth.consumerSecret}" />
    <constructor-arg value="${twitter.oauth.accessToken}" />
    <constructor-arg value="${twitter.oauth.accessTokenSecret}" />
</bean>
```

Spring application context (beans)

Twitter channel adapters

- Spring Integration possui vários adaptadores de entrada e saída (direct message, search, etc.)
  - Usamos o *search-inbound-* para fazer queries

```
<int-twitter:search-inbound-channel-adapter
        twitter-template="twitterTemplate"
        query="#thedevconf OR #tdc2015"
        id="raw-tweets">
            <int:poller fixed-rate="60000"
                        max-messages-per-poll="10" />
</int-twitter:search-inbound-channel-adapter>
```

Automaticamente cria um **direct channel**

Message Translator 1

- Para traduzir o payload (objeto **Tweet** do Spring Social) para uma nova **Message** (do Spring Integration)

```
import org.springframework.social.twitter.api.Tweet;

public class TweetTransformer implements Transformer {

    @Override
    public Message<?> transform(Message<?> msg) {
        Tweet tweet   = (Tweet) msg.getPayload();
        String sender = tweet.getFromUser();
        String text   = tweet.getText();

        Message<String> message = MessageBuilder.withPayload(text)
                .setHeader("sender", sender)
                .build();
        return message;
    }
}
```

```
<int:transformer input-channel="raw-tweets"
                 output-channel="tweets">
   <bean class="br...tdc.TweetTransformer" />
</int:transformer>
```

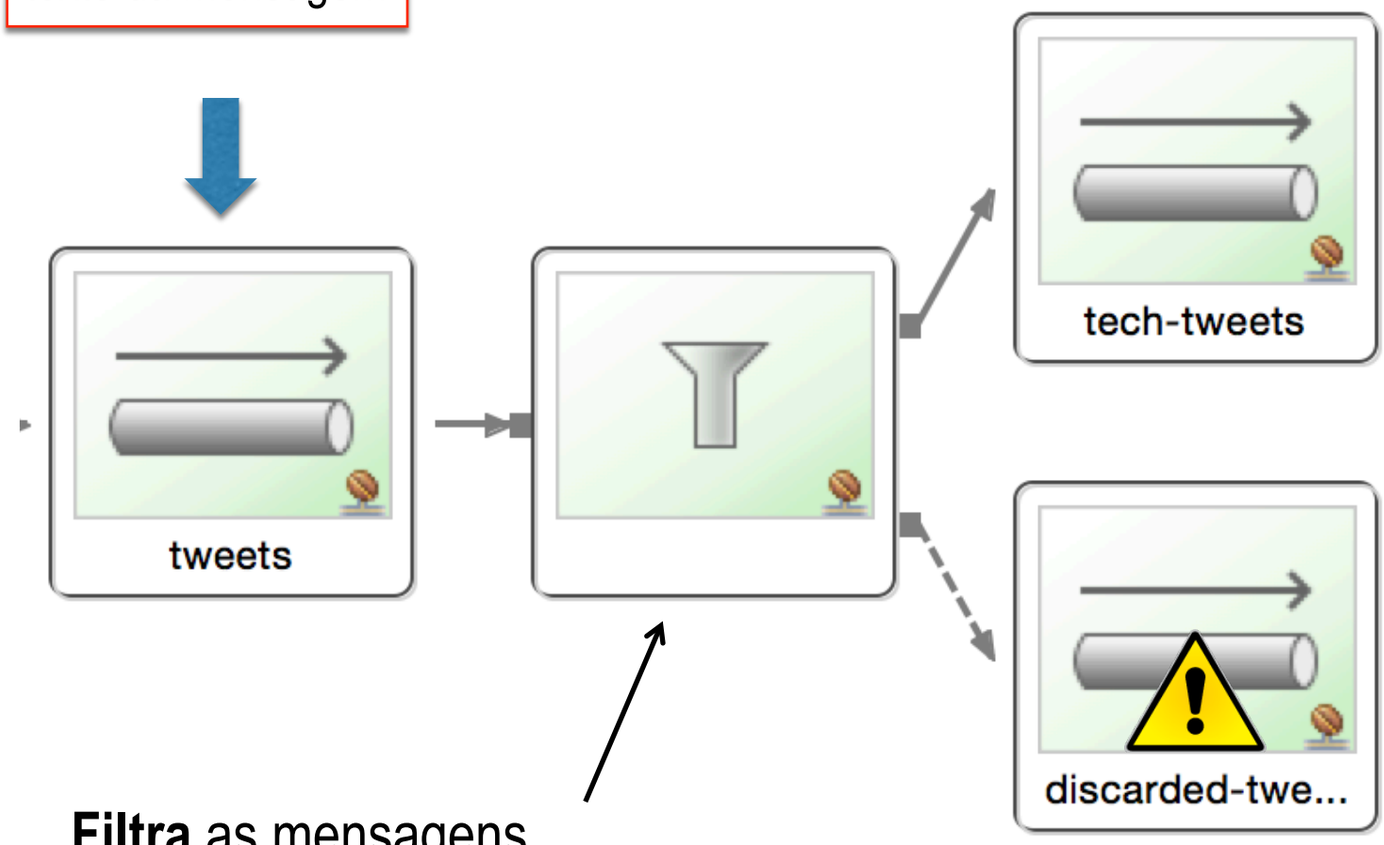
Spring application context (beans)

Message **Filter**

sender: usuario
outros headers: …

Payload:
texto da mensagem

Apenas mensagens com links estão neste canal



Filtra as mensagens selecionando apenas as que têm **links** (http / https)

Invalid-Message Channel (mensagens que nâo tem links)

Message Filters

- Utiliza um **seletor** (bean ou expressão SpEL) para escolher mensagens a serem filtradas
  - Opcionalmente pode enviar as mensagens descartadas para um **Invalid-Message Channel**

