# KAPITEL 4: ENTERPRISE INTEGRATION PATTERNS

# LERNZIELE

- Erklären was Enterprise Integration Patterns sind
- Beispiele für typische Integrationsaufgaben beschreiben
- Notation zur Darstellung von Integrationsansätzen verwenden, um Konzepte darstellen zu können
- Notation verwenden, um damit Ansätze umzusetzen

# 4.1 MOTIVATION

*Interesting applications rarely live in isolation. Whether your sales application must interface with your inventory application, your procurement application must connect to an auction site, or your PDA's PIM must synchronize with the corporate calendar server [...]*

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

# HERAUSFORDERUNGEN BEI VERTEILTEN ANWENDUNGEN

*Netze sind unzuverlässig*

- Integrationslösungen müssen Daten von einer Anwendung zu einer anderen über Netzwerke transportieren
- Im Vergleich zu einem Prozess, der auf einem einzelnen Computer abläuft, muss die verteilte Datenverarbeitung auf möglicher Probleme vorbereitet sein
- Gegebenfalls sind zu integrierende Systeme geografisch stark getrennt und die Daten zwischen ihnen müssen über verschiedene Telefonleitungen, LAN-Segmente, Router, Switches, öffentliche Netze und / oder Satellitenverbindungen übertragen werden
- Jeder Schritt kann zu Verzögerungen oder Unterbrechungen führen.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

*Netzwerke sind langsam*

- Das Senden von Daten über ein Netzwerk ist um mehrere Größenordnungen langsamer als ein lokaler Methodenaufruf
- Wenn verteilte Lösung auf dieselbe Weise entworfen werden, wie es bei einer einzelnen Anwendung der Fall wäre, könnte das Auswirkungen auf die Leistung haben

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

*Zwei beliebige Anwendungen sind unterschiedlich*

- Integrationslösungen müssen Informationen zwischen Systemen übermitteln, die unterschiedliche Programmiersprachen, Betriebsplattformen und Datenformate verwenden
- Eine Integrationslösung muss in der Lage sein, mit all diesen unterschiedlichen Technologien zusammenzuarbeiten

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

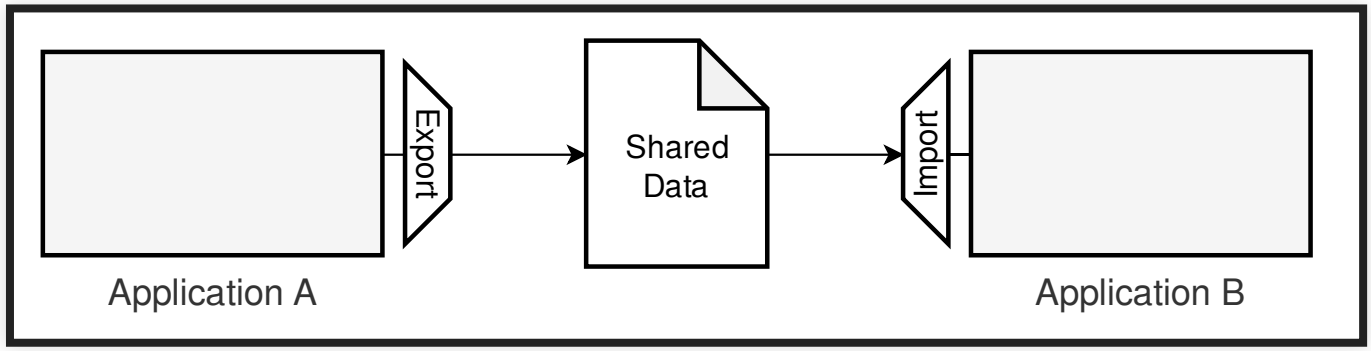
*Änderungen sind unvermeidlich*

- Anwendungen ändern sich mit der Zeit
- Eine Integrationslösung muss mit den Änderungen bei den Anwendungen, die sie verbindet, Schritt halten
- Integrationslösungen können leicht in einen Lawineneffekt von Änderungen geraten - wenn sich ein System ändert, können alle anderen Systeme davon betroffen sein
- Eine Integrationslösung muss die Abhängigkeiten von einem System zum anderen minimieren, indem sie eine lose Kopplung zwischen den Anwendungen herstellt.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

# TYPISCHE LÖSUNGEN

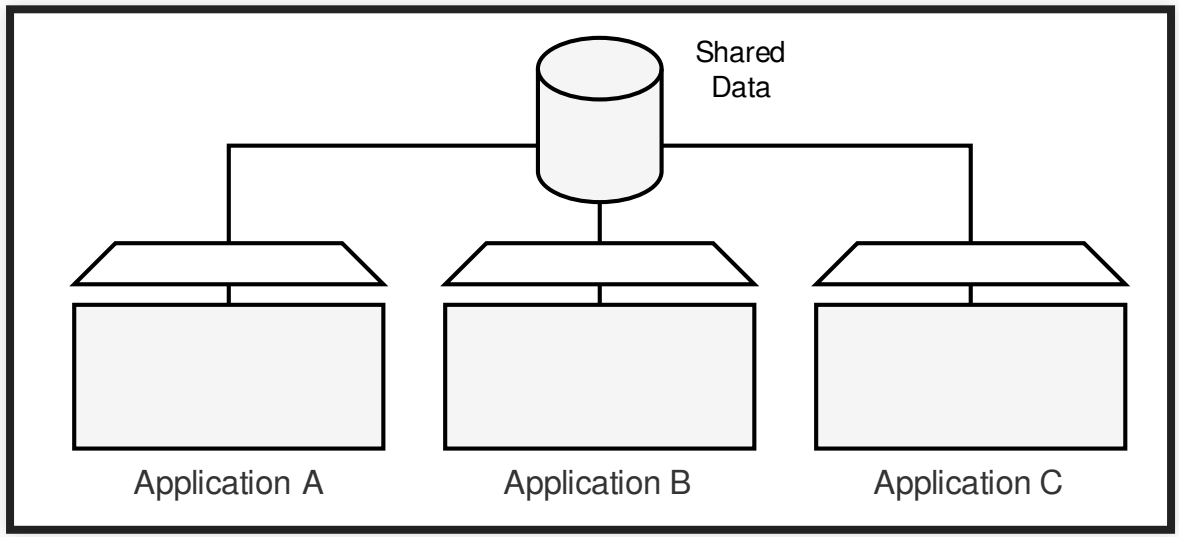
*File Transfer*




Eine Anwendung schreibt eine Datei, die eine andere später liest. Die Anwendungen müssen sich auf den Dateinamen und den Speicherort, das Format der Datei, den Zeitpunkt des Schreibens und Lesens und die Person, die die Datei löscht, einigen.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

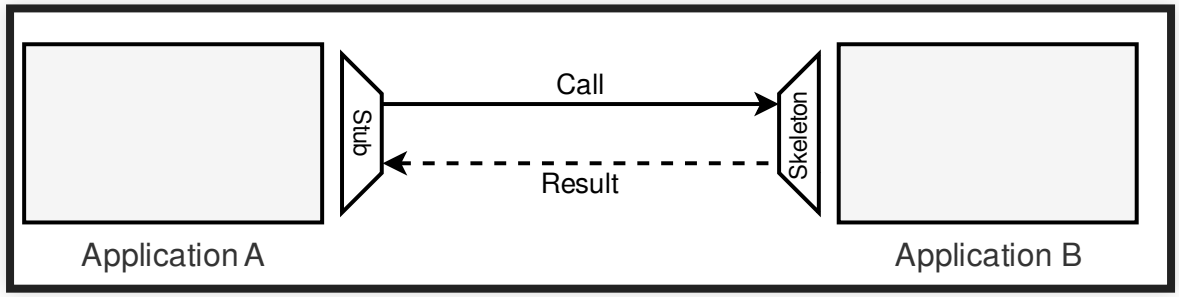
## *Gemeinsame Datenbank*




Mehrere Anwendungen nutzen dasselbe Datenbankschema, das sich in einer einzigen physischen Datenbank befindet. Da es keine doppelte Datenspeicherung gibt, müssen keine Daten von einer Anwendung zur anderen übertragen werden.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

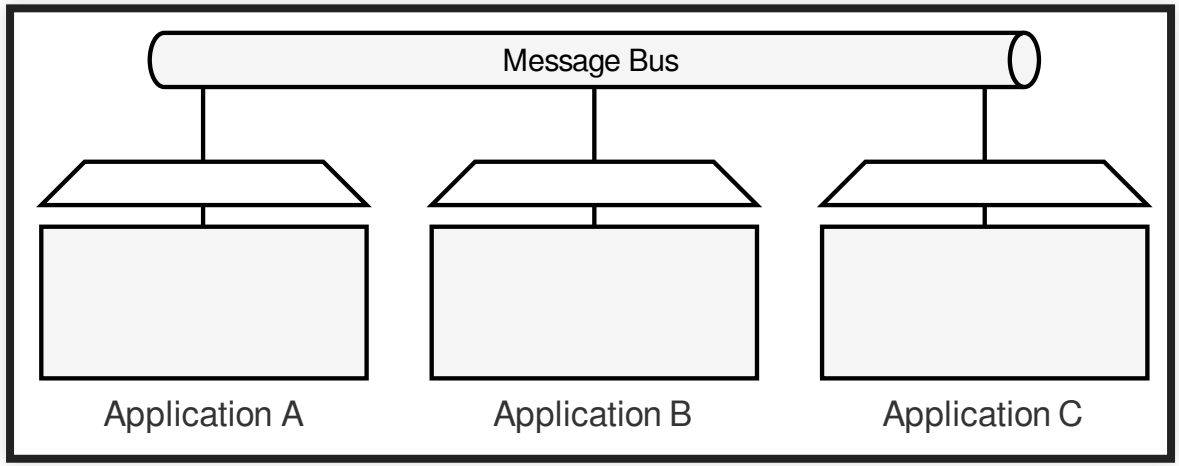
*Remote Procedure Invocation*




Eine Anwendung stellt einen Teil ihrer Funktionalität zur Verfügung, so dass andere Anwendungen über eine Remote Procedure auf sie zugreifen können. Die Kommunikation erfolgt in Echtzeit und synchron.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

