

# **Materialien zu „Serviceorientierte Architekturen“**

## **WS 2008/9**

Version 1.3.0 vom 29. Januar 2009

## Literatur

- Thomas Erl:  
Service-Oriented Architecture: Concepts, Technology, and Design (2005)  
weitgehende Abdeckung der behandelten Themen
- Thomas Erl:  
Entwurfsprinzipien für serviceorientierte Architekturen  
2008, Fokussierung auf den SOA-Entwurf
- Ingo Melzer et al.  
Service-orientierte Architekturen mit Web Services  
3. Auflage (2007)  
Große Bandbreite, teilweise schwer verständlich
- Mark D. Hansen  
SOA Using Java EE5 Web Services, Prentice Hall International, 2007  
Teilaspekt „SOA-Programmierung mit Java“ gut erklärt

# **Inhalte der Lehrveranstaltung**

## **1. SOA Einführung**

- 1.1 Brainstorming: Was ist eine SOA?
- 1.2 Wozu braucht man eine SOA?
- 1.3 Bezug zwischen SOA und EAI
- 1.4 Bezug zwischen SOA und Web-Services
- 1.5 Bezug zwischen SOA und OO-Komponentenmodellen
- 1.6 Begriffsdefinition SOA
- 1.7 Die Mitspieler
- 1.8 SOA und Schichtenmodelle für verteilte Systeme

## **2. Web-Services**

- 2.1      Web-Service-Architektur
- 2.2      WSDL
- 2.3      SOAP
- 2.4      Verzeichnisdienste / UDDI
- 2.5      Die WS-Erweiterungen
  - 2.5.1    Richtlinien und Metadatenaustausch
  - 2.5.2    Sicherheit
  - 2.5.3    Kommunikationsmuster, Koordination und Orchestrierung
- 2.6      Geschäftsprozess-Modellierung

### **3. Serviceorientierte Organisation und SOA-Governance**

- 3.1 Organisatorische Herausforderungen, Silo-Architektur
- 3.2 SOA-Lebenszyklus
- 3.3 SOA-Einführungsstrategien

## **4. Web-Services und Java**

- 4.1 Überblick Java APIs für WS
- 4.2 WSDL first?
- 4.3 JAX-WS
- 4.4 JAXB
- 4.5 ESB und JBI

# **1. Grundlagen und Begriffe**

## **1.1 Anmerkung zur Bezeichnung SOA**

- SOA = „Serviceorientierte Architektur“
- SOA auch Dienstarchitektur/ dienstorientierte Architektur
- 1996 von dem Marktforschungsunternehmen Gartner geprägt
- nicht einheitlich definiert
- Kurzdefinition der OASIS:

SOA ist ein Paradigma für die Strukturierung und Nutzung verteilter Funktionalität, die von unterschiedlichen Besitzern verantwortet wird.

- Begriffsdefinition im Fachlexikon der GI (Gesellschaft für Informatik):

<http://www.gi-ev.de/service/informatiklexikon/informatiklexikon-detailansicht/meldung/118/>

## **SOA für Informatiker? SOA für Betriebswirte?**

SOA ist ein abstraktes Modell einer Unternehmens- und IT-Architektur

- Unternehmensarchitektur:

Aufbau des Unternehmens, Definition der Geschäftsprozesse

- SOA als IT-Architektur:

- SOA ist Komponentenarchitektur für verteilte Systeme
- Komponente: Baustein im Baukastenmodell, Kompositionsprinzip, eigenständige Entwicklung
- Die Komponenten sind die Services (Dienste)

## **Merkmale und Unterschiede zu anderen Komponentenmodellen?**

- SOA unterstützt in besonderer Weise die Integration vorhandener Anwendungen (EAI)
- ist aber Geschäftsprozess-orientiert (nicht aber an Anwendungen orientiert!)
- SOA-Technologie beruht auf offenen, plattformübergreifenden Standards
  - Web-Services sind am besten geeignete Technologie zur SOA-Realisierung
  - aber auch SOA auf CORBA-Basis

## SOA Grundlage: Kooperierende Services

- Rollen: Dienstanbieter / Dienstnutzer

Nutzer *kann* selbst ein Dienst sein

- Kommunikationsparadigmen: asynchron und dokumentenbasiert bzw. auch RPC
- Kommunikationspfade: ggf. komplex durch Zwischendienste
  - Beispiel: Lastverteilungsdienst vermittelt eine von mehreren verfügbaren Dienste-Instanzen an den Nutzer
  - Zwischendienste können Nachrichten verändern
  - Beispiel: Sicherheits-Dienst ver- und entschlüsselt vertrauliche Nachrichten

## 1.2 SOA-Motivation

Herausforderungen an die Organisation und die IT in großen Unternehmen:

- Integration heterogener Anwendungen (EAI)
- Anpassbarkeit der Geschäftsprozesse
- Unternehmensübergreifende Anwendungen (B2B-Integration)
- Nutzung externer Anwendungen (Application Service Provider,  
„Software as a Service“)

## 1.3 Zusammenhänge: SOA und EAI

EAI: Enterprise Application Integration / Unternehmensanwendungsintegration

- prozessorientierte Integration von Anwendungssystemen in heterogenen IT-Anwendungsarchitekturen

Was heißt „prozessorientiert“?

- Unternehmensanwendungen
  - Betrieblich: ERP, PPS, CRM, HR, Supply-Chain . . .
  - Technisch: CAD, CAM, CAFM, . . .
  - Groupware / Kommunikation / Bürossoftware
  - Sonstige
- IT-Organisation: Säulenarchitektur? (Silo-Architektur)
  - Was könnte das sein?
  - Austausch von Daten zwischen den Säulen
  - Problem: Bei geänderten Geschäftsprozessen problematisch Anwendungen ändern/neu programmieren? Probleme?

## 1.4 Zusammenhänge: SOA und Web-Services

Web-Services:

- RESTful WS (einfaches Konzept, basieren nur auf HTTP-Requests)
- SOAP-basierte WS (komplex)

Falls nicht besonders angemerkt: WS = SOAP-basierte WS

WS sind ein auf XML und anderen offenen Standards basierender komplexer Kommunikationsmechanismus und werden meist als Basis für SOA verwendet

## 1.5 Zusammenhänge: SOA und OO

- SOA und OO sind (u.a.) Softwarearchitekturkonzepte
  - teilweise konkurrierende Ansätze
  - teilweise sich ergänzende Technologien
- Lose Kopplung der Komponenten bei SOA durch aufwändiges Nachrichtenkonzept:
  - Typische Nachrichten-Nutzlast: XML-Geschäftsdocument (z.B. Auftrag)
  - Datentypen typischerweise auf verschiedene XML-Schemata verteilt
  - Meta-Information: Policies, Routing-Information, Security-Daten usw.
- Granularität der Kommunikation: SOA grobkörnig, OO feinkörnig (entfernter Methodenaufruf mit Austausch „wenig intelligenter“ Daten)
- SOA favorisiert zustandslose Komponenten
- SOA unterstützt aktivitätsneutrale Dienste, aktivitätsspezifische Semantik steckt in der Nachricht.  
OO favorisiert intelligente aktivitätsbewusste Komponenten

- SOA beruht auf Komponenten-Komposition

OO benutzt Komponenten-Komposition und Vererbung. Vererbung begünstigt enge Kopplung

## 1.6 Detailliertere Begriffsdefinition

Eine SOA ist ein auf Diensten basierendes fachliches Architekturmuster, das die Integration heterogener Anwendungen unterstützt.

Merkmale:

- Ein Dienst hat eine klar umrissene fachliche Funktion und ist eigenständig nutzbar  
(Prinzip: Trennung der Zuständigkeiten, Autonomie der Dienste)
- Ein Dienst hat eine formal definierte veröffentlichte Schnittstelle und verbirgt seine Implementierung
- Ein Dienst ist plattformunabhängig nutzbar
- Die Kommunikation beruht auf offenen Standards
- Ein Dienst ist in einem Verzeichnis registriert.
- Dienste sind in einer SOA lose gekoppelt
- Eine SOA ist grobgranular („wenige“ Dienste)

- Dienste-Schnittstellen sind maschinenlesbar, unterstützen dynamische Bindung und die Konkurrenz verschiedener Diensteanbieter
- Dienste sind wiederverwendbar
- Dienste können Geschäftslogik abstrahieren
- Dienste können Anwendungen abstrahieren

## 1.7 Wer macht denn SOA? Einige „Mitspieler“

- Viele Hersteller für SOA-Produkte, z.B.  
IBM („Websphere“), BEA/Oracle, Microsoft („BizTalk“), Sun („Glassfish/Open ESB“, Java APIs)  
Deutschland: SAP (Netweaver), Software AG (Webmethods)
- Open Source-Gemeinde
- Organisationen:
  - W3C
  - OASIS
  - WS-I
  - UN/CEFACT (UN Centre for Trade Facilitation and E-Business) ebXML
  - OMG: CORBA, (auch UML, MDA ...)

## Verantwortung für Standards und Spezifikationen

- W3C – World Wide Web Consortium:
  - Definiert Standards für WWW-Technologie als SOA-Basis: WWW, XML, XML Schema, XSLT, SOAP, WSDL usw.
  - SOA-spezifisch: WS-CDL = Web-Services Choreography Description Language
- OASIS – Organization for the Advantage of Structured Information Standards
  - ebXML, WS-BPEL, WS-Security, UDDI, SAML (Security Assertions Markup Language), XACML (Extensible Access Control Markup Language).
- Web Services Interoperability Organization (WS-I)
  - „Basic Profile“: Empfehlung für eine Zusammenstellung der WS-Technologien und ihrer Versionen zugunsten einer unternehmenübergreifenden Interoperabilität
  - Anwendungsmuster
  - Beispiel-Implementierungen

## Ergänzung: Begriffe aus dem SOA-Umfeld

- EAI, ERP, HR, CRM, Legacysystem, Silo
- horizontale/vertikale Architektur
- ESB (Business Bus, Integrationsplattform) , „Hub and Spoke“-Architektur
- Orchestrierung, Choreographie
- Web-“Enabling“

## 1.8 SOA und Schichtenmodelle für verteilte Systeme

Komplexität einer SOA erfordert separate Modellierung mehrere Architektur-Aspekte

Welche Aspekte kommen in Frage?

- Klassifikation von Diensten und Bezug zu Geschäftsprozessen und Applikationen
- Modell für den Aufbau und Austausch von Nachrichten
- Kooperationsmodell: Rollen, Dienste, Nachrichten
- Abstraktionsebenen, Schichtenarchitektur
- Sonstige Aspekte: Ressourcenmodell, Richtlinienmodell  
(hierzu mehr in [Melzer])

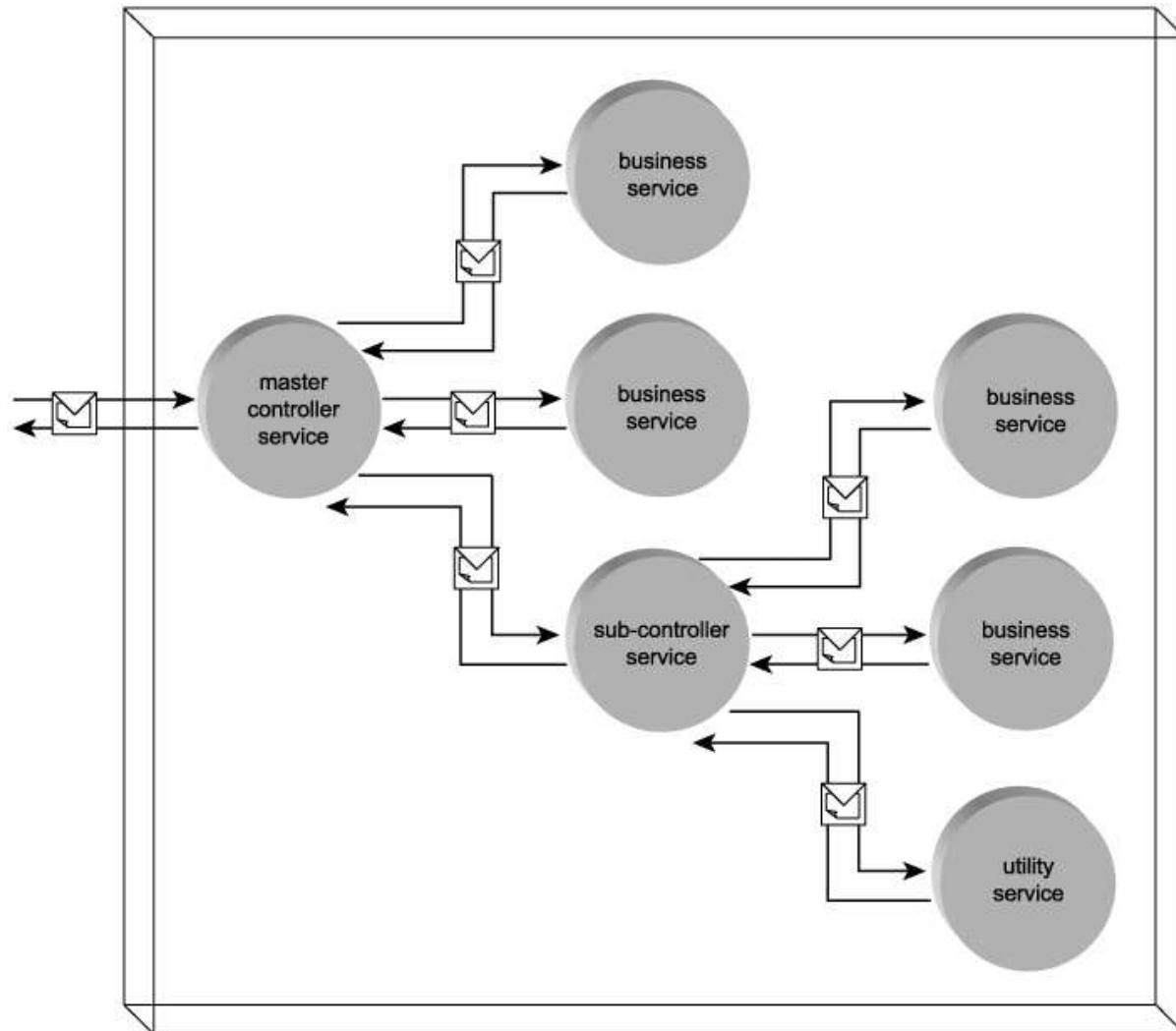
Ist eine SOA eine Alternative zu einer Schichtenarchitektur?

## 1.8.1 Klassifikation von Diensten nach Funktionsbereichen

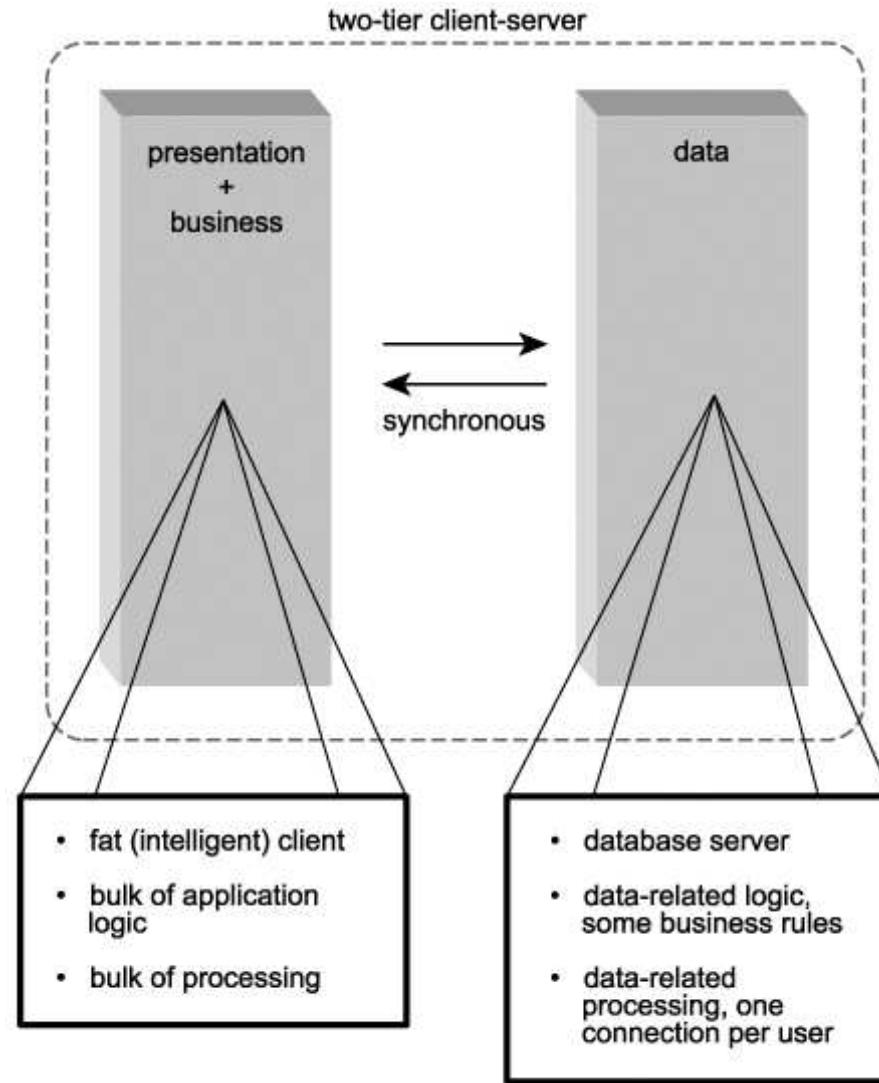
- Geschäftsdienst (Business Service)
  - Abstraktion von Geschäftsprozesslogik
  - Autonomer Dienst
  - kann an Dienst-Komposition beteiligt sein
- Hilfsdienst (Utility Service)
  - Generischer Dienst
  - kapselt wiederverwendbare, nicht anwendungsspezifische Logik
  - autonom
  - mehrfach verwendbar
- Koordinatordienst („Controller Service“)
  - Koordiniert Dienstekomposition
  - Auf Geschäftsprozessebene: Orchestrierungsdienst

Geschäftsdienste und Hilfsdienste können auch koordinieren: Hybrid-Dienste

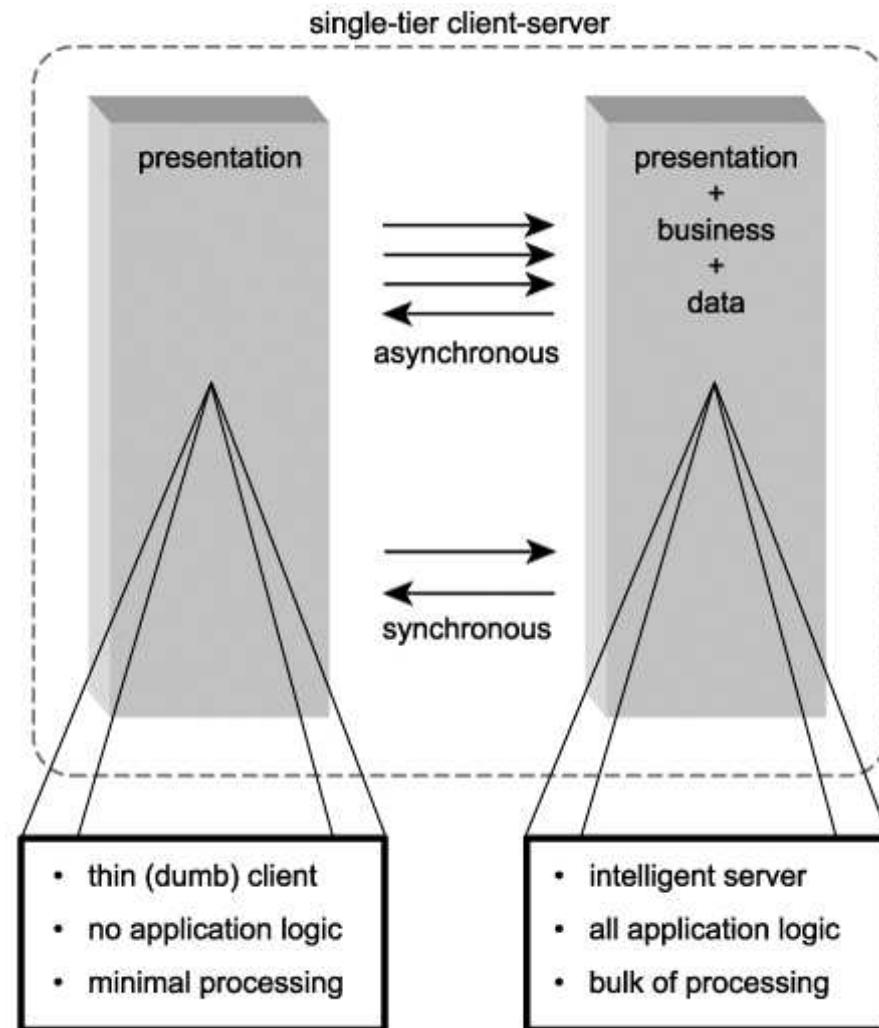
## Dienste-Klassen und Kompositionsprinzip: Mehrstufige Dienstekomposition



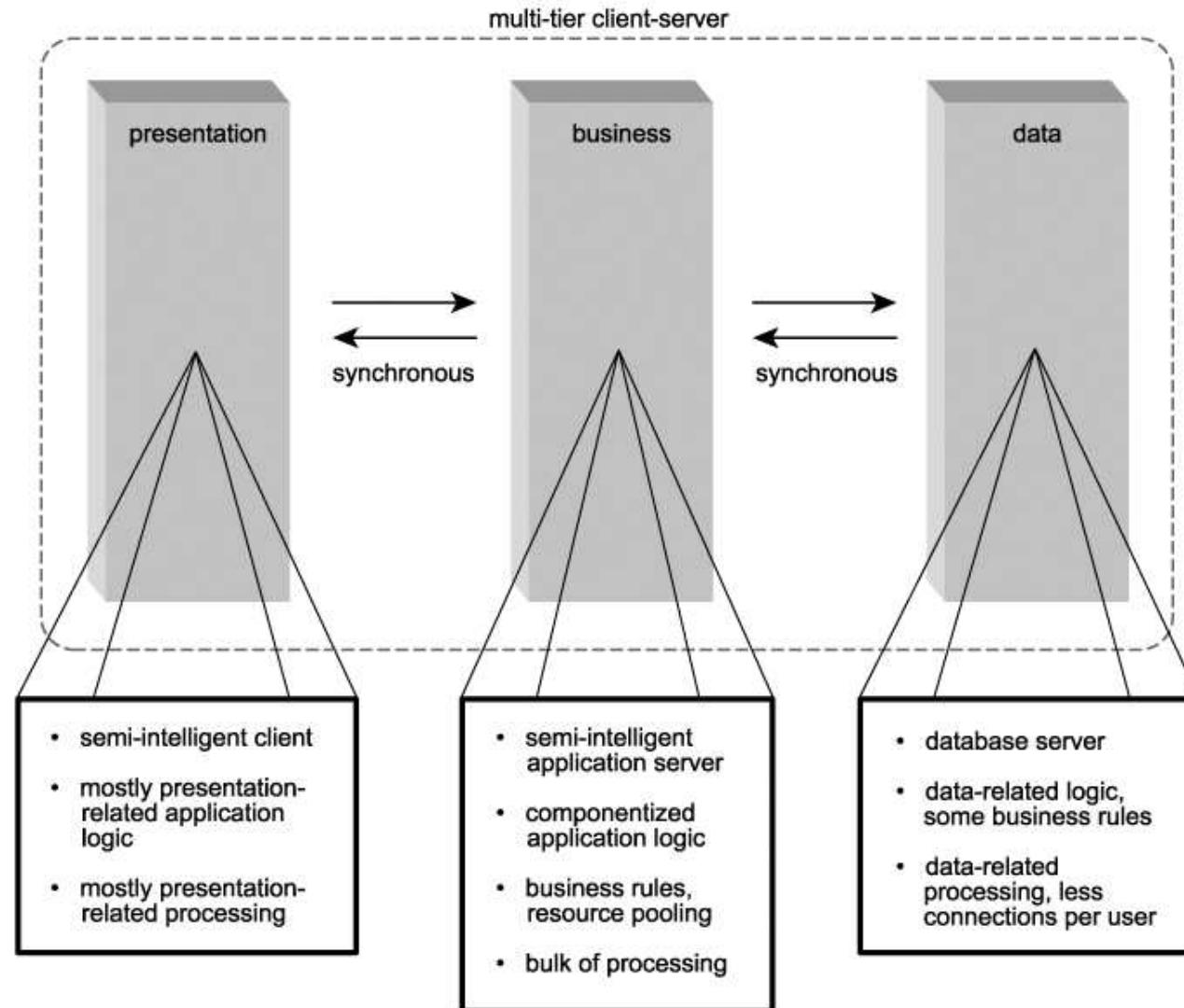
## Schichtenmodelle für verteilte Systeme: Thick Client + Server



