# Orchestrierung in der Praxis

Michael J.M. Wagner

software, design & management
sd&m AG
Carl-Wery-Str. 42
81739 München
michael.wagner@sdm.de

**Abstract:** Die Integration von Anwendungen stellt nach wie vor hohe Ansprüche an die Systemarchitekten. Zum einen ist es eine fachliche Herausforderung, komplexe Geschäftsprozesse auf eine heterogen gewachsene Anwendungslandschaft abzubilden, zum anderen sind unter den vielen angebotenen, oft technisch orientierten Integrationswerkzeugen die passenden auszuwählen. Die serviceorientierte Architektur (SOA) bietet dazu Ansätze, bei denen durch die Orchestrierung Services wahlfrei kombiniert werden können. In der Praxis zeigt sich jedoch, dass Schnittstellen, die als Services in einer SOA verwendet werden sollen, sowohl fachlich als auch technisch zusätzlichen Anforderungen genügen müssen. Diese Arbeit geht insbesondere auf technische Aspekte ein. Sie zeigt, dass technische Eigenschaften wie Transaktionen und Synchronität nicht losgelöst von den fachlichen Anforderungen betrachtet werden können.

## Einleitung

Seit einigen Jahren verspricht die serviceorientierte Architektur (SOA) die Möglichkeit, flexible Anwendungslandschaften gestalten zu können. Flexibilität und Wiederverwendbarkeit erhoffte man sich schon zuvor unter anderem von der objektorientierten Programmierung und der komponentenorientierten Architektur [Si04]. Die Praxis zeigt, dass mit dem Größerwerden und Zusammenwachsen von Softwaresystemen diese Architekturen an ihre Grenzen stoßen. Die SOA will diese bewährten Architekturen für Anwendungslandschaften ergänzen.

## Orchestrierung

Im Rahmen der Orchestrierung werden Services aus fachlichen Gesichtspunkten heraus zu größeren Einheiten, den Geschäftsprozessen, verbunden. BEA, IBM, Microsoft, SAP AG und Siebel Systems haben zu diesem Zweck die Business Process Execution Language for Web Services (BPEL4WS) definiert. Diese Sprache ermöglicht die Definition von Arbeitsabläufen, die dann von entsprechenden Plattformen (Enterprise Service Bus, ESB) auch ausgeführt werden können. BPEL ermöglicht dabei die Formulierung von komplexen Arbeitsabläufen mit Parallelität, Asynchronität und der Definition von Synchronisationspunkten.

## Services

Um bestehende oder auch neue Anwendungsschnittstellen als Services in einer SOA verwenden zu können, müssen neben der reinen Erreichbarkeit auf Protokollebene (SOAP, RMI, …) weitere Voraussetzungen für ein erfolgreiches SOA-Projekt erfüllt sein. Wie in [Ri05] beschrieben, müssen die Schnittstellen folgenden fachlichen Anforderungen genügen:

- Grobgranular: Wenige grobgranulare, allgemeine, fachliche Services sind in der Regel besser als viele spezielle, feingranulare.
- Technikneutral: Ein fachlicher Service sollte nie Interna des implementierenden Systems an der Schnittstelle nach außen zeigen.
- Idempotent: Ein Service wird als idempotent bezeichnet, wenn ein mehrmaliger Aufruf des Service mit denselben Parametern dieselben Effekte hat wie ein einmaliger.
- Frei von Session-Zustand: Ein Service sollte kein Konzept einer user session haben.

Als technische Richtlinie wird genannt:

- Einzeltransaktional: Jeder Aufruf eines Service ist transaktional und genügt somit den ACID-Eigenschaften. Anstatt der Bildung von verteilten oder geschachtelten Transaktionen wird eine lose Kopplung in Verbindung mit *compensating actions* empfohlen.

Gerade der letzte Punkt hat eine Rückwirkung auf den fachlichen Schnitt von Services: Ein Service muss so geschnitten sein, dass er sinnvoll in einer technischen Transaktion abzuarbeiten ist. Er darf nicht zu komplex sein, damit eine *compensating action* einfach zu formulieren ist.

Auch aus dem Punkt der Technikneutralität lassen sich weitere Schlüsse ziehen: Eine Autorisierung des Serviceaufrufs kann nicht über entsprechende Schnittstellenparameter erreicht werden, sondern muss über geeignete Sicherheitsdienste im Hintergrund des fachlichen Ablaufs erfolgen.

