INTRODUCTION TO ENTERPRISE APPLICATION INTEGRATION

Home of All the Laws of Nature

Why Do We Need Integration?

- Enterprise business applications rarely live in isolation
- Users expect instant access to all business functions an enterprise can offer
- Requires disparate applications to be connected into a larger, integrated solution,

What Makes Integration so Hard?

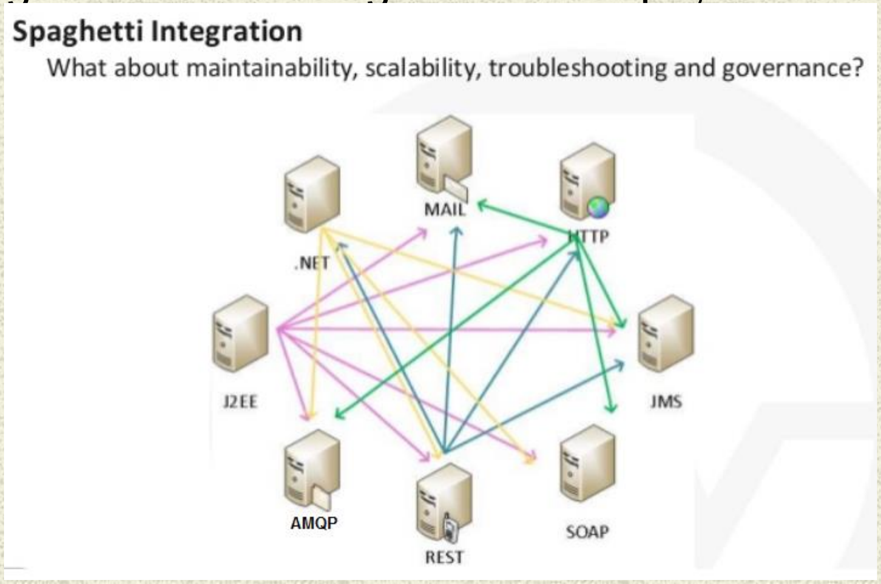
Architecting integration solutions is a complex task
Many conflicting drivers and even more possible 'right' solutions
The success of integration not known, in some cases, for years
No set of underlying guidelines, principles and best practices.

Enterprise Application Integration

- •Been around for as long as disparate computing paradigms have
- •EAI is integration of services and / or data.
- •Enterprise Application Integration platforms are known as:

  Enterprise Service Bus (ESB)

- "Traditional" ESB solutions - TIBCO, Axway, WebMethods
- •Open source / Java-centric solutions : ServiceMix, PEtALS, OpenESB, JBossESB, Mule , Apache Camel
- Non-ESB alternatives: solutions strung together with bailing wire and tape, a veritable Rube Goldberg machine


Spaghetti Integration
What about maintainability, scalability, troubleshooting and governance?
MAIL
HTTP
.NET
J2EE
JMS
AMQP
REST
SOAP

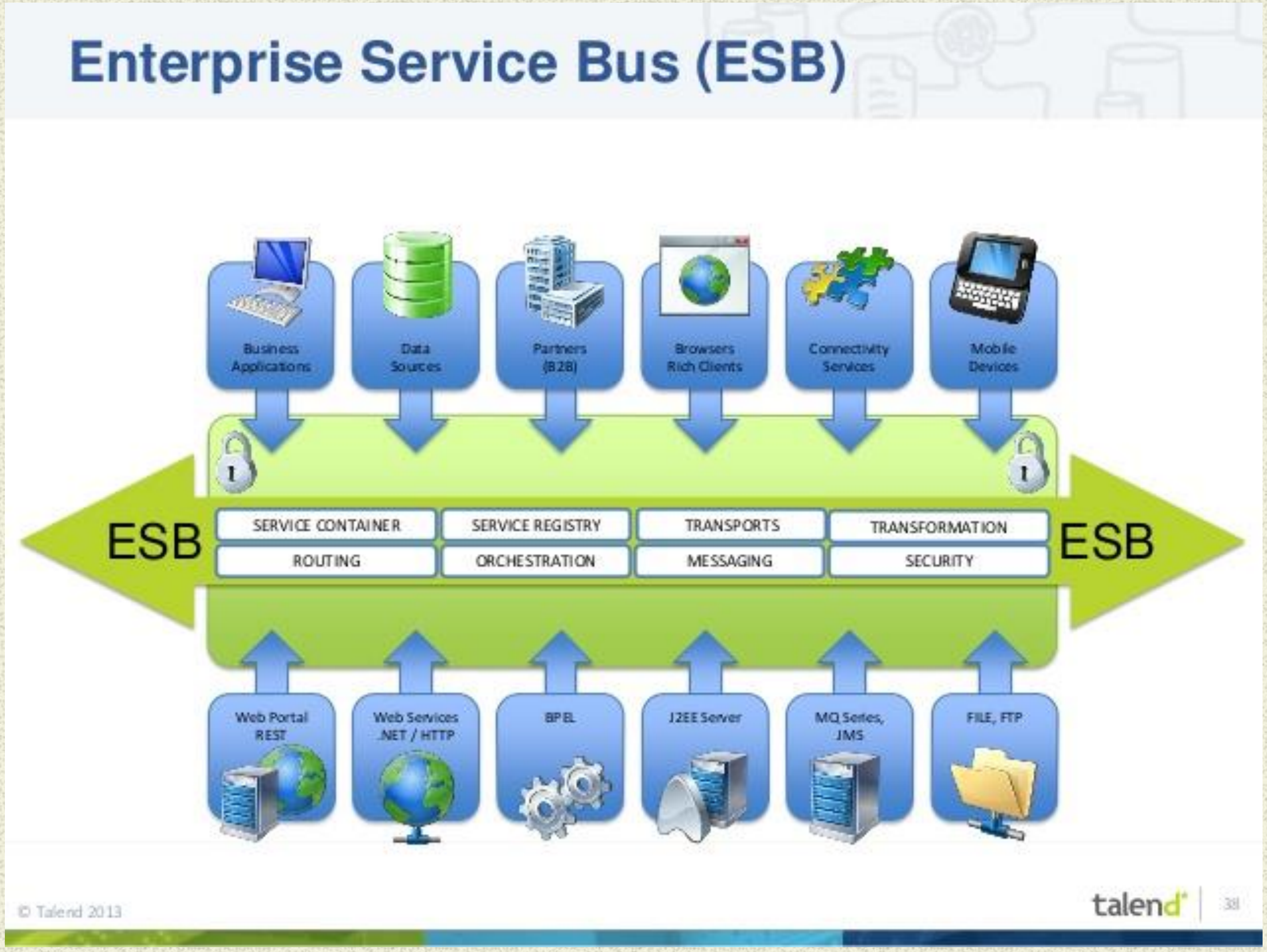
Common ESB Capabilities

Location Transparency
Transport Conversion
Message Transformation / Routing / Enhancement
Security
Monitoring and management
Process management (BPMs, orchestration)
Complex Event Processing

Integration Approaches:

File Transfer
Shared Database
Remote Procedure Call
Messaging

Enterprise Service Bus



Spring Integration

NOT necessarily an ESB….

BUT

Does provide the framework and all the components

needed to do the same things that would make you consider an ESB in the first place.

A Spring Integration application is just a Spring application with some beans in the context that handle the integration concerns.

That means you can deploy anywhere and at any scale.

It is Lightweight

Spring Integration

- **Goals:**
- Provide a simple model for implementing complex enterprise integration solutions.
- Facilitate asynchronous, message-driven behavior within a Spring-based application.
- Promote intuitive, incremental adoption for existing Spring users.

- **Spring Integration is guided by the following principles:**
- Components should be *loosely coupled* for modularity and testability.
- Enforce *separation of concerns* between business logic and integration logic
- Extension points should be abstract & promote *reuse* and *portability*.