```
public class LinkTweetSelector implements MessageSelector {
    public boolean accept(Message<?> message) {
        for (String word : message.getPayload().split(" ")) {
            if (word.startsWith("http://")) { return true; }
        }
        return false;
}}
```

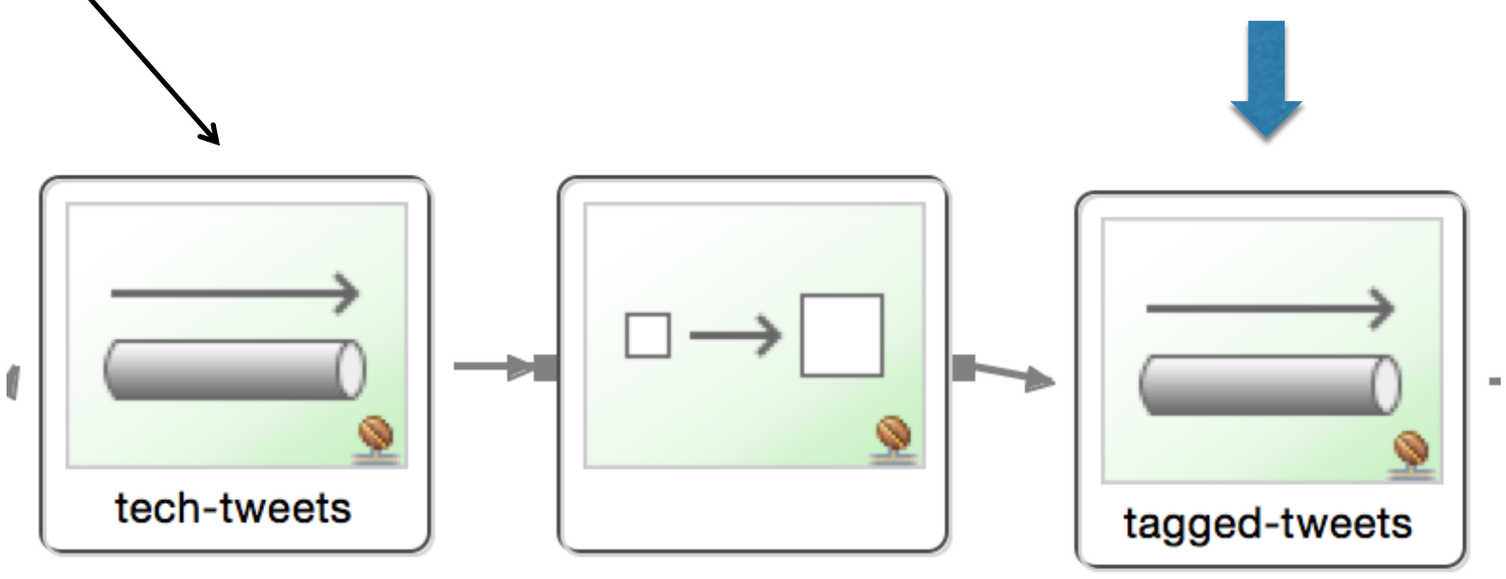
```
<bean id="techTweetSelector"
      class="br.com.argonavis.si.examples.tdc.LinkTweetSelector" />

