PA165
Enterprise Integration

Filip Nguyen
Lab Software Architectures and Information Systems

Seminar 14

December 16, 2014

Today

- Integration
- Motivation
- Integration Criteria and Styles
- Messaging
- SOA
- ESB - JBoss ESB, Apache Camel
- Apache Camel Introduction

Motivation

- No green field projects anymore
- Systems need to communicate with all the issues that arise:
    - Various protocols/database systems (how to share data between your Haskell application and Java app?)
    - Systems might be down
    - Distributed Transactions must be handled
    - Systems APIs change

Integration Criteria

1. Coupling
2. Asynchronicity
3. Data Format
4. Data/Functionality sharing
5. Integration Technology

Style - File Transfer

- File is nice common denominator among systems
- Importing/Exporting CSV files is very common functionality
  - Excel
  - Import CSV into MUNI IS
- Easy to generate
- Need to handwrite everything (transformations, import/export)
- Not very timely
- No functionality sharing

Style - Shared Database

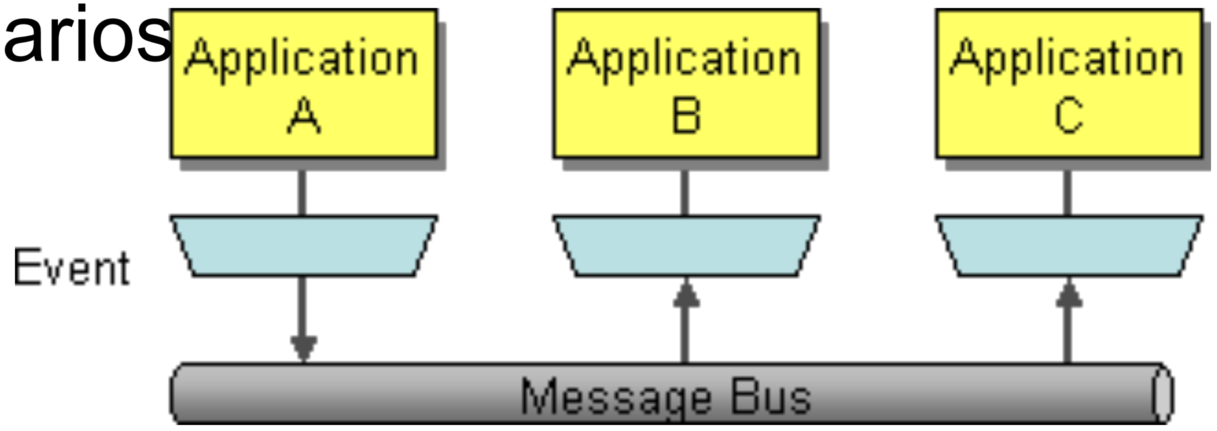
- High Data consistency
- Timely
- Extremely high coupling
- No functionality sharing

Style - Remote Method Invocation

- No asynchronicity
- Lower coupling as opposed to Shared Database. Decoupling provided we have good interfaces
- More extensible
- For example web services that are used as a Service Layer
- REST/SOAP

Style - Messaging

- Asynchronous
- Functionality sharing is more complicated
- Solves most problems in distributed systems
- Forces developer to think asynchronously
- Good APIs (JMS) avoid vendor lock-in scenarios



Messaging in Java

- Java Messaging Service (JSR 914)
- https://www.jcp.org/en/jsr/detail?id=914
- Version 1.0 is prevalent but version 2.0 under adoption
- Apache Active MQ
- JBoss HornetQ, JBoss Messaging
- Every application server bundles some implementation

JMS Basics

- Message
- Works over network
- Queue - many clients may insert and retrieve data
- Topic - publish subscribe implementation
- Ensures transactional behavior - the message is not removed from the queue until you acknowledge it
- Implementations use various backing stores:
    - Database
    - Files
    - Memory

JMS Message

- http://docs.oracle.com/javaee/6/api/javax/jms/Message.html
- TextMessage, ByteMessage,...
- Headers
  - JMSDestionation
  - JMSMessageID
  - ....
- Properties
- Payload

JMS Message Consumption

- javax.jms.MessageConsumer
  - receive()
  - receive(long timeout)
- For Topics: Asynchronous MessageListeners
- Cubersome, while loops needed etc.
- Cannot receive in multiple threads (common requirement!)
- No transparent fail-over - when brooker goes down it is not possible to recover easily

JEE Message Driven Bean

```
public void onMessage
           (Message inMessage)
```

```
@MessageDriven(mappedName="jms/Queue", activationConfig =  {
        @ActivationConfigProperty(propertyName = "acknowledgeMode",
                                  propertyValue = "Auto-acknowledge"),
        @ActivationConfigProperty(propertyName = "destinationType",
                                  propertyValue = "javax.jms.Queue")
    })
public class SimpleMessageBean implements MessageListener {
    @Resource
    private MessageDrivenContext mdc;
```

