Enterprise Application Integration
Spring Integration

Thomas Kruse

Motivation

Enterprise Integration

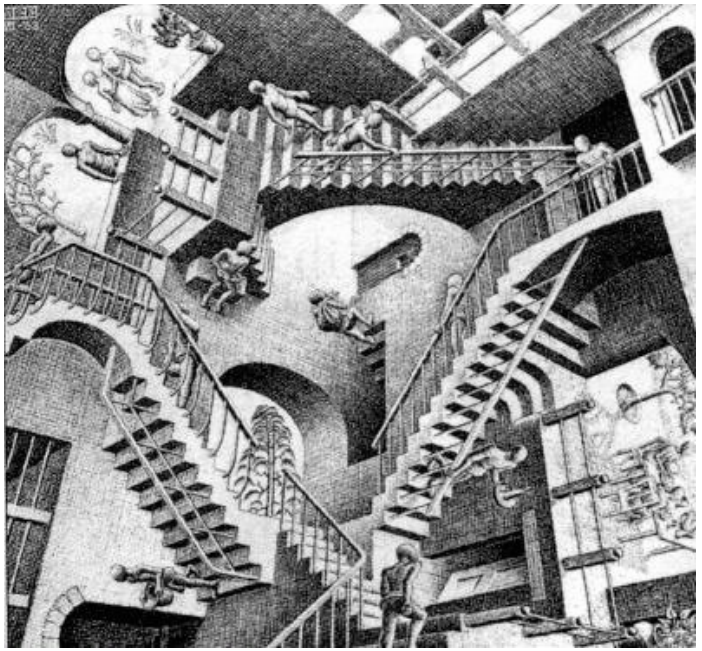
Spring Framework

Spring Integration

Q&A

Anforderungen an Informations Systeme

- Die richtigen Informationen
  - Am richtigen Ort
  - Zur richtigen Zeit
- Ermöglicht Kommunikation
  - Zwischen Menschen
  - Computern und Maschinen
- Basis für Kooperation und Koordination
- Herausforderung: Kontinuierlicher Wandel der Umgebung
- Schwer vorhersehbar bis unplanbar



Beispiel: Airbus A-380

- Airbus A-380 Verzögerung
- Projekt um Jahre verspätet
- Kosten: > 8 Milliarden Euro
- Hintergrund:
  - Firmenkonsortium bis 2001
  - Airbus Standorte verwenden eigene Software
  - Eigene Methoden und Prozesse
  - CAD Software „CATIA“ in Hamburg: 1996, Toulouse: 2001, zusätzlich PMTC
  - Fehleranfällige Konvertierung der Daten nötig

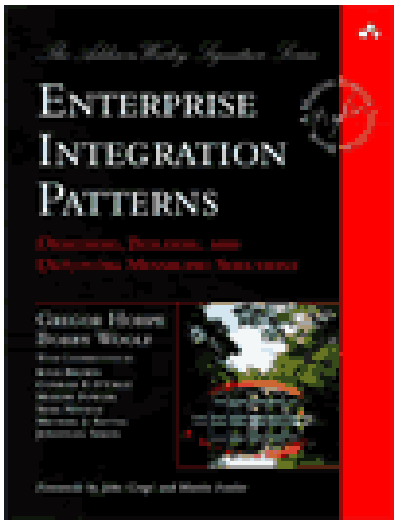
Herausforderung: Heterogene Systeme

- Plattform
  - DBMS, Hardware, Betriebssystem, Netzwerk, …
- Daten
  - Schema Konflikte, Modellierung, Normierung, Daten Typen, Maßeinheiten
- Organisation
  - Verwendung von Drittkomponenten, Abteilungen verwenden jeweils eigene Informationssysteme
  - Ausgestaltung der Prozesse bei verschiedenen Abteilungen und Standorten

- Daten und Dienste miteinander verbinden
  - Firmenfusion oder Zukäufe
  - Neue Dienstleister oder Schnittstellen
  - Neue Geschäftsprozesse
- Legacy Systeme
  - Integration statt Ablösung
- Kapselung von Komplexität
- Integration der Daten/Dienste über Nachrichten