Der Tatsache, ob nun ein Service synchron oder asynchron in einen Arbeitsablauf eingebunden wird, wird oft zu wenig Beachtung geschenkt. Für die Fachlichkeit des Aufrufers ist es aber von großer Bedeutung, ob ein Service nur angestoßen wird, dessen Ausführung gegebenenfalls aber erst Tage später stattfindet, oder ob auf die Ausführung gewartet werden muss. Beide Modi haben für den Aufrufer Vor- und Nachteile:

| **Aufruf** | **Vorteile** | **Nachteile** |
|---|---|---|
| Synchron | • Aufrufer kann auf das Ergebnis des Service aufsetzen<br>• Aufrufer kann das Ergebnis des Service prüfen | • Bei Nicht-Verfügbarkeit des Service muss der Aufrufer abbrechen oder einen aufwändigen Wiederholmechanismus aufsetzen<br>• Bei hoher Belastung des Service können lange Wartezeiten auftreten |
| Asynchron | • „fire and forget“: Aufrufer muss sich nicht um Verfügbarkeit und Auslastung des Service kümmern | • Aufrufer kann nicht auf Ergebnisse des Service aufsetzen |

## Beispiel

BPEL4WS bietet grundsätzlich die Möglichkeit synchroner als auch asynchroner Aufrufe. Dass es sich hierbei nicht um ein nebensächliches technisches Detail handelt, wird durch das folgende Beispiel verdeutlicht.

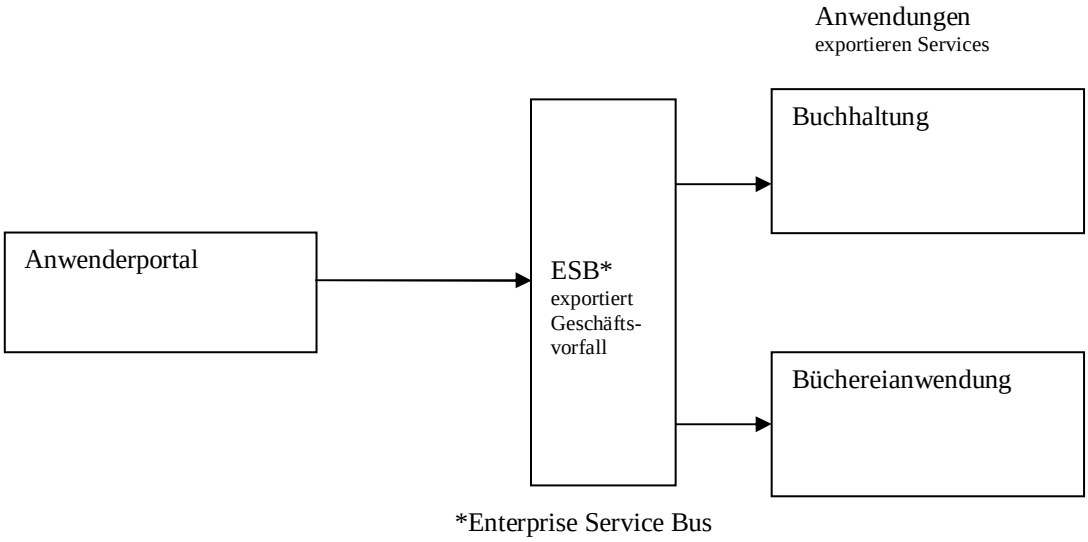


Abbildung 1: Architektur des Büchereisystems

Die Anwendungslandschaft einer Bücherei besteht aus einem Portal, das die in BPEL formulierten Vorfälle nutzt. Diese werden von einem Enterprise Service Bus (ESB) ausgeführt. Die Services werden von zwei Anwendungen, der Buchhaltung und einer Büchereianwendung zur Verfügung gestellt.