## *Messaging*



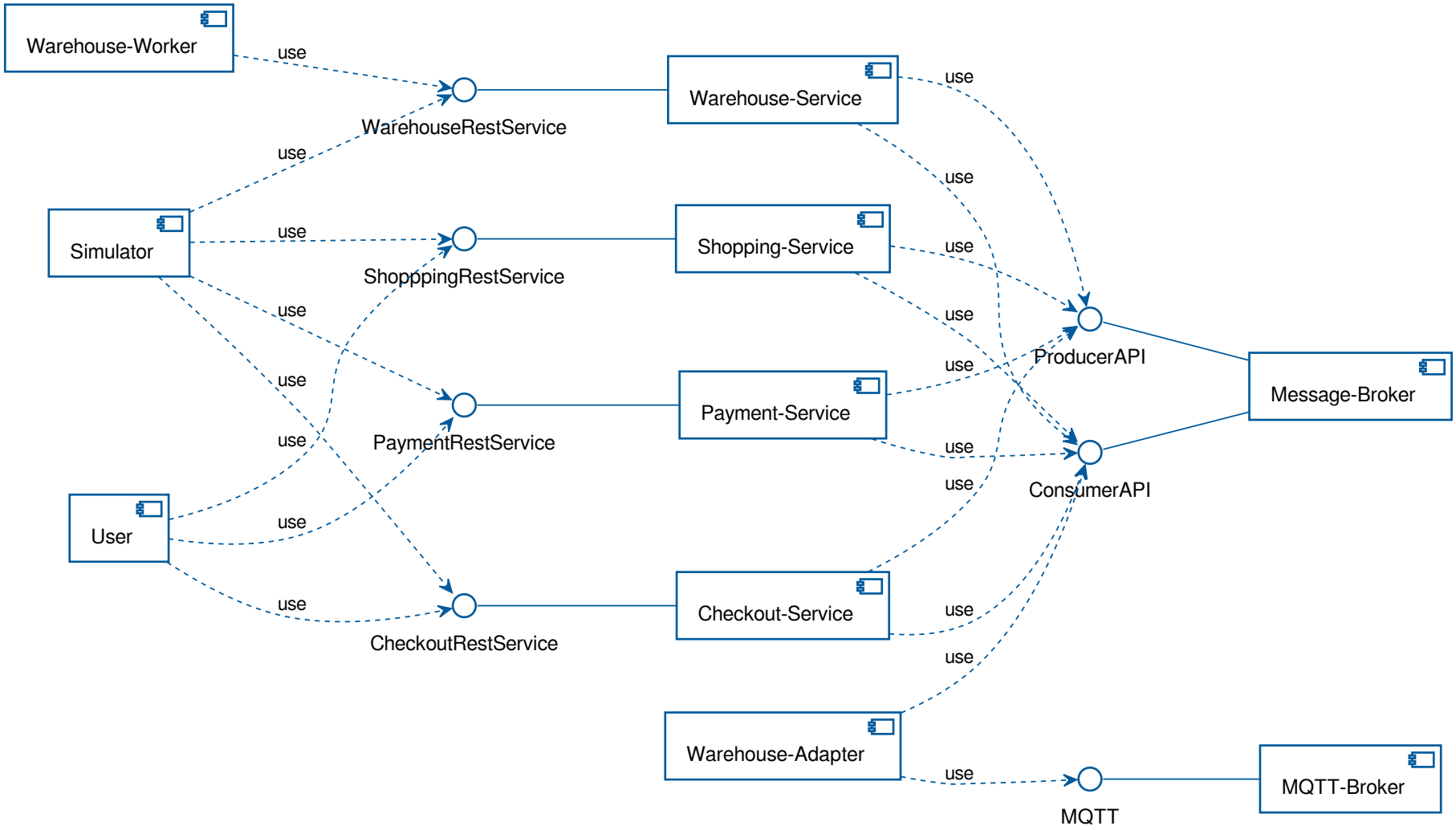

Eine Anwendung veröffentlicht eine Nachricht in einem gemeinsamen Nachrichtenkanal. Andere Anwendungen können die Nachricht zu einem späteren Zeitpunkt aus diesem Kanal lesen.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

Messaging als eine typische Lösung für Herausforderungen bei der Integration von Geschäftsanwendungen.

Dieses Kapitel soll häufige Patterns im Kontext der Integration verschiedener Dienste über nachrichtenbasierte Ansätze einführen.

# Frage: Wo kommt es zu Integrationsaufgaben im Shop-Beispiel?



*Integrationsaufgaben in der Architektur (u.a.):*

- HTTP-Endpunkte an denen Client-Anwendungen mit dem Backend interagieren
- Services die Ereignisse Consumieren und Produzieren
- Adapter die mehrere Systeme miteinander verknüpfen (IoT-Setting mit Queues in Kafka)

# 4.2 BASICS

## Enterprise Integration Patterns

- Orientieren sich an Büchern wie *Design Patterns*, *Pattern Oriented Software Architecture*, *Core J2EE Patterns*, and *Patterns of Enterprise Application Architecture*
- Bieten eine Menge von Patterns organisiert in einer Pattern-Language
- Jedes Pattern steht für eine Entscheidung, die getroffen werden muss, und für die Überlegungen, die zu dieser Entscheidung führen
- Eine Pattern-Language ist ein Netz zusammenhängender Patterns, wobei jedes Pattern zu anderen führt und durch den Entscheidungsprozess leitet

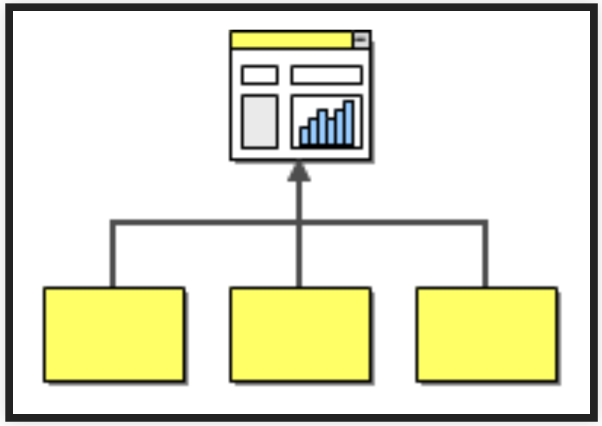
Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

## Beispiele

- Information Portals
- Data Replication
- Shared Business Functions
- Service-Oriented Architectures
- Distributed Business Processes
- Business-to-Business Integration