Enterprise Application Integration (EAI)

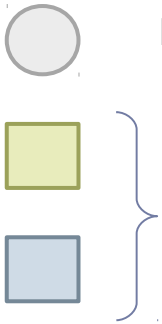
- Patterns
  - Bewährte Lösung für wiederkehrende Probleme
  - Gelten in einem bestimmten Kontext
  - Einheitliches Beschreibungsschema, Sprachunabhängig
  - → vgl. Designpatterns (GoF), Anti-Patterns, ...
- Gregor Hohpe, Bobby Wolf
  - Enterprise Integration Patterns 2003 (Amazon)
  - Homepage: http://www.eaipatterns.com/index.html



- Architekturmuster „Pipes and Filters“
  - Entkoppelt Systeme
    - Änderungen haben lokale Auswirkungen
    - Trennung der Verantwortlichkeiten
  - Pipe: Verbindet Systemkomponenten
  - Filter: Verarbeitungsschritt
- EAI Komponenten: Einfache Bausteine
  - **Nachrichten**
  - **Kanäle** (Pipe)
  - **Endpoints** (Filter)

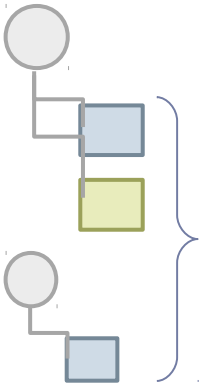
- EAI Patterns: Agnostisch für Technologie
- Beispiele für Integrationstypen
  - Datei Transfer (Netzwerklaufwerk, FTP, ...)
  - Geteilte Datenbank
  - Messaging (JMS, SMTP, POP3, IMAP, ...)
  - Remoting (RPC, REST, SOAP, RMI, …)
- Middleware und Frameworks als Unterstützung
  - Verschiedene Gewichtsklassen
  - TIBCO, Microsoft BizTalk, Websphere ESB
  - OpenSource: OpenESB, Mule, Camel, Spring Integration, …

- Intuitive Notation



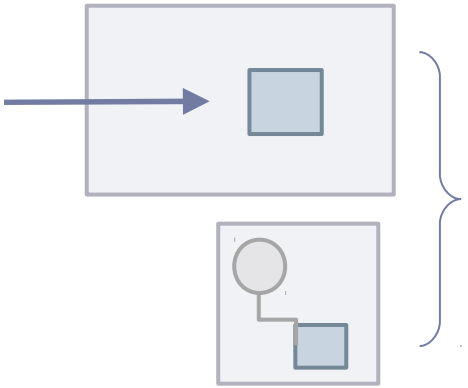
Header

Payload



Nachrichten

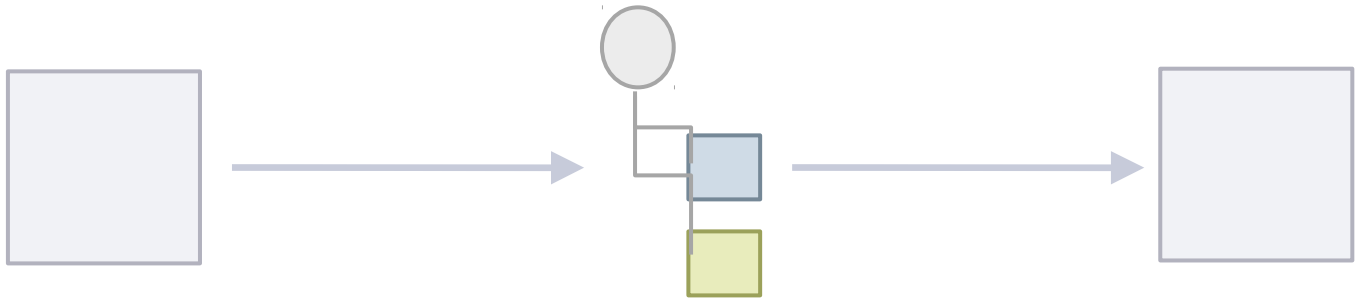
Kanal



Endpoint

- Messaging Realisierungen
  - Synchron
  - Asynchron
  - Point to Point (Queue)
  - Publish Subscriber (Topic)
- Versand
  - Innerhalb eines Systems
  - Über Systemgrenzen hinweg → JMS,
- Messaging Vorteile
  - Reduktion der Komplexität auf handhabbare Bausteine
  - Performance: Asynchrone und parallele Ausführung