<int:filter input-channel="tweets"
            output-channel="tech-tweets"
            discard-channel="discarded-tweets"
            ref="techTweetSelector" />
```

Content **Enricher**

Apenas mensagens com links estão neste canal

sender: usuario
subject: assunto
outros headers: …

Payload:
texto da mensagem



Adiciona um cabeçalho para **rotular** as mensagens por assunto (“java” ou “other”)

Content Enrichers

- Header ou Payload

```
public class TweetSubjectTagger {
    public String setSubjectHeader(String payload) {
        if(payload.toLowerCase().indexOf("java") >= 0) {
            return "java";
        } else {
            return "other"; // web
        }
    }
}
```

Método do bean fornece o valor do header

```
<bean id="tweetTagger"
      class="br.com.argonavis.si.examples.tdc.TweetSubjectTagger" />

<int:header-enricher input-channel="tech-tweets"
                     output-channel="tagged-tweets">
    <int:header name="subject" method="setSubjectHeader"
                ref="tweetTagger" />
</int:header-enricher>
```

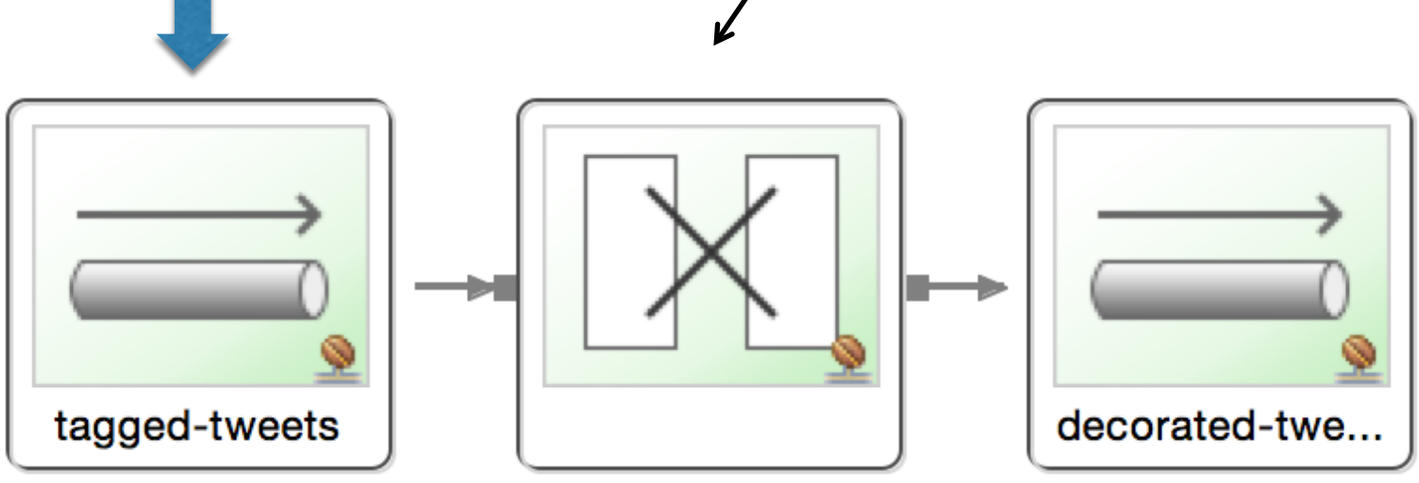
Elementos **<header>** também podem passar valores fixos

Message **Translator**

sender: usuario
subject: assunto
outros headers: …

Payload:
texto da mensagem

Transforma o texto em fragmento HTML
(links em <a href>, hashtags em , etc.



sender: usuario
subject: assunto
outros headers: …

Payload:
** **html da mensagem** **

Palestra de @helderdarocha sobre EIP patterns e SpringIntegration #TheDevConf https://pic.twitter.com/eIcFIYAo8T

** **usuarioPalestra de @helderdarocha sobre EIP patterns e SpringIntegration <span class="hashtag”>#TheDevConf https://pic.twitter.com/eIcFIYAo8T** **

Message Translator 2

```
<bean id="htmlDecoratorTransformer"
      class="br.com.argonavis.si.examples.tdc.HtmlDecoratorTransformer" />

<int:transformer input-channel="tagged-tweets"
                 output-channel="decorated-tweets"
                 ref="htmlDecoratorTransformer" />
```