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

*Informationsportale*

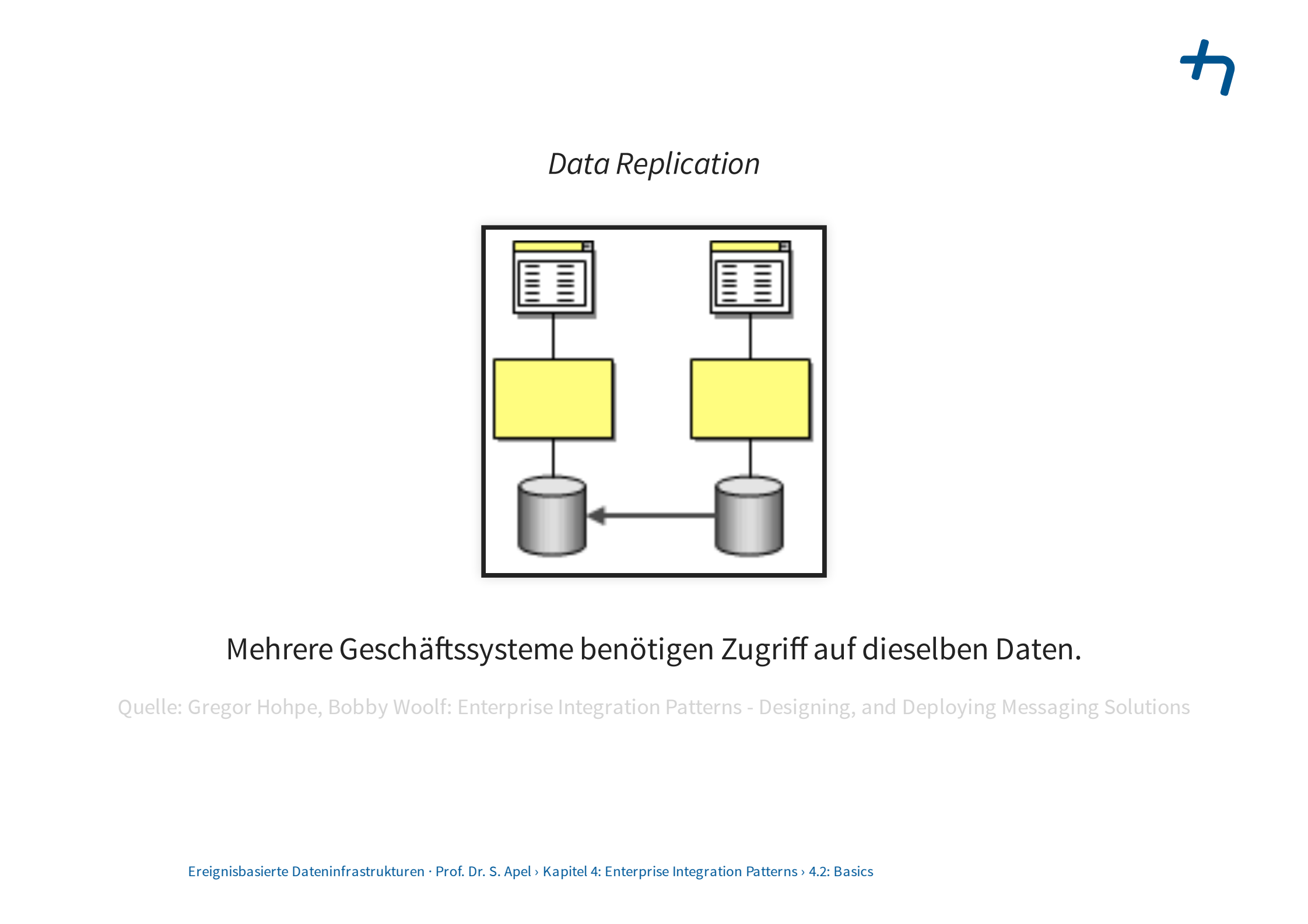


Informationsportale fassen Informationen aus mehreren Quellen in einer einzigen Anzeige zusammen, damit der Benutzer nicht auf mehrere Systeme einzeln zugreifen muss.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

Nutzer müssen meist auf mehr als ein System zugreifen, um eine bestimmte Frage zu beantworten oder um eine einzelne Geschäftsfunktion auszuführen. Um beispielsweise den Status einer Bestellung zu überprüfen, muss ein Kundendienstmitarbeiter möglicherweise auf das Auftragsverwaltungssystem zugreifen und sich außerdem bei dem System anmelden, das die über das Internet aufgegebenen Bestellungen verwaltet.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions




Mehrere Geschäftssysteme benötigen Zugriff auf dieselben Daten.

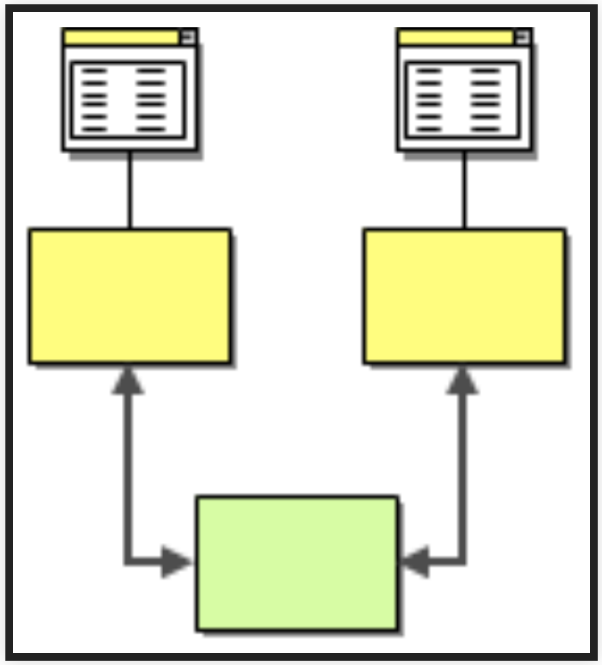
Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

Beispielsweise kann die Adresse eines Kunden im Kundenbetreuungssystem (wenn der Kunde anruft, um sie zu ändern), im Buchhaltungssystem (zur Berechnung der Umsatzsteuer), im Versandsystem (zum Etikettieren der Sendung) und im Rechnungssystem (zum Versenden einer Rechnung) verwendet werden. Viele dieser Systeme haben ihre eigenen Datenspeicher, um kundenbezogene Informationen zu speichern.

Wenn ein Kunde kontakt aufnimmt, um seine Adresse zu ändern, müssen alle diese Systeme ihre Kopie der Adresse des Kunden ändern. Dies kann durch eine Integrationsstrategie erreicht werden, die auf Datenreplikation basiert.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

## *Shared Functions*



Redundante Funktionen zu implementieren ist nach Möglichkeit zu vermeiden. Eine Funktion als gemeinsam nutzbare Geschäftsfunktion darzustellen erlaubt die Verwendung dieser Funktion in verschiedenen Systemen.

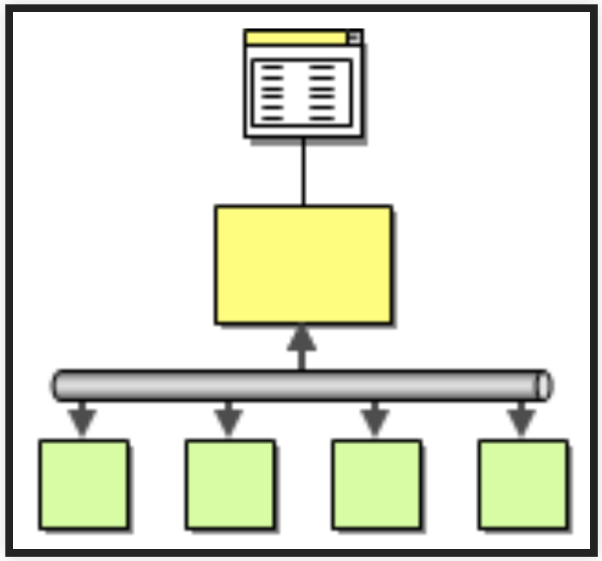
Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

Mehrere Systeme müssen möglicherweise prüfen, ob eine Sozialversicherungsnummer gültig ist, ob die Adresse mit der angegebenen Postleitzahl übereinstimmt oder ob ein bestimmter Artikel auf Lager ist. Es ist sinnvoll, diese Funktionen als gemeinsam genutzte Geschäftsfunktion darzustellen, die einmal implementiert wird und anderen Systemen zur Verfügung steht.

Eine gemeinsam genutzte Geschäftsfunktion kann einige der gleichen Anforderungen erfüllen wie die Datenreplikation. So könnte beispielsweise eine Geschäftsfunktion mit dem Namen "Kundenadresse abrufen" implementiert werden, die es anderen Systemen ermöglicht, die Adresse des Kunden anzufordern, wenn sie benötigt wird, anstatt immer eine redundante Kopie zu speichern.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

*Service Oriented Architecture*



Gemeinsam genutzte Geschäftsfunktionen werden oft als Dienste bezeichnet und in einer Architektur organisiert.

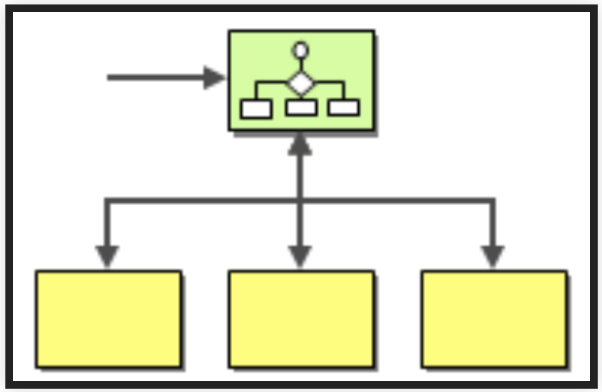
Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

Ein Dienst universell verfügbar und reagiert auf Anfragen von "Dienstnutzern". Sobald ein Unternehmen eine Sammlung nützlicher Dienste zusammenstellt, wird die Verwaltung der Dienste zu einer wichtigen Aufgabe.

Zunächst einmal benötigen Anwendungen eine Art Dienstverzeichnis, eine zentrale Liste aller verfügbaren Dienste. Zweitens muss jeder Dienst seine Schnittstelle so beschreiben, dass eine Anwendung einen Kommunikationsvertrag mit dem Dienst "aushandeln" kann. Diese beiden Funktionen, das Auffinden von Diensten und das Aushandeln von Verträgen, sind die Schlüsselelemente, die eine serviceorientierte Architektur ausmachen.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

*Distributed Business Process*



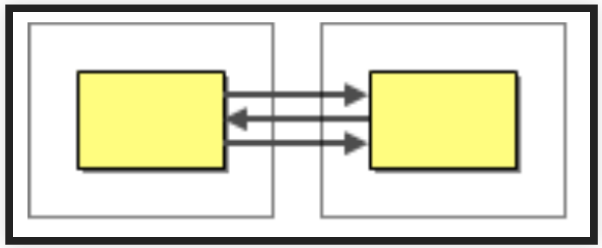
Geschäftsvorgang sind oft über viele verschiedene Systeme verteilt ist.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

Das Beispiel zur "Bestellung aufgeben" hat gezeigt, dass eine einfache Geschäftsfunktion leicht mehrere Systeme berühren kann. In den meisten Fällen sind alle relevanten Funktionen in bestehende Anwendungen integriert. Was fehlt, ist die Koordination zwischen den Anwendungen. Daher können wir eine Geschäftsprozessmanagementkomponente hinzufügen, die die Ausführung einer Geschäftsfunktion über mehrere bestehende Systeme hinweg verwaltet.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

*Business-to-Business Integration*



Geschäftsfunktionen können für externen Anbietern oder Geschäftspartnern zur Verfügung stehen.