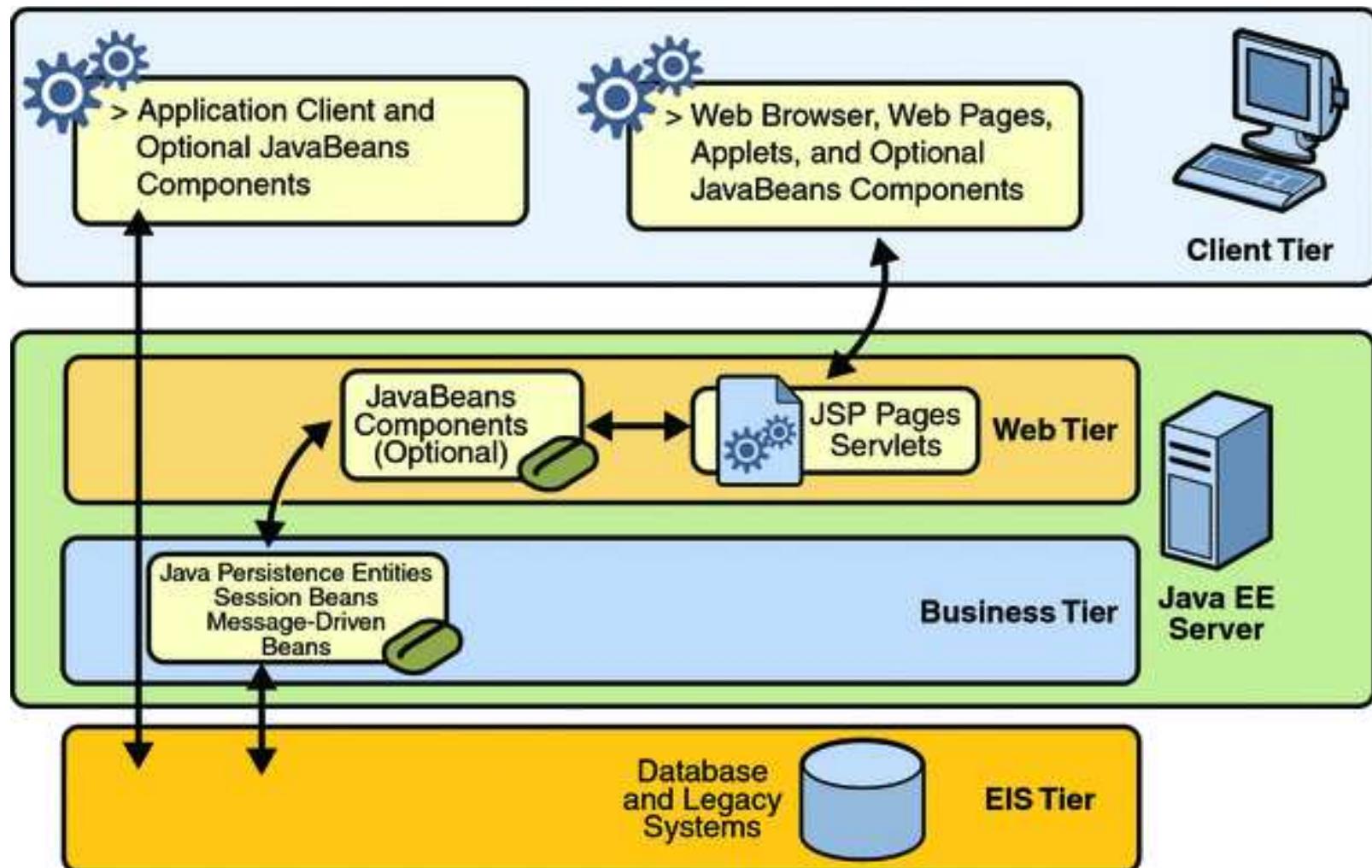
## Schichtenmodelle für verteilte Systeme: Thin Client + Server



# 3-tier-Architektur

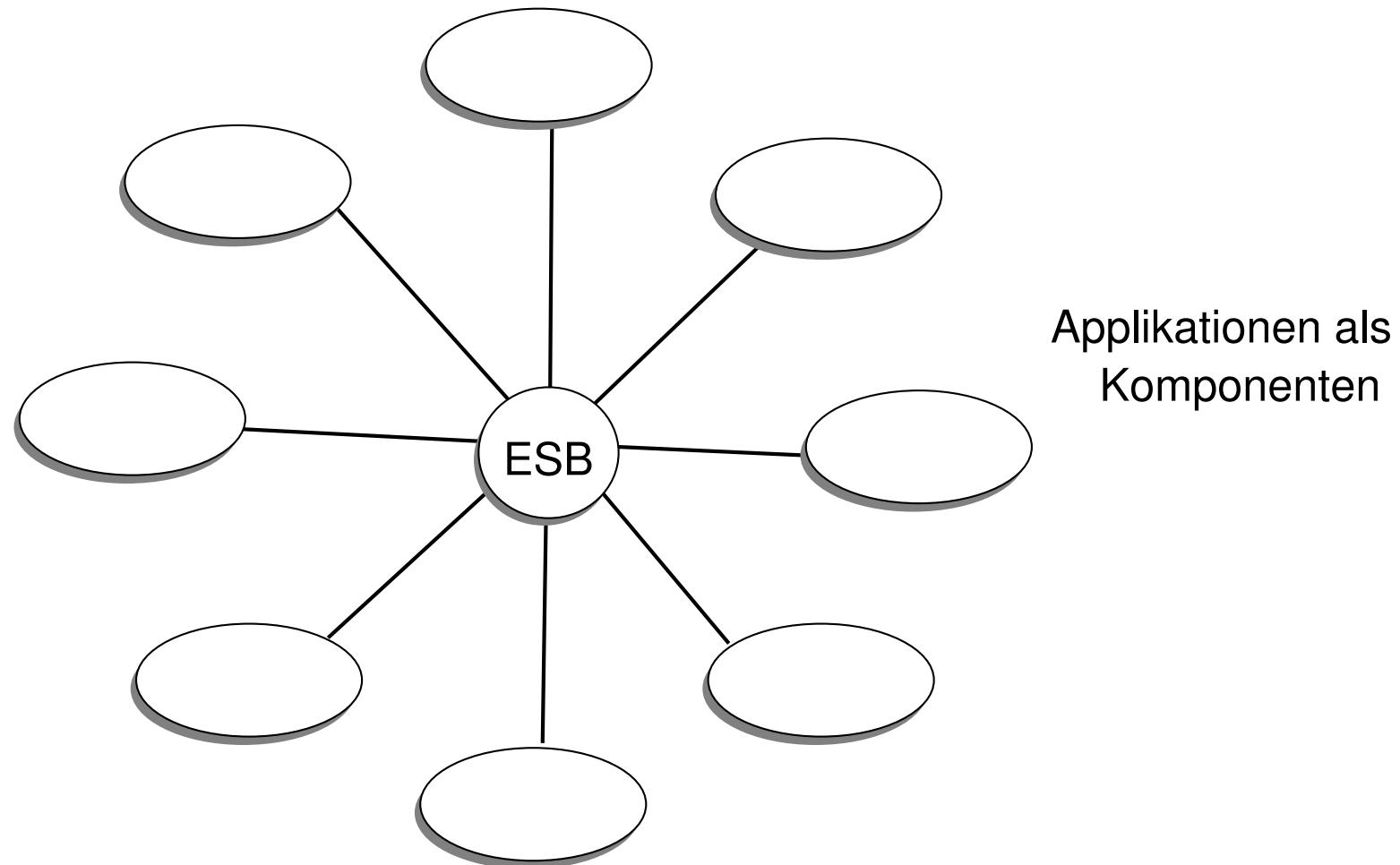


# Java EE 4-tier-Architektur

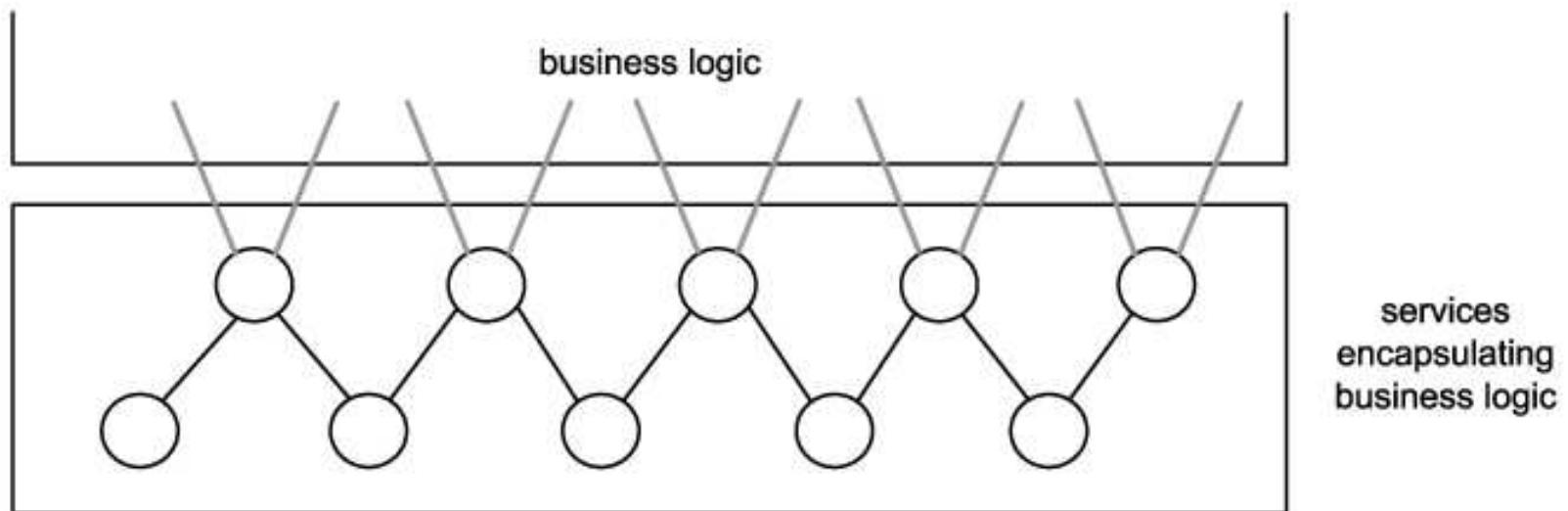


aus <http://java.sun.com/javaee/5/docs/tutorial>

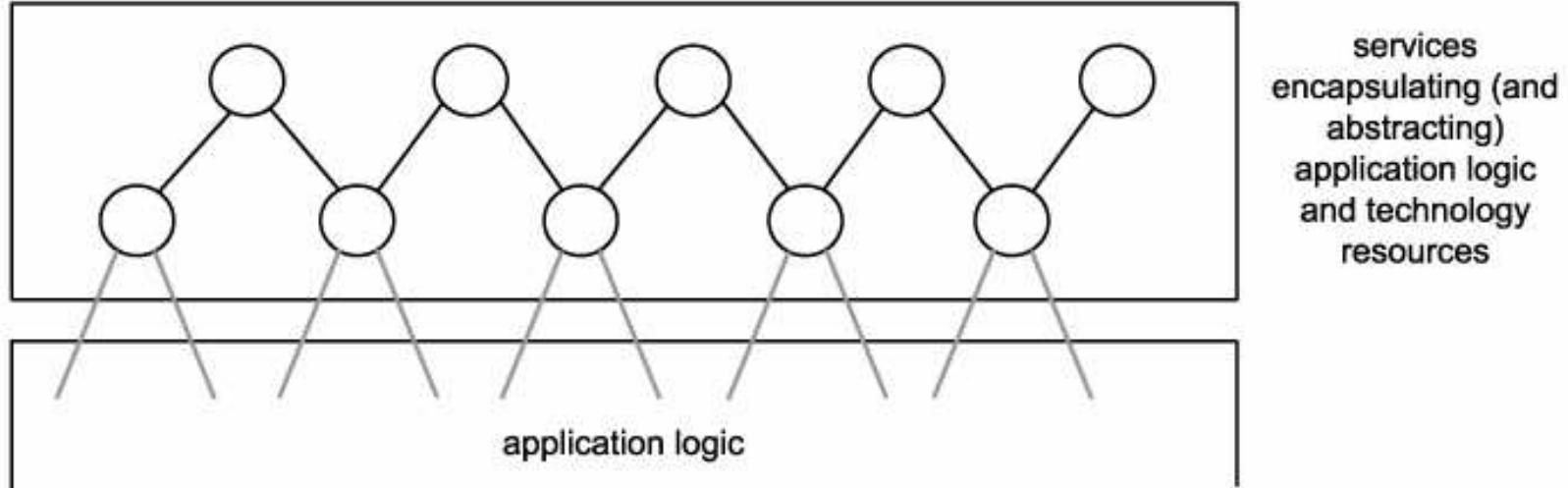
## Nabe-Speiche („Hub- and Spoke“)-Modell



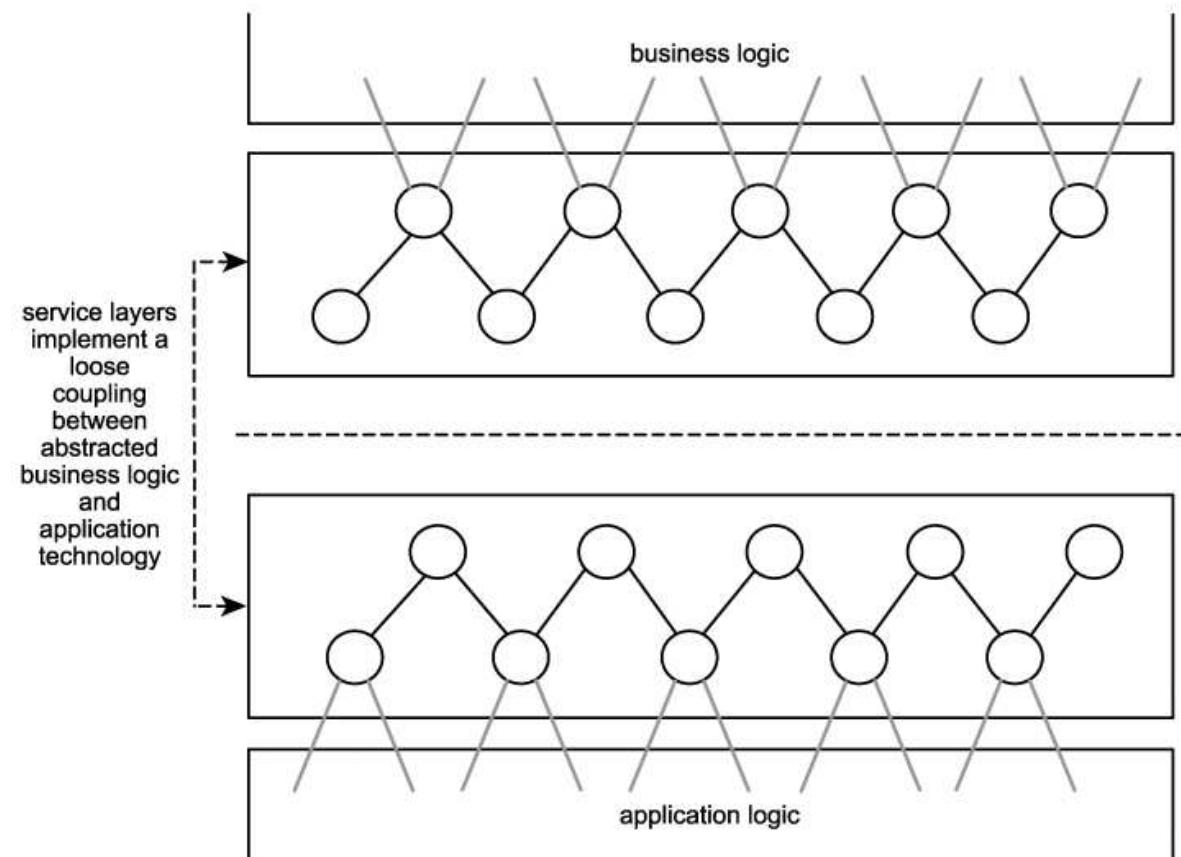
# Geschäftsprozessmodellierung mit SOA



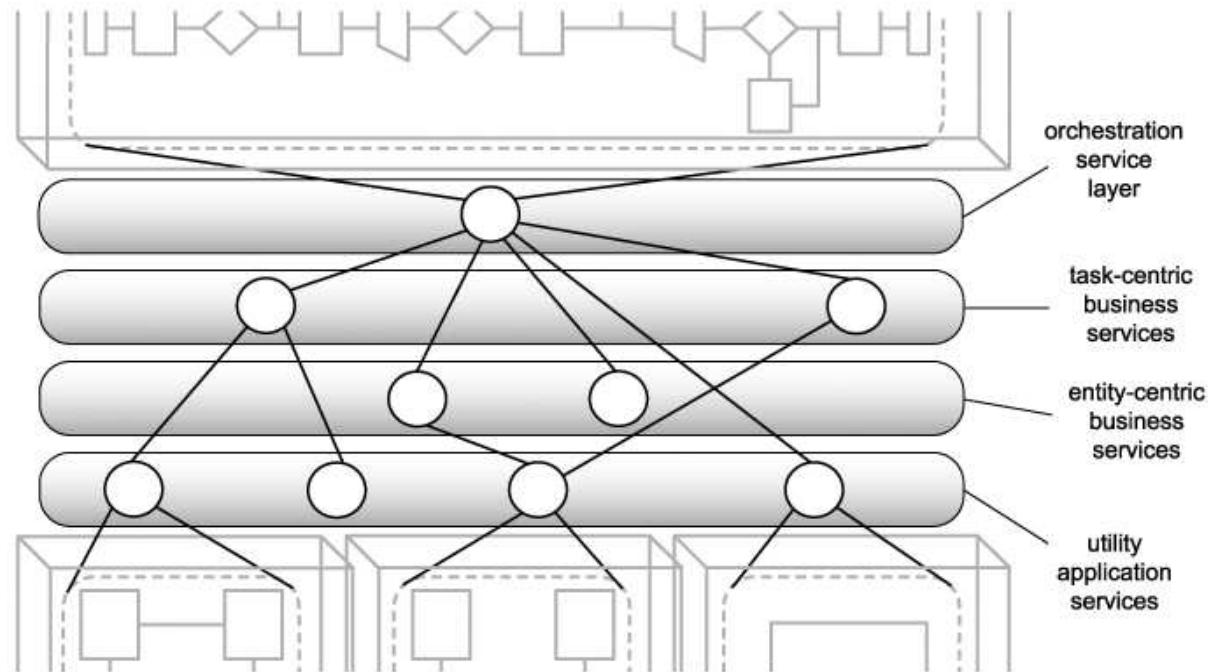
## Integration existierender Anwendungen in eine SOA



# Lose Kopplung von Anwendungen und Geschäftsprozessen



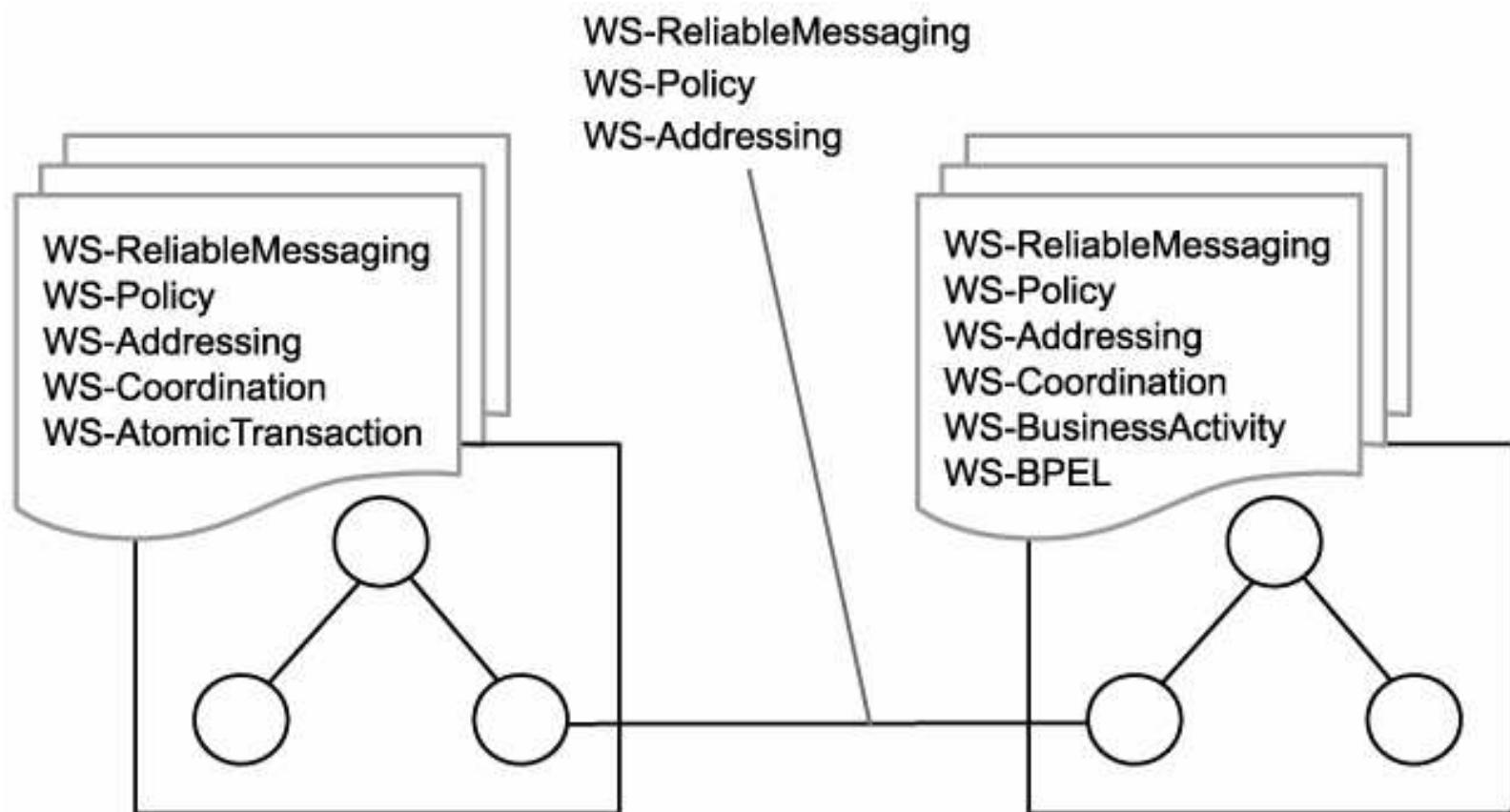
## Schichtenmodell für eine SOA



## 2. Webservices

- Auf XML und anderen offenen Standards basierende Middleware
- Kernkomponenten
  - Nachrichten-System: **SOAP**, Nutzlast sind XML-Dokumente
  - Schnittstellenspezifikation: XML Schema und **WSDL** (Webservice Description Language)
  - Verzeichnisdienst: **UDDI** (Universal Description, Discovery and Integration)
- Erweiterungen: WS-\* (Security, Transaction, ...)
- Diensteorchestrierung mit WS-BPEL oder anderen Geschäftsprozess-Definitionssprachen

## Web-Services als Baukastensystem



## Die einfachere Alternative im Überblick: RESTful „Webservices“

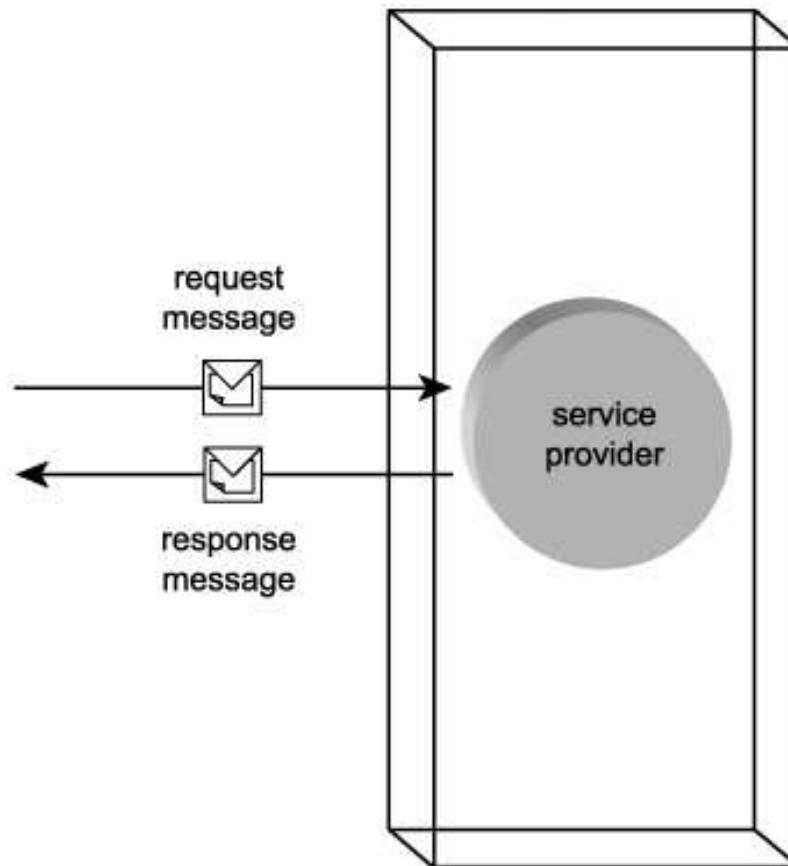
- REST = Representational State Transfer
- Vorbild und technische Basis ist das WWW
- Services bieten Zugriffe auf Web-Ressourcen über URLs
- Operationen sind generisch, nicht Service-spezifisch:  
HTTP-Requesttypen GET, POST, PUT, DELETE
- Ressorcen werden vom Dienstnutzer mit GET abgerufen und enthalten Links auf weitere Ressorcen  
(wie WWW aber XML-Dokumente als Ressorcen-Repräsentanten und Links nach XML-Xlink-Standard)
- Beispiel: Die Webservices von Amazon (<http://aws.amazon.com>) z.B. der Bezahl-dienst FPS, sind alle als RESTful-Services (aber auch via SOAP) nutzbar
- Mehr dazu: <http://www.ibm.com/developerworks/webservices/library/ws-rest1>