- Implementation is an API based on Enterprise Integration Patterns.
- **Lightweight messaging using declarative adapters :**

NOT another name for the Spring remoting APIs _BUT_

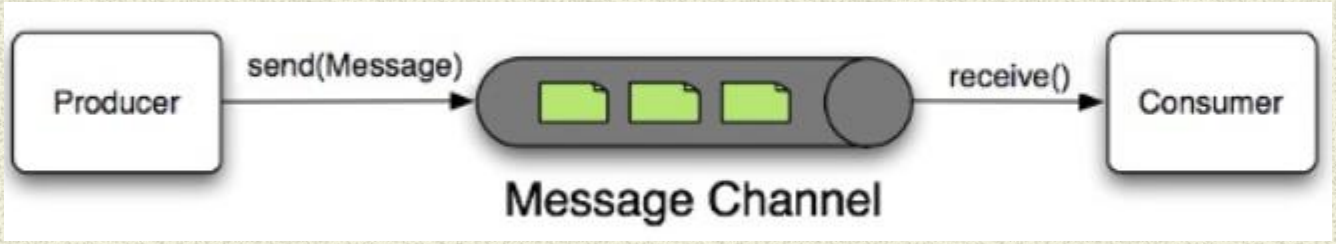
A High level of abstraction over Remoting, Messaging, and Scheduling

Spring Integration Main Components

Message : It is a generic wrapper for any Java object combined with metadata used by the framework while handling that object. It consists of a payload and header(s).

Message Channel : A message channel is the component through which messages are moved so it can be thought as a pipe between message producer and consumer. [PTP or Pub/Sub].

Message Endpoint : A message endpoint isolates application code from the infrastructure. In other words, it is an abstraction layer between the application code and the messaging framework. They are message producers & consumers.



[Some] Message Channels

- Message channels in Spring Integration can be divided into two main categories:

  Subscribe message channels

  Pollable message channels

  PublishSubscribe Channel

    Mainly for events

    Also can be used asynchronously

  Queue Channel

    FIFO Queue PTP

    requires polling [poller]

  Direct Channel [Default]

    Single Consumer [PTP]

Adapters and Gateways

Adapters are unidirectional

Gateways are Bidirectional

- Example:

    Amqp inbound-adapter

    Amqp outbound-adapter

    Amqp inbound-gateway

    Amqp outbound-gateway

Main Point

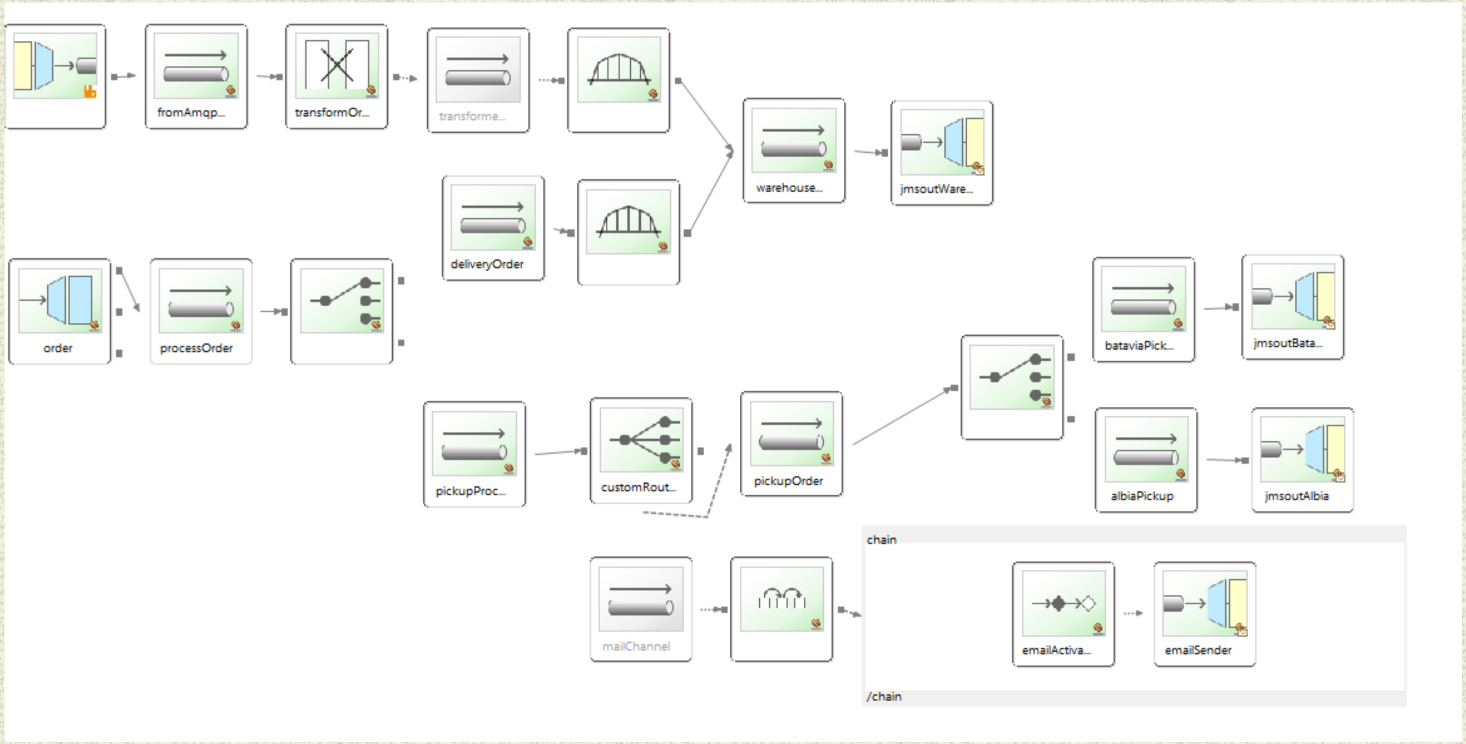
- Spring Enterprise Integration provides for seamless operation of the enterprise by providing adaptive and structured resource coordination.

- *Without a doubt, the structure and adaptive nature of finer levels of consciousness...ultimately, Pure Consciousness makes one's everyday experience seamless and smooth.*

Specific Nomenclature for Demos

MANY,MANY ,MANY ty There are a LOT! rs, endpoints

- **DEMO Diagram:**



[End point Support]

- **Some we will use in Demo:**
- **Messaging Gateway :** entry point for the system [encapsulates Message API]
- **Message Router** decides what channel(s) should receive the Message
- **Messaging Bridge** endpoint that connects two Channels or Channel Adapters.
- **Channel Adapter** is a Message Endpoint that enables connecting a single sender or receiver to a Message Channel

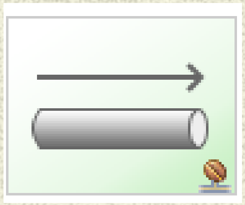
HQ to Warehouse Order Process Use Case

- ***Use Case:***
- Headquarters receives a Customer request via phone
- Headquarters generates a Customer Order [Order]
- It broadcasts the Order to the Warehouse through RabbitMQ
- The Warehouse "listens" for orders on a JMS Queue.
- The Warehouse accept & process an Order as a RouteOrder

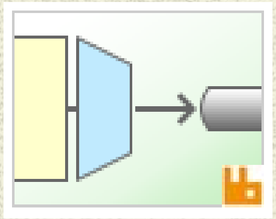
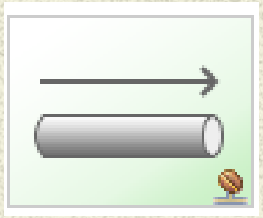
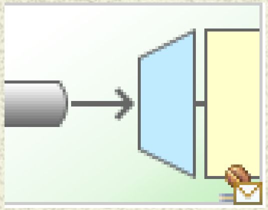
- ***Solution:***
- Listen for RabbitMQ Order message **[Rabbit Adapter]**
- Transform Order Message to RouteOrder Message **[Transformer]**
- Broadcast RouteOrder Message to Warehouse **[JMS Adapter]**