Ein Versandunternehmen kann z.B. seinen Kunden einen Dienst zur Berechnung der Versandkosten oder zur Verfolgung von Sendungen anbieten. Alternativ kann ein Unternehmen einen externen Anbieter nutzen, um Umsatzsteuersätze zu berechnen. Zur Integration vieler Geschäftspartner und derer Systeme in einer elektronischen "Unterhaltung", sind standardisierte Datenformate von entscheidender Bedeutung.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

# 4.3 PATTERNS

## Pattern-Struktur

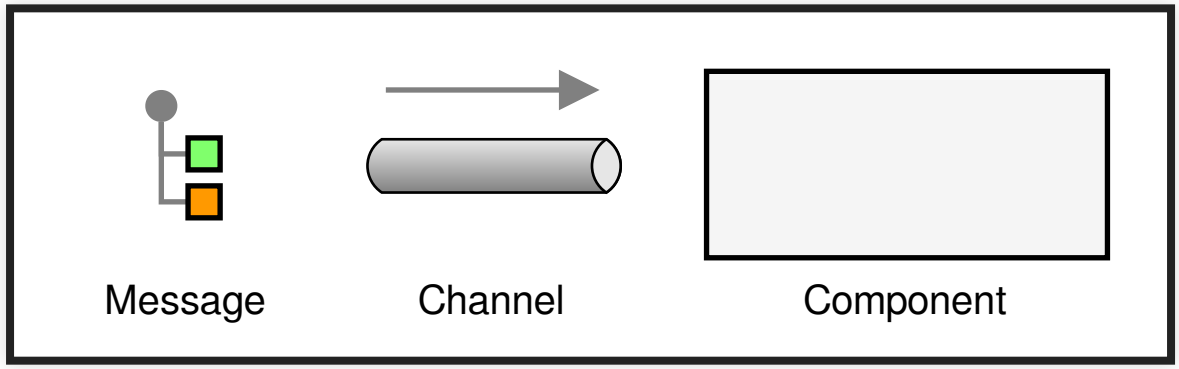
- Name: Bezeichner für das Pattern, der angibt, was das Pattern tut.
- Icon: Viele Pattern sind zusätzlich zum Pattern-Name mit einem Icon verbunden
- Kontext: Erklärt, woran möglicherweise gearbeitet wird, so dass Sie wahrscheinlich auf das Problem stoßen, das dieses Muster löst
- Problem: Schwierigkeit erklärt, mit konfrontiert wird, ausgedrückt als eine Frage, die Sie sich stellen und die dieses Muster löst
- Forces: Die Forces erforschen die Randbedingungen, die die Lösung des Problems erschweren

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

- Solution: Vorlage, die erklärt, was getan werden muss, um das Problem zu lösen
- Sketch: Veranschaulichung der Lösung, um das Wesentliche des Musters zu verstehen
- Ergebnisse: Erläutert die Einzelheiten der Anwendung der Lösung und die Lösung der Randbedingungen
- Next: Verweis auf andere Patterns
- Sidebars: Detailliertere technische Fragen oder Variationen des Musters
- Examples: Anwendungsbeispiele des Patterns

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

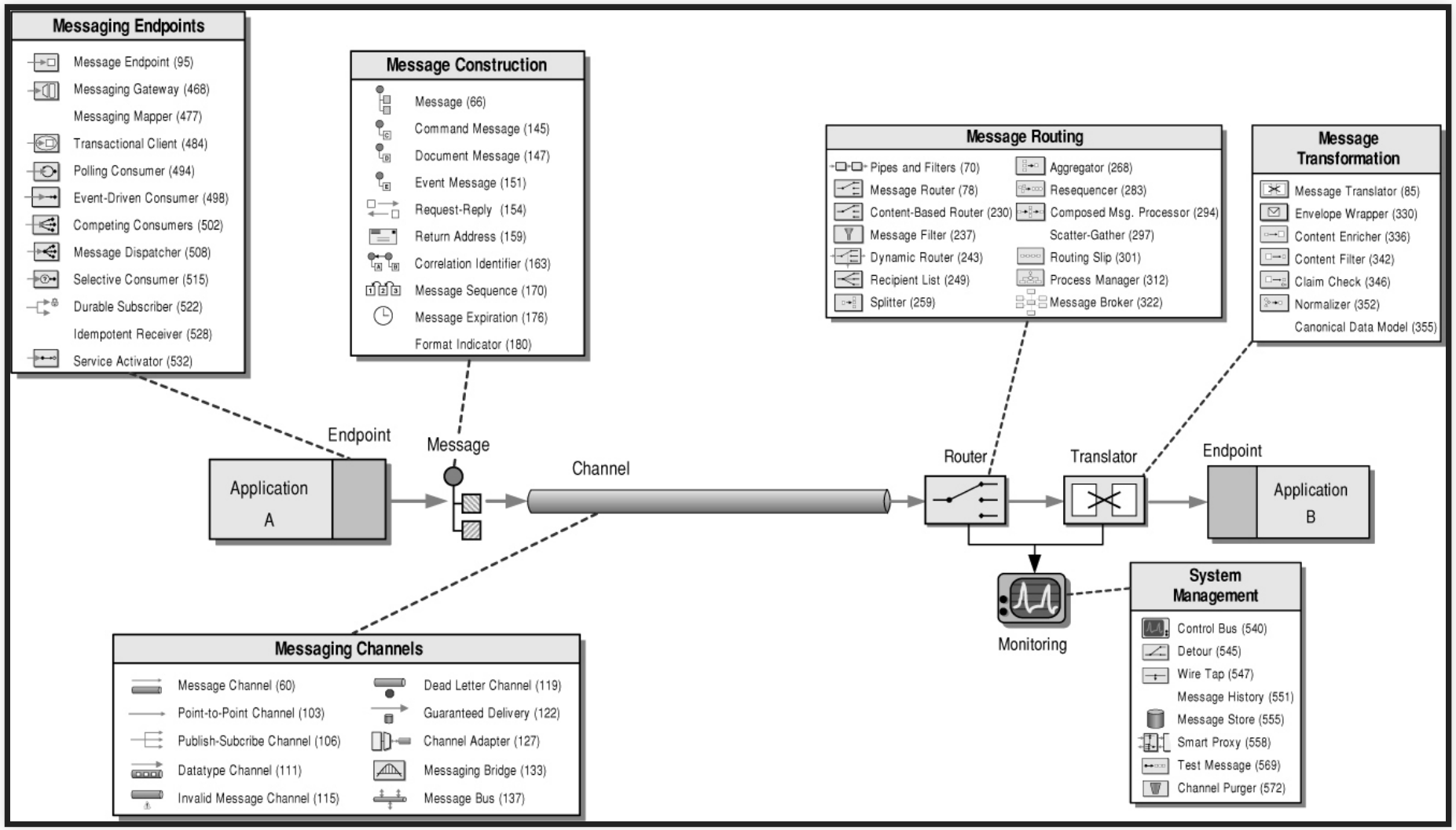
# Notation




Die Notation von Gregor Hohpe und Bobby Woolf erlaubt es Integrationspatterns visuell darzustellen. Jedem Pattern wird dabei eine Symbolik zugeordnet, mithilfe der Pfeile wird der Fluss von Nachrichten dargestellt. Im Beispiel wird eine Nachricht über einen Kanal an eine Komponente transportiert.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

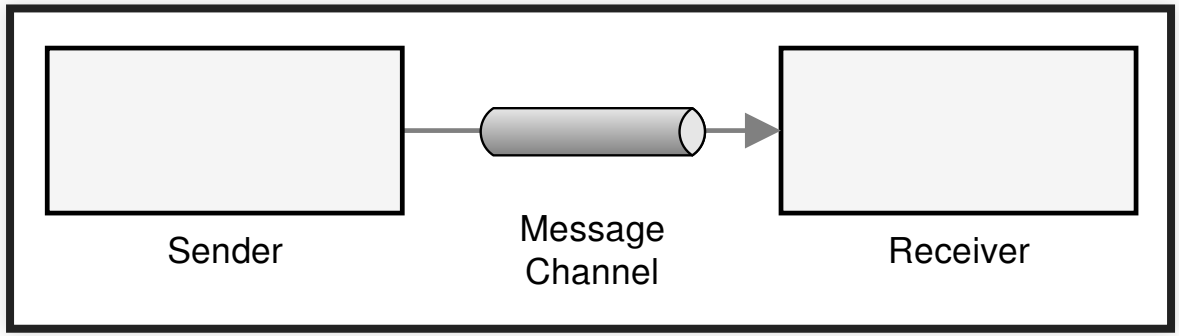
# Übersicht




Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

# MESSAGE CHANNEL

*Wie kommuniziert eine Anwendung mit einer anderen über Messaging?*

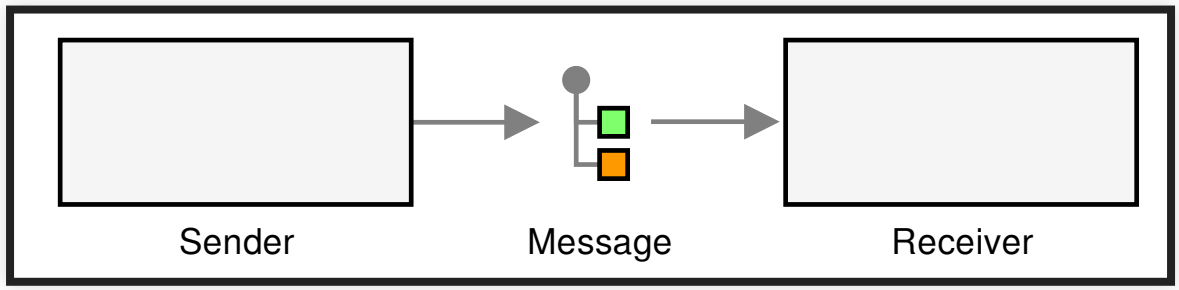



Die Anwendungen über einen Nachrichtenkanal verbinden, wobei eine Anwendung Informationen in den Kanal schreibt und die andere Anwendung diese Informationen aus dem Kanal liest.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

# MESSAGE

*Wie können zwei Anwendungen, die über einen Nachrichtenkanal verbunden sind, eine Information austauschen?*




Die Informationen sind in eine Nachricht zu verpacken, einen Datensatz, den das Nachrichtensystem über einen Nachrichtenkanal übertragen kann.