## Nutzungsmodell für Webservices

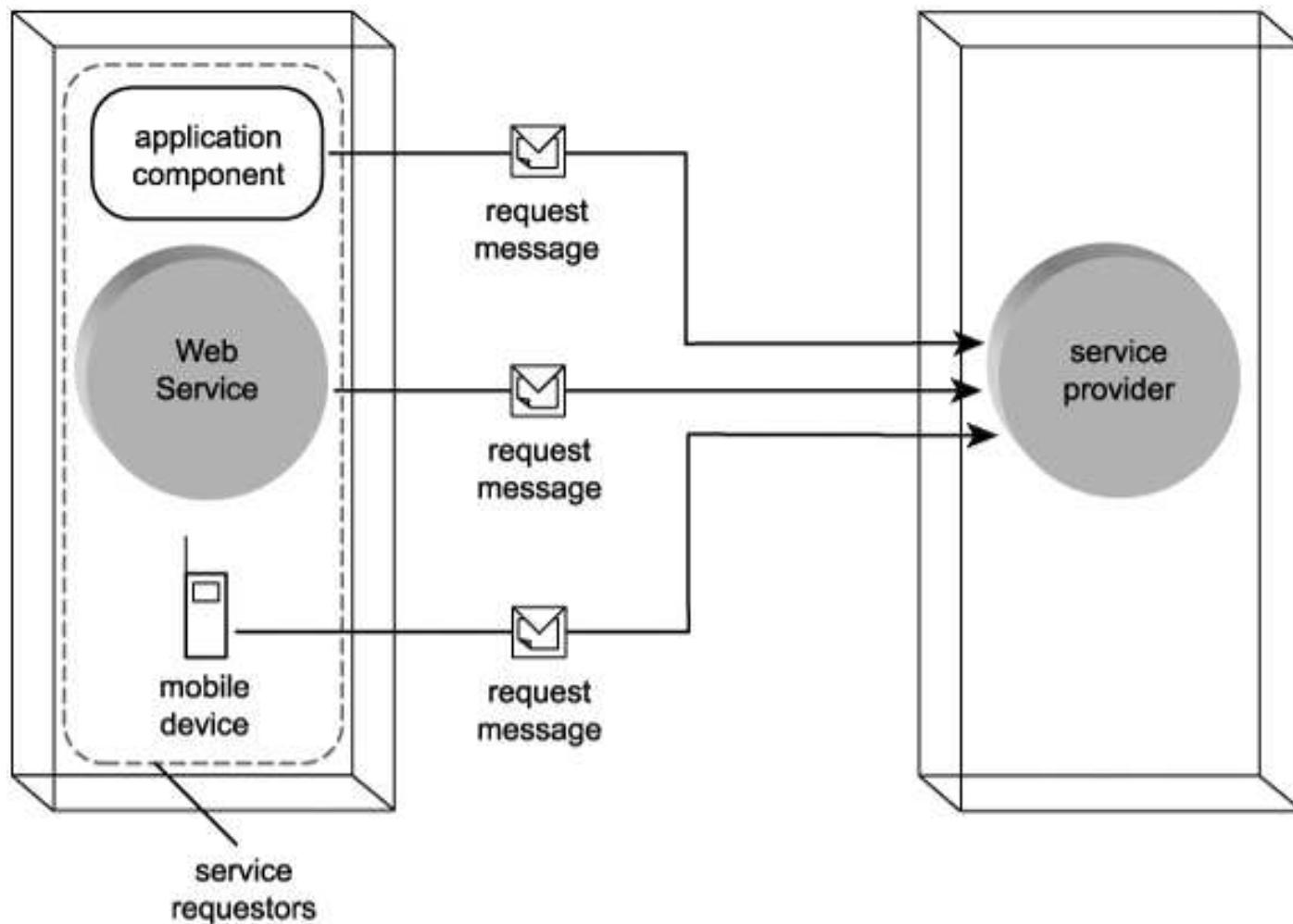
- Jeder Service definiert eine oder mehrere Operationen
- Aktivierung einer Operationen durch Dienstnutzer:  
Senden einer Nachricht an den Dienstanbieter
- Für jede Operation werden festgelegt:
  - ein Muster für den Nachrichtenaustausch und
  - die Typen der ausgetauschten Nachrichten
- Schnittstellenbeschreibung (Nachrichtentypen, Operationen) in WSDL, maschinenlesbar und öffentlich verfügbar
- Nutzer
  - rufen die Schnittstellenbeschreibung ab (z.B. via HTTP-GET)
  - nutzen den Service durch Senden von Nachrichten und Verarbeiten der Antwortnachrichten gemäß Schnittstellenbeschreibung

# Web-Services - Grundlagen

Die Rolle des Dienstanbieters



## Dienstnutzer



## Detaillierte Rollenspezifikation

- Dienstanbieter-Entität (Service Provider Entity)
- Dienstanbieter-Agent (Service Provider Agent ( $\neq$  Service Agent!))
- Dienstnutzer-Entität (Service Requestor Entity)
- Dienstnutzer-Agent (Service Requestor Agent)

Ein Webservice ist immer ein Dienstanbieter.

Ein Webservice kann als Dienstnutzer agieren!

## 2.2 WSDL - Web Service Description Language

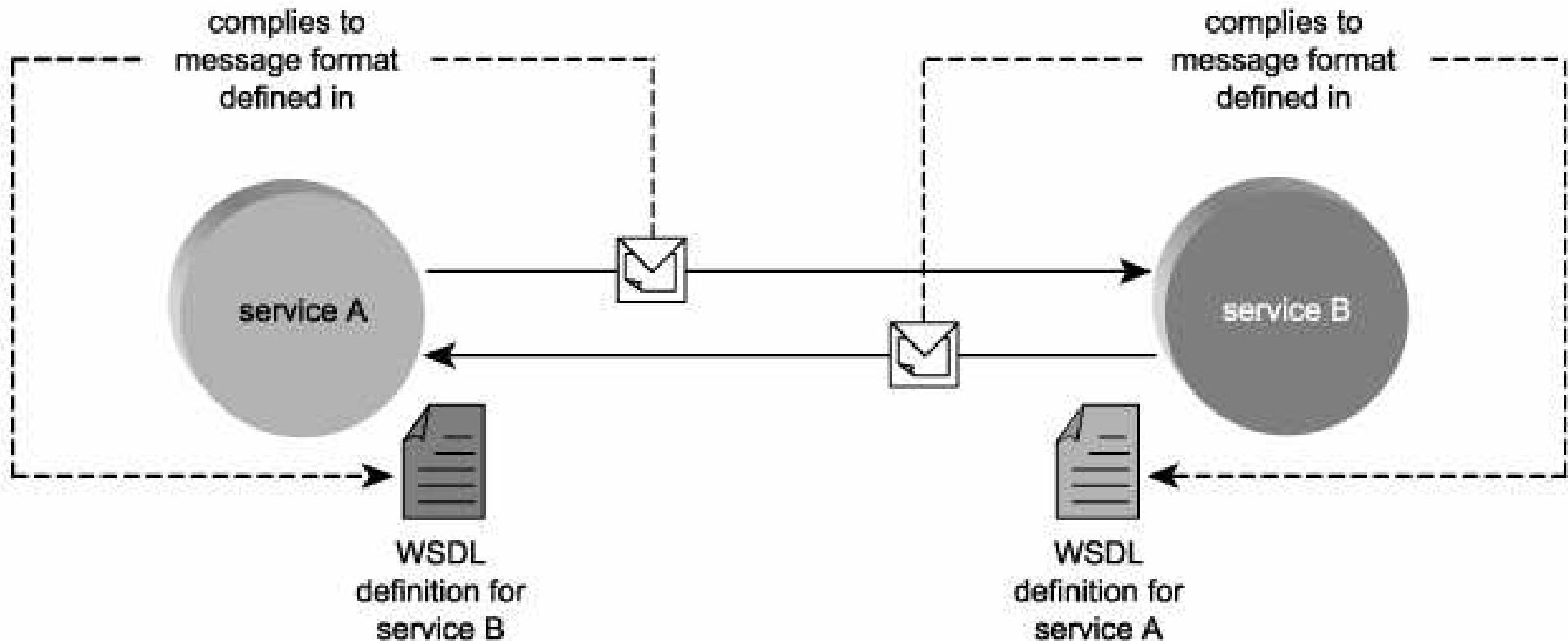
- XML-basierte Sprache zur Beschreibung der Schnittstelle eines Webservice
- Aktuelle Version: W3C-Standard WSDL 2.0

<http://www.w3.org/TR/wsdl120>

Meist wird aber noch Version 1.1 verwendet!

- WSDL unterstützt SOAP-basierte, ab 2.0 auch RESTful Webservices

## Dienstbeschreibung mit WSDL



## WSDL-Aufbau

- Schnittstellenbeschreibung ist ein XML-Dokument
- Wurzelement ist Container für Schema-Definitionen und WSDL-Definitionen

Version 1.1:      <definitions> ... </definitions>

Version 2.0:      <description> ... </description>

- Dem Wurzelement ist ein URI als Attribut „targetNamespace“ zugeordnet.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:description
    targetNamespace="http://example.org/TicketAgent.wsdl20"
    xmlns:wsdl="http://www.w3.org/ns/wsdl" ...>
```

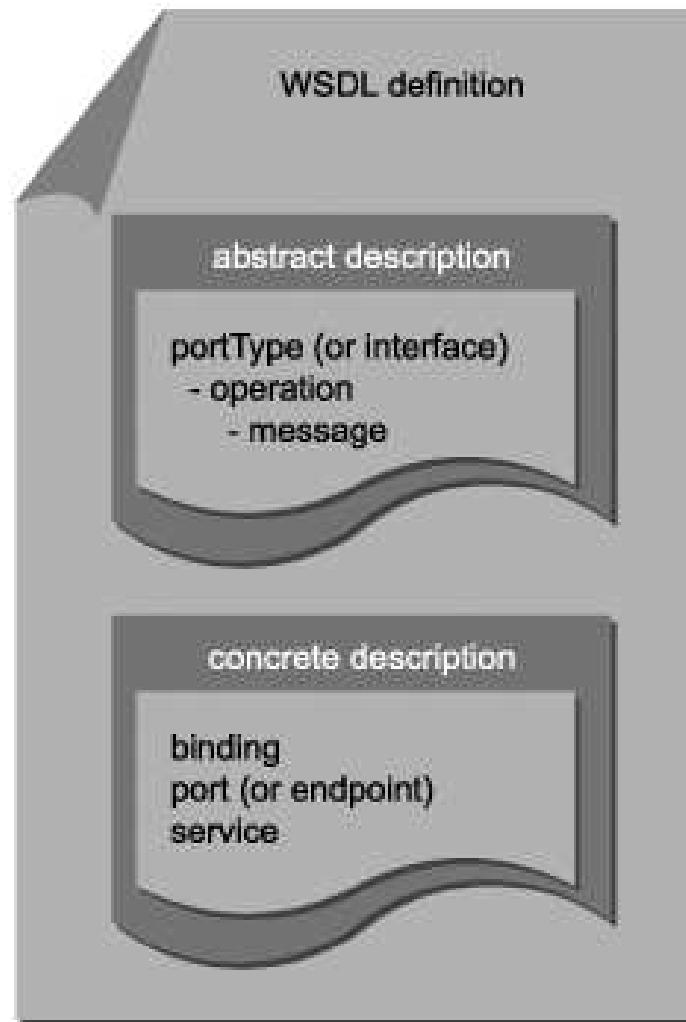
Dies sollte eine dereferenzierbarer URL sein, unter dem eine Schnittstellenbeschreibung erreichbar ist.

## **WSDL-Aufbau**

Die Beschreibung ist zweigeteilt:

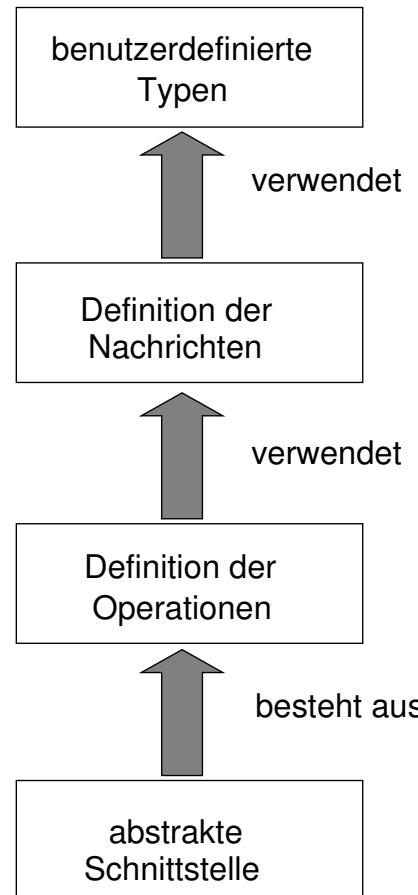
- Abstrakte Definition
  - beschreibt die Operationen des Webservice durch die jeweils zu verwendenden Nachrichtentypen
  - gibt keine Auskunft darüber, wo und wie ein Diensteanbieter erreichbar ist
- Konkrete Definition
  - beschreibt, wie und wo der Service erreichbar ist
  - Service kann über mehrere Transportdienste genutzt werden

## WSDL-Aufbau



# Abstrakte Schnittstellenbeschreibung

Die Definitionen bilden eine 4-stufige Hierarchie:



## Syntaktischer Aufbau der abstrakten Schnittstellenbeschreibung

Der typische Aufbau reflektiert die oben dargestellten Beziehungen der Definitionen:

```
<types>          ...      </types>  
  
<message ...>    ...      </message>  
.  
.  
.  
  
<message ...>    ...      </message>  
  
<interface ...>  
  <documentation>    ...      </documentation>  
  <operation ...>    ...      </operation>  
.  
.  
.  
  <operation ...>    ...      </operation>  
</interface>
```

## Das interface/portType-Element

Syntax (WSDL 2.0, <http://www.w3.org/TR/wsdl120/#component-Interface>):

```
<description>
  <interface
    name="xs:NCName"
    extends="list of xs:QName"?
    styleDefault="list of xs:anyURI"? >
    <documentation />*
    [ <fault /> | <operation /> ]*
  </interface>
</description>
```

- Version 2.0 verwendet den Elementnamen „interface“, Version 1.1 „portType“
- Kern der abstrakten Schnittstellenbeschreibung
- Wesentliche Bausteine: Operationen und Fehlerbehandlung
- Jedes „interface“ benötigt einen eindeutigen Namen
- Vererbungskonzept: Ein Interface kann ein anderes erweitern

- Anwendungsspezifische Typen definiert man besser nicht direkt im WSDL-Dokument:
  - separate XML-Schema-Dateien
  - Import in das WSDL-Dokument <xsd:import schemaLocation=...>
- Operationen sind durch Nachrichten definiert:
  - input message
  - output message

## WSDL Kommunikationsmuster (MEPs Message Exchange Patterns)

- In-Only
- Robust In-Only
- In-Out
- In-Optional-Out
- Out-Only
- Robust Out-Only
- Out-In
- Out-Optional-In

## WSDL Ausnahmebehandlung

- Mit „fault“-Elementen definiert man Nachrichten, die bei Laufzeitfehlern generiert und verschickt werden
- Die Fehlersemantik ist dabei teilweise Programmiersprachen-unabhängig auf der Webservice-Ebene definiert, z.B. bei Nichteinhaltung von maschinell überprüfbarer Webservice-Richtlinien
- Die Fehlerbehandlung betrifft auch die Bindung zwischen Implementierungssprache und Webservice
  - das Versenden von Fehlernachrichten bei Laufzeitausnahmen
  - das Werfen von Ausnahmen beim Empfang von Fehlernachrichten

## Fault-Beispiel (WSDL 1.1)

```
<definitions ...>

<message name="empty"/>
<message name="InsufficientFundsFault">
    <part name="balance" type="xsd:int"/>
</message>

<portType name="Bank">
    <operation name="throwException">
        <input message="tns:empty"/>
        <output message="tns:empty"/>
        <fault name="fault" message="tns:InsufficientFundFault"/>
    </operation>
</portType>
...
</definitions>
```

## WSDL Ausnahmebehandlung

Beispiel: Java Source

```
public class InsufficientFundFault extends java.lang.Exception {  
    private int balance;  
    public int getBalance() {  
        return this.balance;  
    }  
  
    public InsufficientFundFault() {  
    }  
  
    public InsufficientFundFault(int balance) {  
        this.balance = balance;  
    }  
}
```

## WDSL-Beispiel: Abstrakte Schnittstelle Version 2.0

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:description
    targetNamespace="http://example.org/TicketAgent.wsdl20"
    xmlns:xsTicketAgent="http://example.org/TicketAgent.xsd"
    xmlns:wsdl="http://www.w3.org/ns/wsdl"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/ns/wsdl http://www.w3.org/2007/06/wsdl/wsdl20.xsd">

    <wsdl:types>
        <xs:import schemaLocation="TicketAgent.xsd"
            namespace="http://example.org/TicketAgent.xsd" />
    </wsdl:types>

    <wsdl:interface name="TicketAgent">
        <wsdl:operation name="listFlights"
            pattern="http://www.w3.org/ns/wsdl/in-out">
            <wsdl:input element="xsTicketAgent:listFlightsRequest"/>
            <wsdl:output element="xsTicketAgent:listFlightsResponse"/>
        </wsdl:operation>
    </wsdl:interface>
</wsdl:description>
```

```
</wsdl:operation>

<wsdl:operation name="reserveFlight"
    pattern="http://www.w3.org/ns/wsdl/in-out">
    <wsdl:input element="xsTicketAgent:reserveFlightRequest"/>
    <wsdl:output element="xsTicketAgent:reserveFlightResponse"/>
</wsdl:operation>
</wsdl:interface>
</wsdl:description>
```

## Konkrete Definition

- Webservices sind prinzipiell Transport-unabhängig
- Als Transportdienste werden z.B. HTTP, TCP, E-Mail(SMTP, POP, IMAP) genutzt
- Die konkrete Schnittstellenbeschreibung definiert
  - welche Kommunikationsprotokolle der Service unterstützt und
  - unter welcher Adresse (Kommunikations-Endpunkt) der Service für einen bestimmten Transport erreichbar ist
- Ein vom Service unterstütztes Transportprotokoll zusammen mit dem Endpunkt (Transport-spezifische Adresse) ist ein „binding“-Element

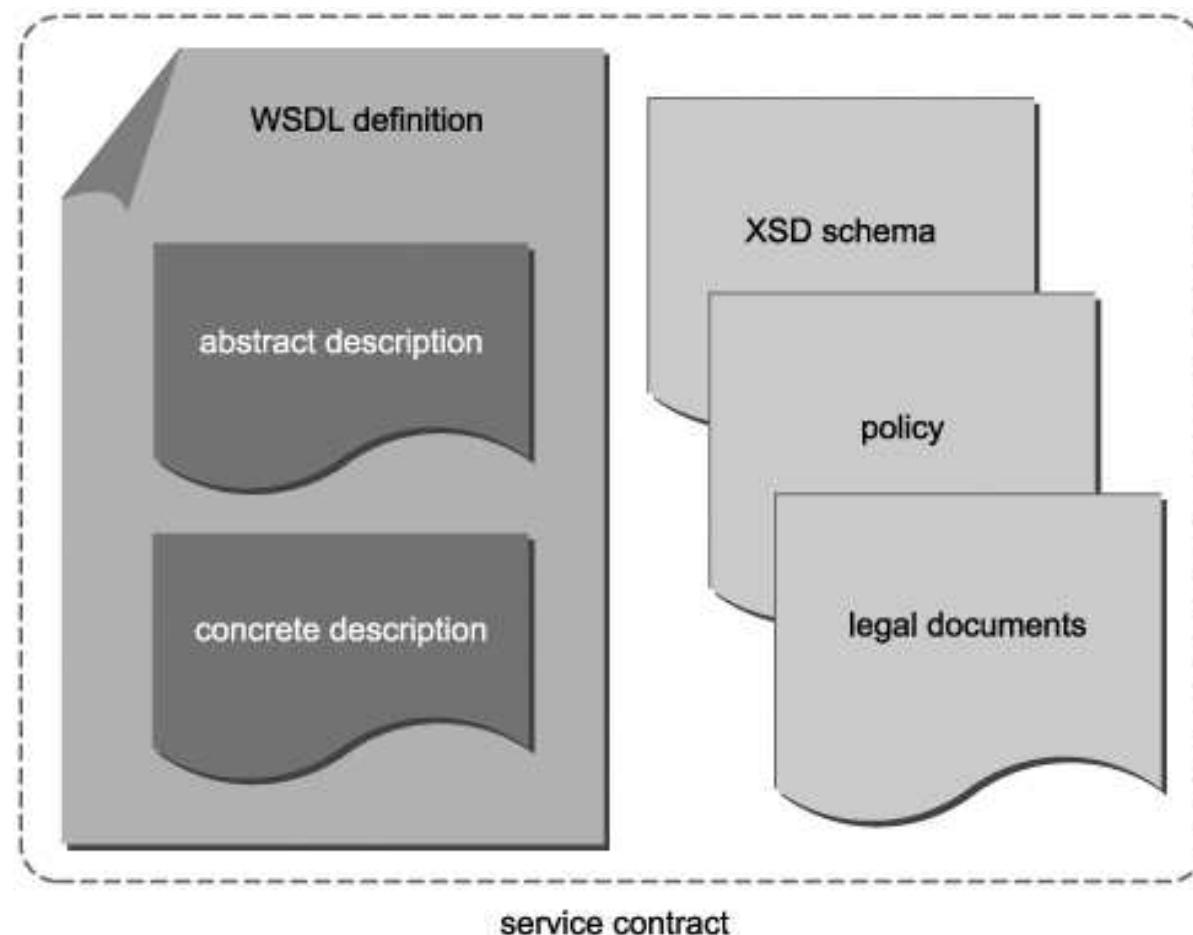
## WDSL-Beispiel - Version 1.1

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="InvoiceProcessing"
  targetNamespace="http://www.xmltc.com/railco/transform/wsdl/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:inv="http://www.xmltc.com/railco/invoice/schema/"
  xmlns:invs="http://www.xmltc.com/railco/invoiceservice/schema/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.xmltc.com/railco/transform/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <types>
    <xsd:schema targetNamespace="http://www.xmltc.com/railco/invoiceservice/schema/">
      <xsd:import namespace="http://www.xmltc.com/railco/invoice/schema/" schemaLocation="Invoice.xsd"/>
      <xsd:element name="SubmitInvoiceType">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="ContextID" type="xsd:integer"/>
            <xsd:element name="InvoiceLocation" type="xsd:string"/>
            <xsd:element name="InvoiceDocument" type="inv:InvoiceType"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </types>
</definitions>
```

```
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
</types>
<message name="receiveSubmitMessage">
    <part name="RequestParameter" element="invs:SubmitInvoiceType"/>
</message>
<portType name="InvoiceProcessingInterface">
    <documentation>
        Initiates the Invoice Processing process.
        Requires either the invoice document location or the document.
    </documentation>
    <operation name="Submit">
        <input message="tns:receiveSubmitMessage"/>
    </operation>
</portType>
<binding name="InvoiceProcessingBinding" type="tns:InvoiceProcessingInterface">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="Submit">
        <soap:operation soapAction="http://www.xmltc.com/soapaction"/>
        <input>
```