Spring EAI Components

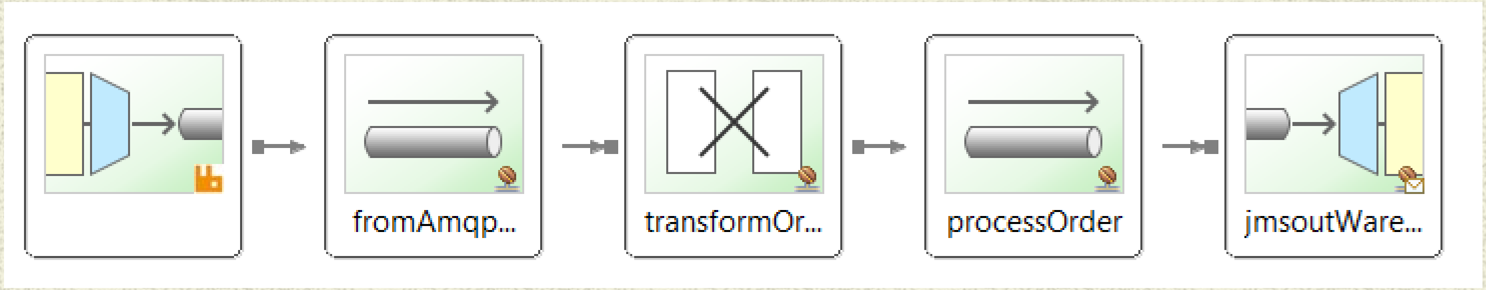
A Channel is a message "pipe“. DirectChannel: simple point-to-point channel



- **An Adapter connects [thru a channel] to another system [Adapter]**

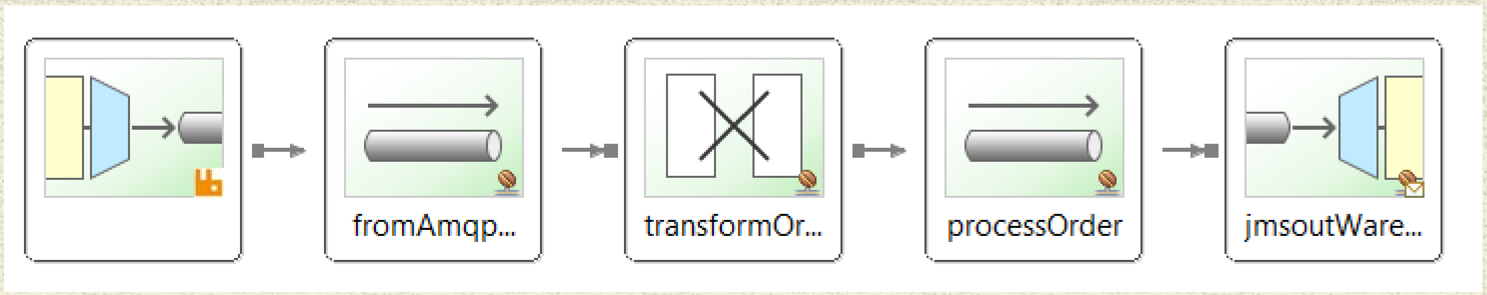




- ***Rabbit Inbound Adapter*** ***JMS Outbound Adapter***
- **A Transformer converts a message payload from one format to another**



Transformer

Configuration Details



```
• <amqp:inbound-channel-adapter channel="fromAmqpOrder"
     queue-names="purchasesPhone" connection-factory="amqpConnectionFactory" />
• <channel id="fromAmqpOrder" />
•  <transformer id="transformOrder" ref="orderTransformer"
         input-channel="fromAmqpOrder" output-channel="processOrder" />
•  <beans:bean id="orderTransformer"
•                    class="edu.mum.integration.OrderTransformerImpl" />
•  <channel id="processOrder" />
• <jms:outbound-channel-adapter id="jmsoutWarehouse"
                  channel="processOrder" destination="warehouseQueue"/>
```

See EAIBusAmqpJms Demo

Transformer Details

```
public class OrderTransformerImpl implements OrderTransformer {

      /**
      * Transform Order from AMQP to RouteOrder for JMS
       */
@Transformer(inputChannel="fromAmqpOrder", outputChannel="processOrder")
public RouteOrder transformOrder(Order order) {

      String name = order.getItems().get(0).getProduct().getName();
      int quantity = order.getItems().get(0).getQuantity();
      int orderNumber = Integer.parseInt(order.getOrderNumber());

      RouteOrder routeOrder = new RouteOrder(name, quantity,
      RouteOrderType.DELIVERY, orderNumber);

      return routeOrder;
}
```

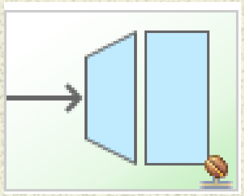
Customer Service RouteOrder to Warehouse Use Case

- ***Use Case:***
- Customer Service handles email orders via desktop [e.g. JavaFx]
- Desktop App generates RouteOrder
- The RouteOrder is broadcast to the Warehouse via JMS
- The Warehouse “listens” for orders on a JMS Queue.
- The Warehouse accept & process a RouteOrder

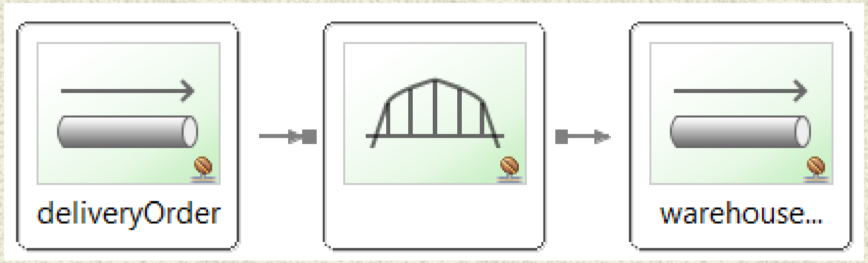
- ***Solution:***
- Receive RouteOrder via Gateway
- Broadcast RouteOrder Message to Warehouse

