Service-orientierte Software-Architekturen

Prof. Dr. U. Hoffmann
FH Wedel

Muster des Nachrichtenaustauschs

Gliederung

Muster des Nachrichtenaustauschs

(Message Exchange Patterns, MEPs)

Message Exchange Patterns

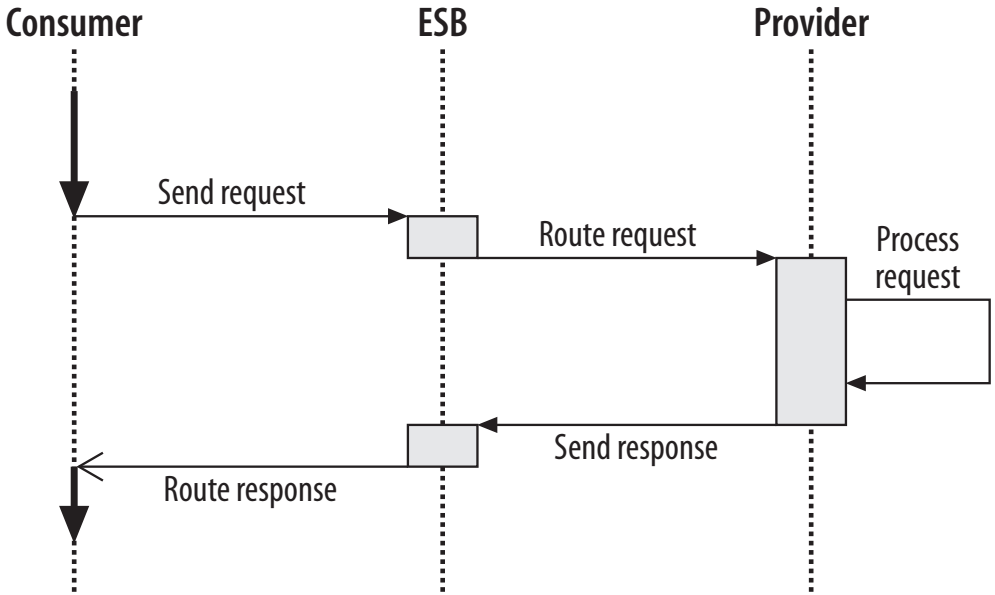
- Basis MEPs
- komplizierte MEPs
- Unzuverlässige Protokolle und Fehler
- MEPs auf unterschiedlichen Ebenen
- Ereignisgesteuerte Architektur

Muster des Nachrichtenaustauschs

Message–Exchange Patterns (MEPs)

- Beschreibung und Kategorisierung unterschiedliche Arten des Nachrichtenaustauschs in verteilten Systemen
- Wie und wann werden die einzelnen Portionen der Daten (Nachrichten) übertragen?
- Generelles Konzept zur Beschreibung der Kommunikation
- Welche Muster sind für Serviceorientierte Architekturen wichtig?