- Komponenten werden über Kanäle verbunden



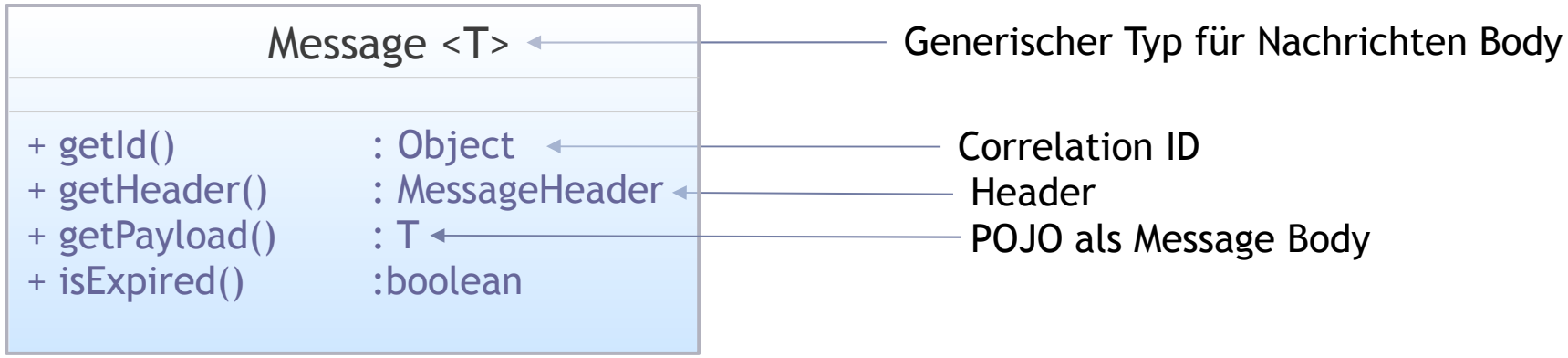
- Eine Nachricht besteht aus
  - Header
  - Payload
- (Payload) Daten werden optional de-/serialisiert für Transport

Spring Integration

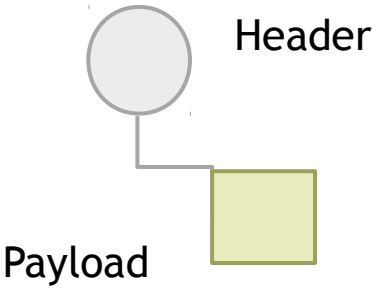
Dezember 2008: 1.0 GA
Mitte 2010: 2.0.0.M7

- Spring Paradigmen für Integrations Domäne
  - Leichtgewichtig, kein Application Server
  - Nicht-invasives POJO Modell
  - Gut Testbar
- Wiederverwendbar Enterprise Integration Pattern Implementierung
- Message und Channel Ansatz auch für Anwendungen ohne Integrationsdomäne
- Spring Integration 2: Spring Expression Language

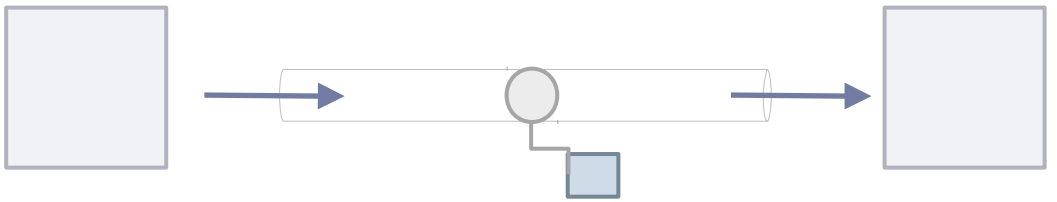
- Spring Message Interface



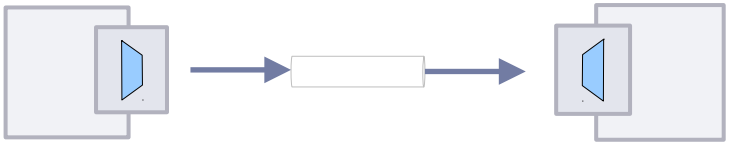
- Innerhalb einer Anwendung: Keine De-/Serialisierung
- Spring verwendet generische Typen für Message Body
- „Typisch Spring“: POJO statt Messages