MDB problems

- You need container that supports MDB
- You need to know how to configure its JNDI and connection to the Queue - quite complicated

Spring JMS handling

- JmsTemplate
- Spring Message Listener Containers
  - can do with POJOs
  - implement onMessage
  - is this enough for message handling?

```
<bean id="jmsContainer"
class="org.springframework.jms.listener.DefaultMessageListenerContainer">
        <property name="connectionFactory" ref="connectionFactory"/>
        <property name="destination" ref="destination"/>
        <property name="messageListener" ref="messageListener" />
        <property name="sessionTransacted" value="true"/>
</bean>
```

SessionAwareMessageListener

```
public interface SessionAwareMessageListener {
    void onMessage(Message message, Session session) throws JMSException;
}
```

Camel JMS

```
from("jms:queue:foo").to("bean:myBusinessLogic");
```

Other Messaging Implementations

- Amazon SQS (SOAP API), cloud
- AMQP (protocol level interoprability, not only interfaces as in the case of JMS)

Amazon SQS

- Bindings to several languages
  - http://sqs.us-east-1.amazonaws.com/doc/2008-01-01/QueueService.wsdl

```
sqs.sendMessage(new SendMessageRequest()
    .withQueueUrl(myQueueUrl)
    .withMessageBody("This is my message
text."));
```

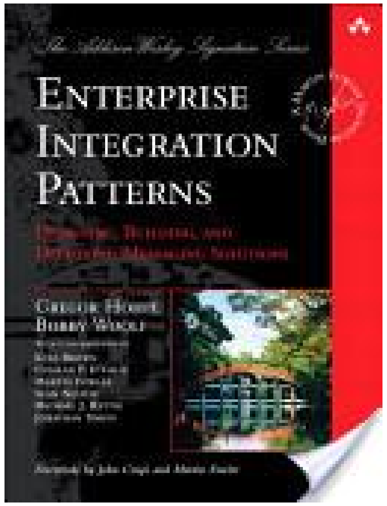
Apache Thrift

- Interprocess integration
- You generated stubs to different languages from 1 contract file *.thrift
- http://thrift.apache.org/

```
      struct UserProfile {
        1: i32 uid,
        2: string name,
        3: string blurb
      }
      service UserStorage {
        void store(1: UserProfile user),
        UserProfile retrieve(1: i32 uid)
      }
```

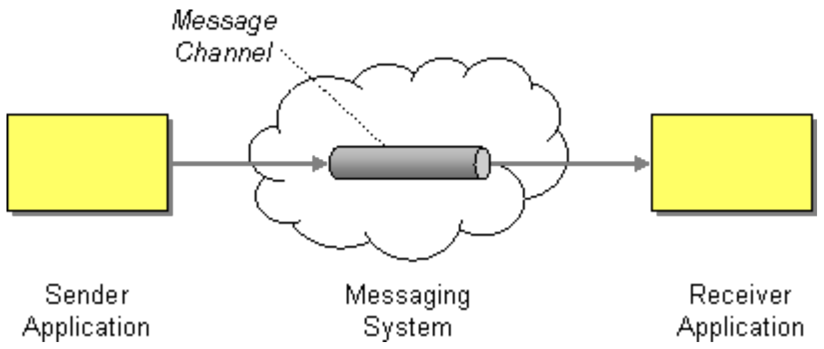
Integration Patterns

- Introduced by Gregor Hophe
  - http://www.eaipatterns.com/http://www.eaipatterns.com/
- 65 patterns
- Notation
- Usage
- Many implementations exist