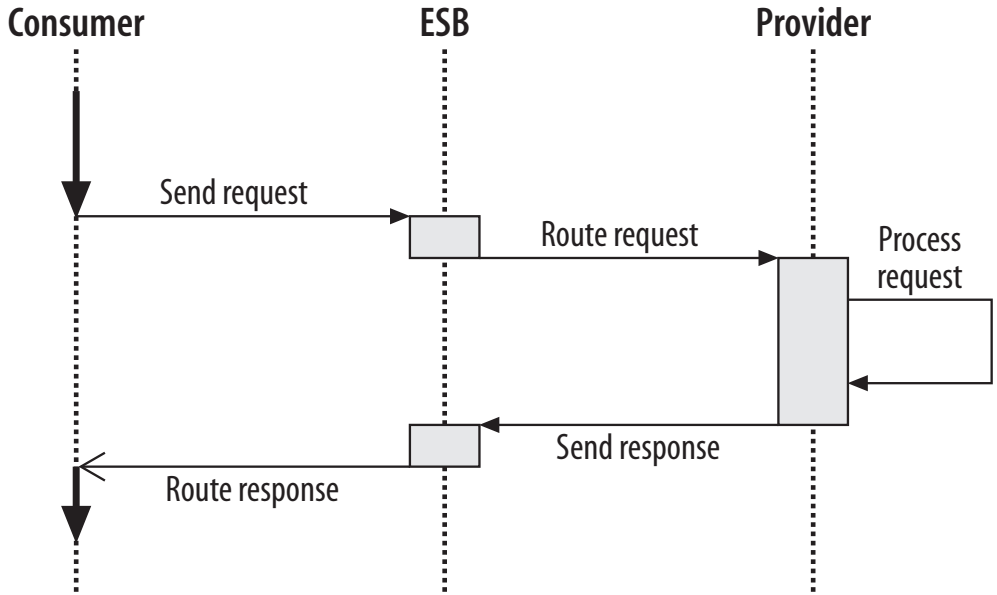
Anfrage/Antwort–Muster



- (synchronous) Request/Reponse–Pattern
- Nutzer (Consumer) und Anbieter (Provider)
- Nutzer sendet Anfrage–Nachricht (Request–Message)
- Antwort–Nachricht enthält Bestätigung bzw. Antwort–Daten
- Nutzersicht: Wie (Remote) Procedure Call (RPC), Nutzer
  - ruft Dienst mit Daten (Parametern) auf.
  - wartet auf Antwort, um sie sofort weiterzuverarbeiten.
- synchron vs. asynchron

Graphik: *SOA in Pratice*, N. Josuttis

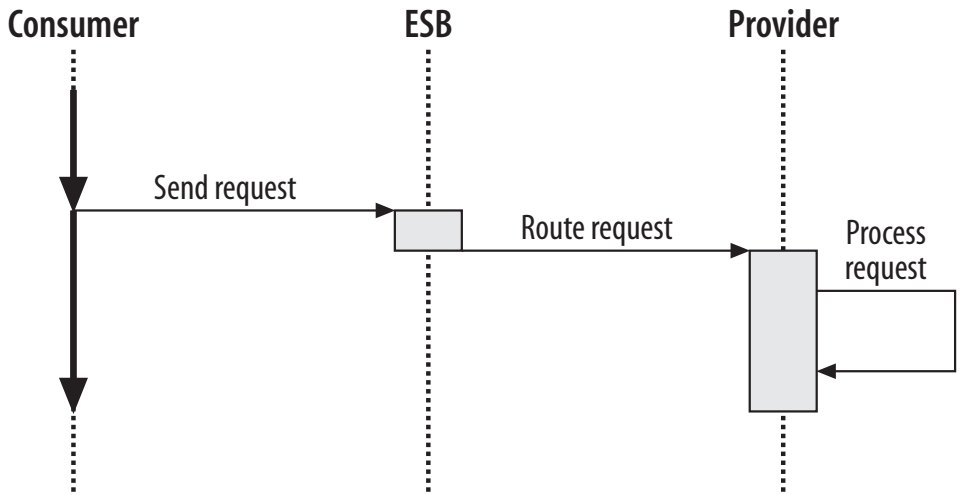
Anfrage/Antwort–Muster



- Vorteil
  - Programmcode einfach (wie Prozedur/Methodenaufruf)
  - warten auf Antwort, um sie sofort weiterzuverarbeiten.
- Nachteil
  - Es muss auf die Antwortnachricht gewartet werden
  - Oft währenddessen nichts Sinnvolles zu tun.
  - Nutzer und Anbieter müssen beide verfügbar sein.
- Geeignet nur bei schnellen Antworten (Warten nicht bedeutend)

Graphik: *SOA in Pratice*, N. Josuttis

Einweg–Nachrichten (One–Way)