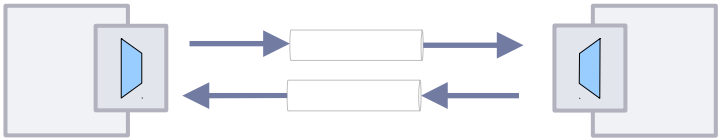


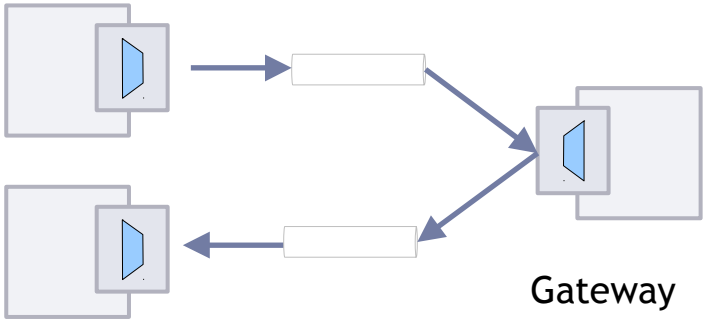
- Channels nicht mit JMS Queue/Topic verwechseln
- Leiten Nachrichten weiter
- Innerhalb einer JVM Instanz
- Channel Typen in Spring Integration (Interfaces)
  - MessageChannel
  - PollableChannel - puffert
  - SubscribeableChannel – kein Puffer
  - Verschiedene Implementierungen
- Channel Interceptor
  - Wire-Tap



- Channel Adapter
  - Unidirektional
  - Inbound: System an Chanbnel
  - Outbound: Channel an System
  - JMS, File, http, Mail
- Messaging Gateway
  - Bidirektional
  - SimpleMessagingGateway
  - GatewayProxyFactoryBean
- Gemischter Einsatz





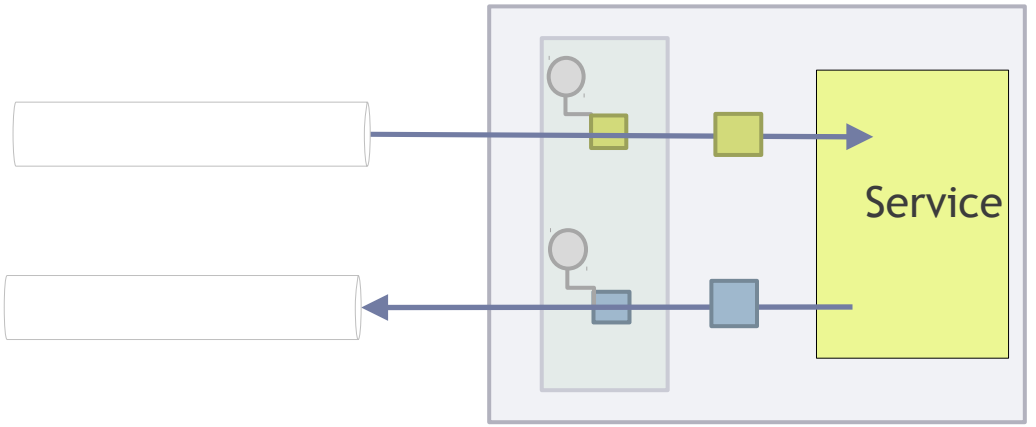


Adapter Konfiguration: XML

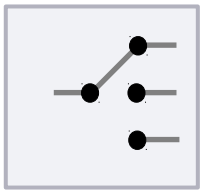
```
<jms-inbound-channel-adapter connection-
factory="jmsConnectionFactory" destination-
name="foo" channel="requests" />


<file-outbound-channel-adapter id="fileWriter"
directory="file:${java.io.tempdir}/out"
channel="processedRequests" />
```