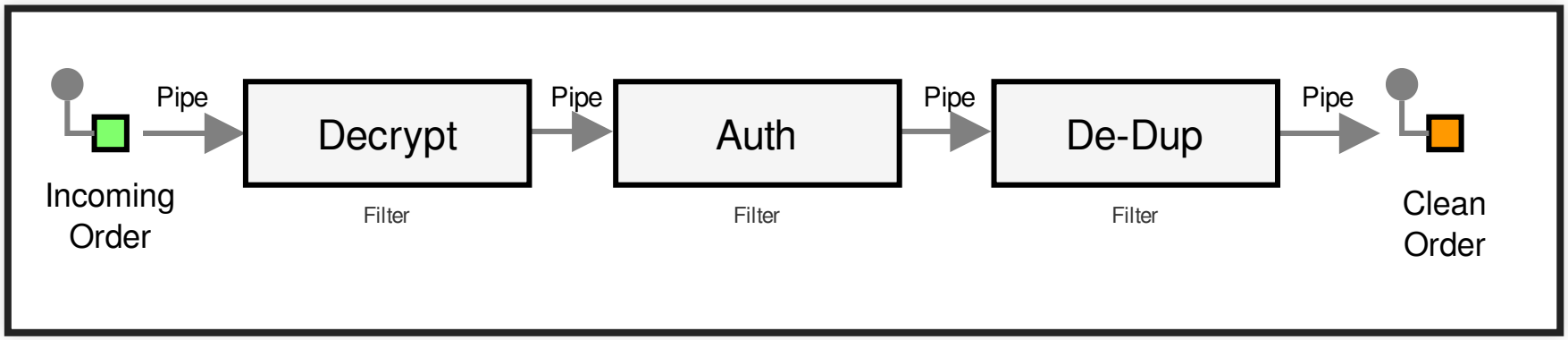
Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

Eine Nachricht besteht aus zwei grundlegenden Teilen:

1. **Header**: Vom Nachrichtensystem verwendete Informationen, die die zu übertragenden Daten, ihren Ursprung, ihr Ziel usw. beschreiben.
2. **Body**: Die zu übertragenden Daten, die im Allgemeinen vom Nachrichtensystem ignoriert und einfach so übertragen werden, wie sie sind.

# PIPES AND FILTERS

*Wie können komplexe Verarbeitungen an einer Nachricht vorgenommen und dabei Unabhängigkeit und Flexibilität gewahrt werden?*



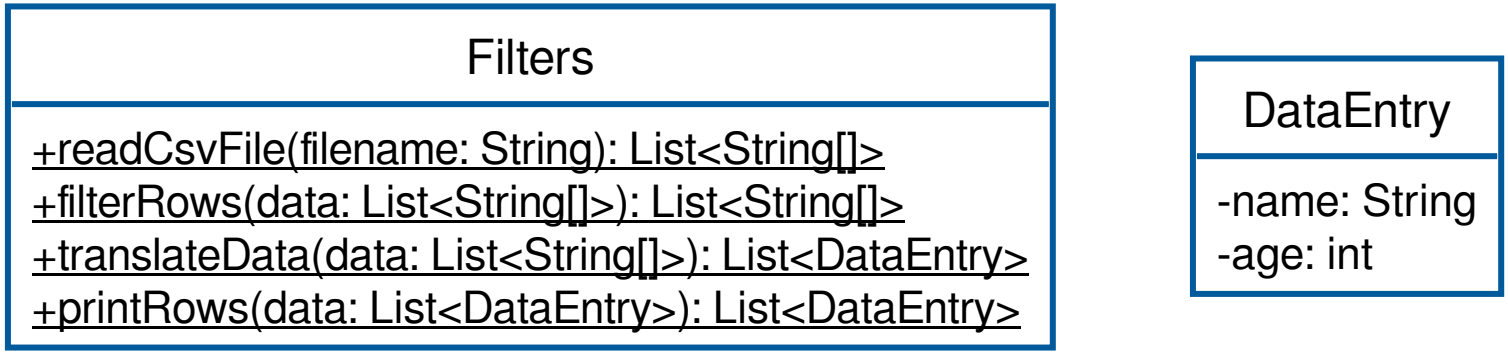

Der Architekturstil Pipes und Filter kann verwendet werden, um eine größere Verarbeitungsaufgabe in eine Abfolge kleinerer, unabhängiger Verarbeitungsschritte (Filter) zu unterteilen, die durch Kanäle (Pipes) verbunden sind.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

- Jeder Filter verfügt über eine einfache Schnittstelle: Er empfängt Nachrichten über die Inbound-Pipe, verarbeitet die Nachricht und veröffentlicht die Ergebnisse in der Outbound-Pipe.
- Die Pipe verbindet einen Filter mit dem nächsten und sendet die Ausgangsnachrichten von einem Filter zum nächsten.
- Da alle Komponenten dieselbe externe Schnittstelle verwenden, können sie zu verschiedenen Lösungen zusammengesetzt werden, indem die Komponenten mit verschiedenen Pipes verbunden werden.
- Wir können neue Filter hinzufügen, bestehende weglassen oder sie in einer neuen Reihenfolge anordnen, ohne die Filter selbst ändern zu müssen.
- In der Grundform hat jede Filterkomponente einen Eingangsanschluss und einen Ausgangsanschluss.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

*Wie kann das Muster mit imperativer Programmierung umgesetzt werden?*




Das Beispiel zeigt eine Sammlung von Verarbeitungsmethoden mit Signaturen, die Listen annehmen oder zurückgeben: Auslesen der Datei, filtern nach einen bestimmten Kriterium, Übersetzen in ein anderes Format und Ausgabe in der Konsole.

Würde man dieses Beispiel verwenden, könnte dies wie folgt aussehen:

```
List<String[]> data;
List<DataEntry> entries;

data = readCsvFile(filename);
data = filterRows(data);
entries = translateData(data);
entries = printRows(entries);
```

Entsprechende Methoden realisieren mit Schleifen die entsprechende Logik. Beim Aufruf werden die Ergebnisse der jeweiligen Filter-Methoden an die nächste Methode weitergeleitet.

*Wie kann das Muster mit Reactive Programming umgesetzt werden?*

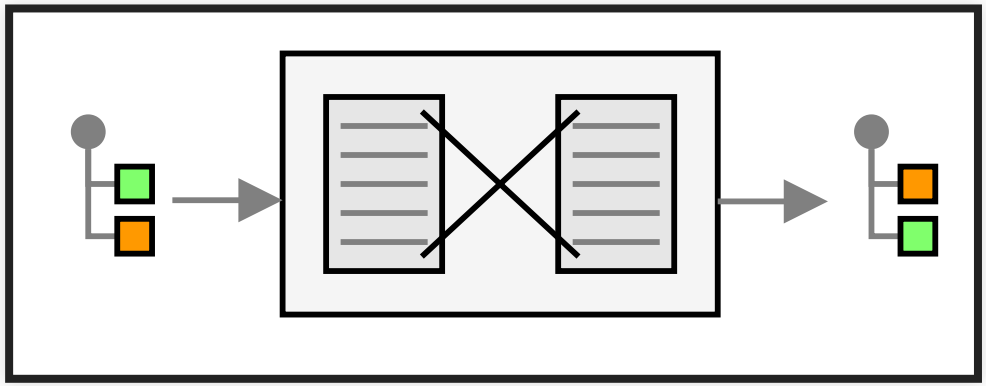
```
Flux.using(
    () -> new BufferedReader(new FileReader(filename)),
    (reader) -> Flux.fromStream(reader.lines()),
    (reader) -> Mono.fromRunnable(() -> {
            try {
                reader.close();
            } catch (IOException e) {}
        }))
  .map(line -> line.split(","))
  .filter(row -> row.length >= 2)
  .map(row -> new DataEntry(row[0], Integer.parseInt(row[1])))
  .subscribe(System.out::println);
```

Mittels `Flux.using` wird ein Flux erzeugt der über eine Initialisierung-, Ereigniserzeugung- und Abschluss-Lambda beschrieben wird.

Anschließend folgen die Operatoren zur Realisierung der Verarbeitungspipeline.