- Beim Service–Aufruf wird keine Antwort/Bestätigung benötigt.
- Nutzer verschickt Nachricht und fährt fort.
- Keine sofortige oder spätere Antwort erwartet
- *Fire and Forget*

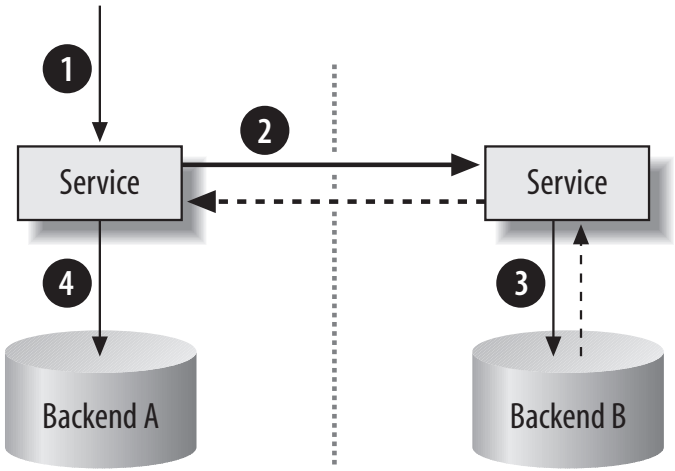
In welchem Verhältnis stehen Anfrage/Antwort–Paare zu zweimaligen Einweg–Nachrichten?

- Nutzer: 2x Einweg wirkt wie asynchrones Anfrage/Antwort–Paar
- Infrastruktur: Sender der 1. Einweg–Nachricht muss zum Empfänger der 2. Einweg–Nachricht werden.

Graphik: *SOA in Pratice*, N. Josuttis

Anfrage/Antwort vs. 2x Einweg

Muss die selbe Instanz über das Resultat informiert werden?



- Synchrone Anfrage/Antwort stellt Antwort der selben Instanz zu.

1. Nutzer wird selbst beauftragt (z. B. Bestellung durchführen)

2a. Nutzer sendet Anfrage–Nachricht an Anbieter
(z. B. prüfen der Kreditwürdigkeit)
Nutzer benötigt Ergebnis für weitere Bearbeitung.

3. Anbieter ermittelt Resultat.

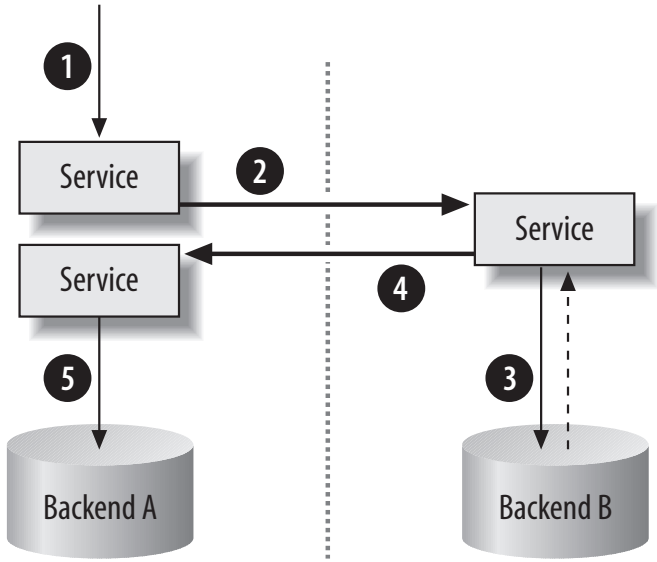
2b. (synchrone) Antwort wird der Anfrage genau zugeordnet.

4. Nutzer fährt mit seiner Bearbeitung fort.

Graphik: *SOA in Pratice*, N. Josuttis

Anfrage/Antwort vs. 2x Einweg

Muss die selbe Instanz über das Resultat informiert werden?