- Service Activator
  - Aufruf von Diensten
  - Antwort ist neue Nachricht
  - Spring: „Method invoking outbound Gateway“
  - Methode auf Objekt aus lokalem Spring Context



- Router
  - Payload
  - Header
  - Recipient

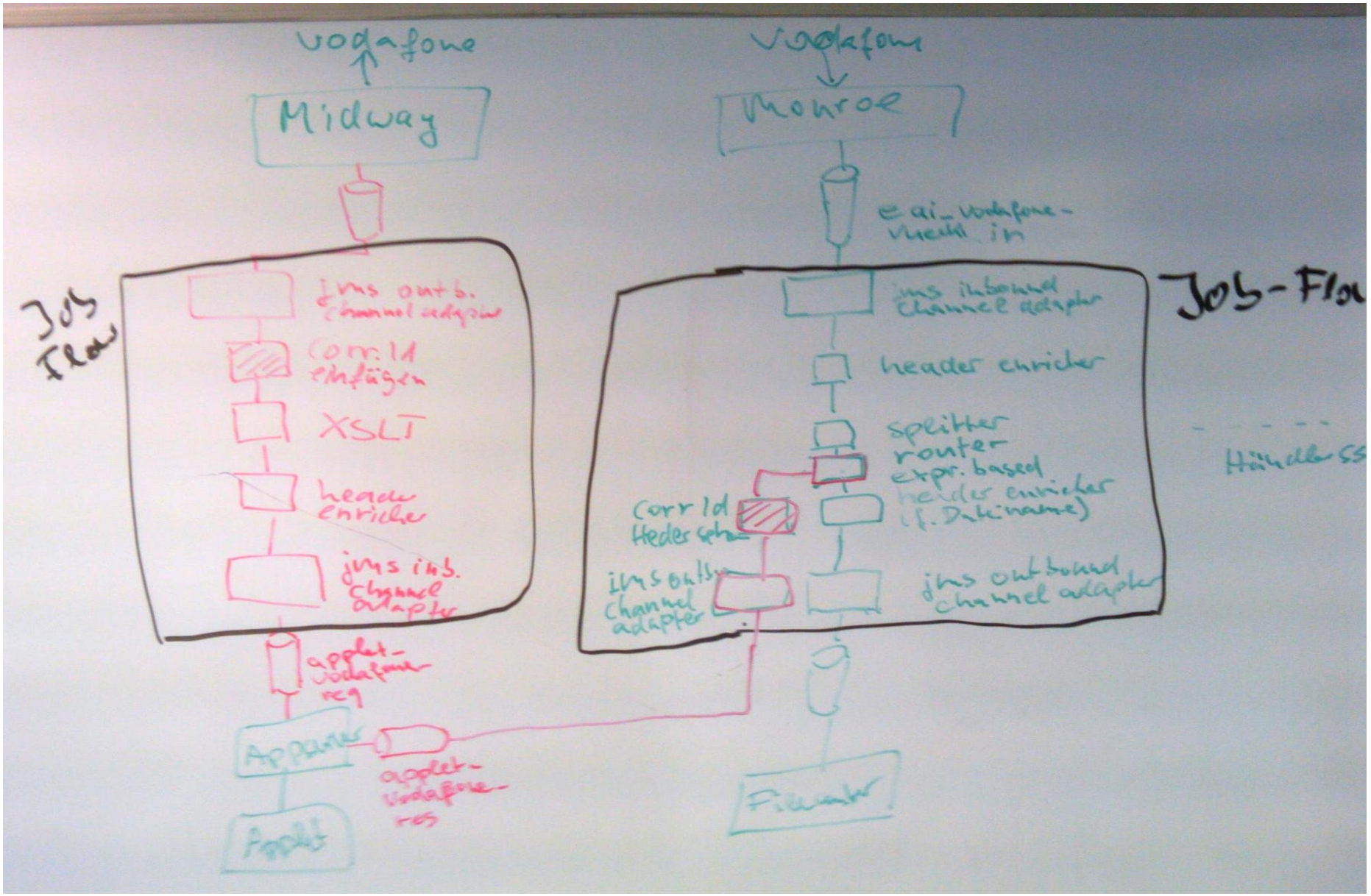


- Transformer (Translator)
  - Ändert Inhalt oder Struktur der Nachricht
  - Vorbereitung auf nachfolgenden Konsumenten