[More] Spring EAI Components

- **A Gateway converts input from local Application to EAI Messaging API**

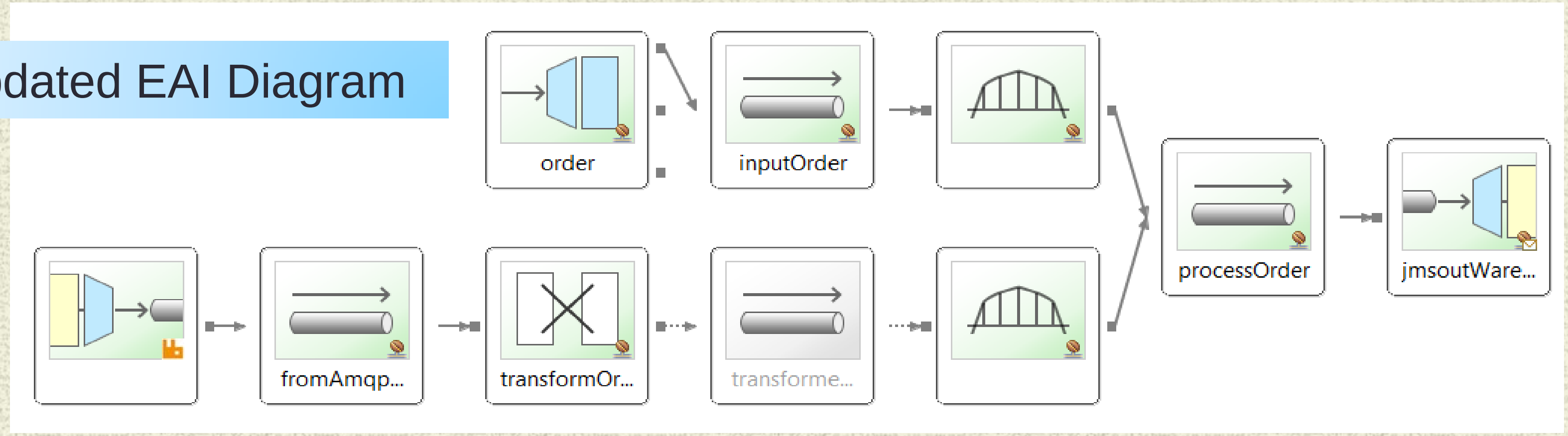


- **A Bridge connects two channels**

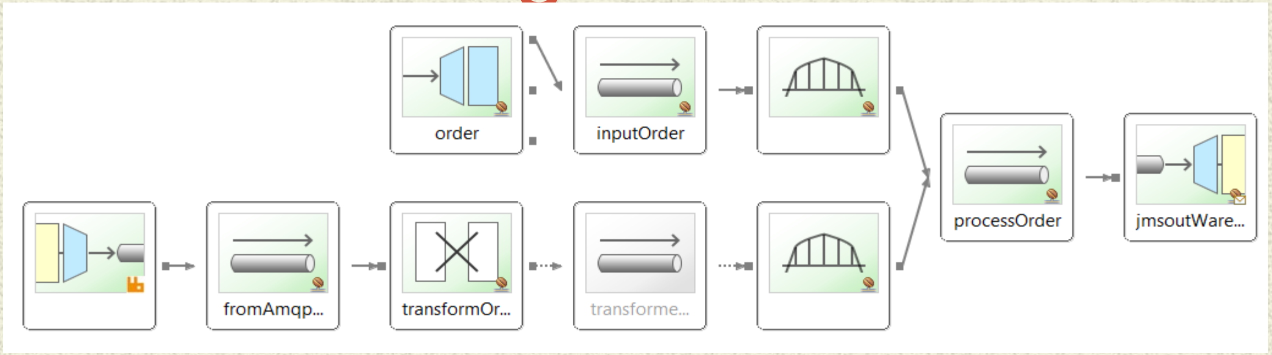


Bridge

Updated EAI Diagram



Configuration Details



```
<gateway id="order" service-interface="edu.mum.integration.OrderGateway"
      default-request-channel="inputOrder"/>
 <channel id="inputOrder" />
<bridge input-channel='inputOrder' output-channel='processOrder'/>

<channel id="fromAmqpOrder" />
 <amqp:inbound-channel-adapter channel="fromAmqpOrder"
        queue-names="purchasesPhone" connection-factory="amqpConnectionFactory" />
<transformer id="transformOrder" ref="orderTransformer"
              input-channel="fromAmqpOrder" output-channel="transformedOrder" />
<beans:bean id="orderTransformer" class="edu.mum.integration.OrderTransformerImpl" />

<!-- input order && transformedOrder channels need to go to JMS -->
<bridge input-channel='transformedOrder' output-channel='processOrder'/>

<channel id="processOrder" />
 <jms:outbound-channel-adapter id="jmsoutWarehouse" channel="processOrder"
      destination="warehouseQueue"/>
```

… from EAIBusAmqpJms Demo

See EAIBusDesktop Demo

Order Gateway

```
// The entry point for the Bus Flow.
public interface OrderGateway {
    //Process a book order.
 @Gateway(requestChannel="processOrder")
  public void process(RouteOrder order);

}
```

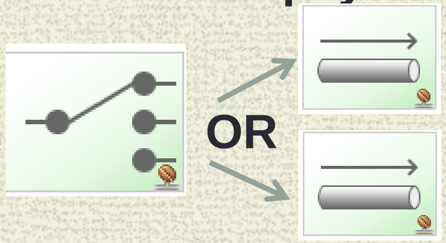
Customer Service Order for Store Pickup Use Case

- ***Use Case:***
- Customer chooses option to pickup order at store
- The RouteOrder is broadcast to the store via JMS
- The Store “listens” for orders on a JMS Queue.
- The Store accepts & processes a RouteOrder
- A notification email is sent to the Customer

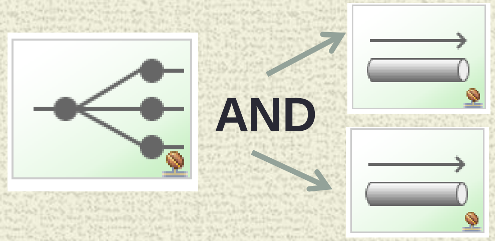
- ***Solution:***
- Determine to send RouteOrder to store [via **Content Router**[
- Branch to dual activities[JMS&Email [via **recipient-list-router**]
- Send Email to Customer [via **Mail Adapter**]
- Broadcast RouteOrder Message to Store [via **JMS Adapter**]

[More] Spring EAI Components

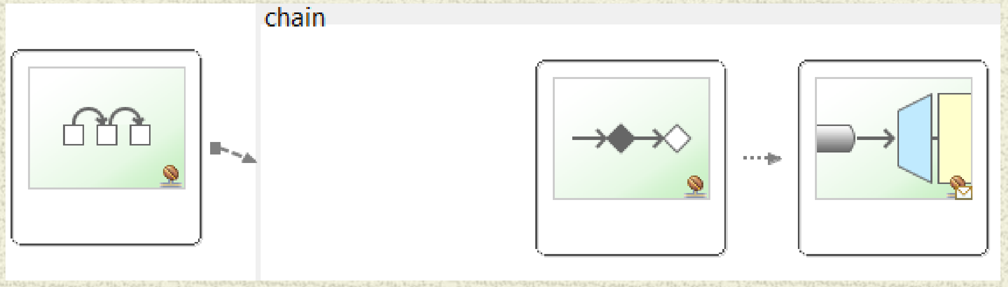
- A Content Router examines the payload to determine message channel