```
    <soap:body use="literal"/>
  </input>
</operation>
</binding>
<service name="InvoiceProcessingService">
  <port binding="tns:InvoiceProcessingBinding" name="InvoiceProcessingPort">
    <soap:address location="http://www.serviceoriented.ws/railco/wsdl/" />
  </port>
</service>
</definitions>
```

## Dienstvertrag: WSDL + XSD + Richtlinien + Weitere Vereinbarungen



## 2.3 SOAP

Asynchroner Nachrichtenaustausch ist in einer SOA **die grundlegende Operation**

⇒ Leistungsfähiges Nachrichtenkonzept ist von essentieller Bedeutung für SOA

Anforderungen:

- Erweiterbarkeit
- Plattformunabhängigkeit
- Unterstützung gängiger Kommunikationsmuster

Effizienz?

„nice to have“!

## SOAP-Grundlagen

- SOAP ist ein „high level“-Transportdienst
- kann z.B. mit HTTP oder TCP als „low level“-Transport kombiniert werden
- W3C-Standard, aktuelle Version: 1.2
- SOAP-Nachrichten sind XML-Dokumente:  
*Envelope* enthält *Header* (optional) und *Body* („Nutzlast“ der Nachricht)
- Nutzlast einer SOAP-Nachricht: anwendungsspezifisch definiertes XML-Dokument  
(→ Synergieeffekte durch Nutzung der XML-Infrastruktur)
- SOAP beruht auf Einweg-Versendung von Nachrichten, bzw. dem Muster „Auftrag-Antwort“.  
Komplexere Nachrichten-Austauschmuster werden oberhalb der SOAP-Ebene implementiert!
- Neben dem Austausch von XML-Dokumenten („document style“) auch:

SOAP RPC („rpc style“): Auftragsachricht + (Antwort oder Fehlernachricht)

## Beispiel für eine SOAP-Nachricht: Gesamtstruktur

```
<?xml version='1.0' ?>

<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">

<env:Header>
    .
    .
    .
</env:Header>

<env:Body>
    .
    .
    .
</env:Body>

</env:Envelope>
```

## Beispiel für eine SOAP-Nachricht: Header

```
<env:Header>

<m:reservation
    xmlns:m="http://travelcompany.example.org/reservation"
    env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
    env:mustUnderstand="true">
    <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
    <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
</m:reservation>

<n:passenger
    xmlns:n="http://mycompany.example.com/employees"
    env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
    env:mustUnderstand="true">
    <n:name>Ake Jogvan Oyvind</n:name>
</n:passenger>

</env:Header>
```

## Beispiel für eine SOAP-Nachricht: Body

```
<env:Body>

<p:itinerary xmlns:p="http://travelcompany.example.org/reservation/travel">
  <p:departure>
    <p:departing>New York</p:departing>
    <p:arriving>Los Angeles</p:arriving>
    <p:departureDate>2001-12-14</p:departureDate>
    <p:departureTime>late afternoon</p:departureTime>
    <p:seatPreference>aisle</p:seatPreference>
  </p:departure>
  <p:return> ... </p:return>
</p:itinerary>

<q:lodging
  xmlns:q="http://travelcompany.example.org/reservation/hotels">
  <q:preference>none</q:preference>
</q:lodging>

</env:Body>
```

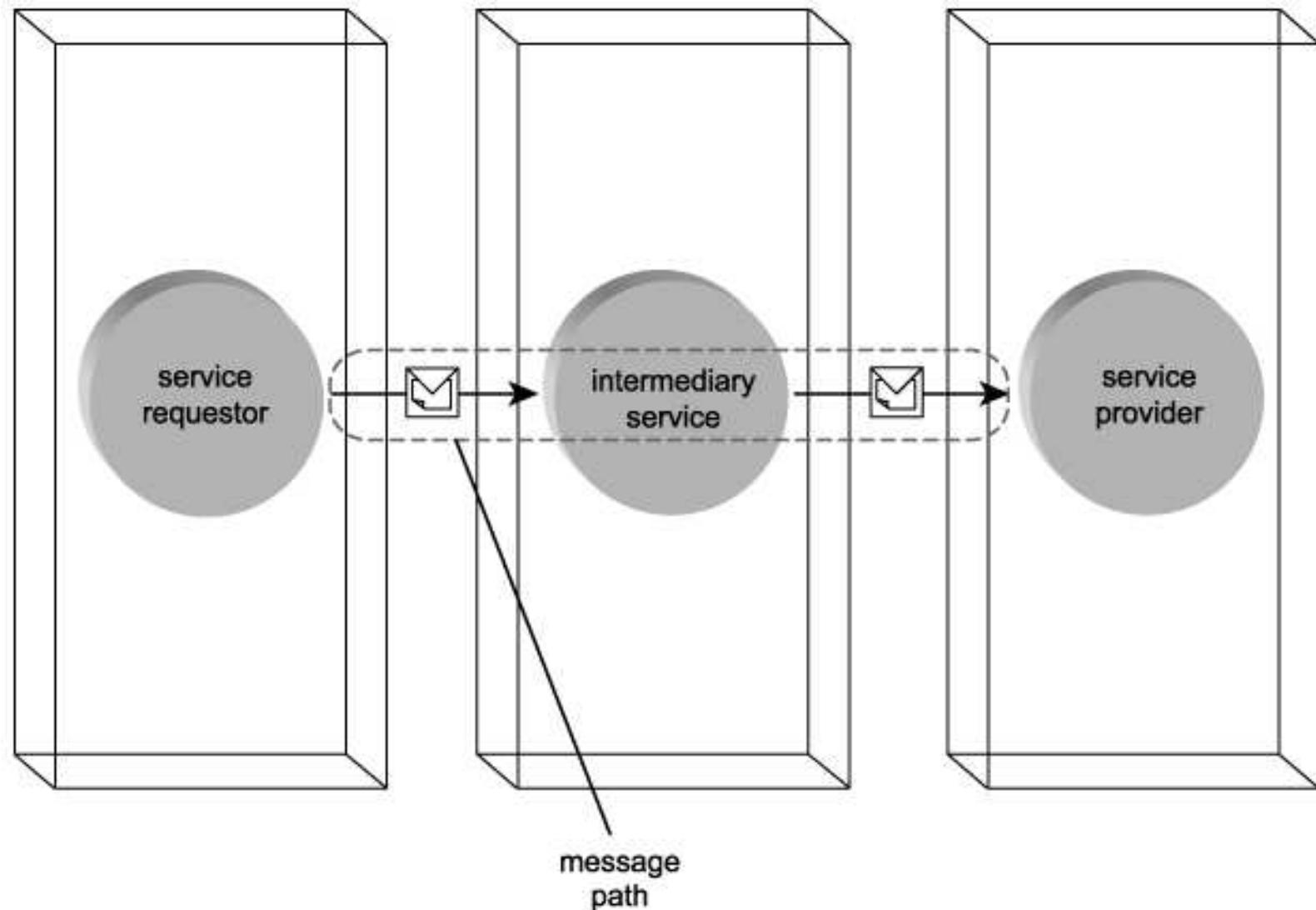
## SOAP-Header

- SOAP-Header enthalten Metadaten, die die Verarbeitung der Nachricht beeinflussen
- Inhalt und Semantik sind im SOAP-Standard **nicht** definiert

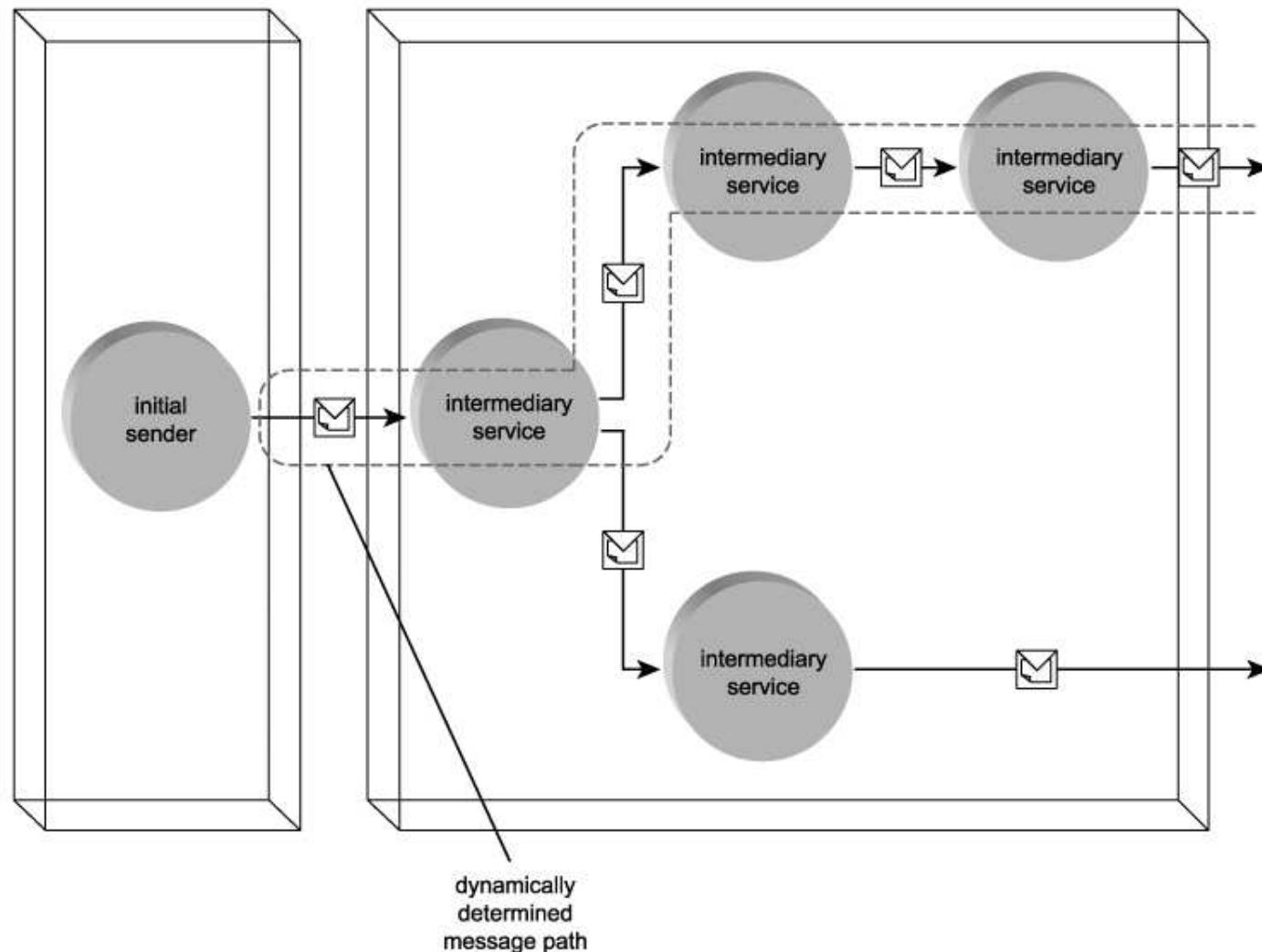
Die Header sind ein Erweiterungskonzept für SOAP:

- Anwendungs-spezifische Erweiterungen
- Erweiterungen im Rahmen des WS-\*-"Baukastens"

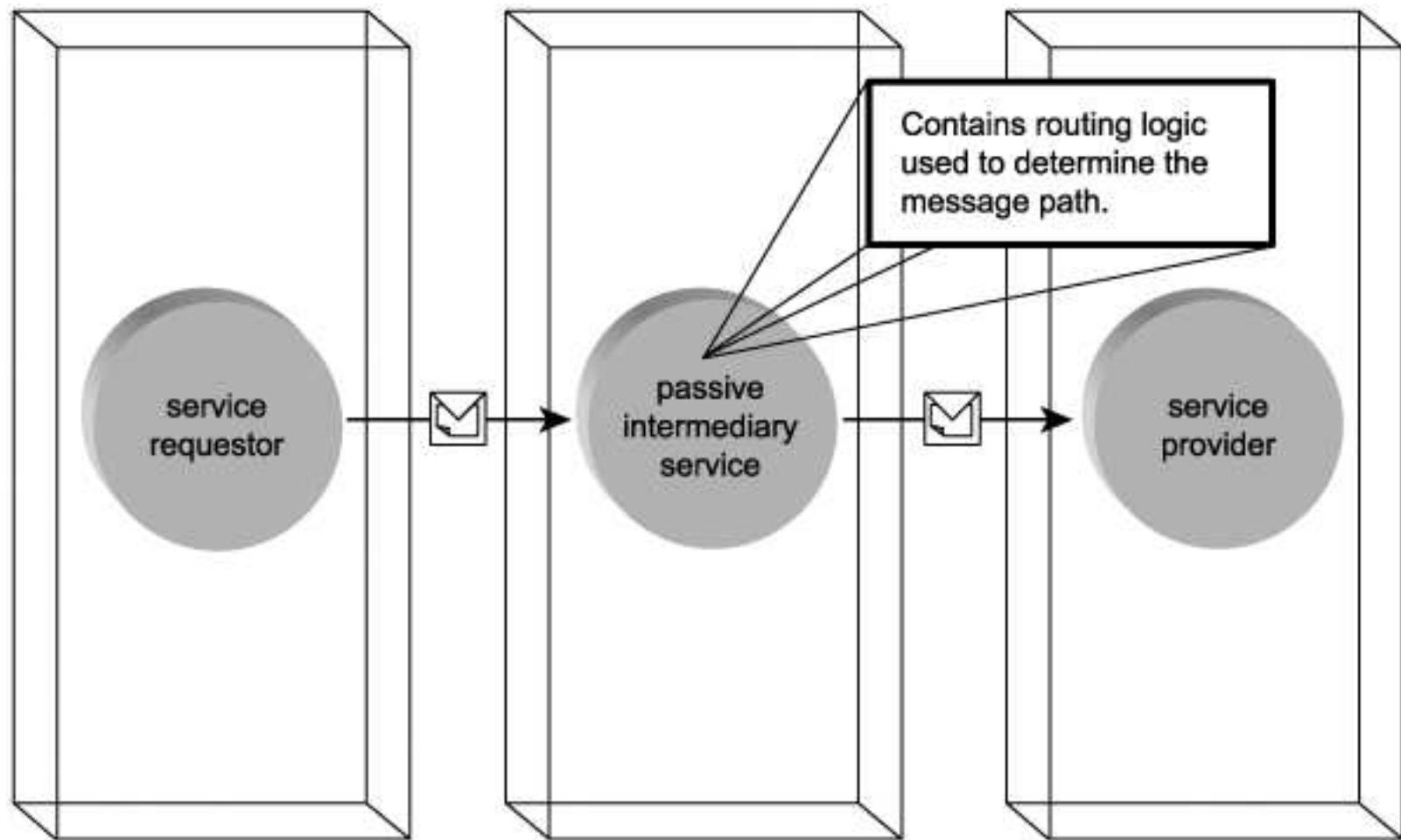
## SOAP Nachrichtenpfade: Pfad mit Zwischendienst



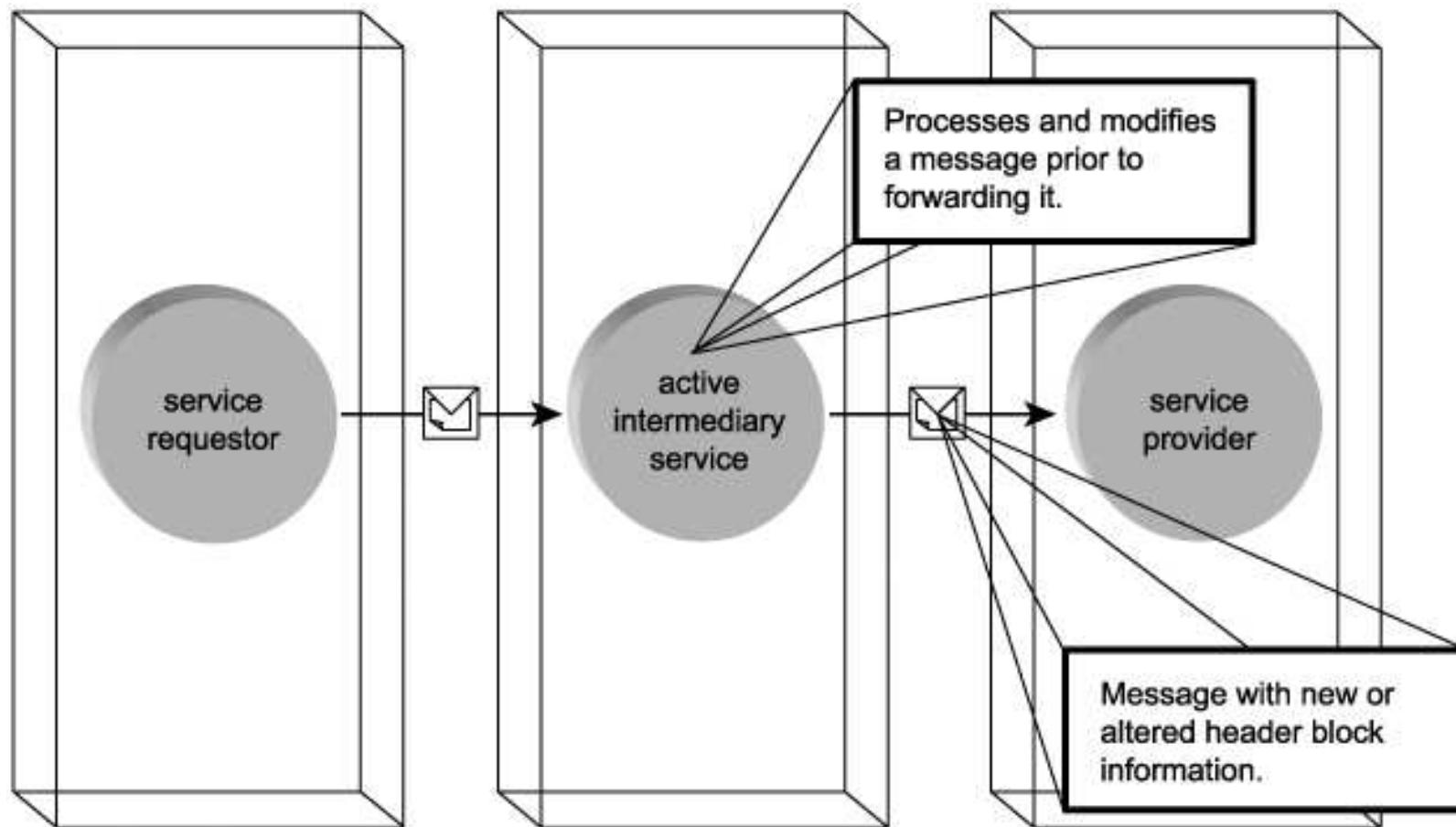
## SOAP unterstützt dynamische Nachrichtenpfade



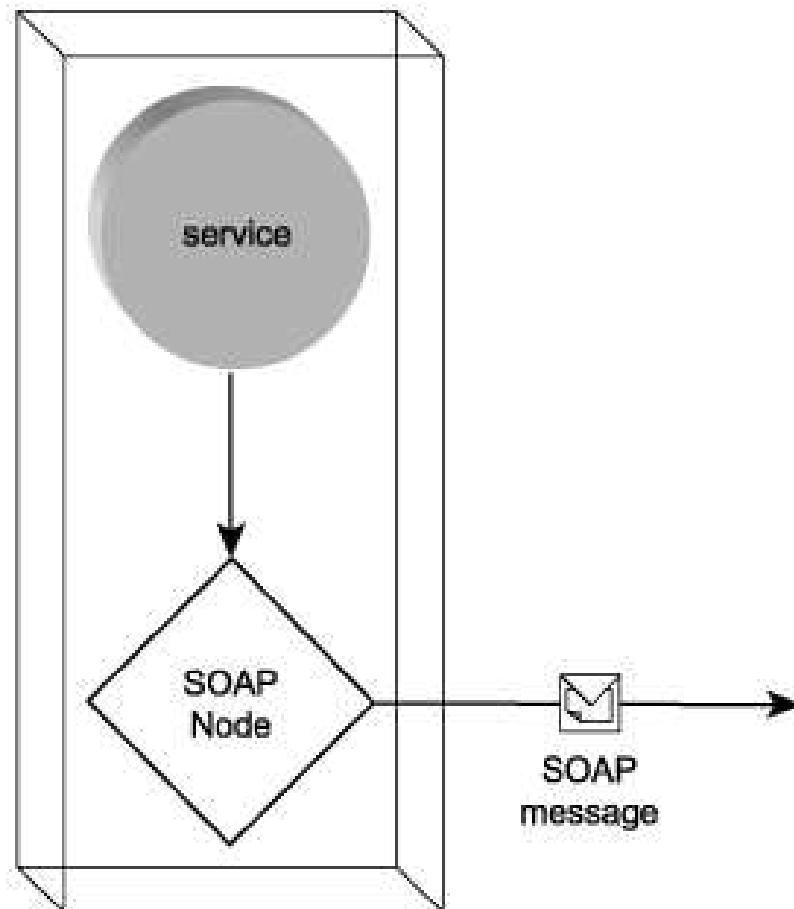
## Passive Intermediäre Services



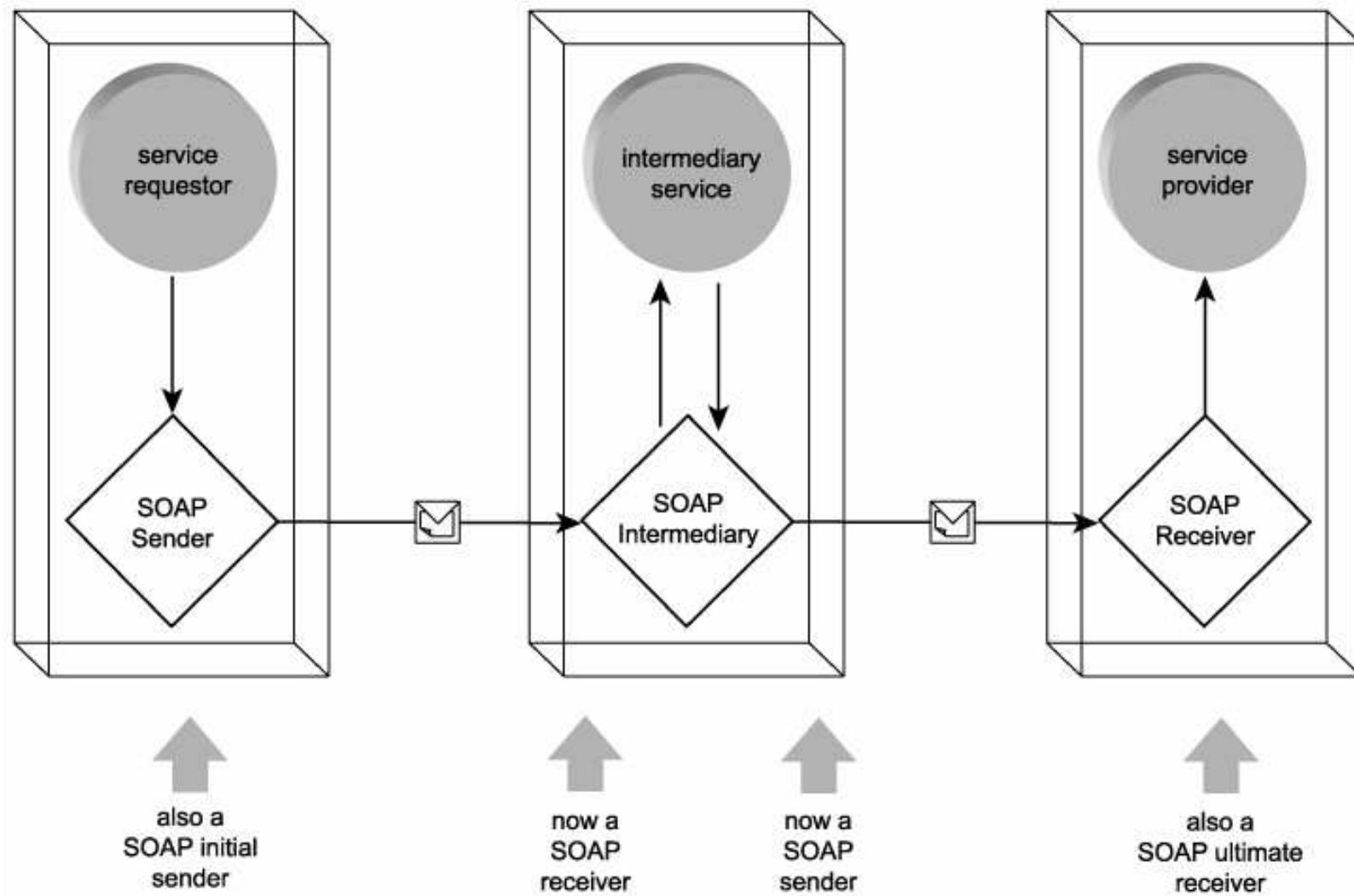
## Aktive Intermediäre Services



## SOAP-Knoten und Dienstanbieter



## Rollen bezüglich SOAP-Vermittlung



## Beispiel für Headernutzung: WS-ReliableMessaging