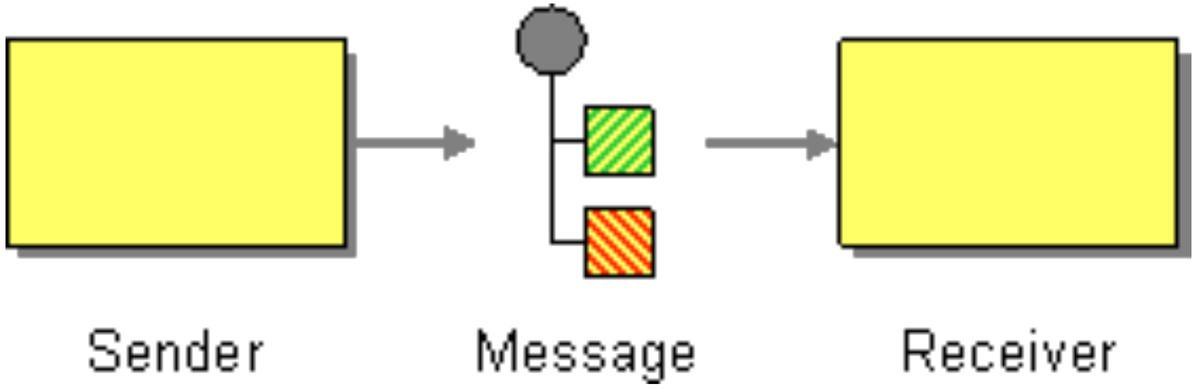
Message Channel

- Identified by name
- Usually a JMS Queue or Topic



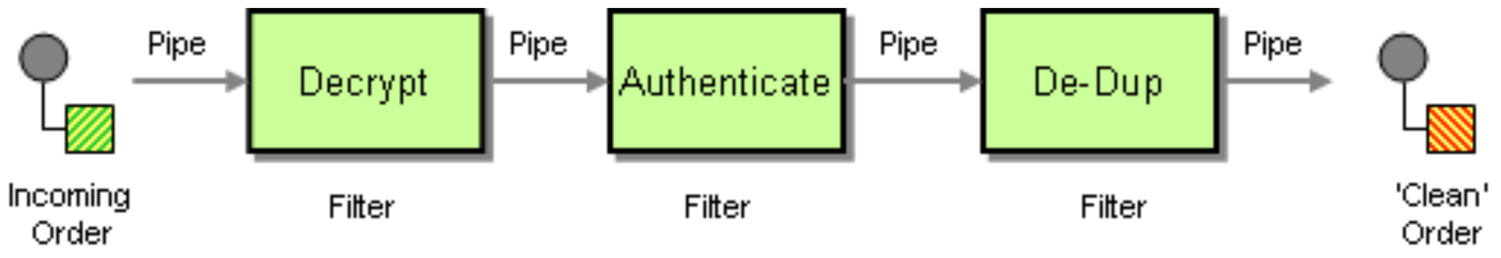
Message

- Body, Headers



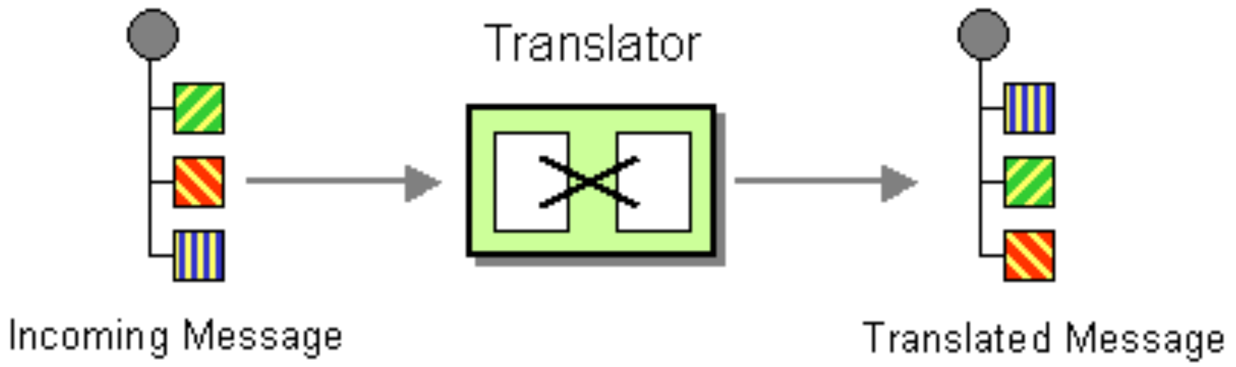