- A recipient-list-router distributes the message to all message channels

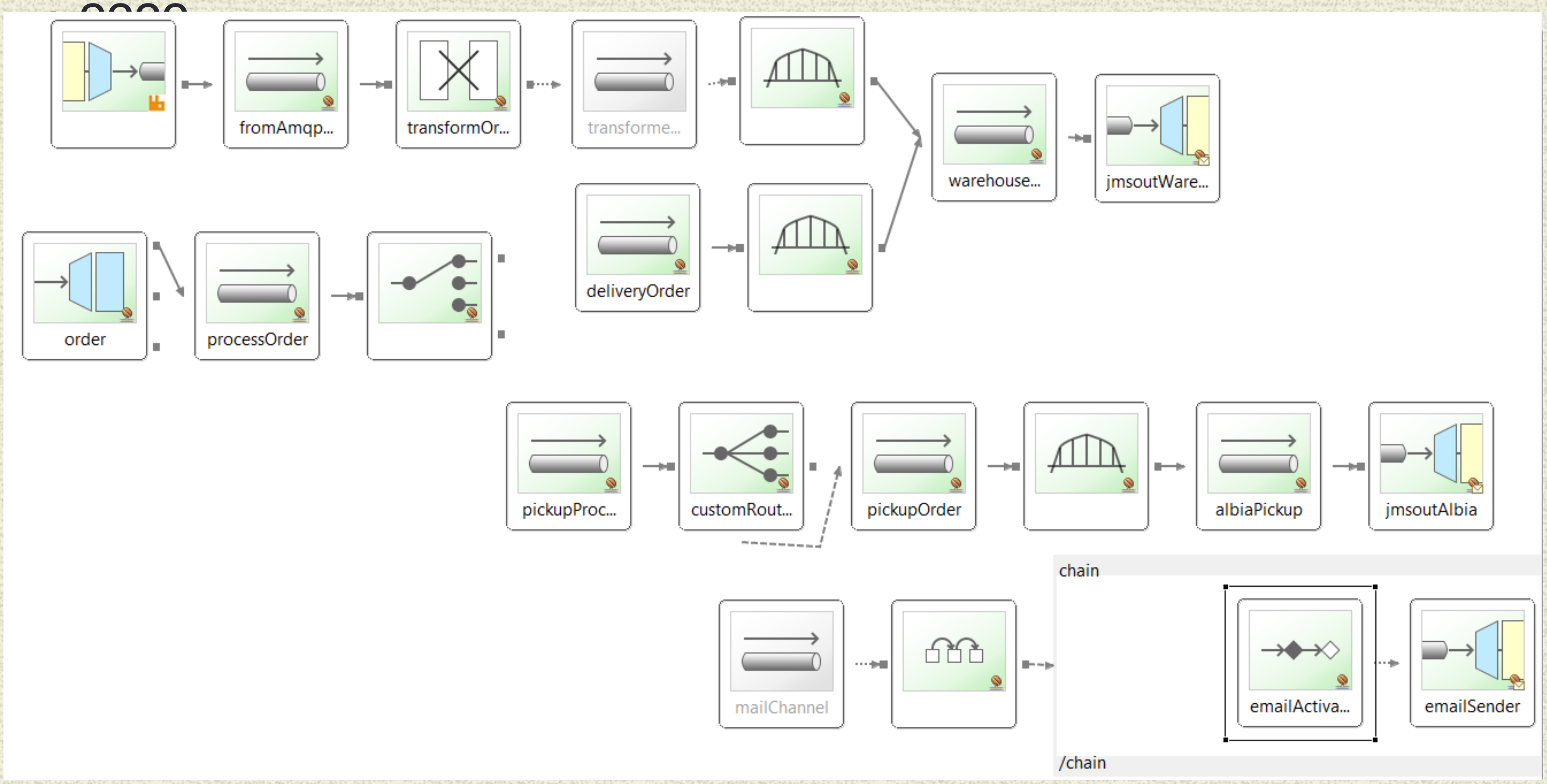


- A chain is a convenience to reduce channels between endpoints
- A Service Activator triggers (or activates) a Spring-managed bean

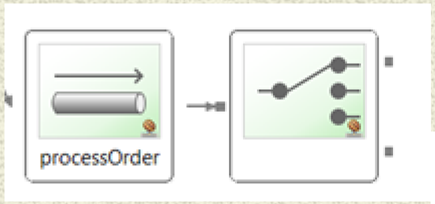


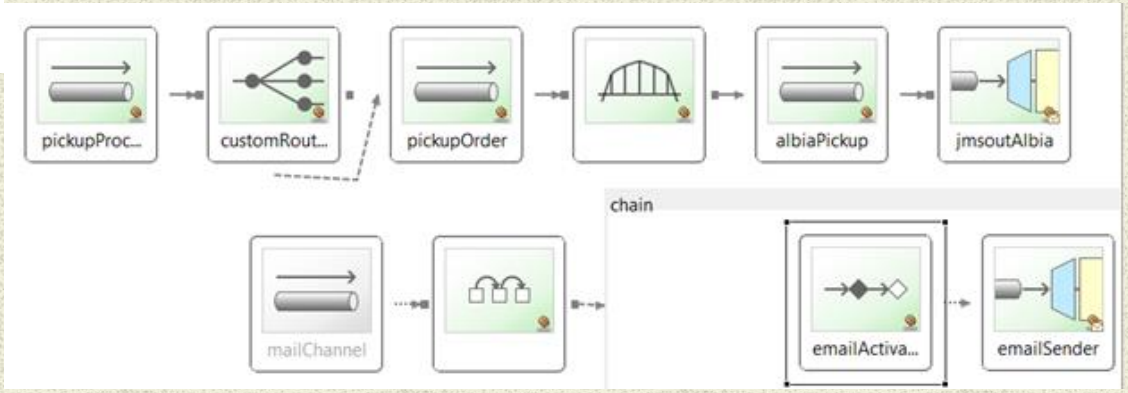
- Chain Service Mail Adapter
- Activator

Store DEMO Integration graph



Configuration Details





```
<!-- "ref" refers to custom Router:ordeRouter "method" is the message channel -->
 <router input-channel="processOrder" ref="orderRouter" method="processOrder"/>

<channel id="pickupProcess" />

<recipient-list-router id="customerRouter" input-channel="pickupProcess" >
  <recipient channel="pickupOrder"/>
  <recipient channel="mailChannel"/>
</recipient-list-router>
<!--  Service Activator triggers emailService Bean to send email-->
 <chain input-channel="mailChannel">
     <service-activator id="emailActivator" ref="emailService"/>
    <mail:outbound-channel-adapter id="emailSender" mail-sender="mailSender" />
</chain>
```

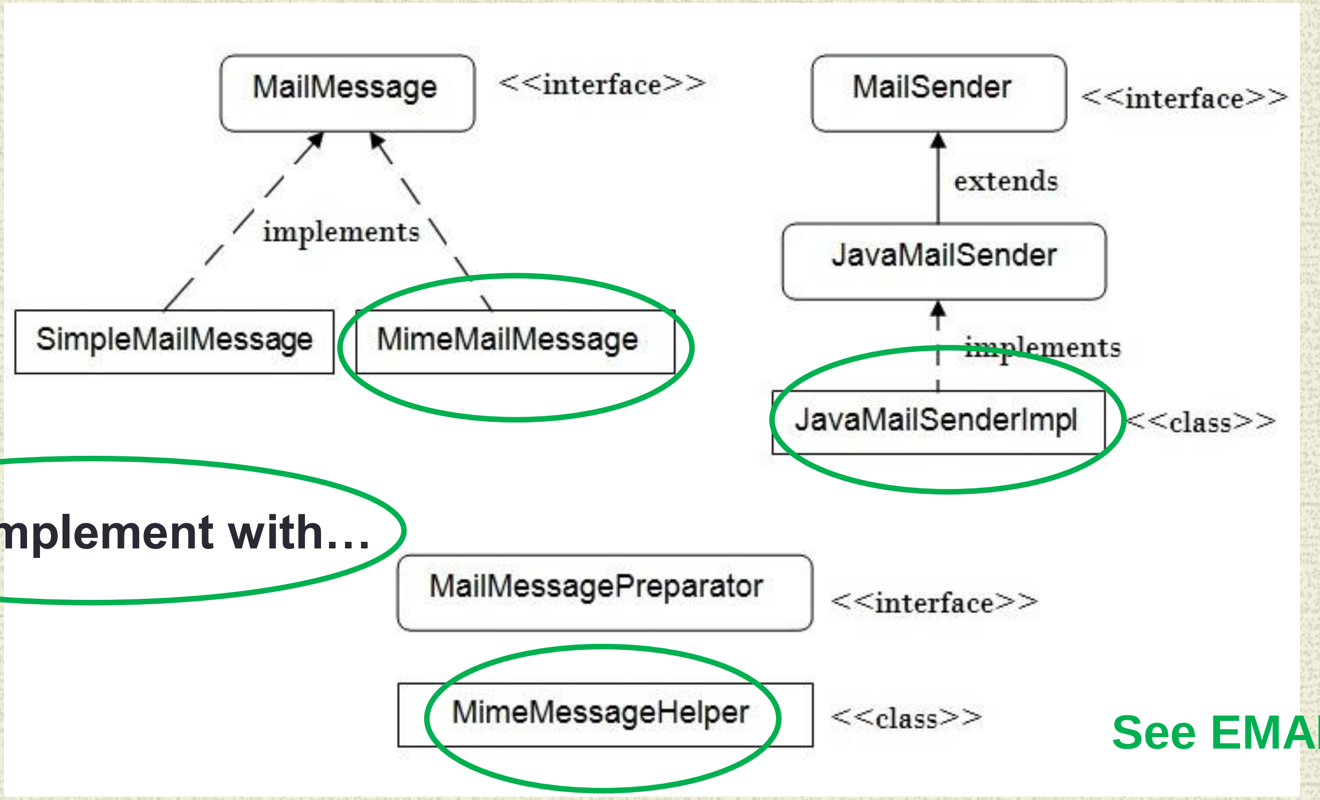
See EAIBusStore Demo

Order Router

```
@MessageEndpoint
public class OrderRouter {

    // Process order.  Routes based on whether or
       * not the order is a delivery or pickup.
        @Router(inputChannel="processOrder")
        public String processOrder(RouteOrder order) {
            String destination = null;

            switch (order.getBookOrderType()) {

            case DELIVERY:
                   destination = "deliveryOrder";
                   break;
           case PICKUP:
                   destination = "pickupProcess";
                   break;
     }
    return destination;
}
```

Spring Email Support
Messaging of a different sort

- Utility library for sending email
- Shields the developer from the specifics of the underlying mailing system
- Responsible for low level resource handling on behalf of the client.