```
<soapenv:Envelope>
  <soapenv:Header>
    <wsa:MessageID> ... </wsa:MessageID>
    <wsa:To> ... </wsa:To>
    <wsa:Action> ... </wsa:Action>
    <wsa:From> ... </wsa:From>
    <wsrm:Sequence soapenv:mustUnderstand="1">
      <wsu:Identifier>
        http://www.ibm.com/guid/a8f7151a091b50a42b38e04437774e11
      </wsu:Identifier>
      <wsrm:MessageNumber>1</wsrm:MessageNumber>
    </wsrm:Sequence>
  </soapenv:Header>
  <soapenv:Body> ... </soapenv:Body>
</soapenv:Envelope>
```

Namespaces: wsu=WS-Utilities, wsa=WS-Adressing, wsrm=WS-ReliableMessaging

## WS-ReliableMessaging: Letzte Nachricht einer Sequenz

```
<soapenv:Envelope>
  <soapenv:Header>
    <wsa:MessageID> ... </wsa:MessageID>
    <wsa:To> ... </wsa:To>
    <wsa:Action> ... </wsa:Action>
    <wsa:From> ... </wsa:From>
    <wsrm:Sequence soapenv:mustUnderstand="1">
      <wsu:Identifier>
        http://www.ibm.com/guid/a8f7151a091b50a42b38e04437774e11
      </wsu:Identifier>
      <wsrm:MessageNumber>4</wsrm:MessageNumber>
      <wsrm:LastMessage/>
    </wsrm:Sequence>
  </soapenv:Header>
  <soapenv:Body> ... </soapenv:Body>
</soapenv:Envelope>
```

## WS-ReliableMessaging: Bestätigung für partiellen Empfang einer Sequenz

```
<soapenv:Envelope>
  <soapenv:Header>
    <wsa:MessageID> ... </wsa:MessageID>
    <wsa:To> ... </wsa:To>
    <wsa:Action> ... </wsa:Action>
    <wsa:From> ... </wsa:From>
    <wsrm:SequenceAcknowledgement>
      <wsu:Identifier>
        http://www.ibm.com/guid/a8f7151a091b50a42b38e04437774e11
      </wsuu:Identifier>
      <wsrm:AcknowledgementRange Lower="1" Upper="2"/>
      <wsrm:AcknowledgementRange Lower="4" Upper="4"/>
      <wsrm:Nack>3</wsrm:Nack>
    </wsrm:SequenceAcknowledgement>
  </soapenv:Header>
  <soapenv:Body> ... </soapenv:Body> (response body)
</soapenv:Envelope>
```

## WS-ReliableMessaging: Bestätigung für kompletten Empfang einer Sequenz

```
<soapenv:Envelope>
  <soapenv:Header>
    <wsa:MessageID> ... </wsa:MessageID>
    <wsa:To> ... </wsa:To>
    <wsa:Action> ... </wsa:Action>
    <wsa:From> ... </wsa:From>
    <wsrm:SequenceAcknowledgement>
      <wsuu:Identifier>
        http://www.ibm.com/guid/a8f7151a091b50a42b38e04437774e11
      </wsu:Identifier>
      <wsrm:AcknowledgementRange Lower="1" Upper="4"/>
    </wsrm:SequenceAcknowledgement>
  </soapenv:Header>
  <soapenv:Body> ... </soapenv:Body>
</soapenv:Envelope>
```

## **SOAP-Fehler / Ausnahmebehandlung**

- Für den Fall von Verarbeitungsproblemen beim Nachrichtenempfang
- Definition von Fehlercodes und Fehlermeldungs-Nachrichten („fault“-Sektion im Nachrichtenrumpf)
- Automatische Rücksendung von Fehlermeldungs-Nachrichten folgender Kategorien:

VersionMismatch	Sender benutzt falsche SOAP-Version
MustUnderstand	Verarbeitung eines „Muss-“Headers nicht implementiert
DataEncodingUnknown	Unbekannte Codierung
Sender	Falsches Nachrichtenformat
Receiver	Server-Problem verhindert Bearbeitung

## Beispiel für eine SOAP-Fault-Nachricht

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header/>
  <env:Body><env:Fault>
    <env:Code><env:Value>env:Sender</env:Value></env:Code>
    <env:Reason>
      <env:Text xml:lang="en-US">
        Message does not have necessary info
      </env:Text>
    </env:Reason>
    <env:Role>http://gizmos.com/order</env:Role>
    <env:Detail>
      <P0:order xmlns:P0="http://gizmos.com/orders/">
        Quantity element does not have a value
      </P0:order>
      <P0:confirmation xmlns:P0="http://gizmos.com/confirm">
        Incomplete address: no zip code
      </P0:confirmation>
    </env:Detail>
  </env:Fault></env:Body>
</env:Envelope>
```

## **SOA Fallbeispiel: TLS und sein Zulieferer RailCo**

Firmenprofil RailCo:

- RailCo Ltd. stellt Luftdruckbremsen für die Eisenbahn her, verleiht fallweise Techniker für Installationen und Reparaturen.
- 40 Mitarbeiter, 6 feste IT-Mitarbeiter, diese unterhalten PCs und einige Server. Keine eigene SW-Entwicklung.
- 2-tier Architektur für Buchhaltung und Materialwirtschaft, manuelle Erfassung der Auftrags- und Rechnungsdaten aus nichtelektronischen Geschäftsdokumenten.
- Kontakte-Datenbasis mit Datenbank und Web-Frontend.

## RailCo: Buchführungssoftware

- Klassische Windows-Applikation mit GUI zum Verbuchen und für Reports
- 3 Benutzer, keine Performanceprobleme
- älteres Datenbanksystem, lief meist problemlos
- Nach mehreren Updates auf neuere Windows-Versionen unregelmäßige unerklärliche Abstürze und Fehler der Applikation
- Veraltete PCs, außer der Buchführungssoftware läuft wenig, Produktivität der Mitarbeiter leidet
- Eine notwendige Änderung im Rechnungsstellungsprozess wird von der Software nicht unterstützt und muss manuell ergänzt werden.
- Ziel: Buchführungssoftware auf Server verlagern, SOA zu besseren Erweiterbarkeit

## RailCo: Geschäftsziele und Probleme

- Umsatzrückgang durch Abwanderung von Kunden
- Ursachen-Analyse ergibt: Verwaltungsaufwand zu hoch, ein Konkurrent bietet vergleichbare Produkte günstiger an, hat offensichtlich effizientere Geschäftsprozesse
- Konkurrent nutzt B2B-Schnittstellen der größeren Kunden für elektronische Auftragsabwicklung, hat Buchhaltungspersonal erheblich reduziert und gleichzeitig Bearbeitungszeiten verkürzt
- Konkurrent beliefert jetzt RailCos Hauptkunden TLS.
- Geschäftsziel: Geschäftsprozesse modernisieren, um besser konkurrieren zu können.  
Primäres Ziel: Online-Transaktionen mit TLS ermöglichen.

## **TLS - Transit Line Services**

Geschäftsmodell:

- Hauptgeschäft: Betrieb privater Eisenbahnlinien mit 1800 Angestellten in 4 Städten.
- Technischer Service im Eisenbahnbereich (Verleih technischen Servicepersonals)
- Maschinenbau: Teileherstellung für andere Industrien  
(nach Übernahme eines wichtigen Lieferanten)
- Touristische Angebote  
(in Zusammenarbeit mit Hotels und Fluggesellschaften)

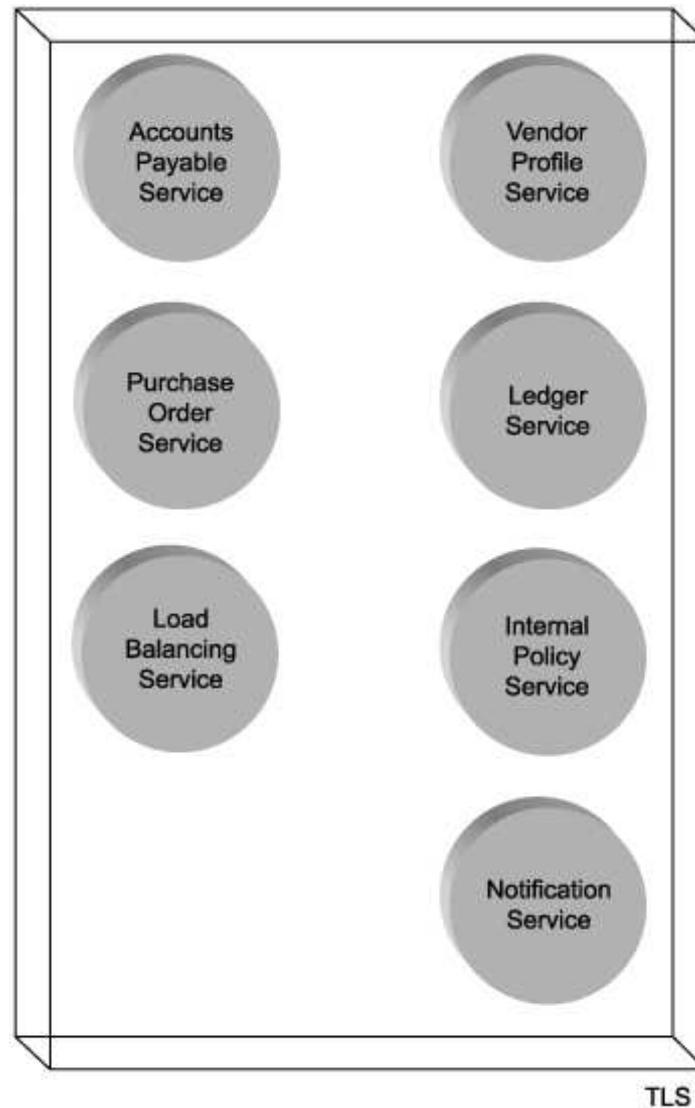
## IT-Infrastruktur von TLS (Ausschnitt)

- 200 IT-Profis, davon 100 fest angestellt.
- Moderne DV-Infrastruktur mit E-Business-Server-Cluster, Mainframes und Windows-Servern mit älteren Client-Server-Anwendungen
- Buchhaltungssystem mit 400 Benutzern, Standardsoftware mit diversen TLS-spezifischen Erweiterungen und Anpassungen. Web-Frontend und Desktop-Analyse- und Reportfunktionen. Webservice-Adapter für verschiedene Module der Anwendung.
- Zeiterfassungssystem für die an Kunden verliehenen Techniker. Manuelle Übertragung der Zeiten in das Buchführungssystem.

## **TLS – Geschäftsziele und Probleme**

- Nach Aquisition zweier Firmen große Integrationsprobleme. IT-Infrastruktur schlecht wartbar, Kostenexplosion im Bereich des Geschäftsprozessautomatisierung, langsame Reaktion auf Änderungen im Geschäftsumfeld
- Lösungsansatz: Auf SOA basierende Neukonzeption.
  - SOA-konforme Entwicklung neuer Anwendungen
  - SOA-konforme Integration vorhandener Anwendungen
  - SOA-Ansatz als generelles Modell für automatisierte Geschäftsprozesse

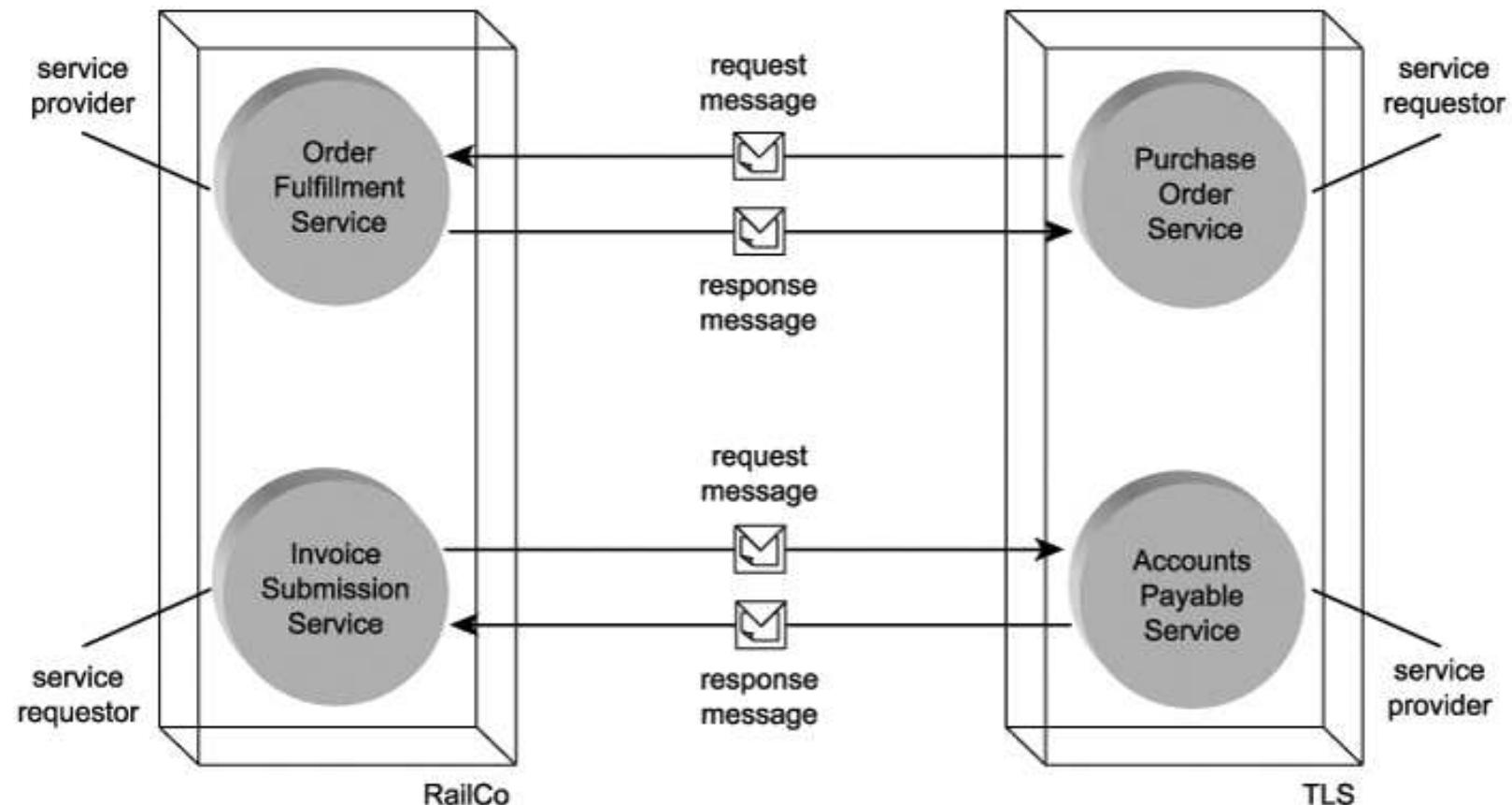
## TLS: Anfangs vorhandene Webservices



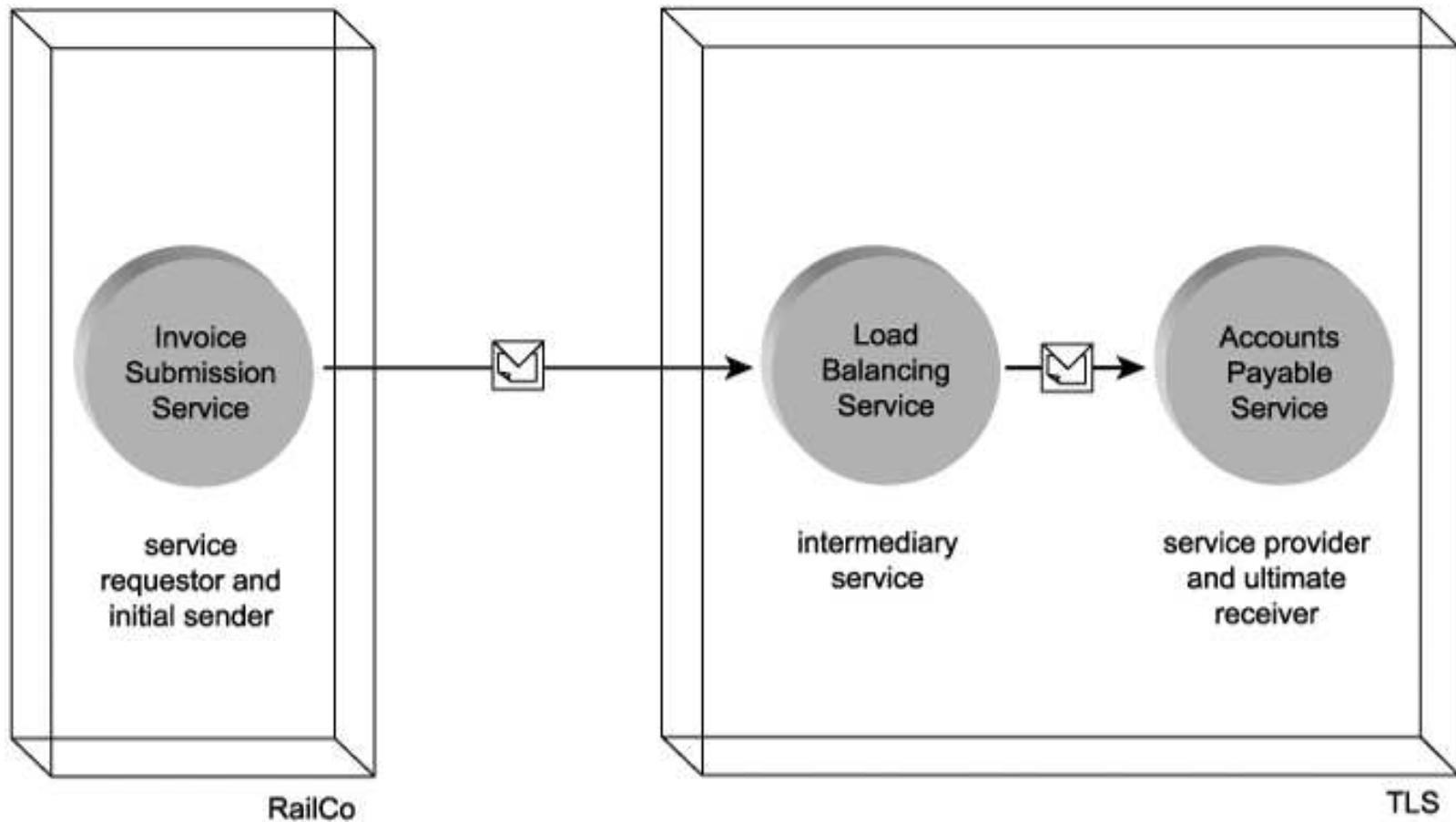
## Anmerkungen zu den BWL-Begriffen:

„accounts payable“	Kreditorenbuchhaltung (Verbindlichkeiten)
„purchase order“	Bestellungen / (Kauf-) Auftragerteilung
„order fulfillment“	Bestellannahme / Auftragsabwicklung
„ledger“	Journal
„purchase ledger“	Wareneingangsbuch
„invoice submission“	Rechnungserstellung

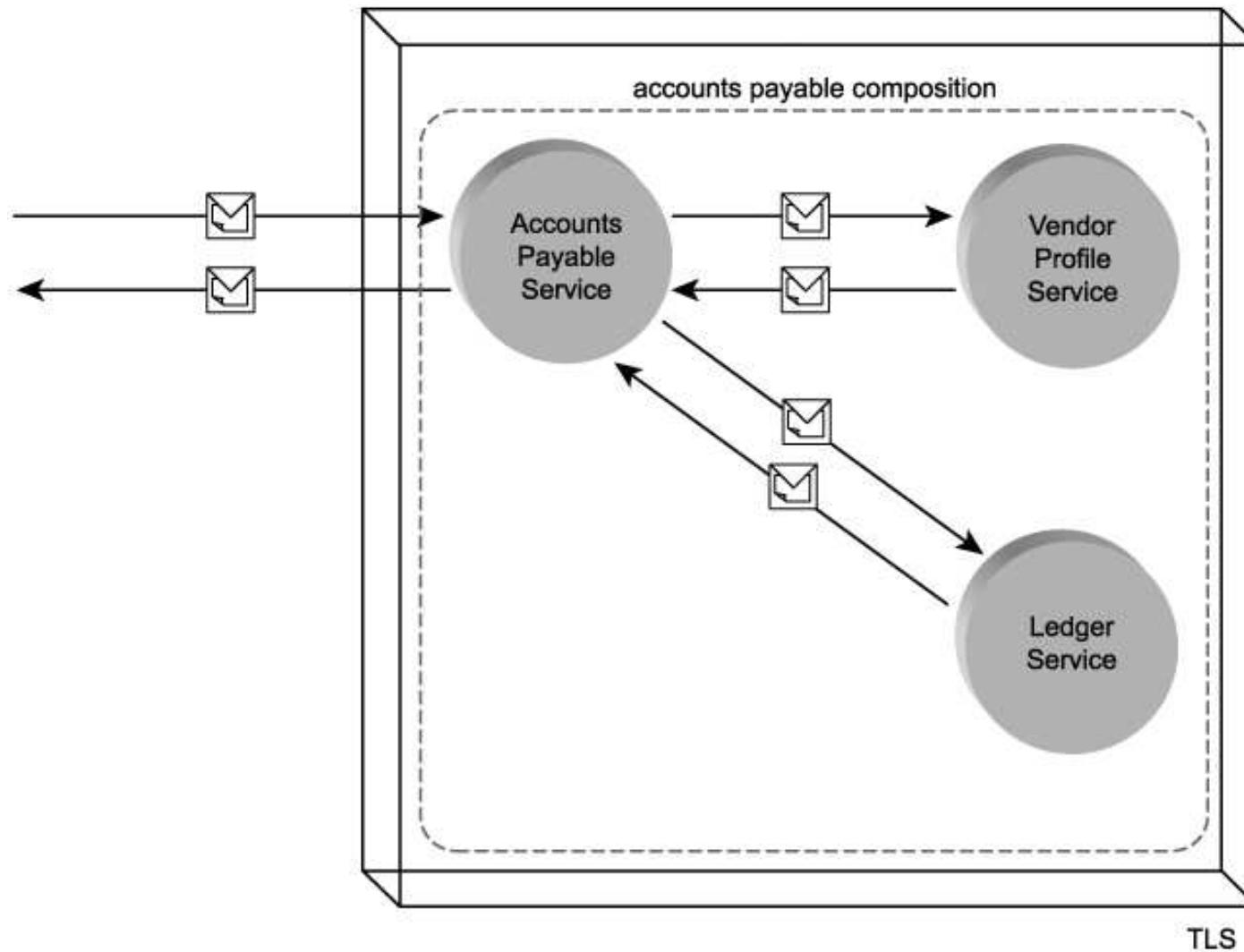
## Nachrichtenaustausch im Fallbeispiel



## Rollen im Fallbeispiel



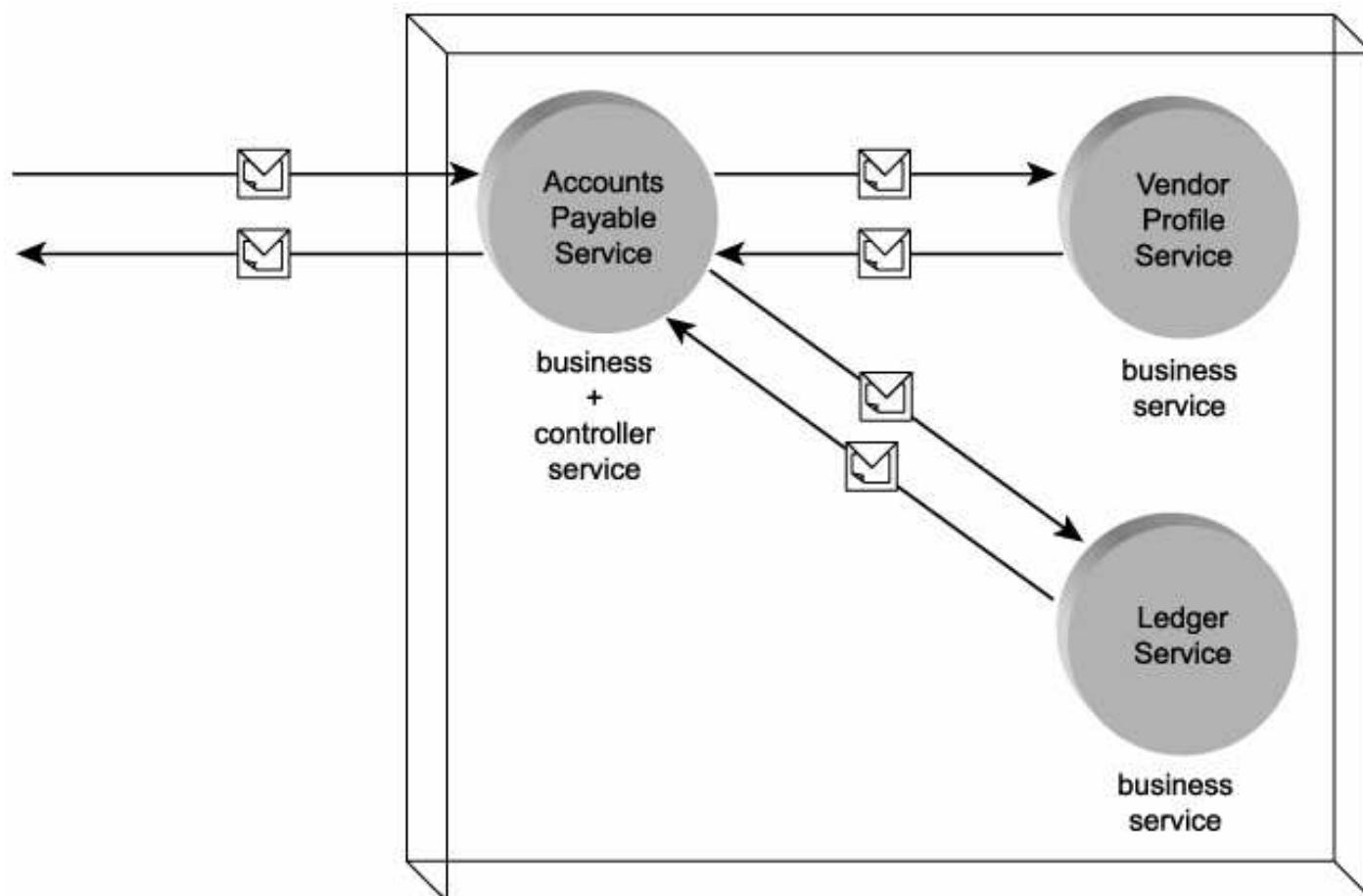
## Komposition von Diensten im Fallbeispiel