- Filter
  - Entscheidet ob Nachricht weitergeleitet werden soll

- Splitter
  - Nachricht aufteilen
- Aggregator
  - Nachricht aus mehreren Einzelnachrichten erzeugen
- Resequenzer
- Delayer
- Chain
- Bridge

Real Life Example



```
@Splitter

public List<Document> orderItems(Document doc){
    return splitOrder(doc);
}


@Router

public String resolveTransport(Mailing mail) {
    if (mail.isInternational()) {
        return "airMail";
    }
    return "domesticMail";
}
```

- Service Activator

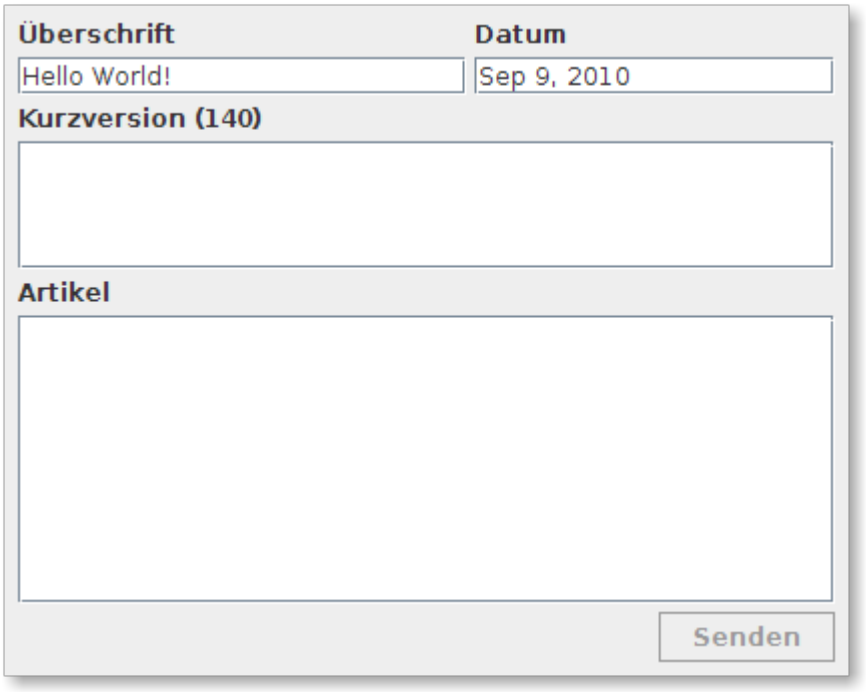
```
@MessageEndpoint
public class FooService {
    @ServiceActivator
    public void processMessage(Message message) {
        …
    }
}
```

- Header Auswertung

```
@ServiceActivator
public void do(String payload, @Header(„foo“)
int fooValue) {
        …
}
```

- Transaktionen
  - Mit Messaging (JMS) realisierbar
  - Ende an Systemgrenzen
- Security
  - Innerhalb jedes Systems zu regeln
- Error Handling: try-catch bei „DirectChannel“ (synchron)
- Sonst (asynchroner Fall/Messaging System):
  - Auswertung Header „errorChannel“
  - Globaler „errorChannel“ (Spring Bean)
  - `ErrorMessageExceptionTypeRouter` verwenden

- Verfügbar über Mercurial Repo der JUG Münster
- Twitter und Mail Anbindung für Nachrichten
- Maven Projekt mit Spring Integration
- Leichter Einstieg
- Blog Artikel Serie: Bald!
- http://www.jug-muenster.de/



- An Systemgrenzen XML als Datenformat
- Keine Objekt-Messages versenden
  - Versionierungsprobleme
  - Java Abhängigkeit
- White Board nutzen!