Pipes and Filters

- Sequence of actions done on a message



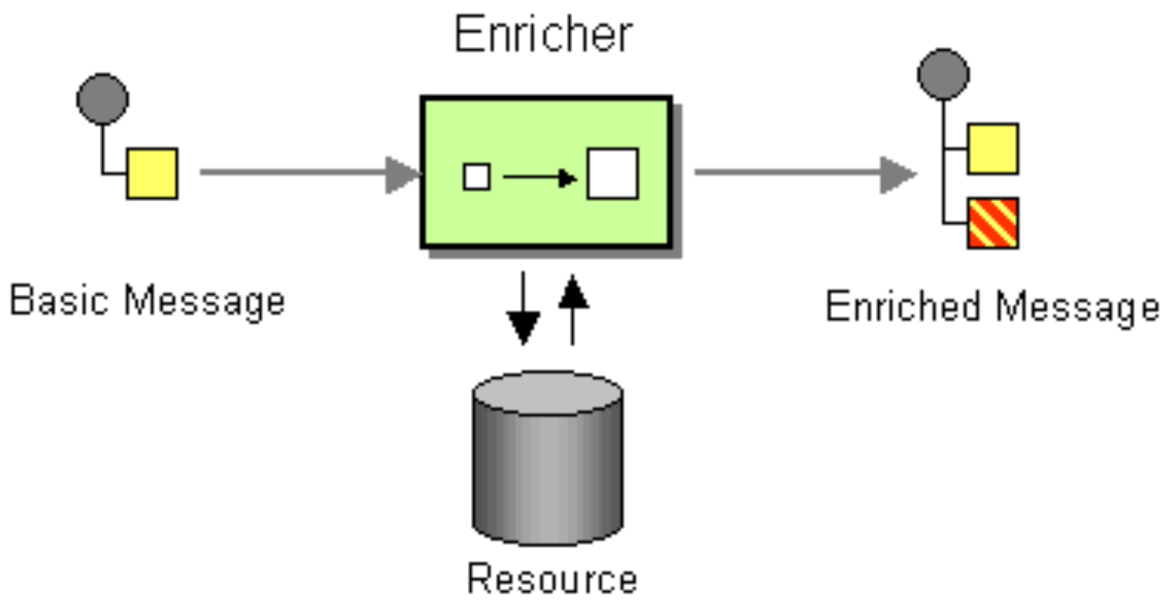
Message Translator

- Reformatting of a message



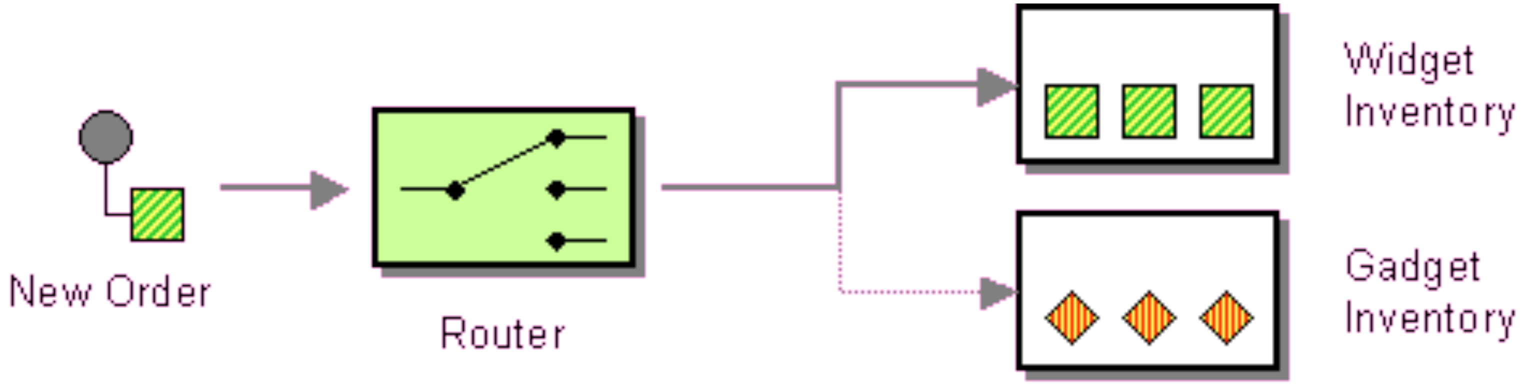
Enricher

- Adds data to a message