# MESSAGE TRANSLATOR

*Wie können Systeme, die unterschiedliche Datenformate verwenden, mittels Messaging miteinander kommunizieren?*

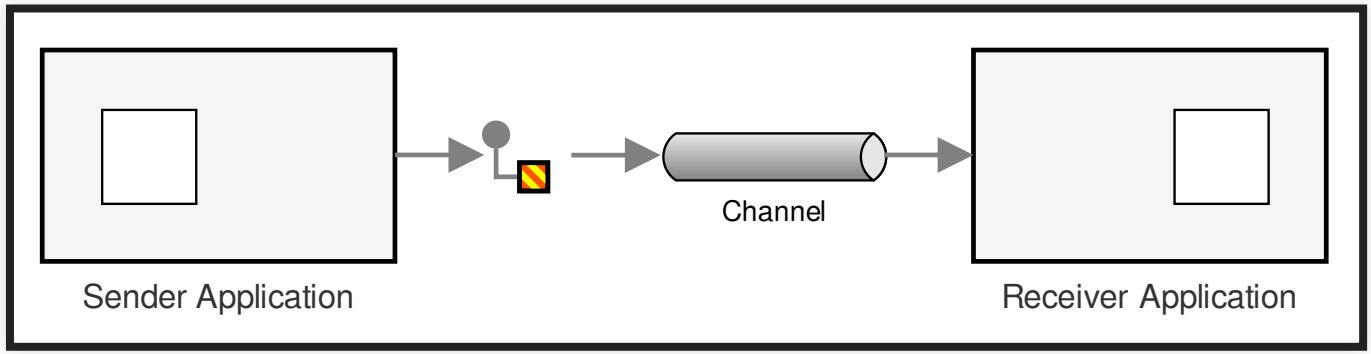


Verwendet werden kann ein spezieller Filter, welcher einen Message Translator darstellt, zwischen anderen Filtern oder Anwendungen, um ein Datenformat in ein anderes zu übersetzen.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

# MESSAGE ENDPOINT

*Wie stellt eine Anwendung eine Verbindung zu einem Messaging-Kanal her, um Nachrichten zu senden und zu empfangen?*

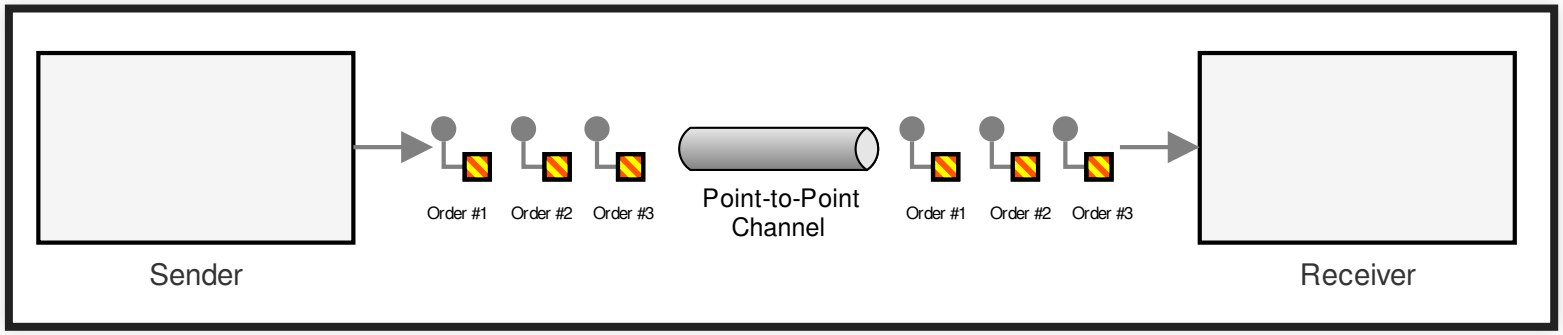



Eine Anwendung ist mit einem Messaging-Kanal über einen Message-Endpunkt zu verbinden, den die Anwendung dann zum Senden oder Empfangen von Nachrichten verwenden kann.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

# POINT-TO-POINT CHANNEL

*Wie kann der Anrufer sicher sein, dass genau ein Empfänger das Dokument erhält oder den Anruf ausführt?*




Die Nachricht ist über einen Punkt-zu-Punkt-Kanal zu versenden, der sicherstellt, dass nur ein Empfänger eine bestimmte Nachricht erhält.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

*Wie verhalten sich hier MQTT-Broker und Apache Kafka?*

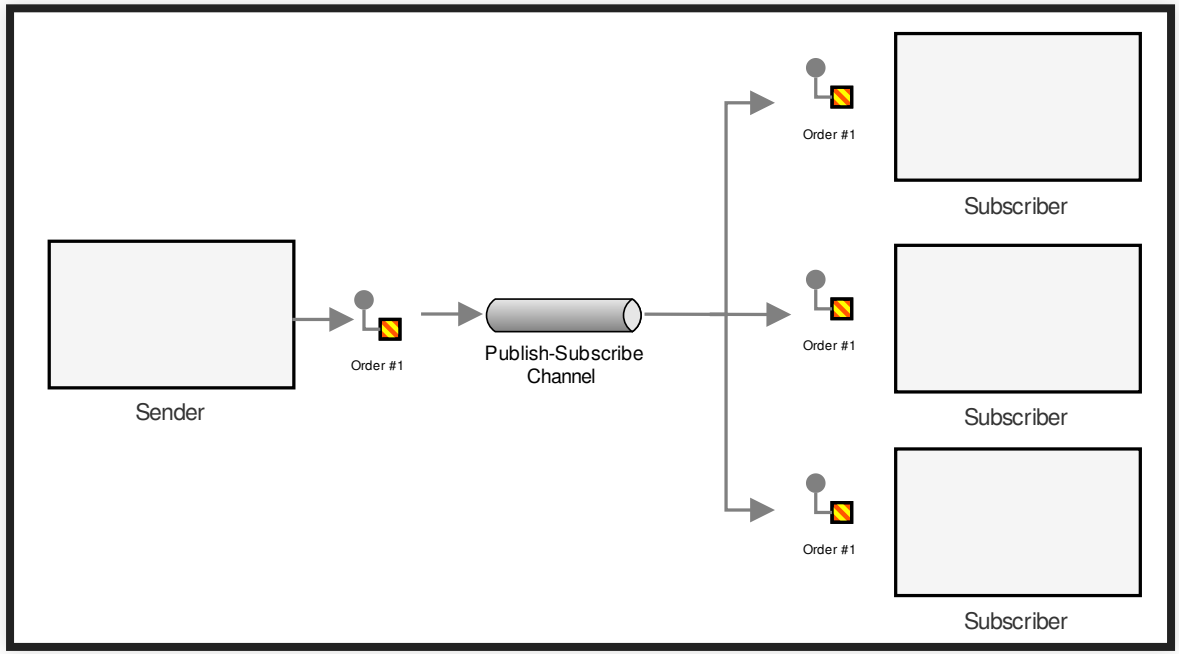
MQTT-Broker folgen einem Publish-Subscribe-Ansatz, es ist schwierig sicherzustellen, dass hier nur ein Empfänger die Nachrichten verarbeitet. In Apache Kafka erhält grundsätzlich nur ein Consumer aus einer Gruppe eine Nachricht. Wenn alle Consumer der selbe Gruppe angehören, ergibt sich damit ein Point-to-Point-Verhalten.

*Wie kann dies mit Reactive Programming erreicht werden?*

Datenströme müssen als Unicast-Kanäle erzeugt werden, damit es nur eine Subscription geben kann und damit auch nur ein Consumer.

# PUBLISH-SUBSCRIBE CHANNEL

*Wie kann der Absender ein Ereignis an alle interessierten Empfänger senden?*




Ereignis werden über einen Publish-Subscribe-Channel versendet, der jedem Empfänger eine Kopie eines bestimmten Ereignisses liefert.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

*Wie verhalten sich hier MQTT-Broker und Apache Kafka?*

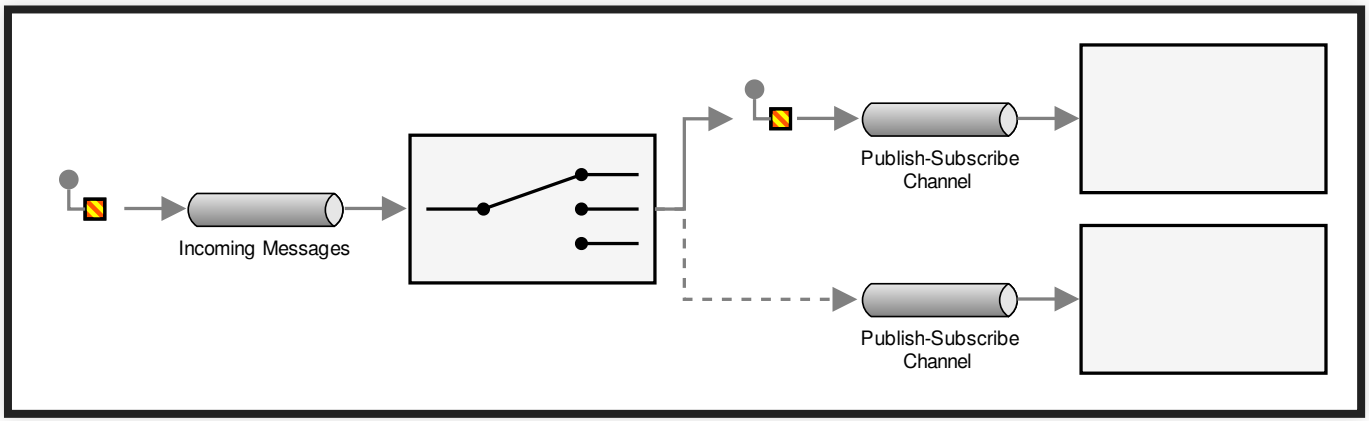
MQTT-Broker folgen diesem Ansatz bereits, jeder Client erhält für seine Subscriptions alle Nachrichten. In Apache Kafka ist mit unterschiedlichen Gruppen zu arbeiten, damit Nachrichten mehrfach konsumiert werden.