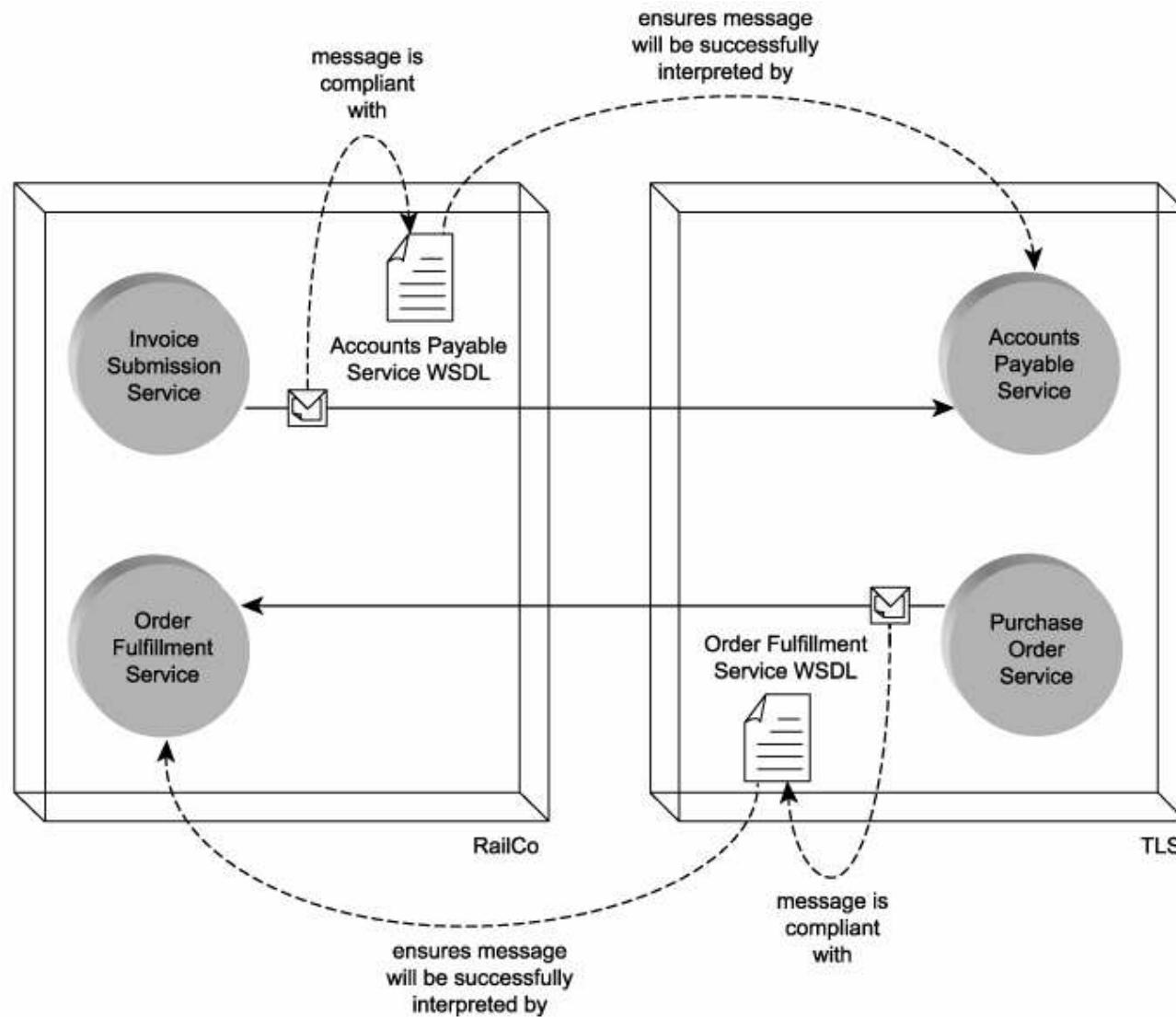
## Dienstklassen im Fallbeispiel

- Geschäftsdienste:  
Kreditorenbuchhaltung (accounts payable), Rechnungsstellung (invoice submission), Journal (ledger), Auftragsbearbeitung (order fulfillment), Auftragserteilung (purchase order), Lieferantenprofile
- Hilfsdienste:  
Policy-Dienst (internal policy), Lastverteilung (load balancing)

## Service mit Geschäfts- und Controllerfunktion



# Die Rolle von WSDL im Fallbeispiel



## WSDL im Fallbeispiel

TLS Accounts Payable Service:

- Ein Interface mit einer Operation: **SubmitInvoice**
- SubmitInvoice hat eine Input- und eine Output-Nachricht:
  - Die Input-Nachricht enthält die Rechnung vom Lieferanten
  - Die Output-Nachricht enthält eine Bestätigung für
    - \* den Empfang und
    - \* die erfolgreiche Validierung der Input-Nachricht

## **SOAP-Nachrichten: Fallbeispiel Rechnungsstellung**

- SOAP-Header

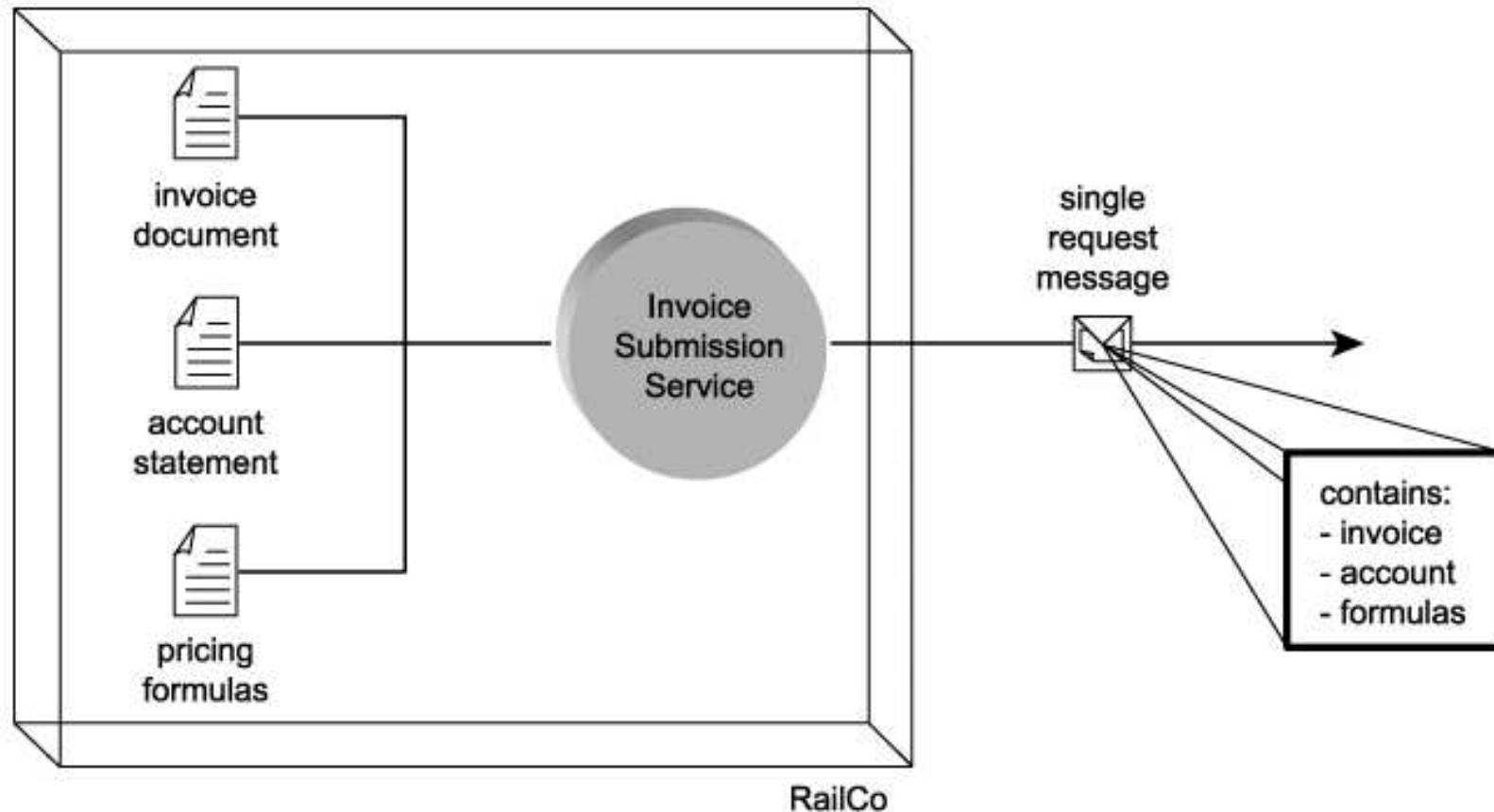
Beim Einreichen einer Rechnung an den TLS Accounts Payable Service:

- Korrelationsbezeichner zur Zuordnung der Antwort
- Authentifizierungsdaten

- Nutzlast

- Rechnungsdokument
- Liste der Außenstände
- Mengenrabatt-Tabelle zur Erinnerung

## Rechnungsstellung im Fallbeispiel



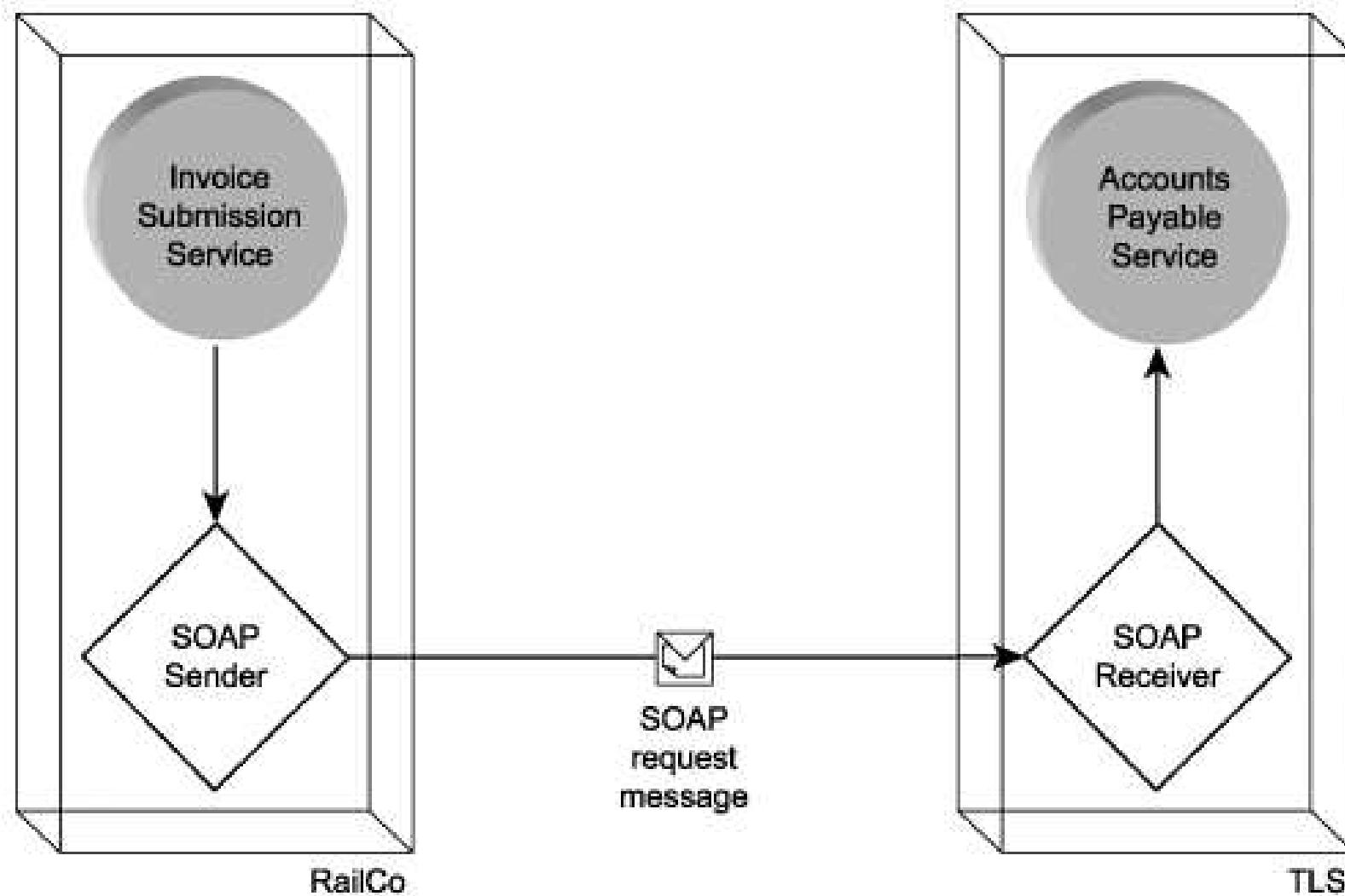
## Anlagen: SOAP-Attachments

- Mechanismus zur Übertragung von Binärdaten (z.B. Bilddateien) als Nutzlast von SOAP-Nachrichten
- Fallbeispiel:
  - Aufträge über 100000 US-\$ müssen von einem TLS-Prokuristen handschriftlich signiert werden. TLS erlaubt keine rein elektronischen Aufträge dieser Größenordnung aus Angst vor Irrtümern.
  - Der unterschriebene Auftrag wird eingescannt und in einem Dokumentenserver dauerhaft archiviert.
  - Das eingescannte Dokument wird als Anhang in der Auftragsnachricht mitverschickt.

## **SOAP-Fehler im Fallbeispiel**

Spezifische Benachrichtigung von TLS, falls das binäre Attachment beim Auftragnehmer nicht verarbeitet werden kann.

## SOAP-Vermittlung im Fallbeispiel



## **Logischer und physikalischer Nachrichtenpfad im Fallbeispiel**

Für Rechnungen von RailCo an TLS ist der logische Nachrichtenpfad fest:

- RailCo InvoiceSubmission Service
- TLS LoadBalancing Service
- TLS Accounts Payable Service

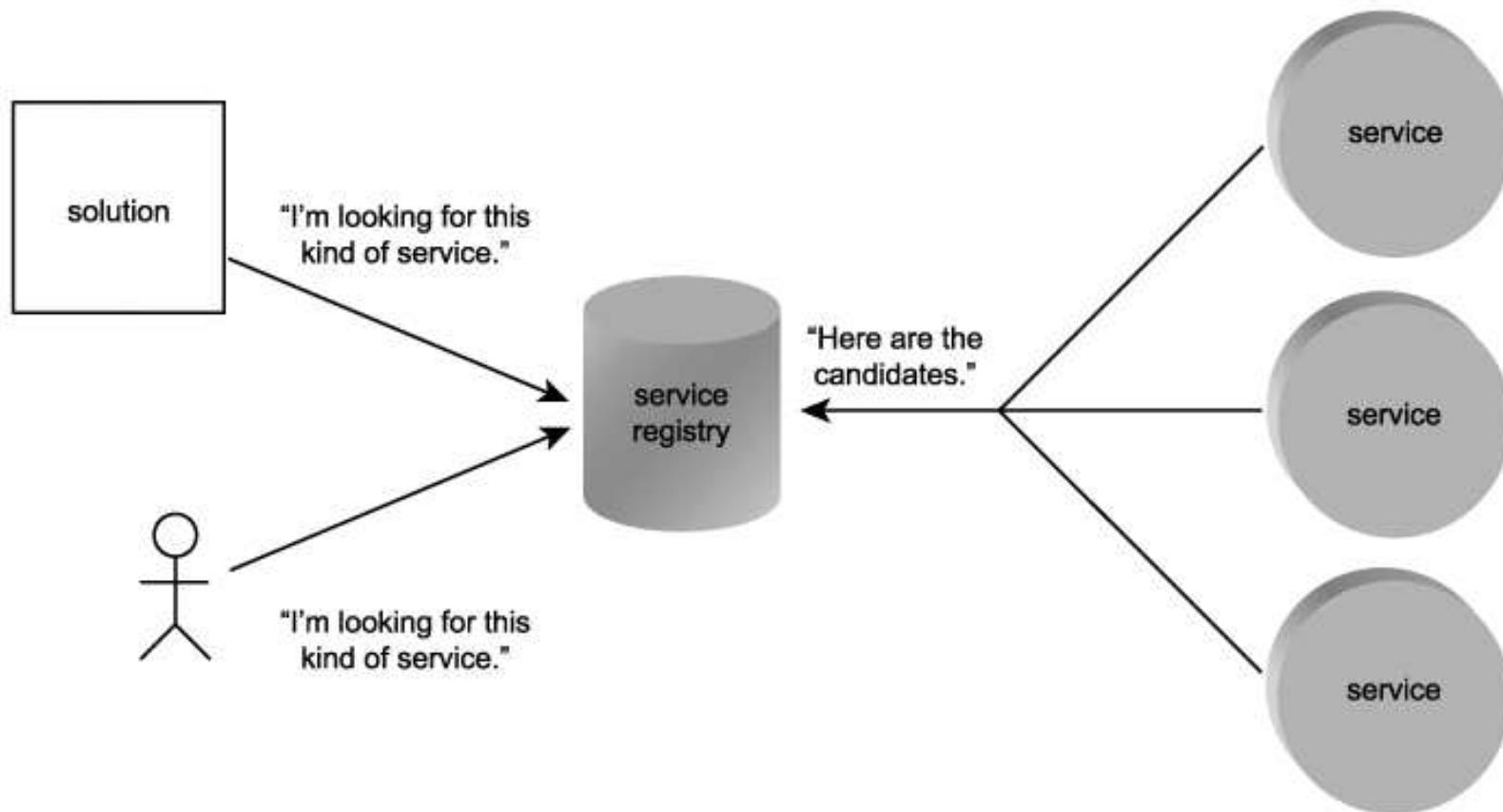
Aber:

- Die für die Bearbeitung der SOAP-Nachrichten zuständigen SOAP-Knoten werden vom Lastverteilungsdienst dynamisch bestimmt (Servercluster)
- Der tatsächliche Nachrichtenpfad steht erst zur Empfangszeit fest!

## Fazit der „high level“-Diskussion von SOAP

- Vielseitige Erweiterbarkeit des Nachrichtenkonzepts
- Beziehung zwischen Nachrichtenköpfen und WS-\*-Erweiterungen
- Nichttriviale Nachrichtenpfade
- Unterscheidung physikalischer/logischer Nachrichtenpfad

## 2.4. Finden von Dienstanbietern / Verzeichnisdienste



## Verzeichnisdienste für Web-Services

Verzeichnisdienste erleichtern das Finden von Webservices.  
Notwendig, wenn deren Anzahl unüberschaubar wird.

- Service-Verzeichnisse im Intranet
  - Teile einer unternehmensweiten SOA-Infrastruktur
  - Rollenbasierte Zugriffsrechte zur verteilten Pflege nötig
  - Einbindung in SOA-Governance-Prozess
- Service-Verzeichnisse im Internet
  - „Gelbe Seiten“-Funktionalität
  - Basis: Kategorisierung von Diensten
  - Attributierung von Diensten
    - Anwendungsdomänen-spezifisch
  - Suchmöglichkeiten innerhalb einer Kategorie über Attribute
    - Beispiel: Suche Dienst in der Kategorie „route planner“, „free=true“ „ AND „germany“ IN „available-regions“

## Standards: WS-Inspection

- dezentralisiertes Verzeichnisdienste-Konzept
- „leichtgewichtiges“, Anbieter-lokales Verzeichnis am „Angebotspunkt“:  
Anbieter von Webservices stellt auf seiner Webseite Liste von Links zur Verfügung
  - zu WSDL-Beschreibungen und/oder
  - zu UDDI-Verzeichnissen

Grundidee: Anbieter pflegt seine Service-Verzeichnisse!