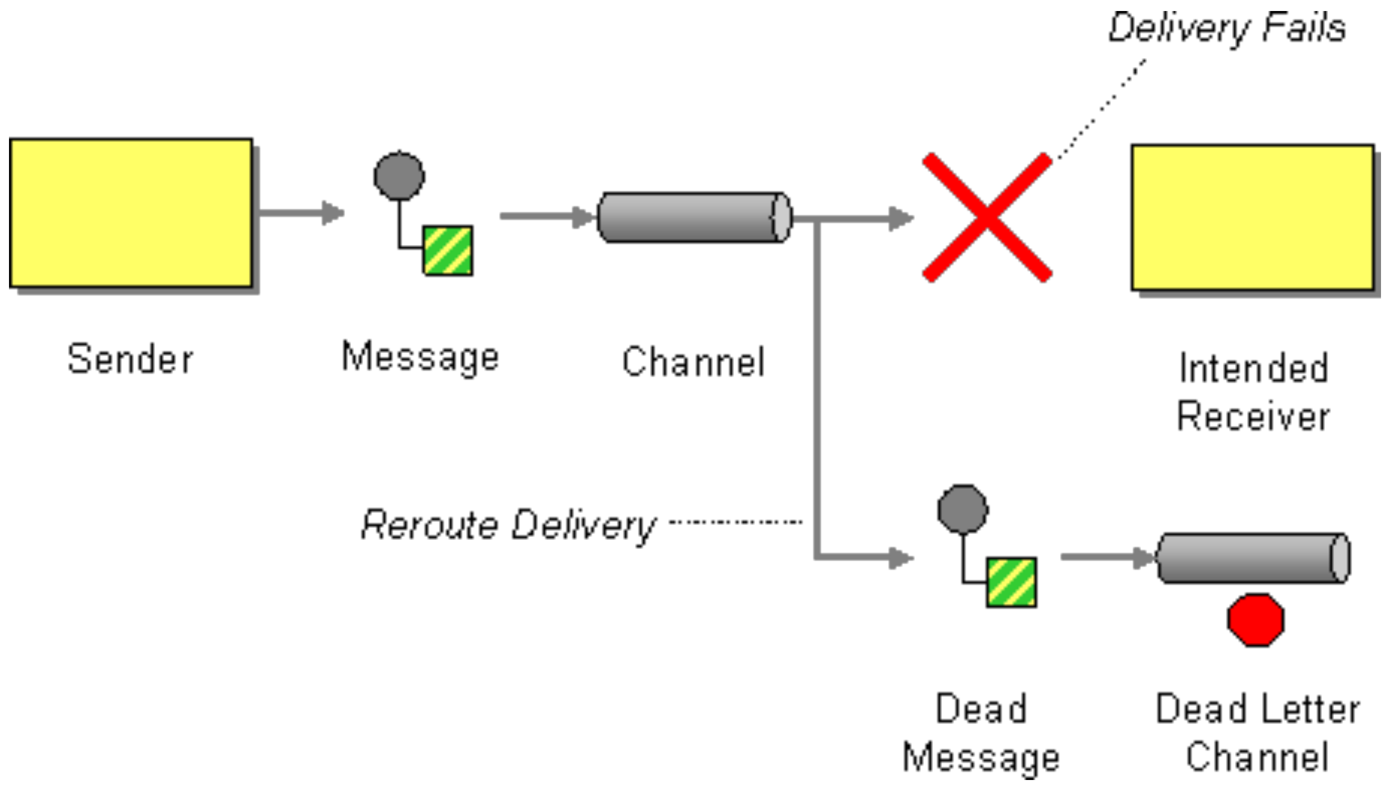
Content Based Router

- Usually XPath or similar selector



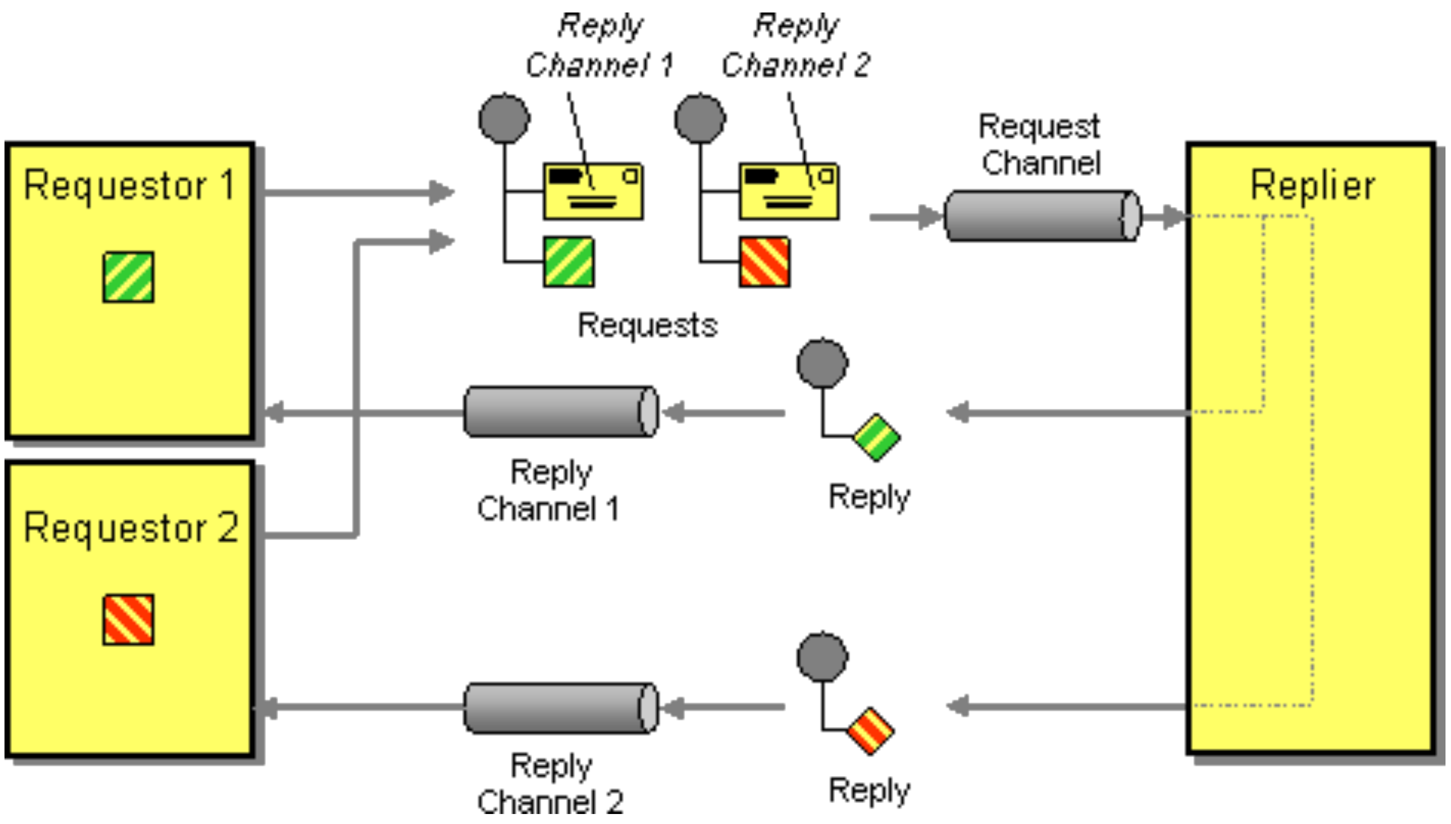
Dead Letter Channel

- Error handling!



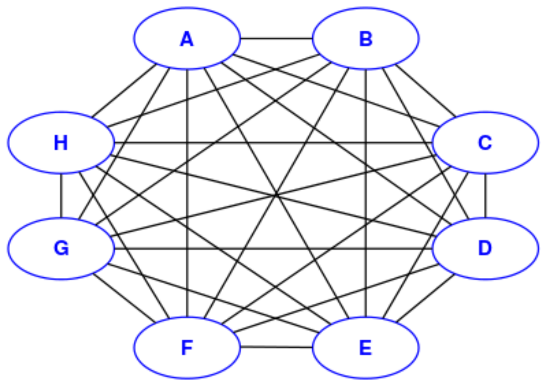
Return Address

- How to do request-reply?

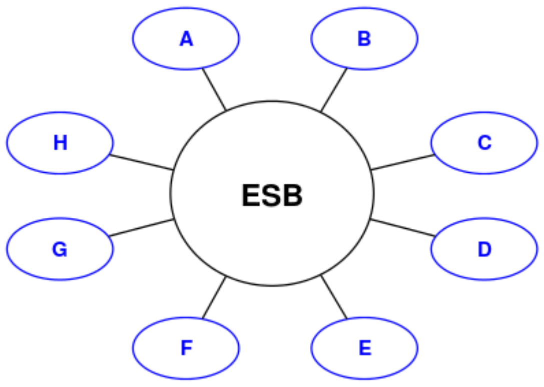


Enterprise Service Bus

- Very Important Archtiectural Style
- Helps implementing SOA
- Connects all the resources from the organization in one place