```
public class HtmlDecoratorTransformer implements Transformer {

   public Message<?> transform(Message<?> msg) {
      String newPayload = /* several transformations */
      String html = "<div class='tweet'>" + newPayload + "</div>";
      return MessageBuilder.withPayload(html)
                           .copyHeaders(msg.getHeaders())
                           .build();
   }
}
```

Message **Router**

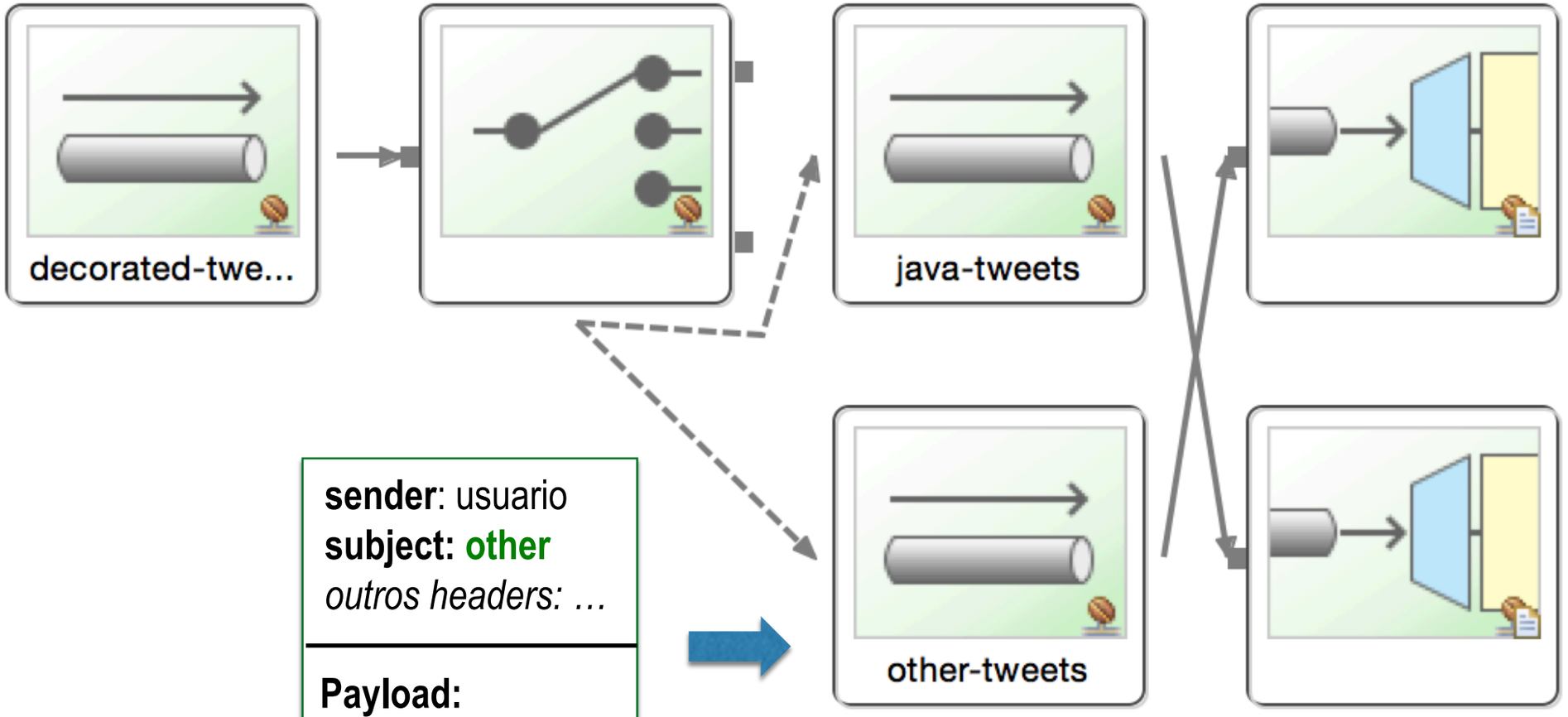
+ Outbound **Channel Adapter**

sender: usuario
subject: assunto
outros headers: …

Payload:
** **html ** **

sender: usuario
subject: java
outros headers: …

Payload:
** **html ** **



Arquivo web/tweets/ **JavaTweets.html**

Arquivo web/tweets/ **OtherTweets.html**

sender: usuario
subject: other
outros headers: …

Payload:
** **html ** **

Message Router

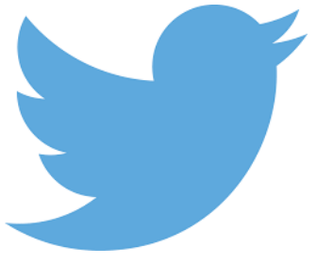
- **Header Value Router** mapeia valor de um cabeçalho selecionado a diferentes canais

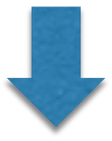
```
<int:header-value-router input-channel="decorated-tweets"
                         header-name="subject">
    <int:mapping value="java"  channel="java-tweets" />
    <int:mapping value="other" channel="other-tweets" />
</int:header-value-router>
```

- **Outbound Channel Adapters** gravam no arquivo