- Die zweite Einweg–Nachricht kann an andere Instanz gehen

1. Nutzer wird selbst beauftragt (z. B. Bestellung durchführen)
2. Nutzer sendet 1. Einweg–Nachricht an Anbieter (z. B. Rechnung drucken), Nutzer hat keinen Bedarf an (sofortigem) Ergebnis und fährt mit seiner Bearbeitung fort.
3. Anbieter ermittelt Resultat.
4. 2. Einweg–Nachricht geht an andere Instanz (im Bestellsystem) Nachricht kann dem Vorgang zugeordnet werden
5. Ergebnis wird im Vorgang nachgetragen

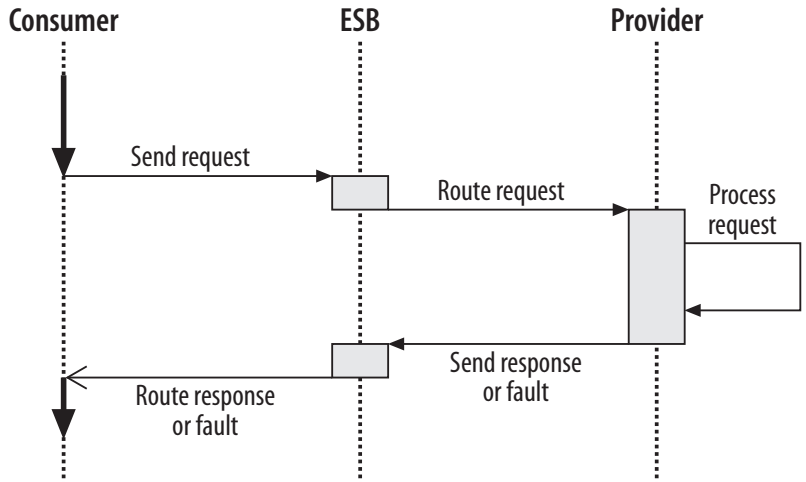
Anfrage/Rückruf–Muster

- *nichtblockierende* oder *asynchrone* Anfrage/Antwort
- *Request/Callback*
- Nutzer wartet nicht auf Antwort sondern
- fährt mit der Bearbeitung fort (z.B. in zusätzlichem Thread)
- Nutzer gibt bei der Anfrage eine *Callback–Funktion* an, an die die Antwort gesendet werden soll.
- Zusätzlich zu lösende Fragen:
  - **Reihenfolge der Antworten bei mehreren gleichzeitigen Anfragen**
    Wie kann eine Zuordnung der Antworten zu den Anfragen erfolgen? (*Correlation–IDs*)
  - **Existenz des Anfrage–Kontexts**
    Kontext aus dem die Anfrage gesendet wurde, muss erhalten bleiben oder wieder hergestellt werden, damit Antwort sinnvoll weiterbearbeitet werden kann.
  - **Umgang mit nicht eintreffenden Antworten**
    Es muss festgestellt werden, wenn Antworten nicht eintreffen und passend darauf reagiert werden.
- höhere Aufwand aber losere Kopplung

Veröffentlichen/Abonnieren–Muster

- *Publish/Subscribe*–Pattern (auch *Observer*–Pattern)
- Benachrichtigung von interessierten Teilnehmern (*Beobachtern*) über *Ereignisse* im System (*Notifications*)
- Bestimmung der Beobachter:
  - Nutzer registrieren sich (zur Laufzeit) für (eine bestimmte Art) von Ereignissen.
  - Häufig im SOA–Umfeld: Zur Modellierungszeit des Geschäftsprozesses wird festgelegt, wer welche Benachrichtigungen bekommen soll. Zur Laufzeit steht das dann bereits fest und wird häufig nicht mehr geändert.
- Zur Benachrichtigung werden Einweg–Nachrichten verschickt.
- Kommunikations–Initiative geht vom Anbieter aus (anders als bei Anfrage/Antwort).
- mögliche Kommunikationsstrukturen:
  - Zentraler Versand von Benachrichtigungen über einen Benachrichtigungsdienst (*Notification–Service*) oder im ESB (in einem *Event–Manager*)
  - direkter Versand von Benachrichtigungen von einem Anbieter an seine Beobachter