We will implement with…

See EMAIL Demo

Template Engine

Spring Email capability - simple text emails

- Enterprise application need more "substance"
  - Writing HTML-based email content in Java code is tedious and error prone
  - There is no clear separation between display logic and business logic
  - Changing the display structure of the email requires writing Java code, recompiling, redeploying etc…

- **Template Engines:**

  FreeMarker : Velocity : Thymeleaf : Handlebars : Mustache

- **View Templates** (For rendering views in your browser)
- **Email Templates** - with support for both HTML and Text emails

Thymeleaf

- Java XML / XHTML / HTML5 template engine
- Complete substitute for JSP with Natural Templating abilities
- Provides full Spring MVC web integration
- Can be used offline
- HTML email is an example offline application
- Create feature-rich HTML email templates with expressions,
  internationalization, date/number formatting, etc.
- Email templates can be designed by UI designers as usual HTML pages Displayable on a browser as working templates.
-

Thymeleaf.org

Online Order to Warehouse Use Case

Use Case:

Customer makes a purchase on E-commerce web site

Web Site generates an Order and sends it via HTTP

A RouteOrder is broadcast to the Warehouse via JMS

The Warehouse “listens” for orders on a JMS Queue.

The Warehouse accept & process a RouteOrder

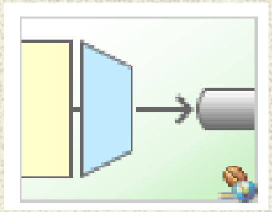
Solution:

Receive HTTP Order via **HTTP Adapter**

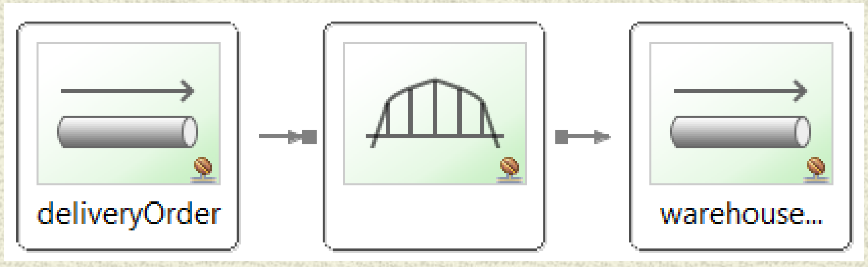
Merge Order with HQ Order to transform to RouteOrder