## **Standards: UDDI - Universal Description, Discovery, and Integration**

- Zentralisiertes Verzeichnisdienste-Konzept
- Wenige UDDI-Verzeichnisse enthalten Dienste vieler Anbieter
- White Pages (z.B. [www.whitepages.com](http://www.whitepages.com))
  - Namensregister, sortiert nach Namen
  - Auflistung der Anbieter mit allen Detailangaben
  - Kontaktinformationen (Telefon, Telefax,...)
- Yellow Pages (z.B. [www.yellowpages.com](http://www.yellowpages.com))
  - Branchenverzeichnis
  - Spezifische Suche gemäß verschiedener Taxonomien (Ort, Dienststart,...)
  - Verweist auf White Pages
  - Klassifiziert die Services anhand internationaler Standards wie UNSPSC
- Green Pages
  - Informationen über das Geschäftsmodell des Unternehmens
  - Technische Details zu den angebotenen Web Services
  - Auskunft über Geschäftsprozesse

## Verzeichnisdienste in der Praxis

- Anforderungen für unternehmensweite Verzeichnisdienste und „Gelbe Seiten“ im Internet sind sehr unterschiedlich
- UDDI gilt derzeit als gescheitert:
  - komplex
  - Für unternehmensinterne Verzeichnisse nicht geeignet  
(Einbindung in Governance-Prozesse schwierig, fehlende Rollen-Unterstützung)
  - Internet: Enüberschaubare Konkurrenz der Anbieter gleichartiger Dienste nur bei wenigen Dienstekategorien  
(Welche?)
  - Internet: Wer betreibt „das eine“ zentrale Verzeichnis?  
IBM+Microsoft+...?  
wurde ausprobiert und wieder eingestellt
- Hersteller entwickeln spezifische Konzepte für unternehmensweite Verzeichnisse
- Ruf nach neuen Standards

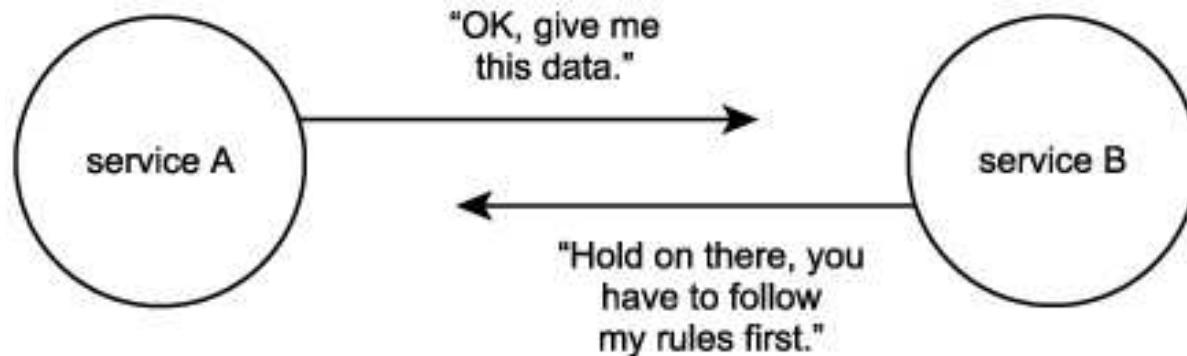
## 2.5 Der WS-\*-"Baukasten" für SOAen

Eine Vielzahl von WS-Standards erweitern die Funktionalität von Webservices

- Metainformationen zur Service-Nutzung, z.B.
  - Richtlinien (spezifizieren nichtfachliche Fähigkeiten und Anforderungen)
  - Sicherheitskonzepte
  - Metadatenaustausch
- Muster zur Modellierung komplexer Geschäftsprozesse, z.B.
  - MEPs („Message exchange patterns“)
  - Koordination
  - Orchestrierung
  - Choreographie

## 2.5.1 Richtlinien bei der Kooperation von Diensten

Beschreibungssprache: WS-Policy



## Wozu Richtlinien?

Richtlinien sind formalisierte Zusatzvereinbarungen für die Nutzung von Diensten

- Mechanismus zur Spezifikation von Regeln, Einschränkungen und Präferenzen
- Verarbeitung durch Mensch oder Maschine möglich (und sinnvoll)
- Eine Richtlinie bezieht sich auf beliebige Anforderungen für ein *Richtliniensubjekt*:
  - Dienst
  - Bindung
  - Operation
  - Nachricht
- Verschiedene Richtliniendomänen: Sicherheit, Koordination usw.
- Richtliniendomäne definiert spezifische, maschinell verarbeitbare Basisanforderungen (policy assertion), z.B. dass eine Nachricht verschlüsselt sein muss
- Richtliniensprache definiert Konstrukte zur Formulierung komplexer Richtlinien

## Richtlinien-Terminologie

- Jede Richtlinie kann mehrere Richtlinienalternativen vorsehen
- Eine Richtlinienalternative ist eine Menge von einzelnen Richtlinienanforderungen (policy assertions)
- Ein *Richtlinienausdruck* ist eine in XML gemäß WS-Policy formulierte Richtlinienanforderung
- Ein „*Policy Attachment*“ verknüpft eine Richtlinienanforderung mit einem Richtliniensubjekt

Näheres: <http://www-128.ibm.com/developerworks/webservices/library/ws-policy.html>

## Beispiele für Richtlinien

- Die Nutzung einer Service-Operation erfordert Authentifizierung durch
  - X.509-Zertifikat *oder*
  - Benutzername und Passwort
- Die Nutzung einer Operation eines Dienstes muss in eine Transaktion eingebunden werden
- Die Dienstenutzung (alle Operationen) erfordert die Beachtung einer bestimmten Version von WS-Security
- Die Richtlinie bestimmt technische Daten wie Timeout-Werte für die Übermittlung von Nachrichtensequenzen gemäß WS-ReliableMessaging
- Die Input-Nachricht einer Operation erfordert auf Transportebene verschlüsselte Übertragung
- Die Nutzung eines Dienstes erfordert UTF-8-Zeichencodierung

## Beispiel einer WSDL-Beschreibung mit Richtlinien

```
<wsdl:definitions name="StockQuote"
    targetNamespace="http://www.fabrikam123.example.com/stock/binding"
    xmlns:tns="http://www.fabrikam123.example.com/stock/binding"
    xmlns:fab="http://www.fabrikam123.example.com/stock"
    xmlns:rmp="http://schemas.xmlsoap.org/ws/2005/02/rm/policy"
    xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsoap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
(02) <wsp:Policy wsu:Id="RmPolicy" >
(03)   <rmp:RMAssertion>
(04)     <rmp:InactivityTimeout Milliseconds="600000" />
(05)     <rmp:BaseRetransmissionInterval Milliseconds="3000" />
(06)     <rmp:ExponentialBackoff />
(07)     <rmp:AcknowledgementInterval Milliseconds="200" />
(08)   </rmp:RMAssertion>
(09) </wsp:Policy>
(10) <wsp:Policy wsu:Id="X509EndpointPolicy" >
```

```
(11)    <sp:AsymmetricBinding>
(12)        <wsp:Policy>
(13)            <!-- Details omitted for readability -->
(14)            <sp:IncludeTimestamp />
(15)            <sp:OnlySignEntireHeadersAndBody />
(16)        </wsp:Policy>
(17)    </sp:AsymmetricBinding>
(18) </wsp:Policy>
(19) <wsp:Policy wsu:Id="SecureMessagePolicy" >
(20)     <sp:SignedParts>
(21)         <sp:Body />
(22)     </sp:SignedParts>
(23)     <sp:EncryptedParts>
(24)         <sp:Body />
(25)     </sp:EncryptedParts>
(26) </wsp:Policy>
(27) <wsdl:import namespace="http://www.fabrikam123.example.com/stock"
(28)   location="http://www.fabrikam123.example.com/stock/stock.wsdl" />
(29) <wsdl:binding name="StockQuoteSoapBinding" type="fab:Quote" >
(30)   <wsoap12:binding style="document"
(31)     transport="http://schemas.xmlsoap.org/soap/http" />
(32)   <wsp:PolicyReference URI="#RmPolicy" wsdl:required="true" />
```

```
(31)      <wsp:PolicyReference URI="#X509EndpointPolicy" wsdl:required="true" />
(32)      <wsdl:operation name="GetLastTradePrice" >
(33)          <wsoap12:operation soapAction="http://www.fabrikam123.example.com/stock/0
(34)              <wsdl:input>
(35)                  <wsoap12:body use="literal" />
(36)                  <wsp:PolicyReference URI="#SecureMessagePolicy"
(37)                      wsdl:required="true" />
(38)              </wsdl:input>
(39)              <wsdl:output>
(40)                  <wsoap12:body use="literal" />
(41)                  <wsp:PolicyReference URI="#SecureMessagePolicy"
(42)                      wsdl:required="true" />
(43)              </wsdl:output>
(44)          </wsdl:operation>
(45)      </wsdl:binding>
(46)  </wsdl:definitions>
```

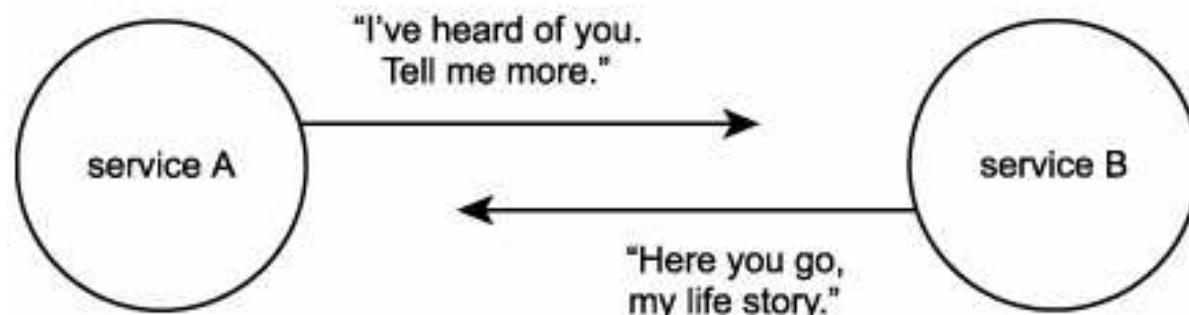
## Richtlinien im Fallbeispiel

- TLS verwendet ab sofort eine neuere Version von WS-ReliableMessaging
- In einer Richtlinie für den InvoiceSubmission-Dienst wird festgelegt
  - der Dienst akzeptiert WS-ReliableMessaging Header nach alter und neuer Syntax
  - die neue Syntax wird vom Dienst bevorzugt
  - unabhängig von der Version wird eine bestimmte Zeichencodierung gefordert

## Richtlinien im Fallbeispiel

```
<wsp:Policy
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy">
  <wsp:ExactlyOne>
    <wsp:All ID="Invoice1">
      <wsp:SpecVersion wsp:Usage="wsp:Required"
        wsp:Preference="10" wsp:URI=
          "http://schemas.xmlsoap.org/ws/2004/03/rm"/>
      <wsp:TextEncoding wsp:Usage="wsp:Required"
        Encoding="iso-8859-5"/>
    </wsp:All>
    <wsp:All ID="Invoice2">
      <wsp:SpecVersion wsp:Usage="wsp:Required"
        wsp:Preference="1" wsp:URI=
          "http://schemas.xmlsoap.org/ws/2003/02/rm"/>
      <wsp:TextEncoding wsp:Usage="wsp:Required"
        Encoding="iso-8859-5"/>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

## Metadaten-Austausch

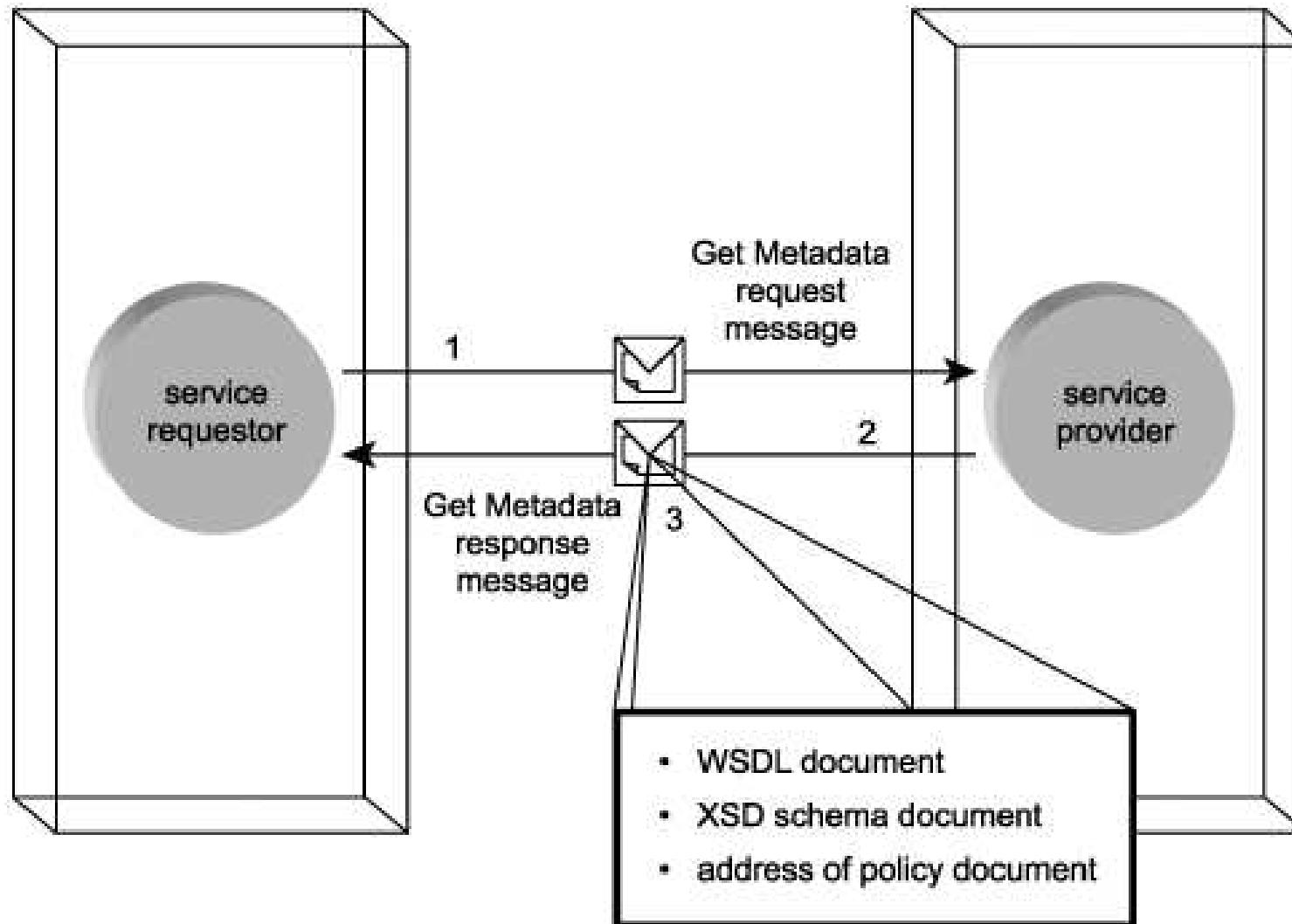


## Metadaten-Austausch

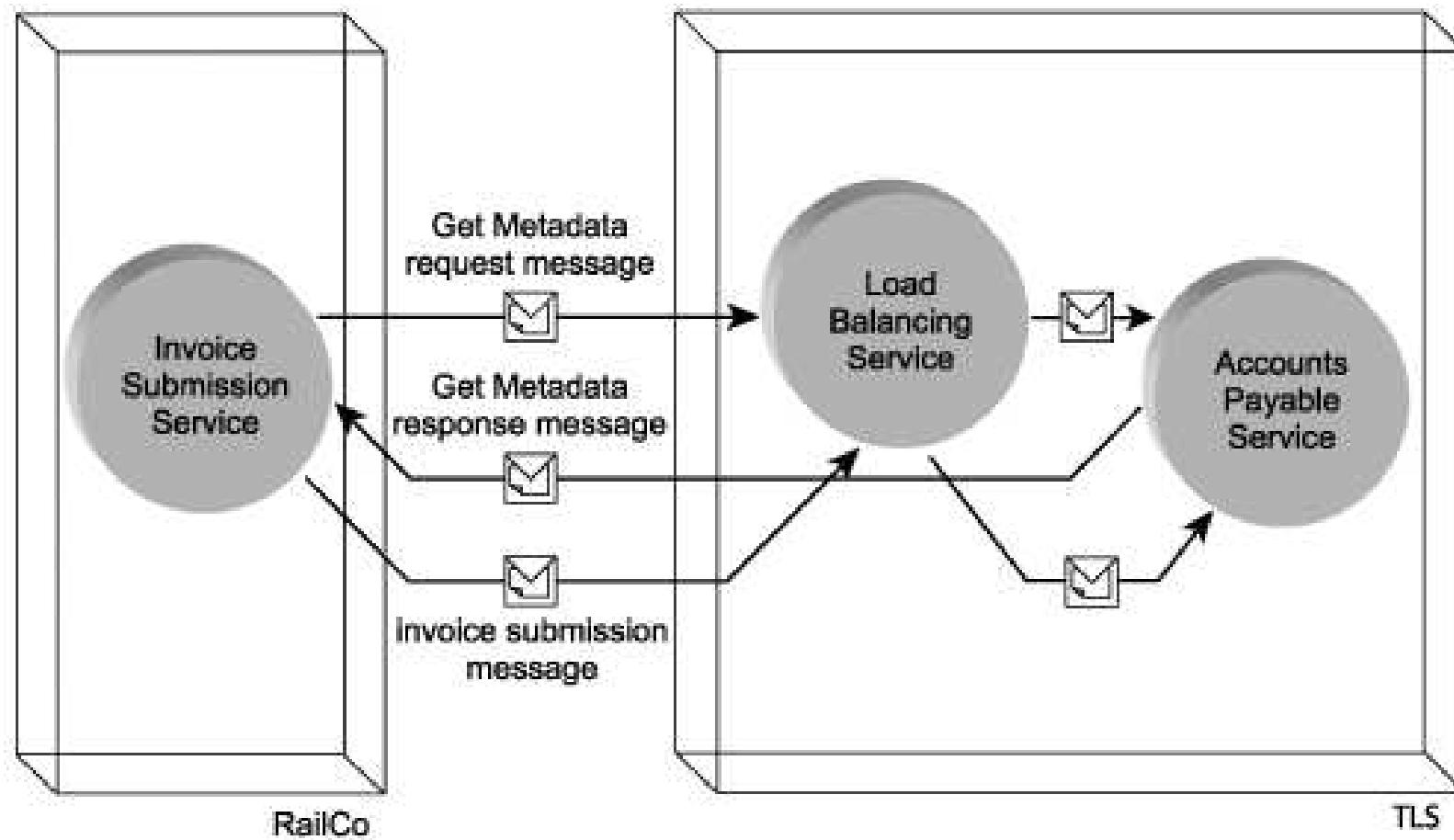
Mechanismus zur Erfragen der Metadaten zu einem Dienst

- Selektive oder komplette Zusendung von Metadaten:
  - WSDL (auch über UDDI erfragbar)
  - zugrunde liegende XSD-Schemata
  - Richtlinien
- Anwendungsbeispiele:
  - Einsicht der Metadaten zur Vorbereitung der ersten Dienstenutzung
  - Regelmäßige Kontrolle der Metadaten, um Änderungen des Dienstvertrags durch den Anbieter automatisch entdecken (Merkmal loser Kopplung)

## Übertragung aller Metadaten



## Metadaten-Austausch im Fallbeispiel



## Sicherheit von Webservices

Webservices ermöglichen einfachen Zugang zu Daten und Dienstleistungen über Firewall-Grenzen hinweg

→ **hohe Sicherheitsanforderungen**

- Allgemeine Sicherheitsrisiken gelten auch für Webservices, z.B.
  - unautorisierte Zugriffe
  - Ausspionieren vertraulicher Daten
  - „Denial of Service“-Attacken
  - Malware (Viren, Spyware/Trojaner, Würmer)
- Webservices sind oft Schnittstellen zu Backend-Systemen, machen diese von außen angreifbar
- Webservices realisieren unternehmenskritische Geschäftsprozesse
  - Vertraulichkeit und Integrität der Daten besonders wichtig  
(Besonders kritisch: Schutz vor Industriespionage)

- Externe Webservices sind von der Sicherheit her schwer einzuschätzen, entsprechen ggf. nicht internen Sicherheitsrichtlinien
- Firewalls sind kein geeignetes Schutzkonzept
  - für HTTP-basierte WS-Nutzung meist offen
  - klassische Zonenkonzepte „unsicher“/DMZ/„sicher“ nicht hilfreich
- Spezifische „Denial of Service“-Attacken, z.B.
  - XML-Parser zum Absturz bringen  
(z.B. Speichermangel beim DOM-Parser erzeugen)
  - XML-Schema manipulieren (XML-“Vergiftung“)
  - Massenhaft nicht gültige XML-Dokumente senden  
(erhöhter Bearbeitungsaufwand beim Dienstanbieter)
  - Manipulation von Dienstverzeichnissen

## Sicherheitsanforderungen für Webservices

- Identifikation und Authentifizierung der Teilnehmer
- Integrität der Daten, Schutz gegen Manipulation
- Vertraulichkeit der Daten
  - Verschlüsselung beliebiger Teile von Dokumenten
- Autorisierungsmechanismen
  - Zugriff auf Services
  - Zugriff auf verschlüsselte Teile von Geschäftsdokumenten
- Verbindlichkeit (Nichtabstreitbarkeit der Teilnahme an einer Transaktion)
- Single Sign-On / Weitergabe von Autorisierungsinformation
- Ende-zu-Ende Sicherheitskonzepte für komplexe Nachrichtenpfade

## **Wichtige Sicherheits-Standards für Webservices**

- XML Digital Signature (XDSIG), W3C
- XML Encryption (XENC), W3C
- WS-Security (WSS), OASIS
- Security Assertion Markup Language (SAML), OASIS
- XML Key Management Specification (XKMS), W3C
- Extensible Access Control Markup Language (XACML), OASIS

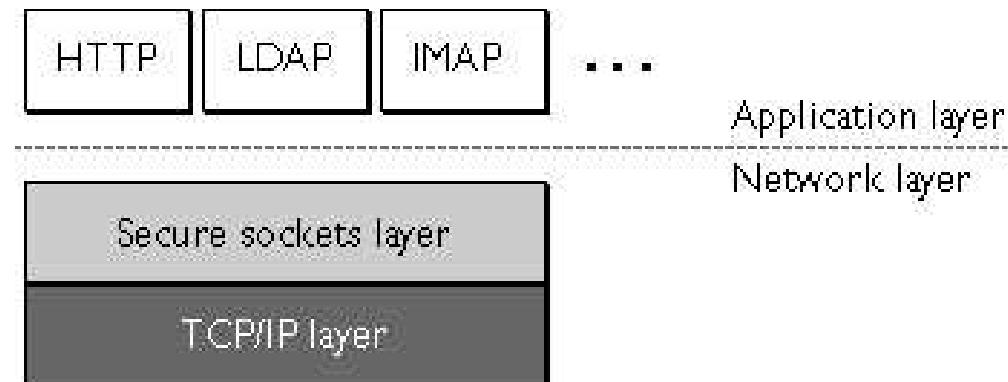
## Sicherheitsebenen

- Gleichzeitige Realisierung von Sicherheitsmechanismen auf verschiedenen Kommunikationsschichten ist kein Widerspruch:
- Sicherung der Schicht 3 (VPN), falls bereits die Existenz der Kommunikation sicherheitskritisch ist
- Sicherung des Transports: SSL/TLS
- Sicherung auf der Nachrichtenebene (XML/SOAP)
- Sicherheitsrichtlinien für die Dienstenutzung

## Transportsicherheit durch SSL/TLS

- SSL = „Secure Socket Layer“-Protokoll
- TLS = „Transport Layer Security“
- Begriff TLS benutzbar
  - als Weiterentwicklung von SSL oder
  - als Bezeichnung für neuere SSL-Versionen
- Ursprünglich von Netscape entwickeltes Sicherungsprotokoll
- Protokoll wird zwischen Transportschicht und Anwendung „eingeschoben“
  - SSL wird aus Anwendungssicht wie ein Transportprotokoll behandelt
  - SSL wird vom Transportdienst wie ein Anwendungsprotokoll behandelt

## SSL im Schichtenmodell



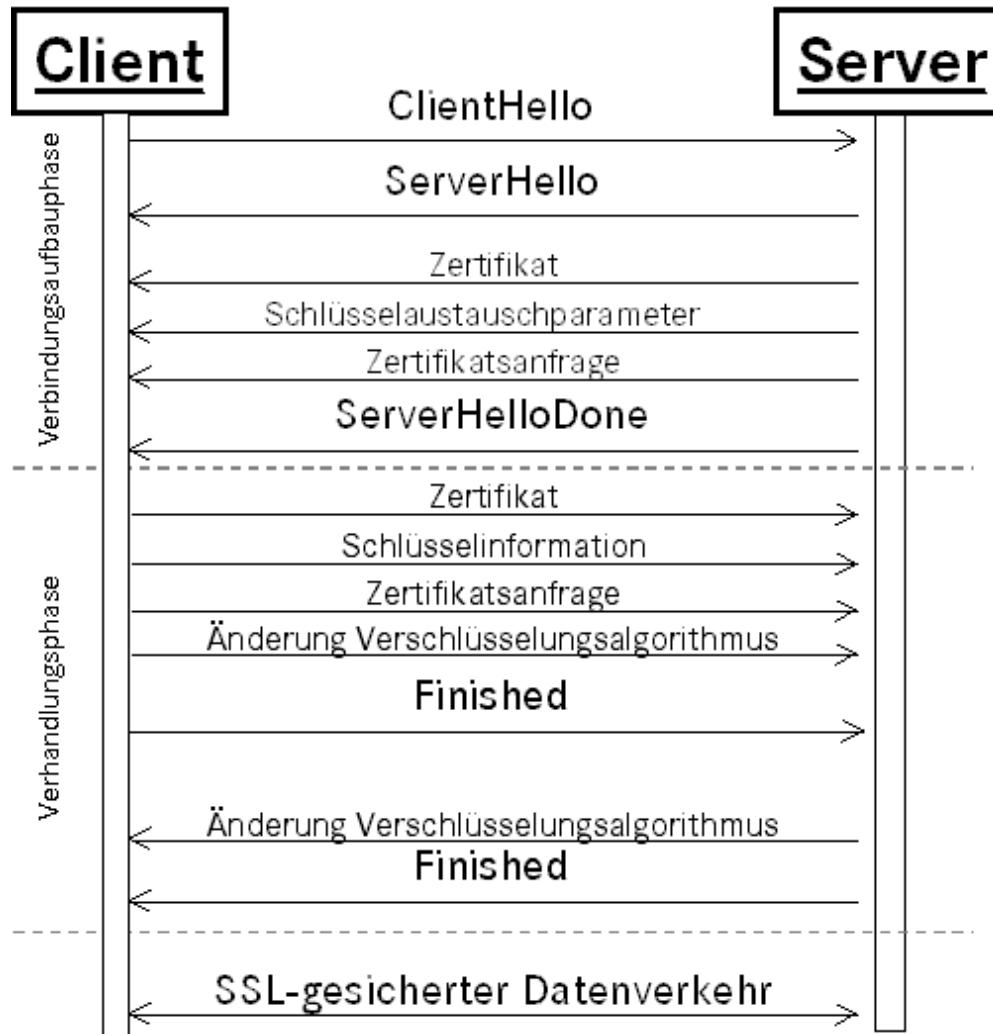
## SSL-Anwendungsbeispiele

- Verschlüsselung von WWW-Sitzungen
- Verschlüsselung von Dateiübertragungen
- Verschlüsselung von E-Mail
- Authentifizierung der Kommunikationspartner
  - Server-Authentifizierung durch Client oder
  - gegenseitige Authentifizierung
  - mit oder ohne Zertifikatsprüfung

## SSL - Technische Merkmale

- Zwei Sub-Protokolle:
  - SSL-Record-Protocol definiert Datenformat für die Nachrichten
  - SSL-Handshake-Protocol definiert
    - \* Parameter-Verhandlung beim Sitzungsaufbau
    - \* Authentifizierung beim Sitzungsaufbau
    - \* Generierung und Versendung eines Sitzungsschlüssels
- diverse Verschlüsselungsverfahren
  - Kombination aus asymmetrischen und symmetrischen Verfahren, sowie Hash-Verfahren zur Dokumentauthentifizierung
  - Beispiele: DES, Triple DES, DSA, KEA, MD5, RSA

# SSL-Verbindungsauftbau



## Schema für SSL-Handshake in Java