Unzuverlässige Protokolle und Fehler

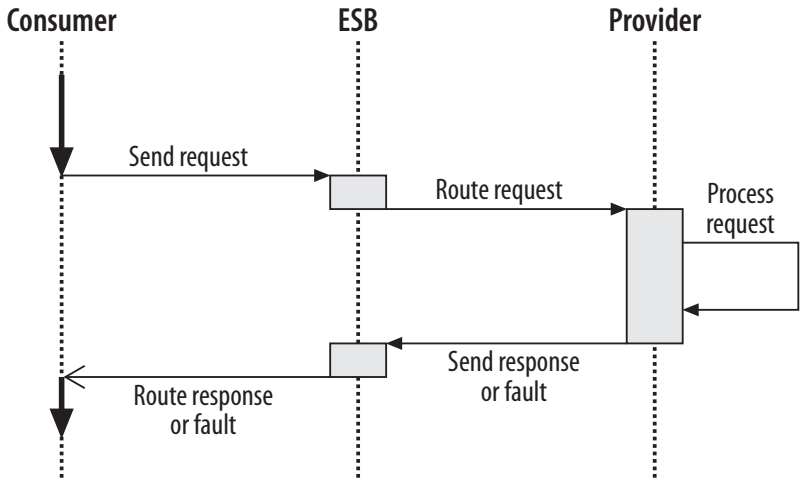


Mögliche Fehlerbilder:

- Anbieter stellt einen Fehler fest und sendet Fehlermeldung anstatt einer regulären Antwort (führt im Nutzer eventuell zu einer Exception).
- Anbieter oder Nutzer sind nicht verfügbar: keine (rechtzeitige) Annahme/Verarbeitung
- Nachrichten gehen verloren (evtl. weil die Transportschicht der Infrastruktur unzuverlässig ist).

Graphik: *SOA in Pratice*, N. Josuttis

Fehlernachrichten

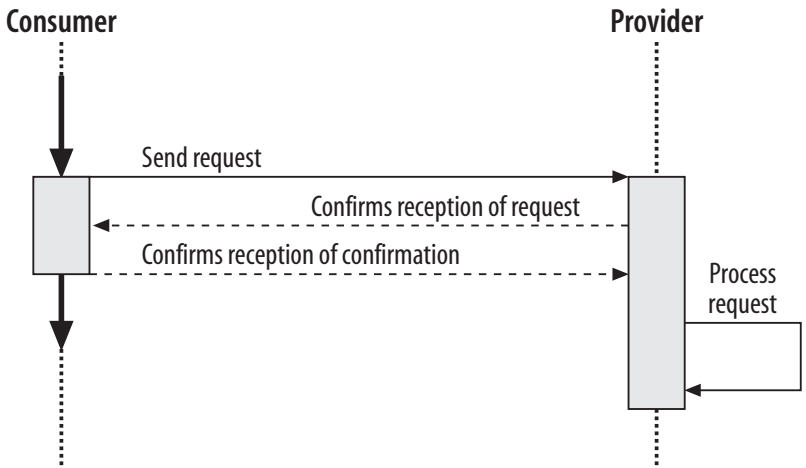


Fehlernachrichten (für vom Anbieter erkannte Fehler)

- Vorhandensein und Art von Fehlernachrichten ist abhängig vom verwendeten Kommunikationsprotokoll.
- Gegebenfalls lassen sich zu einer Fehlernachricht weitere Attribute festlegen, die den Fehler weiter beschreiben.
- Web–Services erlauben die Beschreibung von Input–, Output– und Fault–Messages.
- Abbildung der Fehlernachrichten auf Programmiersprach–Fehler (z.B. Exceptions) ist abhängig von der Infrastruktur.