[More] Spring EAI Components

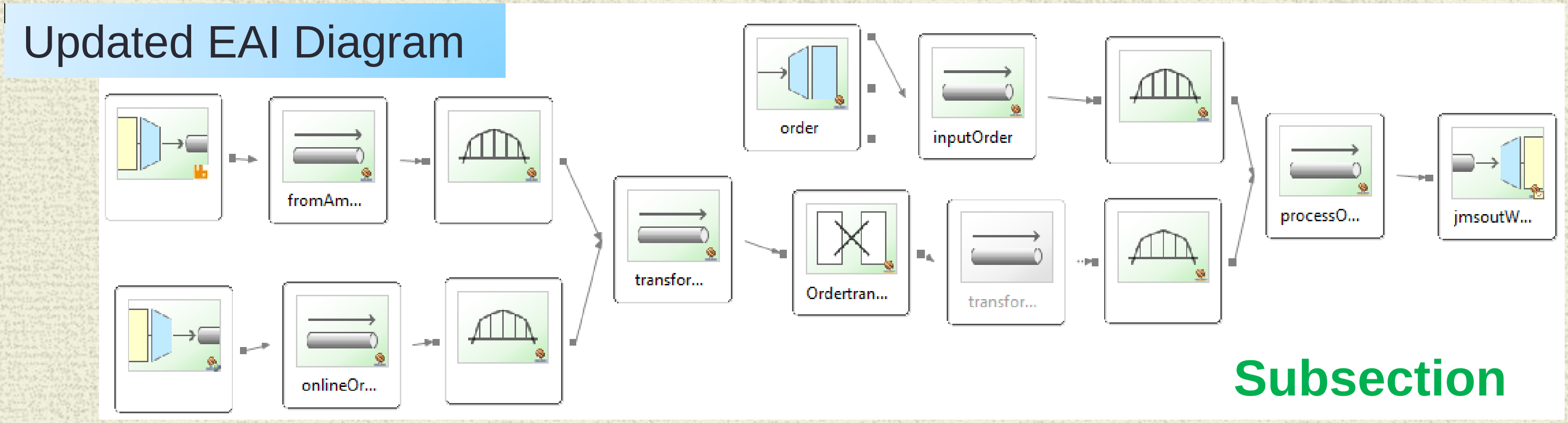
- A HTTP Inbound Channel Adapter



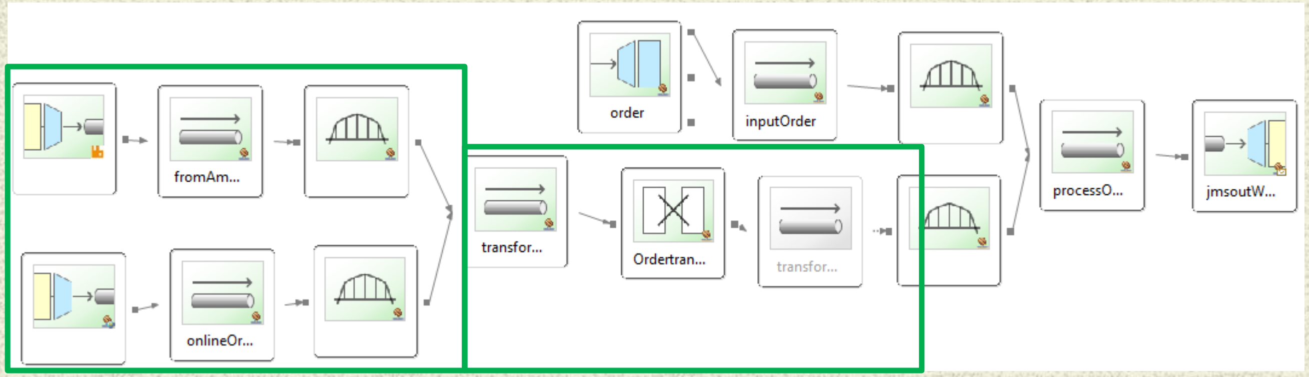
- A Bridge is for connecting channels



Bridge



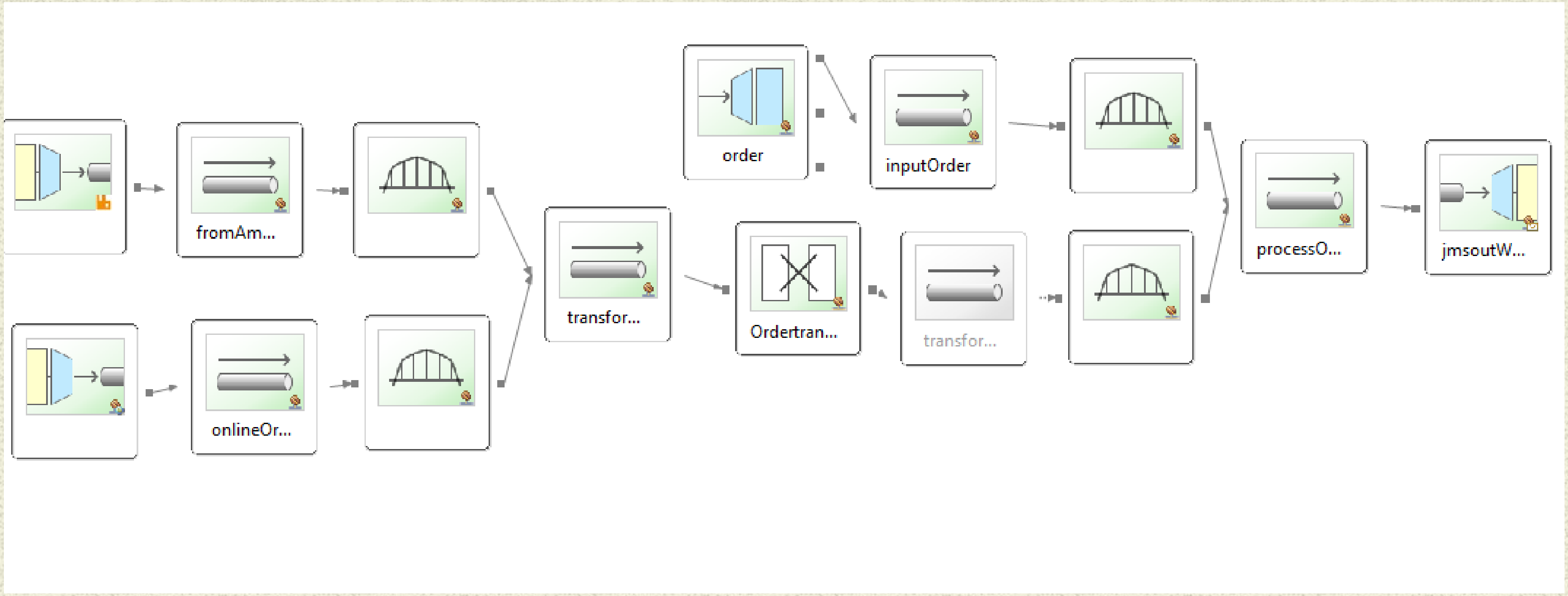
Configuration Details



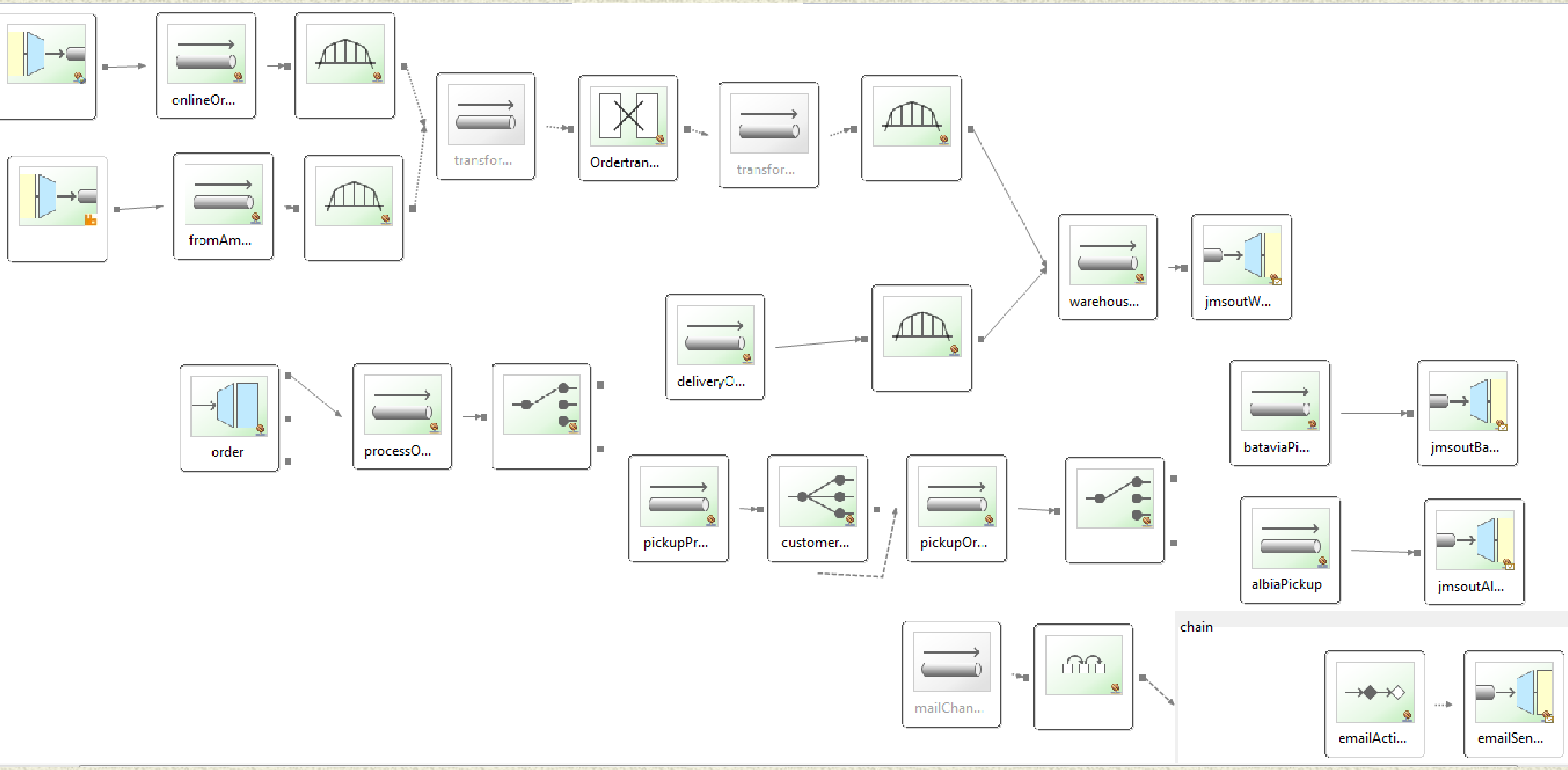
```
<channel id="fromAmqpOrder" />
<amqp:inbound-channel-adapter channel="fromAmqpOrder"
           queue-names="purchasesPhone" connection-factory="amqpConnectionFactory" />
<bridge input-channel='fromAmqpOrder' output-channel='transformOrder' />

<channel id="onlineOrder"/>
 <http:inbound-channel-adapter    channel="onlineOrder"
      status-code-expression="T(org.springframework.http.HttpStatus).NO_CONTENT"
      supported-methods="POST" path="/onlineOrder"
      request-payload-type="edu.mum.domain.Order">
     <http:request-mapping consumes="application/json" />
 </http:inbound-channel-adapter>
<bridge input-channel='onlineOrder' output-channel='transformOrder' />

<transformer id="transformOrder" ref="orderTransformer"
                  input-channel=" transformOrder " output-channel="transformedOrder" />
<beans:bean id="orderTransformer" class="edu.mum.integration.OrderTransformerImpl" />
```