Betrachten wir drei Geschäftsvorfälle am Anwenderportal für die Mitarbeiter der Bücherei:

- Bestellung eines Buches
- Zahlung einer Mahngebühr
- Abrechnung eines Buchkaufs in der Buchhaltung mit Freigabe in der Büchereianwendung

Die Geschäftsvorfälle stützen sich auf Services der Backendsysteme: Das Einbuchen einer Kontenbewegung in der Buchhaltung, sowie verschiedene weitere Operationen der Büchereianwendung. Für beide Systemschnittstellen bieten sich die oben genannten Prinzipien für Services an: Grobgranular, technikneutral, idempotent, zustandsfrei, einzeltransaktional. Folgende Abbildung zeigt die Signaturen der Services als Java-Interfaces. Die Parametertypen (Konto, Zahlungsvorgang, …) müssen dabei so ausgeprägt sein, dass daran der Vorgang eindeutig identifizierbar ist. Nur dann ist es möglich, einen erneuten Aufruf als Wiederholung zu erkennen.

```
public interface Buchhaltung{

   Kontoinformation buchen( Konto konto, Zahlungsvorgang vorgang );

}

public interface Buechereianwendung{

   Nutzer nutzerPflegen( Nutzer  nutzer );

   Nutzer mahngebuehrEinzahlen( Zahlungsvorgang vorgang );

   Reservation buchBestellen( Titel titel, Nutzer nutzer );

   void bestandPflegen( Titel titel, Buch buch );

}
```

Abbildung 2: Signatur der Services

Es zeigt sich, dass es für einen reibungslosen Ablauf des Büchereibetriebs nötig ist, Services sowohl synchron als auch asynchron verwenden zu können.

Betrachten wir hierzu die Konsequenzen der Nichtverfügbarkeit der Backendsysteme:

- Bei der Nichtverfügbarkeit der Büchereianwendung kann dem Büchereikunden nicht gesagt werden, ob und wann sein Buch zur Abholung bereit stehen wird. Ohne diese Information macht aber der gesamte Geschäftsvorfall keinen Sinn. In diesem Fall wird also die synchrone Antwort des Backendsystems benötigt.

- Bei der Nichtverfügbarkeit der Buchhaltung soll die Zahlung der Mahngebühr angenommen und der Kunde in der Büchereianwendung entsperrt werden. Das Einbuchen in das Kontensystem erfolgt zu dem Zeitpunkt, an dem die Buchhaltung wieder verfügbar ist.

- In der Buchhaltungsanwendung soll eine Kontenbewegung sofort an der GUI sichtbar sein. Daher erfordert die Abrechnung eines Einkaufs einen synchronen Aufruf der Buchhaltung. Die Freigabe des neuen Buches in der Büchereianwendung soll nachgelagert erfolgen.

Im Folgenden werden die beiden systemübergreifenden Arbeitsabläufe (Zahlung einer Mahngebühr, Abrechnung eines Buches) näher betrachtet:

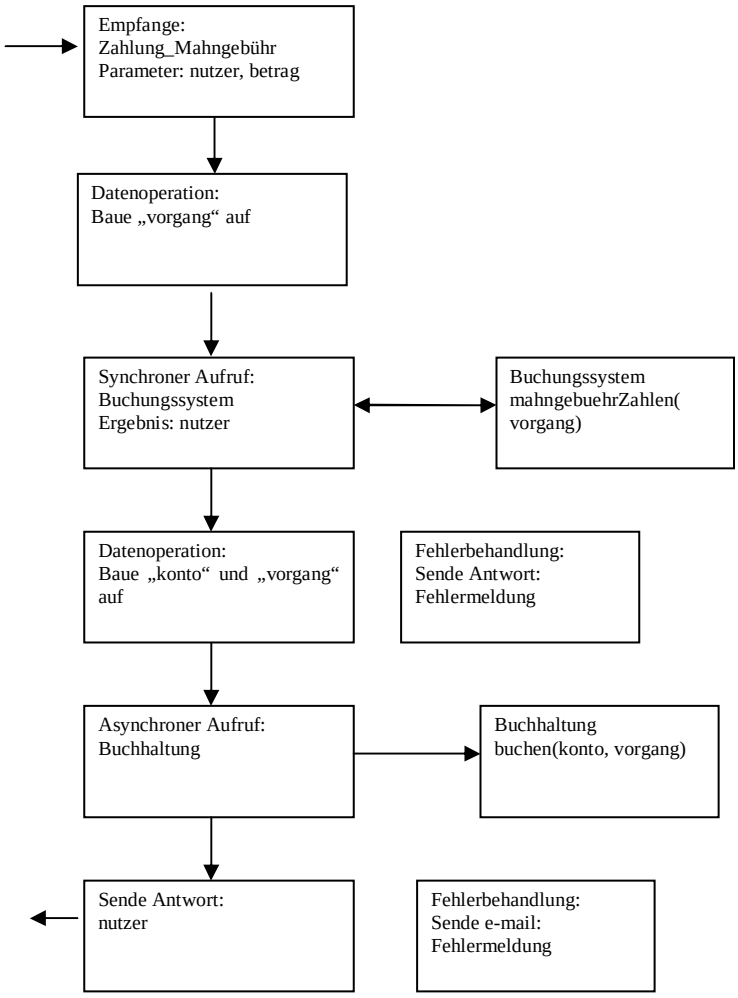


Abbildung 3: Arbeitsablauf „Zahlung einer Mahngebühr“

Abbildung 3 zeigt den Arbeitsablauf „Zahlung einer Mahngebühr“. Nach dem Empfang des neuen Vorgangs mit Nutzerdaten und gezahltem Betrag, wird der Datensatz `vorgang` aufgebaut. Er muss all die Daten enthalten, die den Zahlungsvorgang identifizieren (Datum, Nutzer, Betrag, …). Dieser Datensatz wird der Büchereianwendung mitgeteilt. Der Aufruf ist idempotent, d.h. auch bei mehrmaligem Aufruf wird der gezahlte Betrag nur einmal gutgeschrieben. Als Ergebnis wird der aktualisierte Nutzer-Datensatz zurückgegeben, der unter anderem ein Kennzeichen enthält, ob der Nutzer all seine Rückstände beglichen hat oder noch gesperrt ist. Im Fehlerfall wird eine entsprechende Meldung an den Anwender zurückgegeben.

Vor dem asynchronen Aufruf der Buchhaltung müssen die entsprechenden Buchungssätze aufgebaut werden. Üblicherweise werden die Datensätze für den Zahlungsvorgang der Büchereianwendung und der Buchhaltung unterschiedliche Formate haben, so dass an dieser Stelle die Umwandlung erfolgt. Dazu können einfache lesende Datenoperationen nötig sein, die hier nicht näher betrachtet werden. Nach dem Aufruf der Buchhaltung wird der Nutzer-Datensatz an den Aufrufer zurückgegeben. Tritt bei der Bearbeitung ein Fehler auf, so wird ein Sachbearbeiter über diesen Vorfall per e-mail informiert. Der Geschäftsvorfall „Zahlung einer Mahngebühr“ konnte dennoch erfolgreich beendet und der Nutzer entsperrt werden. Die fachliche Anforderung, dass das Entsperren des Nutzers nicht von der Verfügbarkeit der Buchhaltung abhängen soll, ist also erfüllt.

Abbildung 4 zeigt den Arbeitsablauf für die Abrechnung eines Einkaufs in ähnlicher Weise. Hier liegen die Gewichte umgekehrt: Die synchrone Antwort der Buchhaltung wird an den Anwender zurückgegeben, was nicht von der Verfügbarkeit der Büchereianwendung abhängen soll.