Graphik: *SOA in Pratice*, N. Josuttis

Technische Fehler

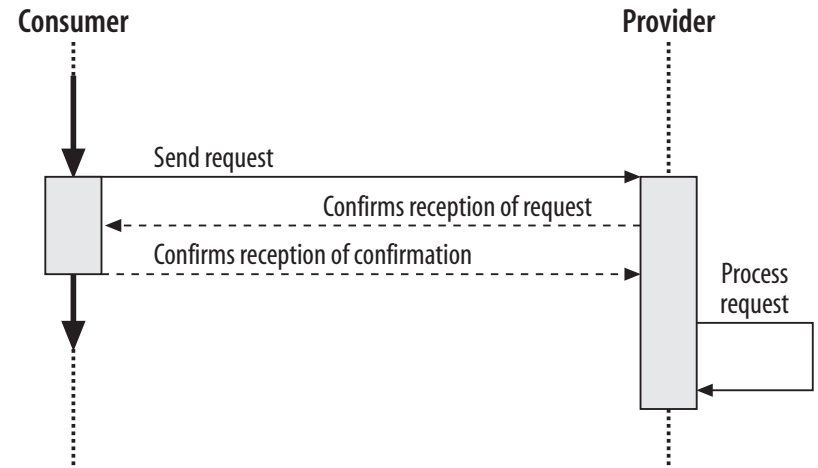


Technische Fehler (keine Fehler in der Applikationslogik)

- Fehlerfreies Zustellen von Nachrichten muss sichergestellt sein, oder fehlerhaftes erkannt werden.
- Doppelte Bestätigung (*Double Confirmation*):
- Ziel: Beide Teilnehmer haben gemeinsames Verständnis über die Kommunikation
- Anfrage kann verloren gehen.
- Bestätigung kann verloren gehen.
- Bestätigung der Bestätigung kann verloren gehen.
- Nur wenn alle 3 Nachrichten erfolgreich zugestellt werden, sind beide Partner sicher, dass die Kommunikation erfolgreich war.

Graphik: *SOA in Pratice*, N. Josuttis

Technische Fehler



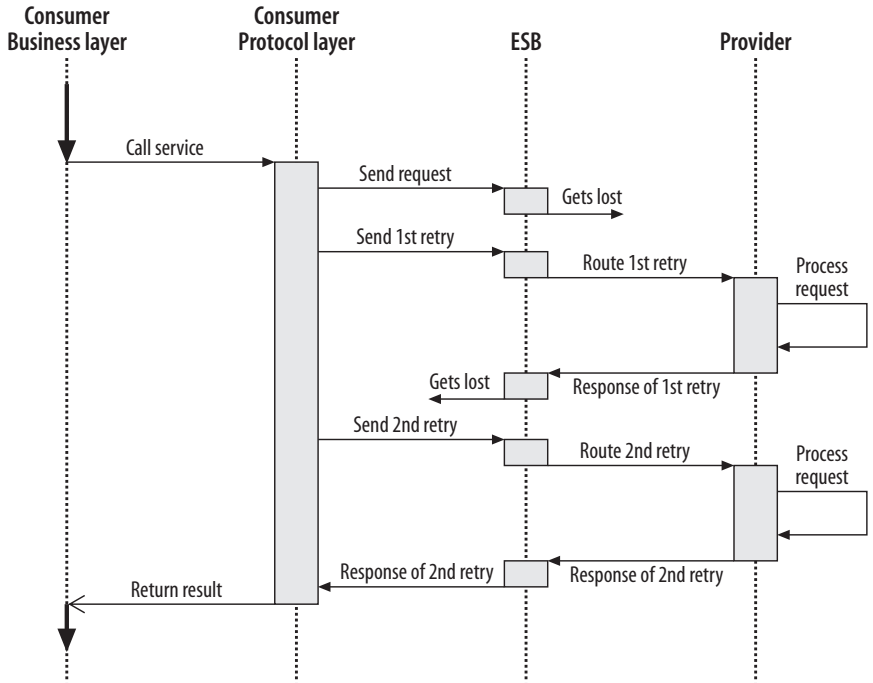
Technische Fehler (keine Fehler in der Applikationslogik)