See EAIBusHttpOnline Demo

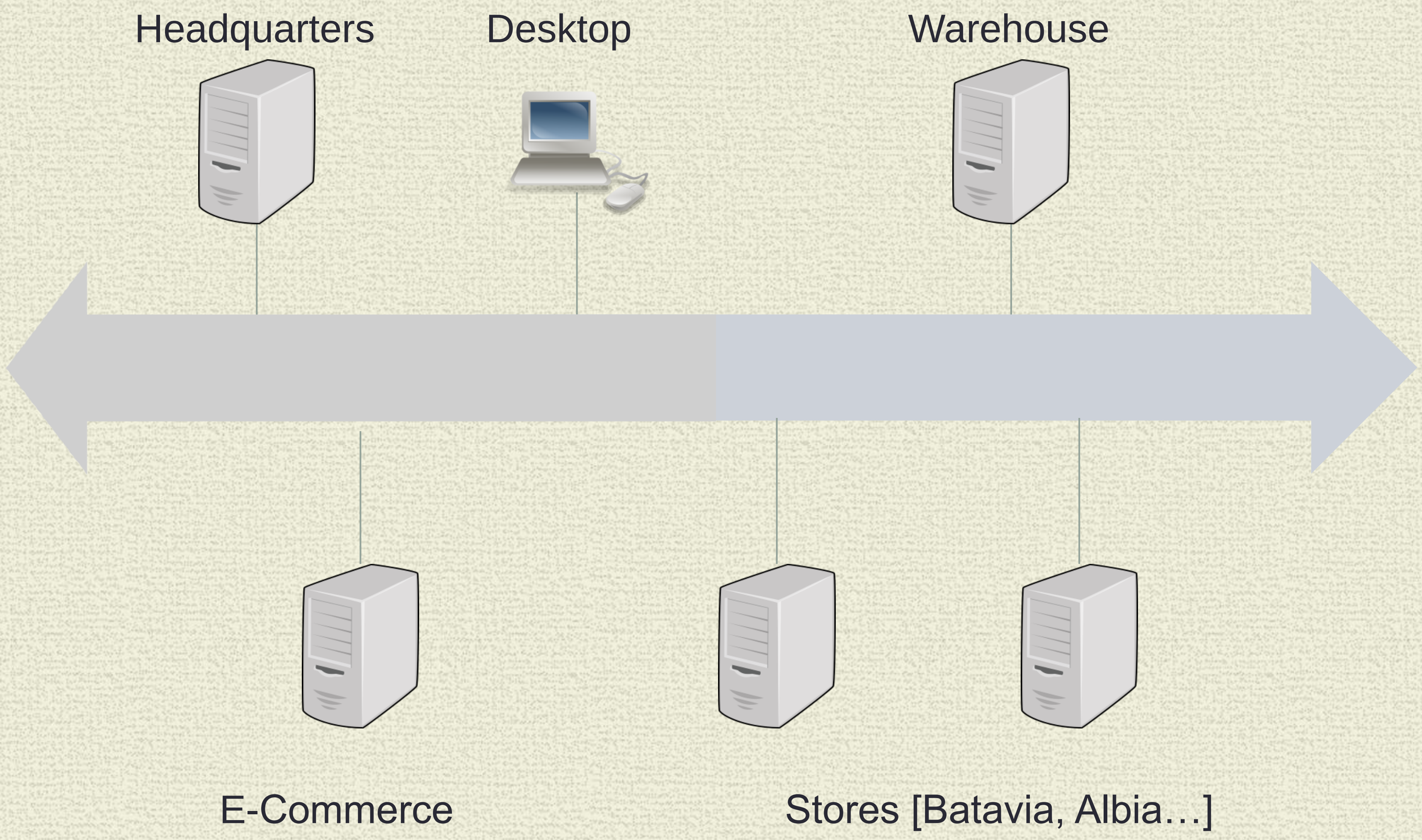
Online order "Subsection"



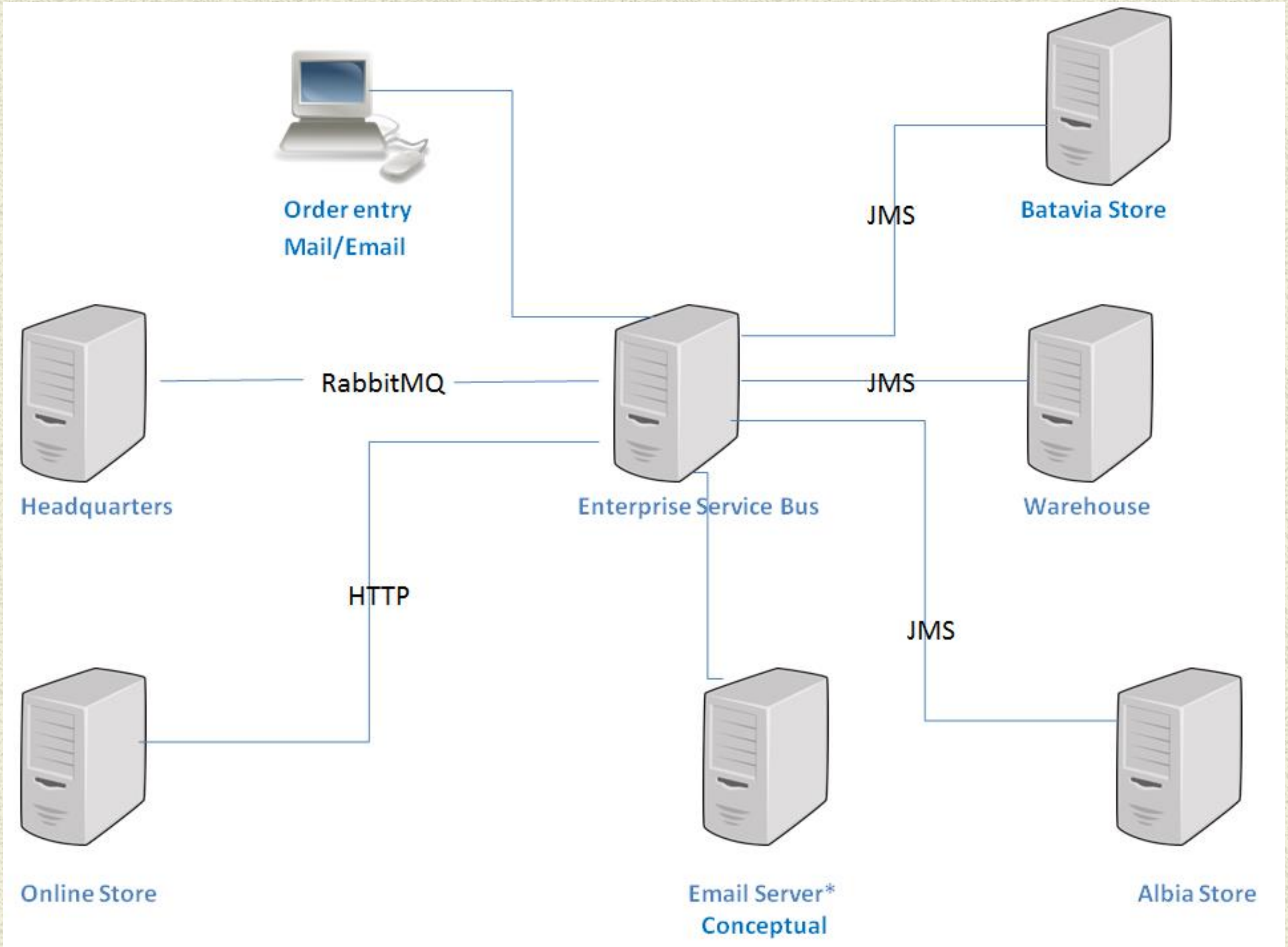
Final Spring EAI Demo



“Bus” view of Spring Integration Demo



ESB Order Demo Network Diagram



Main Point

- Enterprise Application Integration provides an “almost” endless variety of components.

- *Life is characterized by an infinite range of possibilities*