*Wie kann dies mit Reactive Programming erreicht werden?*

Datenströme müssen als Multicast-Kanäle erzeugt werden, damit es nur eine Subscription geben kann und damit auch nur ein Consumer.

# MESSAGE ROUTER

*Wie kann man einzelne Verarbeitungsschritte entkoppeln, so dass Nachrichten in Abhängigkeit von einer Reihe von Bedingungen an verschiedene Filter weitergeleitet werden können?*

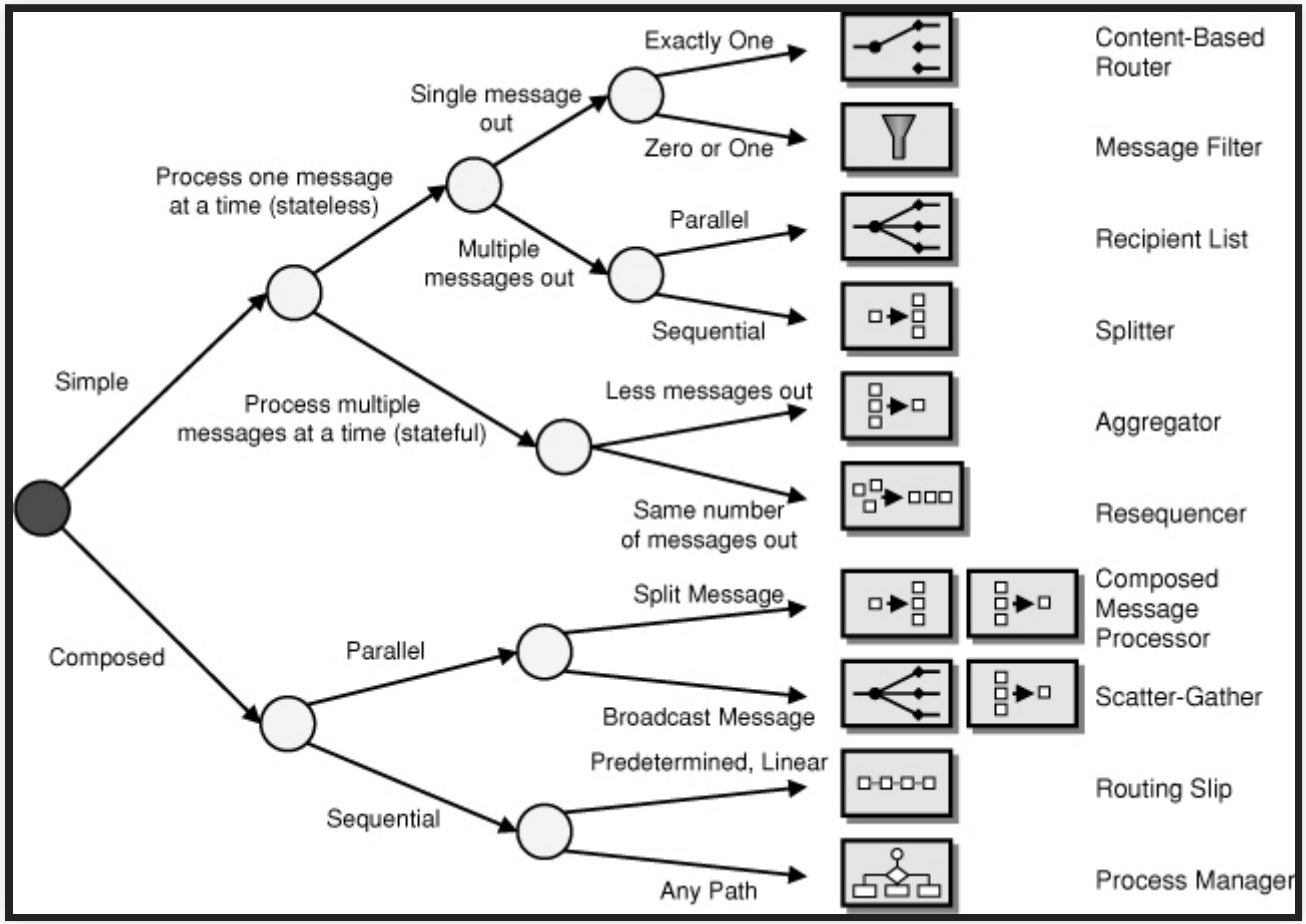



Ein spezieller Verarbeitungsschritt kann verwendet werden, der Nachrichten-Router, der eine Nachricht aus einem Nachrichtenkanal konsumiert und sie in Abhängigkeit von einer Reihe von Bedingungen in einem anderen Nachrichtenkanal wieder veröffentlicht.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

- Der Nachrichten-Router unterscheidet sich vom Grundkonzept der Pipes and Filters dadurch, dass er mit mehreren Ausgangskanälen verbunden ist
- Dank der Architektur von Pipes and Filters wissen die Komponenten, die den Message Router umgeben, jedoch nichts von der Existenz eines Message Routers: Sie konsumieren einfach Nachrichten von einem Kanal und geben sie an einen anderen weiter.
- Eine definierende Eigenschaft des Message Routers ist, dass er den Inhalt der Nachricht nicht verändert; er kümmert sich nur um das Ziel der Nachricht.

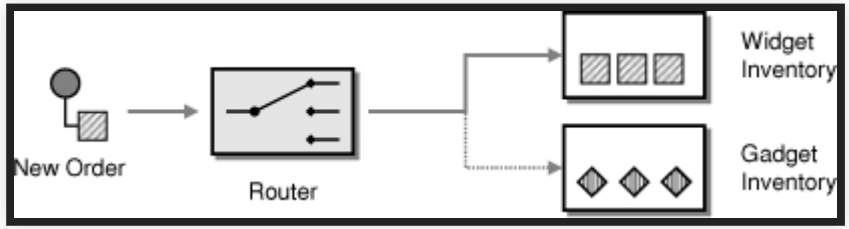
Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions




Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

## Content-Based Router

*Wie gehen wir mit einer Situation um, in der die Implementierung einer einzigen logischen Funktion über mehrere physische Systeme verteilt ist?*

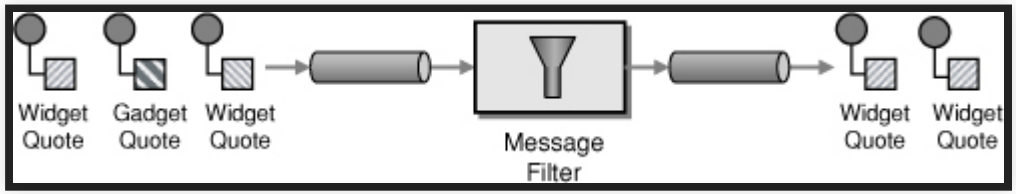



Inhaltsbasierten Router werden verwendet, um jede Nachricht auf der Grundlage ihres Inhalts an den richtigen Empfänger weiterzuleiten.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

# Message Filter

*Wie kann eine Komponente den Empfang uninteressanter Nachrichten vermeiden?*

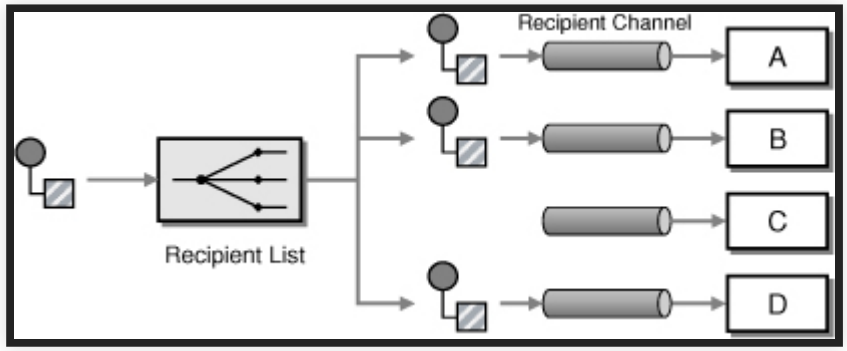



Spezielle Art von Nachrichten-Router wird verwendet, ein Nachrichten-Filter, um unerwünschte Nachrichten aus einem Kanal auf der Grundlage einer Reihe von Kriterien zu eliminieren.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

# Recipient List

*Wie leitet man eine Nachricht an eine dynamische Liste von Empfängern weiter?*

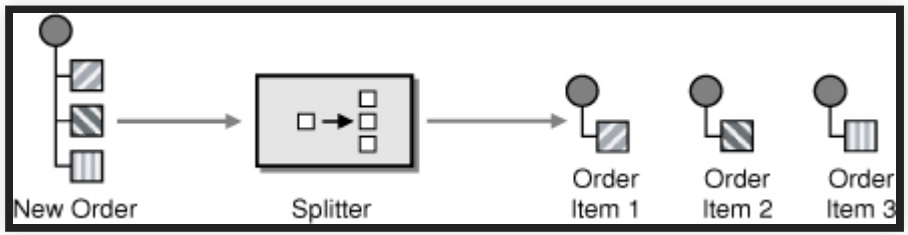



Für jeden Empfänger einen Kanal definieren. Eine Empfängerliste verwenden, um eine eingehende Nachricht zu prüfen, die Liste der gewünschten Empfänger zu bestimmen und die Nachricht an alle Kanäle weiterzuleiten, die mit den Empfängern in der Liste verbunden sind.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