- synchrone Kommunikation: Behandlung oft auf Protokoll–Ebene
- Schwierigere Behandlung bei asynchroner Kommunikation:
- Warteschlangen/Puffer für Nachrichten (*Message–Queues*) oft auch persistent sorgen für sicheres Zustellen der Nachrichten.
- Maximalzeiten für die Zustellung, Behandlung unzustellbarere Nachrichten? Fehlerarbeitsplatz oder verwerfen?
- Der Ursprüngliche Absender ist eventuell gar nicht mehr verfügbar, wenn der Zustellungsfehler erkannt wird
- oder er hat eventuell dafür gar keine Schnittstelle.
- Wiederholung der Zustellung (*retries*) — Idempotenz

Graphik: *SOA in Pratice*, N. Josuttis

MEPs auf unterschiedlichen Ebenen

Die einzusetzenden Nachrichtenaustausch–Muster hängen von den Eigenschaften der verwendeten Protokoll– und Transport–Ebene ab.

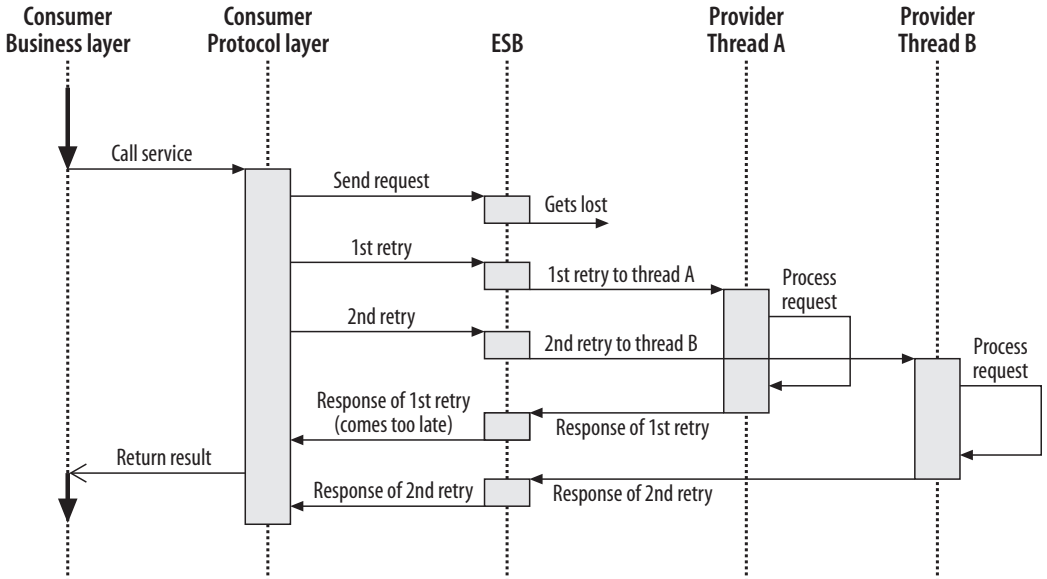


- Beispiel: Wie kann man sichere Zustellung von Nachrichten realisieren, wenn das Protokoll unzuverlässig ist?
- Behandlung von Retries im Nutzer (Kommunkationsebene)
- Welche MEPs unterstützt das Protokoll?
- Welche MEPs unterstützen die verwendeten APIs?