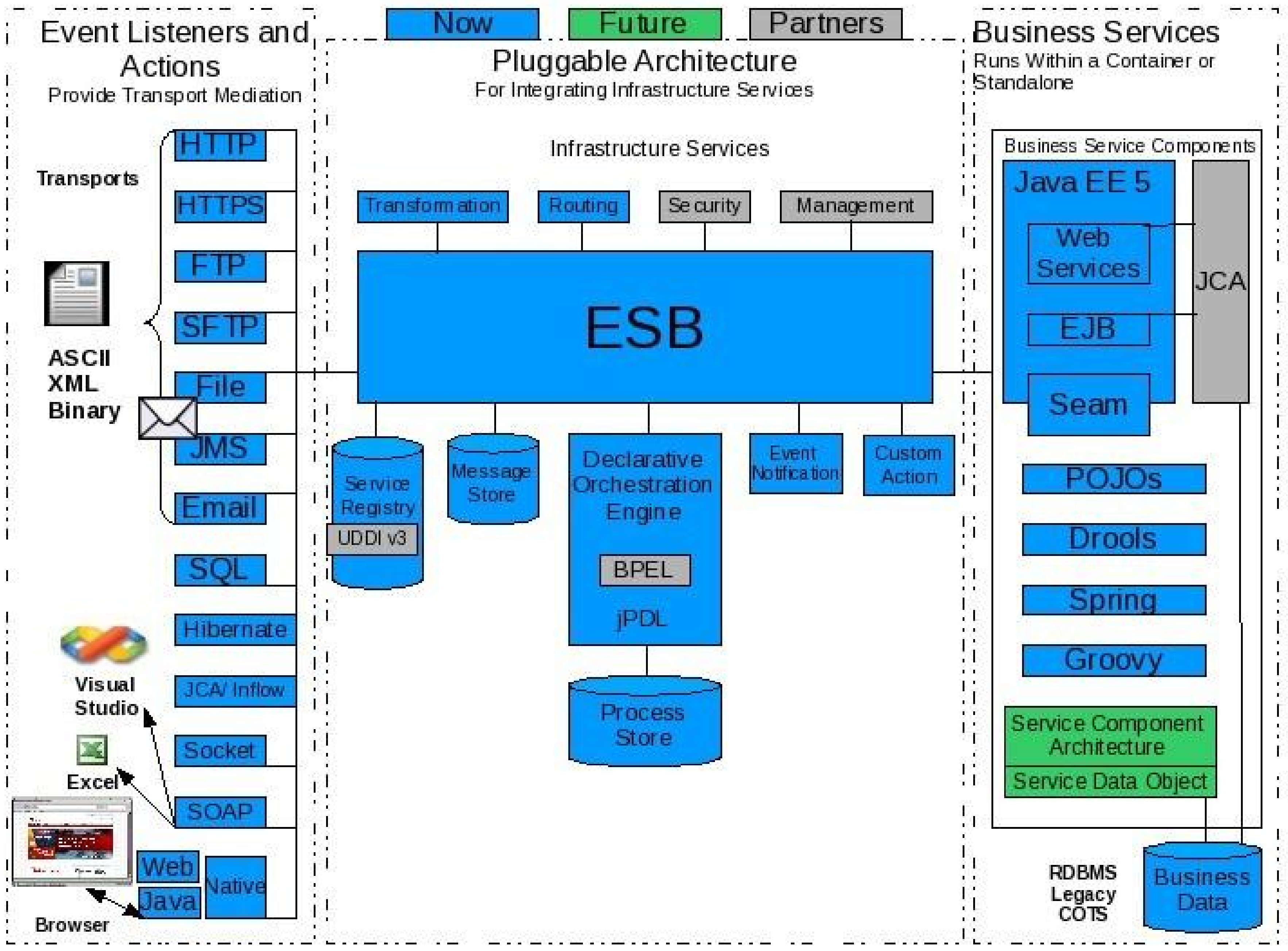
Impact = N



Impact = 1

Enterprise Service Bus

- JBoss ESB (Application Server + ESB)
- Apache Service Mix (Proprietary server)
- ESB Achieves
  - Mediation
  - Routing
  - Marshalling
  - Versioning


Now
Future
Partners
Event Listeners and Actions
Provide Transport Mediation
Transports
HTTP
HTTPS
FTP
SFTP
File
JMS
Email
SQL
Hibernate
JCA/ Inflow
Socket
SOAP
Web
Java
Native
ASCII
XML
Binary
Visual Studio
Excel
Browser
Pluggable Architecture
For Integrating Infrastructure Services
Infrastructure Services
Transformation
Routing
Security
Management
ESB
Service Registry
UDDI v3
Message Store
Declarative Orchestration Engine
BPEL
jPDL
Event Notification
Custom Action
Process Store
Business Services
Runs Within a Container or Standalone
Business Service Components
Java EE 5
Web Services
EJB
Seam
JCA
POJOs
Drools
Spring
Groovy
Service Component Architecture
Service Data Object
RDBMS
Legacy
COTS
Business Data

Apache Camel

- Integration patterns Implementation, subset of ESB
- http://camel.apache.org
- Book: Camel In Action
- User Manual

What Can You Do with Camel

from("file://inputdir/").**to**("file://outputdir")

from("file://inbox/order").**to**("jms:queue:order?jmsMessageType=Text");

from("imaps://imap.gmail.com?username=YOUR_USERNAME@gmail.com&password=YOUR_PASSWORD&delete=false&unseen=true&consumer.delay=60000")
.**process**(new MyMailProcessor());