# Splitter

*Wie können wir eine Nachricht verarbeiten, wenn sie mehrere Elemente enthält, von denen jedes auf eine andere Weise verarbeitet werden muss?*

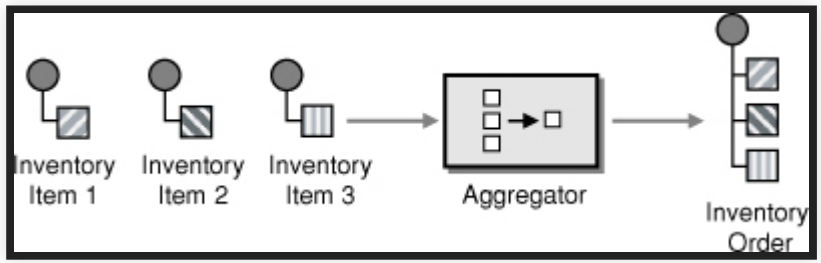



Einen Splitter verwenden, um die zusammengesetzte Nachricht in eine Reihe von Einzelnachrichten aufzuteilen, die jeweils Daten zu einem Element enthalten.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

# Aggregator

*Wie kombinieren wir die Ergebnisse einzelner, aber zusammenhängender Nachrichten, so dass sie als Ganzes verarbeitet werden können?*




Einen zustandsabhängigen Filter verwenden, einen Aggregator, um einzelne Nachrichten zu sammeln und zu speichern, bis er einen vollständigen Satz zusammengehöriger Nachrichten erhält. Dann veröffentlicht der Aggregator eine einzelne Nachricht, die aus den einzelnen Nachrichten destilliert wurde.

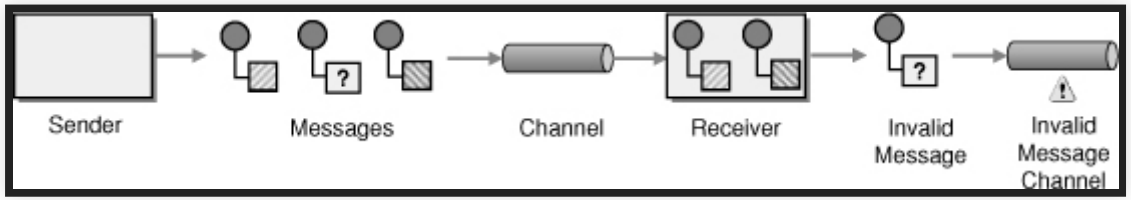
Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

*Weitere Router*

- Resquencer
- Compos. Msg. Processor
- Scatter-Gather
- Routing Slip
- Process Manager

# INVALID MESSAGE CHANNEL

*Wie kann ein Nachrichtenempfänger mit dem Empfang einer unsinnigen Nachricht umgehen?*

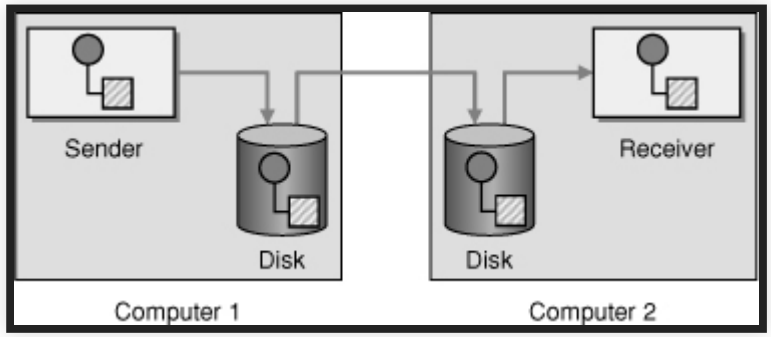



Der Empfänger sollte die unzulässige Nachricht in einen Kanal für ungültige Nachrichten verschieben, einen speziellen Kanal für Nachrichten, die von ihren Empfängern nicht verarbeitet werden konnten.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

# GUARANTEED DELIVERY

*Wie kann der Absender sicherstellen, dass eine Nachricht zugestellt wird, auch wenn das Nachrichtensystem ausfällt?*

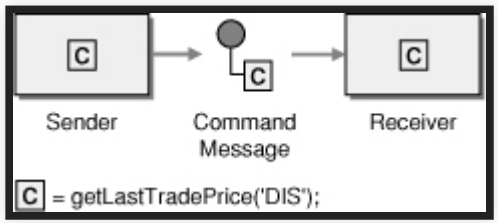



Guaranteed Delivery wird verwendet, um sicherzustellen, dass Nachrichten auch bei einem Ausfall des Nachrichtensystems nicht verloren gehen.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

# COMMAND MESSAGE

*Wie kann eine Nachricht verwendet werden, um eine Prozedur in einer anderen Anwendung aufzurufen?*

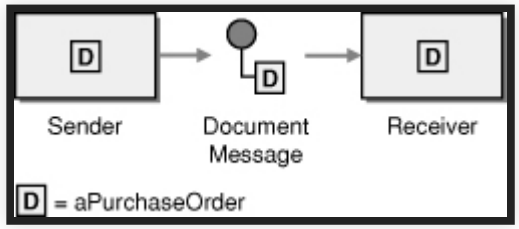



Eine Befehlsnachricht verwenden, um eine Prozedur in einer anderen Anwendung zuverlässig aufzurufen.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

# DOCUMENT MESSAGE

Wie kann Messaging zur Übertragung von Daten zwischen Anwendungen genutzt werden?

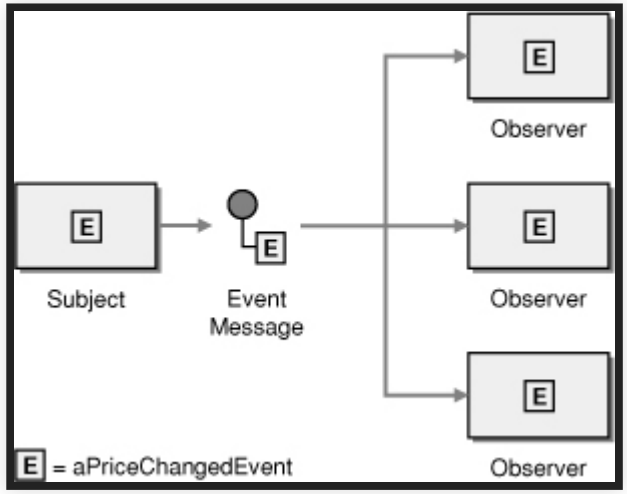



Eine Dokumentnachricht verwenden, um eine Datenstruktur zuverlässig zwischen Anwendungen zu übertragen.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

# EVENT MESSAGE

Wie kann Messaging verwendet werden, um Ereignisse von einer Anwendung zu einer anderen zu übertragen?

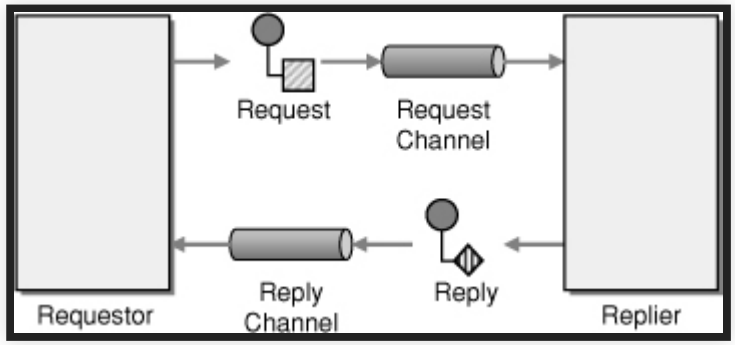



Ereignismeldung können für eine zuverlässige, asynchrone Ereignisbenachrichtigung zwischen Anwendungen verwendet werden.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

# REQUEST-REPLY

*Wenn eine Anwendung eine Nachricht sendet, wie kann sie dann eine Antwort vom Empfänger erhalten?*

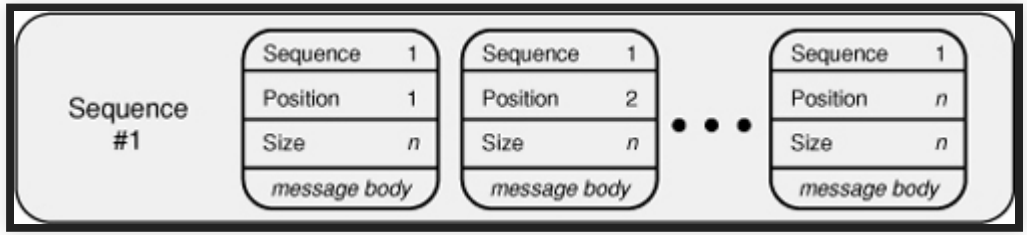



Ein Paar von Request-Reply-Nachrichten senden, jede auf ihrem eigenen Kanal.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

# MESSAGE SEQUENCE

*Wie kann eine beliebig große Datenmenge mit Hilfe von Nachrichten übertragen werden?*




Wenn eine große Datenmenge in nachrichtengroße Stücke aufgeteilt werden muss, können die Daten als Nachrichtenfolge versendet und jede Nachricht mit Sequenzidentifikationsfeldern kennzeichnet werden.

Quelle: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns - Designing, and Deploying Messaging Solutions

# 4.4 APACHE CAMEL