```
<int-file:outbound-channel-adapter
       channel="other-tweets" charset="UTF-8" mode="APPEND"
       directory="web-document-root/tweets"
       filename-generator-expression="'OtherTweets.html'" />
<int-file:outbound-channel-adapter
       channel="java-tweets" charset="UTF-8" mode="APPEND"
       directory="web-document-root/tweets"
       filename-generator-expression="'JavaTweets.html'" />
```

Resultado





JavaTweets.html

OtherTweets.html

Ajax

Outras formas de configurar

- Spring Integration também pode ser parcialmente ou totalmente configurado via **anotações** em classes e métodos
  - @Aggregator, @Filter, @Router, @ServiceActivator, @Transformer, @InboundChannelAdapter, etc.

```
public class Servico {
   @ServiceActivator(inputChannel="entrada", outputChannel="saida”)
   public void metodo(String payload, @Headers Map<String, Object> head) {
          …
   }
}
```

- As rotas, desde Spring Integration 4.0, podem ser expressas em **Java DSL**, que as torna mais legíveis

```
@Bean public IntegrationFlow splitAggregateFlow() {
    return IntegrationFlows.from("tweets”)
            .split(null).channel(MessageChannels.executor(this.taskExecutor()))
            .resequence().aggregate().get();
}
```

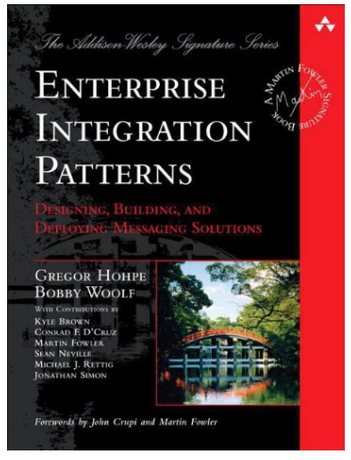
  - Camel normalmente declara rotas usando DSLs (não apenas Java)

Camel ou Spring Integration?

- Resposta baseada em **opinião**
  - Ambos implementam bem vários padrões EIP e possuem gateways e adapters para muitos serviços
  - Se você já usa a plataforma Spring, pode usar **Camel** (com Spring) ou **Spring Integration** (se preferir)
  - Se usa Spring XD, **SI** é melhor pois XD é construído sobre SI
  - Se não usa Spring, Camel é uma alternativa melhor (mais fácil de aprender, flexível e completa – não deve nada ao SI)
  - Se tem um projeto SI e prefere usar Camel, existe uma **Ponte Camel – Spring-Integration** (https://camel.apache.org/springintegration.html)

Conclusões

- Esta palestra apresentou uma introdução a padrões de integração de sistemas e Spring Integration
  - Uma **visão geral dos padrões** de destacando os principais
  - Uma possível **solução teórica** usando padrões para um problema de integração
  - Uma **implementação da solução** em Java usando Spring Integration
  - Breve discussão sobre outros recursos e **alternativas** a Spring Integration



Referências

- Gregor Hohpe, Bobby Woolf, et al **Enterprise Integration Patterns** http://www.eaipatterns.com/
- **Spring Integration** http://projects.spring.io/spring-integration/
- Kai Waehner. **Integration Framework Comparison – Spring Integration, Mule ESB or Apache Camel** http://www.javacodegeeks.com/2012/03/integration-framework-comparison-spring.html
- Jim White. **Spring Integration Tutorial**. Intertech, 2014. http://www.intertech.com/Blog/spring-integration-part-1-understanding-channels/ Ótimo tutorial passo-a-passo sobre Spring Integration

obrigado!

helder@argonavis.com.br

Palestra

http://www.argonavis.com.br/download/tdc_2015_eip-si.html

Código-fonte

https://github.com/argonavisbr/SpringIntegrationExamples