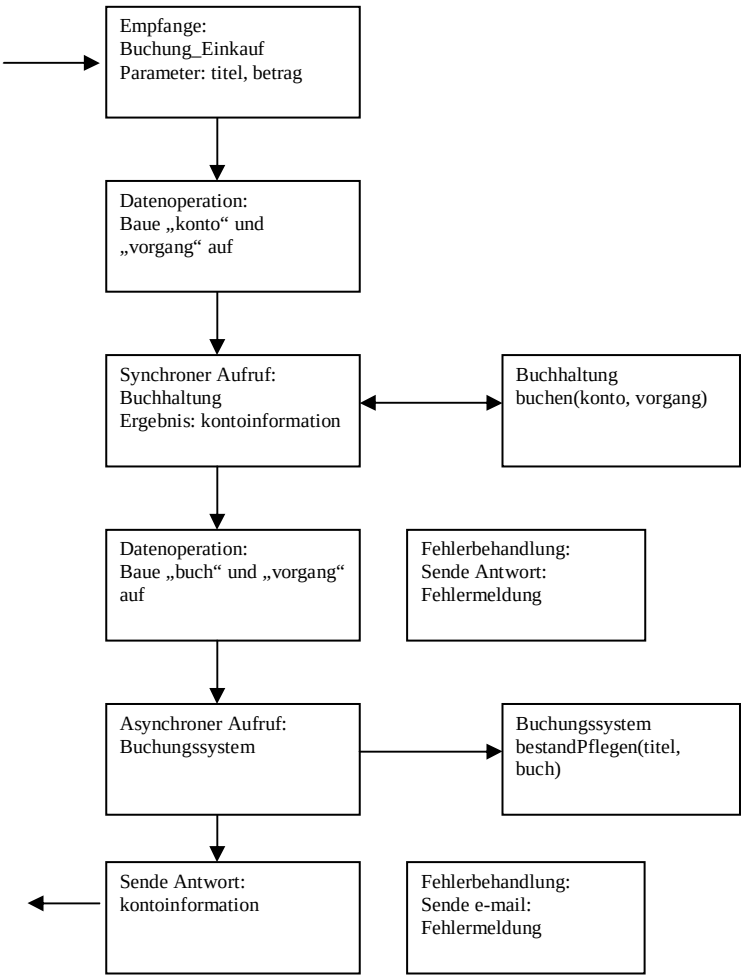


Abbildung 4: Arbeitsablauf „Abrechnung eines neuen Buches“

## Folgerungen

Dieses Beispiel unterstreicht, dass die Synchroneigenschaft einer Anwendungsschnittstelle fachlichen Anforderungen unterliegt. Soll nun eine Schnittstelle für eine Orchestrierung zur Verfügung stehen und damit auch in zukünftigen Anwendungsszenarien ohne Anpassung eingesetzt werden, muss sie sowohl synchron, als auch asynchron ansprechbar sein.

Der Service-Anbieter hat zur Erfüllung dieser Forderung verschiedene Möglichkeiten:

- Die Schnittstelle wird nur synchron zur Verfügung gestellt. Die asynchrone Aufrufvariante wird durch die Abspaltung eines eigenen Threads im ESB erzeugt.

- Die Schnittstelle wird nur asynchron angeboten. Der Anbieter arbeitet allerdings Anfragen sofort ab und sendet das Ergebnis, wenn erwünscht, an den Aufrufer zurück. In diesem Falle kann der ESB ein entsprechend synchrones Verhalten emulieren.

- Die Schnittstelle wird sowohl synchron, als auch asynchron angeboten. Der ESB wählt den geeigneten Typ. Nur in diesem Fall kann der Service-Anbieter die asynchronen Anfragen zu einer Hintergrundverarbeitung zu lastarmer Zeit bündeln.

Für den ESB-Hersteller leitet sich daraus ab, dass er zum einen dem fachlichen Anwender sowohl eine synchrone, als auch eine asynchrone Variante eines jeden Service anbietet. Zum anderen muss im ESB die Abbildung auf die real zur Verfügung stehenden Services durchgeführt werden.

Dieses Beispiel zeigt gleichfalls, dass bei geschickter Anordnung des Arbeitsablaufs die Verwendung von *compensating actions* vermieden werden kann. Bei komplexeren Abläufen lässt sich aber deren Einsatz zur Reaktion auf Fehlersituationen nicht grundsätzlich vermeiden.

Eine Ausnahme bilden dabei rein lesende Operationen. Ein asynchrones Lesen, sowie dessen Kompensation muss selbstredend nicht in Betracht gezogen werden.

Desweiteren ist durch den ESB eine geeignete Sicherheitsinfrastruktur zur Verfügung zu stellen, damit die Services durch die Verwendung dieser Sicherheitsmechanismen die Autorisierung eines Aufrufs prüfen können.

## Fazit

Eine vollständige Serviceschnittstelle, die in einer SOA flexibel orchestriert werden kann, muss also aus Anwendersicht folgenden Umfang aufweisen:

- synchrone Nutzschnittstelle

- asynchrone Nutzschnittstelle

- *compensating action*

Im Rahmen der Orchestrierung wählt der Nutzer die geeigneten Servicetypen aus seiner fachlichen Sicht.

Für den Anwender transparent verlaufen die Transaktionssteuerung, die Prüfung der Autorisierung, sowie eine allfällige Umsetzung der Synchroneigenschaft.

Für die effiziente Anwendung einer SOA mit freier Orchestrierung haben Anbieter von SOA-Produkten und Service-Anbieter ihre Systeme auf diese Anforderungen hin zu prüfen!

## Literaturverzeichnis


[Ri05] Richter, J.-P. *Wann liefert eine Serviceorientierte Architektur echten Nutzen?* Software Engineering 2005, Fachtagung des GI-Fachbereichs Softwaretechnik

[Si04] Siedersleben, J. *Softwarearchitektur*, dpunkt.verlag 2004