Configuration

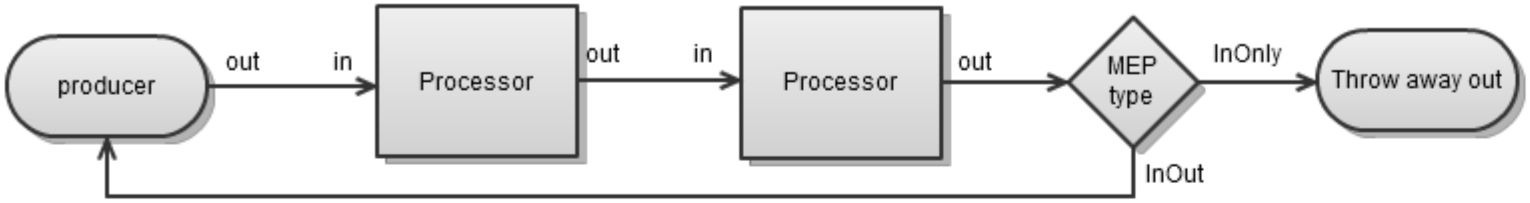
- Via XML
- Directly in Spring Config File
- Via Java DSL

Java DSL

```
CamelContext camelContext = new DefaultCamelContext();
camelContext.addRoutes(new RouteBuilder() {
   @Override
   public void configure() {
        from("file://old-input-orders?recursive=true&flatten=true")
        .to("file://outputDir");
    }
  });
 camelContext.start();
 Thread.sleep(100000);
```

Message vs Exchange

- Message
  - Attachments
  - Headers
  - Body
- Exchange: InMessage OutMessage



Creating an Exchange manually

```
Endpoint inputDir2 = camelContext.getEndpoint("file://old-input-orders");
Exchange fileExchange = inputDir2.createExchange();
fileExchange.getIn().setBody("<order><item>xyz</item></order>");
```

Camel Components

- Have string identifiers
- We have seen “file” component
- Many components:
  - JMS
  - Mail
  - WebServices
  - Database
  - EJB
  - IRC
  - SSH

File Component

- New version: http://camel.apache.org/file2.html
- Read Files
- Writes Files
- By setting a message header “CamelFileName” you can control the resulting file

```
fileExchange.getIn().setHeader("CamelFileName", "myArtificialOrder.txt");
```

Routes and Direct Component

- Java in-memory endpoint, for joining various routes

Route1:

```
from("file://old-input-orders).to(“direct:commonPipeline");
```

Route NameX:

```
from("direct:commonPipeline").id(“NameX”).to("file://outputDir")
```

Processor

- Use Java code to enrich/change the message
- Implementing import org.apache.camel.Processor
- You will get Exchange

```
from("direct:commonPipeline")
.process(new TimeAddingProcessor())
.to(...
```

Changing something in a message

- You must copy everything to Out message! Or otherwise you are loosing headers+attachments

Wrong:

```
public void process(Exchange exchange) throws Exception {
    String request = exchange.getIn().getBody(String.class);
    request = request.replace("<order>", "<order>" + timeElement);
    exchange.getOut().setBody(request);
}
```

Validator Component

- New version: http://camel.apache.org/file2.html
- Validates XML Payload against data schema on classpath (src/main/resources)

```
.to("validator:orderSchema.xsd")
```

JAXB Marshalling

- Object representation of your XML payloads
- jaxb.index

JAXB Class:

```
    @XmlElement(required = true)
     private String created;
```

Camel Route:

```
JaxbDataFormat jaxb = new JaxbDataFormat(OrderBean.class
                    .getPackage().getName());
.unmarshal(jaxb)
...
.marshall(jaxb)
```

Spring Integration

- <camel:camelContext>

```
extends SpringRouteBuilder{
....
@Override
    public void configure() throws Exception {
```

@EndpointInject

- Injects a ProducerTemplate for sending messages to a route

```
@EndpointInject(uri="file://new-orders-input")
private ProducerTemplate newFileEndpoint;

public void sendSomethingThere(){
      newFileEndpoint.sendBody(
  ...
```

spring-ws component

- Hook to an existing Spring WS and process requests

```
from("spring-ws:rootqname:{http://cz.fi.muni.order}orderRequest?
endpointMapping=#endpointMapping")
```

spring-ws component

- org.springframework.ws.wsdl.wsdl11.DefaultWsdl11Definition
- XSD defines the contract for the service (format of Request and Response). WSDL is automatically generated
- You can create your JAXB classes for Request and Response, based on this XSD, again jaxb.index needs to be created

Camel Testing

- Extend AbstractCamelTestNGSpringContextTests
- AdviceWith to mock endpoints

```
retailStoreCamelContext.getRouteDefinition("commonRoute").adviceWith(retailStoreCamelContext,
        new AdviceWithRouteBuilder() {
            @Override
             public void configure() throws Exception {
                    interceptSendToEndpoint("file://outputDir")
                    .skipSendToOriginalEndpoint()
                        .to("mock:sendOrderTestMock");
            }
        });
```

Error Handling

- Using various policies
  - Dead Letter Channel
  - LoggingErrorHandler

```
errorHandler(deadLetterChannel("file://errored"))
.maximumRedeliveries(3).redeliveryDelay(5000));
```