Fazit

- Vorhandene Frameworks nutzen: Rad nicht neu erfinden!
- Standards verwenden, auch auf Architekturebene
- Integration erlaubt Weiterverwendung bestehender Systeme
- Anbindung neuer Systeme wird vereinfacht
- Modularisierung, lose Koppelung spart Kosten
- Spring ist leichtgewichtige Alternative zu Middleware

Folien und weitere Informationen

http://www.jug-muenster.de/

- http://www.eaipatterns.com/toc.html
- http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/
- http://www.springsource.org/spring-integration
- http://static.springsource.org/spring-integration/docs/2.0.x/spring-integration-reference/htmlsingle/
- http://refcardz.dzone.com/refcardz/enterprise-integration
- http://refcardz.dzone.com/refcardz/soa-patterns
- http://de.wikipedia.org/wiki/Pipes_und_Filter
- http://de.wikipedia.org/wiki/Rube-Goldberg-Maschine

- Enterprise Service Bus (ESB)
  - Plattform für EAI
  - Dolmetscher Rolle
  - Statt Point-To-Point Einsatz von Bus-Architektur
- Bereitgestellte Dienste
  - Routing: Festes und bedingtes Routing
  - Messaging: Transformation
  - Mediation: Adapter, Service-Mapping
  - Ereignisverarbeitung: Korrelierende Ereignisse auf dem Bus
  - Dienstaufruf: Anbieten und Aufrufen von Diensten
- Spring Integration: ESB in a Box

Spring Framework

- Antwort auf Komplexität von J2EE / EJB
- Dependency Injection (IoC)
- Aspekt orientierte Programmierung
  - Security, Transaktionen
- Templates
  - Vereinfachung von APIs
- POJO
  - Nicht invasiv, leicht zu testen
- Inzwischen kompletter Enterprise „Stack“
- Portierungen: Spring .NET

- Service instantiiert Abhängigkeiten

```
Service service = new MyService();
```

- Factory Pattern

```
Service service = factory.constructServiceInstance();
```

- Hollywood-Prinzip: „Don't call us, we call you!“

```
public class Collaborator {
    private Service service;  // interface Service
    public void setService (Service service) {
      this.service = service;
    }
}
```

- Konfiguration über XML oder Annotationen

```
public class Collaborator {
    @Autowired
    private Service service;
}
```

- Spring hat die Entwicklung von Java beeinflusst
- JSR-299, JSR-330 und Guice
  - JSR-299: Contexts and Dependency Injection (CDI): JEE
  - JSR-330: Dependency Injection For Java: Annotationen
  - Google Guice: Nur IoC Container, kein XML
- Spring erlaubt DI für alle POJOs, nicht nur Services
- Nicht nur DI/IoC: Zusätzlich Konfiguration
- Aber: Annotationen nicht JSR-330 kompatibel

- `@Autowired` oder XML
- XML erlaubt Konfiguration der Beans

```
<bean id="caller" class="example.Caller">
    <property name="service" ref="myService"/>
</bean>
<bean id="myService" class="example.MyServImpl"/>
<bean id="mail" class="example.MyMailImpl">
    <property name="host" ref="localhost"/>
</bean>
```

- Standard: Singleton Objekte
  - Weitere Scopes: Prototype, Request, Session
- Fine Tuning: Init- und Destroy-Methoden

- Bootstrapping der Application über Spring
  - Spring Servlet bei Webanwendungen
- Programmatisch: Spring ApplicationContext

```
public class Example {
  public static void main (String args[]) {
    ApplicationContext context = new
    ClassPathXmlApplicationContext("context.xml");

    Service sv = context.getBean("service", Service.class);
  }
}
```

- Unit/Integration Tests

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration({"/applicationContext.xml",
"/applicationContext-test.xml"})
public class TestClass { … }
```

Spring Integration Channel Realisierungen

- DirectChannel
- QueueChannel
- Pub Sub
- Rendezvous
- Executor

Adapter Realisierungen

- JMS
- RMI
- Http Invoker
- HTTP
- File
- Spring Web Services
- Mail
- Spring Application Events

- Note:
  - Blue 3
  - Red 5
  - Green