Graphik: *SOA in Pratice*, N. Josuttis

MEPs auf unterschiedlichen Ebenen

Problem: Retry bei zu später Antwort



- erste Anfrage geht verloren
- Antwort auf erstes Retry kommt zu spät
- zweites Retry wird initiiert
- Nutzer erhält zwei Antworten
- Wie geht man mit der zweiten Antwort um?
- Variation der **Idempotenz**–Frage
  hier mehrere Antworten, die beim Nutzer eintreffen

Graphik: *SOA in Pratice*, N. Josuttis

Ereignisgesteuerte Architektur

Neuer Architekturstil:

Ereignisgesteuerte Architektur (*Event–Driven Architecture, EDA*)

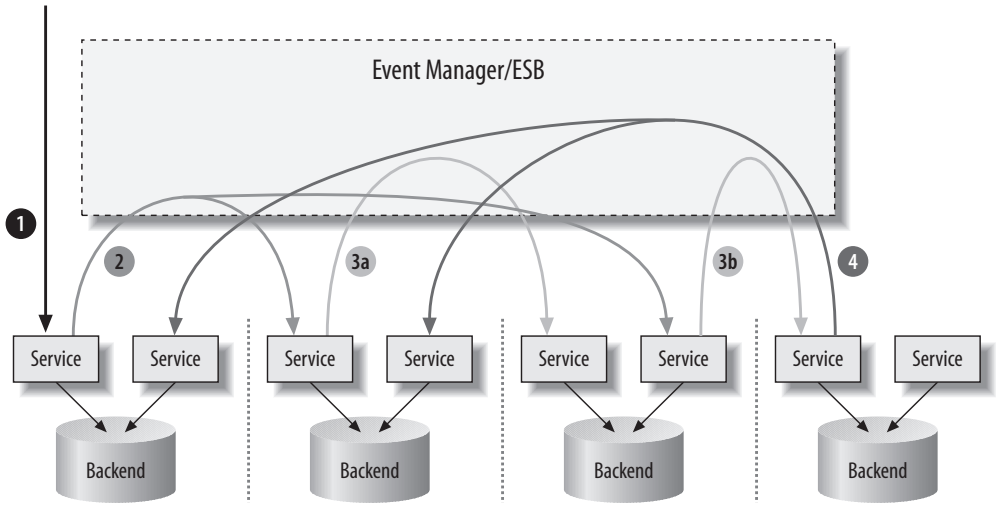
- Einweg–Nachrichten und Benachrichtigungen werden als Ereignisse aufgefasst.
- informieren über Zustandsänderungen
- lösen passende Reaktion bei interessierten Komponenten aus

- Ist EDA eine spezielle Form von SOA?
- oder eine Erweiterung/Verbesserung von SOA?
- oder grundlegend verschieden von SOA?
- *SOA 2.0*, *Advanced SOA*
- Unterschiede in der Art der Ereignisse
  - Ereignis informiert, *dass* sich Daten geändert haben.
  - Ereignis enthält Details darüber *was* sich im einzelnen geändert hat.
- In der Praxis wird man einzelne Aspekte ereignisorientiert lösen.

Graphik: *SOA in Pratice*, N. Josuttis

Ereignisgesteuerte Architektur

Sollen die Nutzer (Empfänger der Ereignisse) dem Anbieter (Sender der Ereignisse) bekannt sein?



- Nein (die Nutzer sind den Anbietern unbekannt)
  - Nutzer melden sich bei einem Ereignis–Manager (Event–Manager) an.
  - führt zu loserer Kopplung
  - Problem: Anbieter kennt seine Abhängigkeiten nicht mehr

  $\Rightarrow$ Führt leicht zu unänderbaren Systemen
- Ja (die Nutzer sind den Anbietern bekannt)
  - engere Koppplung, aber explizite Abhängigkeiten
- EDA–Prozessmodell: Geschäftsprozessketten (Choreographie)

Zusammenfassung

Message Exchange Patterns

Basis MEPs
komplizierte MEPs
Unzuverlässige Protokolle und Fehler
MEPs auf unterschiedlichen Ebenen
Ereignisgesteuerte Architektur

- Fragen?
- nächste Woche: Service Lebenszyklus