```
KeyManager []km = null;
TrustManager []tm = {new MyX509TrustManager()};
SSLContext sslContext = SSLContext.getInstance("SSL");
sslContext.init(km,tm,new java.security.SecureRandom());
SSLSocketFactory sf = sslContext.getSocketFactory();
Socket sock=new Socket("alice",443);
SSLSocket sslsock=(SSLSocket)sf.createSocket(sock, "alice", 443, true);

sslsock.startHandshake();

X509Certificate[] c = sslsock.getSession().getPeerCertificateChain();

//Prüfung der Berechtigung und Authentisierung
//Falls Berechtigung erteilt, Speicherung des Zertifikates
//...
KeyStore ks;
//...
ks.setCertificateEntry ("myCert", c[0]);
URL url = new URL("https://alice:443/soap/servlet/rpcrouter");
//unveränderter SOAP-Aufruf
```

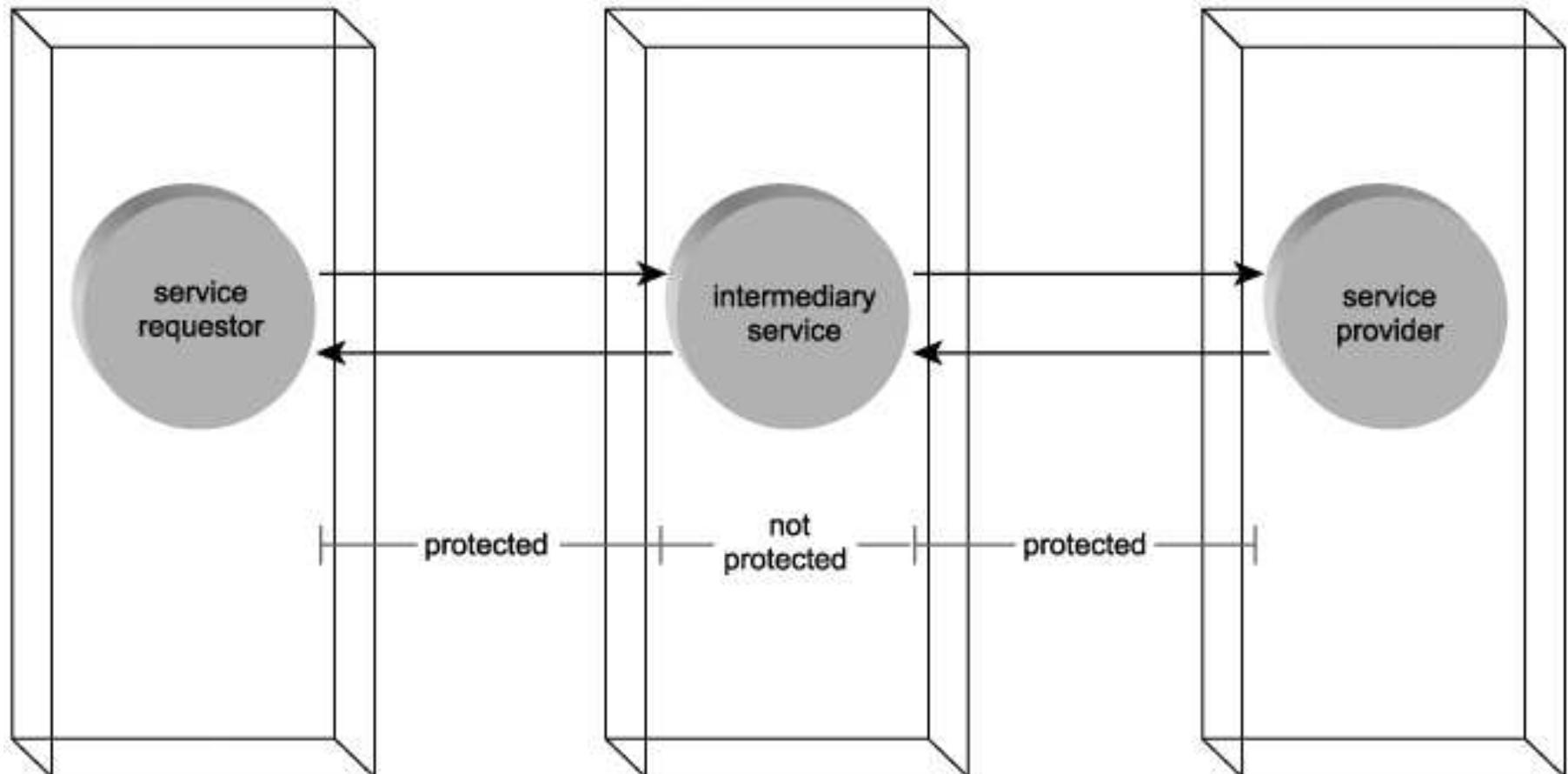
## **Transportsicherheit ist nicht immer ausreichend**

- In vielen Fällen ist die SOAP-Verwendung auf der Basis von SSL-gesicherten Transporten (HTTP, SMTP) sinnvoll
- Die SSL-Nutzung löst nicht alle Probleme:
  1. Komplexe Nachrichtenpfade mit intermediären Diensten
    - Webservices definieren eine Ende-zu-Ende-Nachrichtenübertragung
    - Transportsicherheit bezieht sich nur auf die Transportwege zwischen je zwei Diensten innerhalb des Nachrichtenpfads
    - Vertraulichkeit: Nur ein Teil der an der Verarbeitung eines Dokuments beteiligten Dienste soll Zugriff auf vertrauliche Daten erlangen

Beispiel: In einer Bestellung sind verschlüsselte Kreditkartendaten enthalten, die nur der Dienst zur Zahlungsabwicklung kennen darf
    - kein Ende-zu-Ende Absenderbeweis

2. Granularität der Vertraulichkeit zu grob: In manchen Fällen möchte man Teile einer Nachricht verschlüsseln, während andere Teile für intermediäre Bearbeitung unverschlüsselt sein müssen
3. Keine Integration des SSL-Handshake in die SOAP-Ebene vorhanden
4. Server-Zertifikate nötig → Zusatzaufwand
5. Laufzeitaufwand durch SSL-Handshake
6. Zertifikatsaustausch vor Dienstenutzung widerspricht der Idee loser Kopplung
7. Asynchrone Diensteaufrufe nicht möglich

## Transport-Sicherheit und intermediäre Dienste



## WS-Security Framework

### Komplexes Framework für Sicherheit auf Nachrichtenebene

- Ergänzt Sicherheit auf Transportebene (SSL/TLS)
- XML-bezogene Sicherheitsmechanismen
- WS-bezogene Sicherheitsmechanismen
- Zum Framework gehört der OASIS-Standard WS-Security (IBM, Microsoft, Verisign).

Er regelt die Nutzung von XML-Sicherheitsmechanismen für SOAP durch Definition entsprechender SOAP-Header.

## WS-Security Framework: Die Standards

- WS-Security
- WS-SecurityPolicy
- WS-Trust
- WS-SecureConversation
- WS-Federation
- Extensible Access Control Markup Language (XACML)
- Extensible Rights Markup Language (XrML)
- XML Key Management (XKMS)
- XML-Signature
- XML-Encryption
- Security Assertion Markup Language (SAML)
- .NET Passport
- Secure Sockets Layer (SSL)
- WS-I Basic Security Profile

## **XML Encryption: Vollständige oder teilweise Verschlüsselung von XML-Dokumenten**

- Wahl zwischen verschiedenen asymmetrischen oder symmetrischen Chiffrierverfahren:  
Triple-DES-CBC, AES-CBC, RSA
- Verschlüsselung einzelner Elemente mit unterschiedlichen Schlüsseln möglich  
→ selektive Sichtbarkeit entlang komplexer Nachrichtenpfade
- Verschlüsselung von Elementinhalten unter Bewahrung der XML-Tags → partielle Verarbeitung ohne Dechiffrierung
- Zur Verschlüsselung von SOAP-Nachrichten benutzt: Übertragung der Verschlüsselungsparameter (Chiffrierverfahren, Initialisierungsparameter) in der Nachricht
- Verschlüsselung von angehängten Binärdaten (Bilder usw.)

## **XML Encryption - Grundelemente**

EncryptedData	umschließendes Tag für die Verschlüsselung
EncryptionMethod	Der verwendete Verschlüsselungsalgorithmus
CipherData	Die verschlüsselten Daten (direkt oder Referenz)

## Beispiel für partielle Verschlüsselung

Verschlüsselung der Kreditkartenzahlung:

```
...
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number>
      <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
        Type='http://www.w3.org/2001/04/xmlenc#Content'>
        <CipherData>
          <CipherValue>A23B45C56</CipherValue>
        </CipherData>
      </EncryptedData>
    </Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

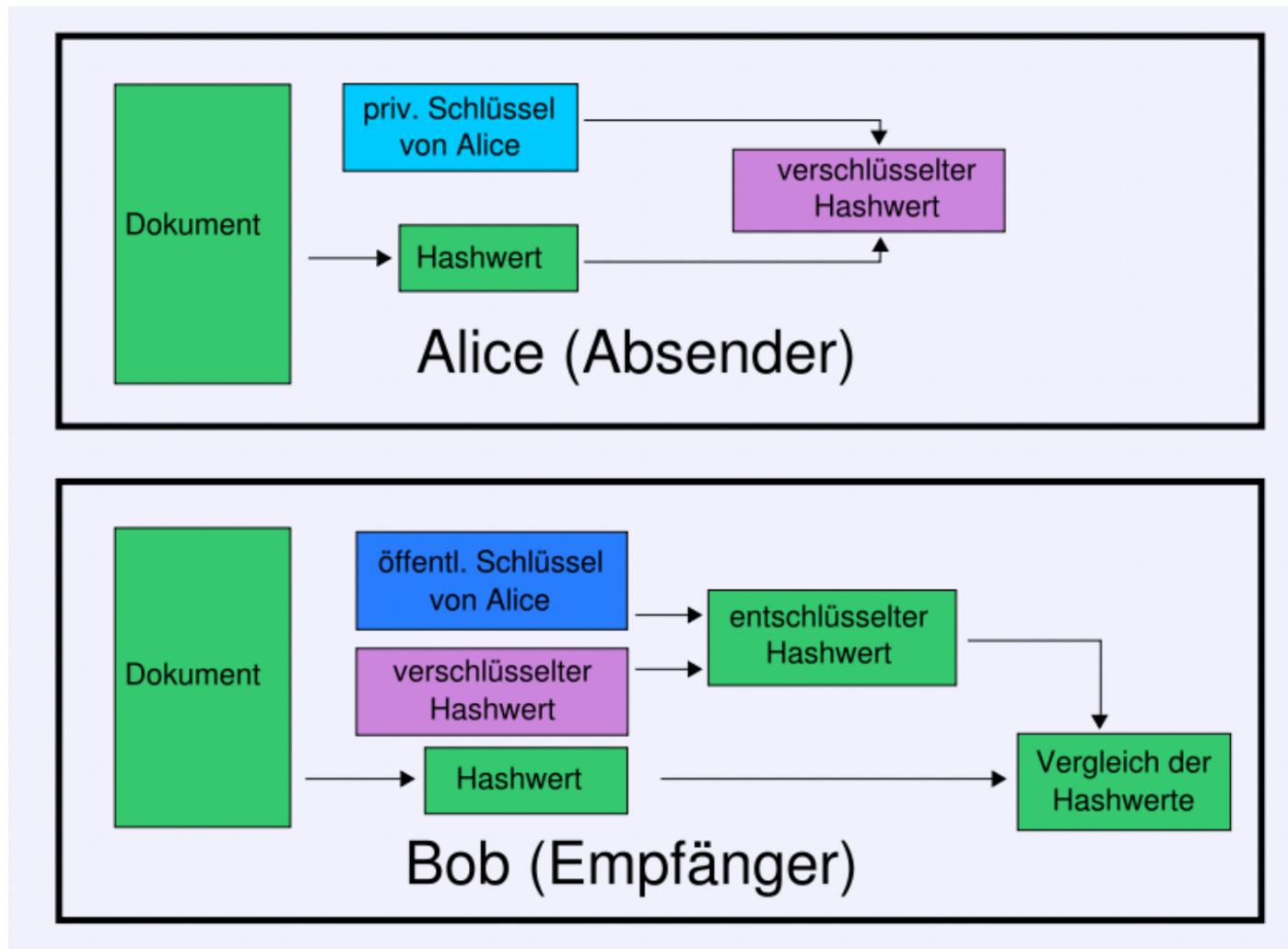
## Elektronische Signatur

- Ersatz für herkömmliche Unterschrift
- unabhängig von einer verschlüsselten Übertragung
- Garantiert beim Versenden eines Dokuments:
  - Datenintegrität:  
Erkennung von Verfälschungen durch Übertragungsfehler oder Manipulationen
  - Absenderauthentizität
  - Nichtabstreitbarkeit des Nachrichtenversands

## Technische Voraussetzungen

- Asymmetrische Verschlüsselung („private key“+“public key“)
- PKI-System zur Verwaltung von öffentlichen Signaturschlüsseln und Zertifikaten (z.B. X.509, PGP)
- Hash-Verfahren zur Erstellung eines „message digest“  
(Prüfsumme des Dokuments)
- Alternativ:  
Symmetrische Verschlüsselung („secret key“) mit zentraler Verwaltung der Schlüssel  
(„Kerberos“)

## Beispiel für ein Implementierungsschema



(aus Wikipedia)

## XML-Signature - Standard für signierte XML-Dokumente

- Unterstützt elektronische Signatur von
  - XML-Dokumenten
  - Abschnitten oder
  - einzelnen Elementen
- Einzelne Teile können mit unterschiedlichen Schlüsseln signiert werden
- Signierte Dokumente enthalten ein XML-Element „Signature“ mit
  - signierten Daten („SignedInfo“)
  - Angaben zum Signierschlüssel („KeyInfo“)
  - Signatur („SignatureValue“)
- Kanonische XML-Repräsentation erforderlich
  - Zeichencodierung
  - „Whitespace“-Zeichen

## Signaturvarianten

Die Bindung der Signatur an die signierten Daten kann unterschiedlich erfolgen:

- **Eingebettete Signatur** („enveloped signature“)

Signatur-Element ist Unterelement des signierten Dokuments

- **Einbettende Signatur** („enveloping signature“)

Signatur-Element ist Eltern-Element des signierten Dokuments

- **Separate Signatur** („detached signature“)

Signatur und signierte Daten stehen in separaten Dokumenten

## Beispiel-Struktur für einbettende Signatur

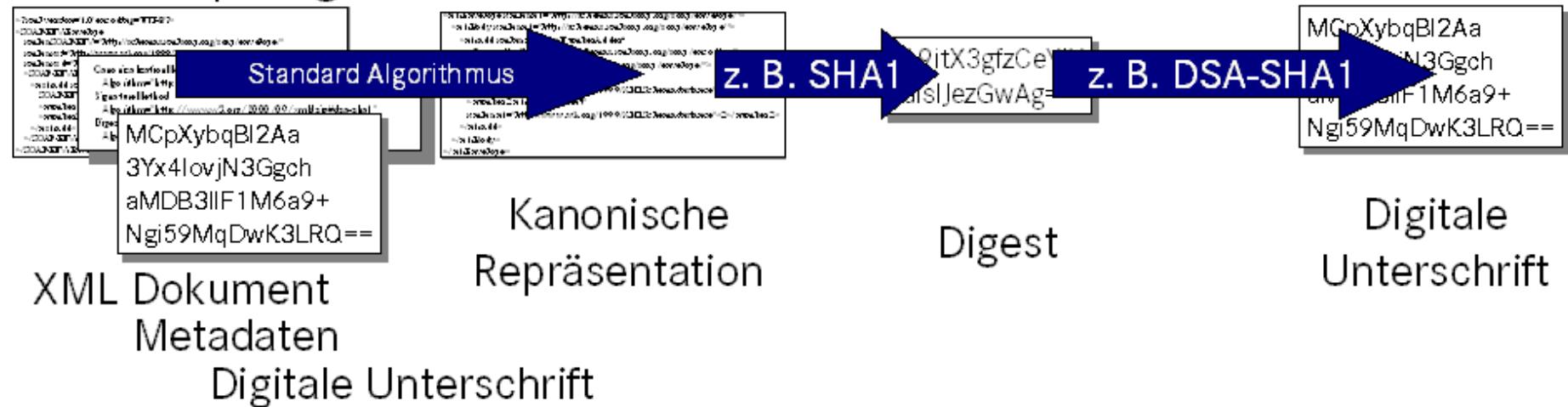
```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo> ...
        </SignedInfo>
    <KeyInfo> ...
        </KeyInfo>
    <SignatureValue> ...
        </SignatureValue>
</Signature>
```

# Kanonische XML-Repräsentation und Digitale Signatur

## ● Sender:



## ● Empfänger:



(aus Melzer et al.)

## Beispiel für XML-Signatur

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/..."/>
    <SignatureMethodAlgorithm="http://www....xmldsig#rsa-sha1"/>
    <Reference URI="#PurchaseOrder">
      <DigestMethod Algorithm="http://www.w3...xmldsig#sha1"/>
      <DigestValue>qZk+nkcGcWq6piVxeFdczQ2J=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>
    IWijxQjUrcXBYc0ei4QxjWo9Kg8Dep9tlWoT4SdeRT87GH03dgh
  </SignatureValue >
  <KeyInfo>
    <X509Data>
      <X509SubjectName>
        CN=Smith, STREET=742 P.Av,L=New York, ST=NY, C=US
      </X509SubjectName>
    </X509Data>
  < /KeyInfo>
</Signature>
```

## SOAP-Security

- SOAP-Header zur Nutzung von XML-Sicherheitskonzepten XENC, XDSIG u.a.
- Entlang komplexer Nachrichtenpfade agierende Services benötigen ggf. spezifische sicherheitsrelevante Metadaten

Ein Security-Header kann einem Aktor zugeordnet werden

- Zwischendienste können Security-Header hinzufügen/modifizieren
- Security-Token-Elemente im Header repräsentieren für Verschlüsselung und Signaturen benötigte Zusatzinformation

## **Security-Token-Kategorien für Verschlüsselung und Signaturen**

- **UsernameToken** – zur Identifikation
- **BinarySecurityToken** – Binär codierte Information, z.B. X.509-Zertifikat oder Kerberos-Ticket
- **EncryptedDataToken** – Verschlüsselte Header-Information
- **SecurityTokenReference** – Verweis auf Schlüssel
- **SecurityTimestamp** – Zeitstempel für Verschlüsselungszwecke

## Beispiel für signierte SOAP-Nachricht

```
<s12:Envelope xmlns:s12="http://www.w3.org/2003/05/soap-envelope">
  <s12:Header>
    <wsse:Security
      xmlns:wsse="http://schemas.xmlsoap.org/soap/security/2000-12"
      s12:actor="http://www.example.com/soapEndpoint"
      s12:mustUnderstand="1">
      <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo>
          <ds:CanonicalizationMethod
            Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
          <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
          <ds:Reference URI="#Body">
            <ds:Transforms>
              <ds:Transform Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-20001026">
                </ds:Transforms>
              <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
              <ds:DigestValue>zA9itX3gfzCeYIYyalslJezGwAg=</ds:DigestValue>
            </ds:Reference>
          </ds:SignedInfo>
```

```
<ds:SignatureValue>
    MCpXybqBI2Aa3Yx4IovjN3GgchaMDB3lIF1M6a9+Ngi59MqDwK3LRQ==
</ds:SignatureValue>
</ds:Signature>
</wsse:Security>
</s12:Header>
<s12:Body xmlns:wsse="http://schemas.xmlsoap.org/soap/security/2000-12"
wsse:id="Body">
<ns1:add xmlns:ns1="urn:NumberAdder"
    s12:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <number1 xsi:type="xsd:int">1</number1>
    <number2 xsi:type="xsd:int">2</number2>
</ns1:add>
</s12:Body>
</s12:Envelope>
```

## Beispiel für Verwendung von Verschlüsselung und Signatur

```
<?xml version="1.0" encoding="utf-8"?>
<S11:Envelope ...>
  <S11:Header>
    <wsse:Security>
      <wsu:Timestamp wsu:Id="T0">
        <wsu:Created>2001-09-13T08:42:00Z</wsu:Created>
      </wsu:Timestamp>

      <wsse:BinarySecurityToken
        ValueType="http://...#X509v3"
        wsu:Id="X509Token"
        EncodingType="...#Base64Binary">
        MIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
      </wsse:BinarySecurityToken>

    <xenc:EncryptedKey>
      <xenc:EncryptionMethod Algorithm=
        "http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
```

```
<ds:KeyInfo>
    <wsse:SecurityTokenReference>
        <wsse:KeyIdentifier
            EncodingType="...#Base64Binary"
            ValueType="http://...#X509v3">
            MIGfMa0GCSq...
        </wsse:KeyIdentifier>
    </wsse:SecurityTokenReference>
</ds:KeyInfo>

<xenc:CipherData>
    <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
    </xenc:CipherValue>
</xenc:CipherData>

<xenc:ReferenceList>
    <xenc:DataReference URI="#enc1"/>
</xenc:ReferenceList>
</xenc:EncryptedKey>

<ds:Signature>
    <ds:SignedInfo>
```

```
<ds:CanonicalizationMethod
    Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
<ds:SignatureMethod
    Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />

<ds:Reference URI="#T0">
    <ds:Transforms>
        <ds:Transform
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
    </ds:Transforms>
    <ds:DigestMethod
        <ds:DigestValue>LyLsF094hPi4wPU...
        </ds:DigestValue>
    </ds:Reference>

<ds:Reference URI="#body">
    <ds:Transforms>
        <ds:Transform
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
    </ds:Transforms>
    <ds:DigestMethod Algorithm="..." />
    <ds:DigestValue>LyLsF094hPi4wPU... </ds:DigestValue>
```

```
</ds:Reference>
</ds:SignedInfo>

<ds:SignatureValue>
    Hp1ZkmFZ/2kQLXDJbchm5gK...
</ds:SignatureValue>

<ds:KeyInfo>
    <wsse:SecurityTokenReference>
        <wsse:Reference URI="#X509Token"/>
    </wsse:SecurityTokenReference>
</ds:KeyInfo>
</ds:Signature>

</wsse:Security>
</S11:Header>

<S11:Body wsu:Id="body">
    <xenc:EncryptedData
        Type="http://www.w3.org/2001/04/xmlenc#Element"
        wsu:Id="enc1">
        <xenc:EncryptionMethod
```

```
Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
<xenc:CipherData>
  <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...</xenc:CipherValue>
</xenc:CipherData>
</xenc:EncryptedData>
</S11:Body>
</S11:Envelope>
```

## WS-Trust

- Definiert Möglichkeiten zum initialen Austausch von Sicherheitsdaten („security token“)
- Spezifiziert Format der Daten (z.B. X.509v3)
- Spezifiziert Vertrauensbeziehungen
- definiert „Security Token Service“ mit PKI-Funktionalität

## WS-SecureConversation

- Konzept für sichere „Sitzung“ mit mehrfachem Nachrichtenaustausch
- Sitzungsschlüsselerstellung zu Beginn der Konversation (Security Context Token)
- Absicherung der Konversation mit Sitzungsschlüssel
- 3 Varianten:
  - Sicherheitskontext wird durch vertrauenswürdigen externen Dienst etabliert (z.B. Kerberos-Server)
  - Sicherheitskontext wird durch einen vertrauenswürdigen Kommunikationspartner etabliert
  - Sitzungsschlüssel wird nach Diffie-Hellman erzeugt und ausgetauscht

# SAML - Security Assertion Markup Language

OASIS-Standard mit drei Kernbereichen:

- SAML-Assertions

Sprache zur Formulierung von Sicherheitsaussagen

- Authentication Assertion = Zugriffsberechtigung auf Ressourcen
- Attribute Assertion = Aussagen über eine Rolle eines Benutzers
- Authorization Decision Assertion = Formulierung von Voraussetzungen für den Zugriff auf eine Ressource

(z.B. „Die Berechtigung des Dienstenutzers wurde erfolgreich geprüft“)

- Protokolle zur Anforderung und Übermittlung von Zertifikaten oder SAML-Assertions
- Bindungen und Profile

spezifizieren die Möglichkeiten zur Einbindung von SAML-Aussagen in andere Dokumente

## **SOA und Single Sign-On**

aus Wikipedia:

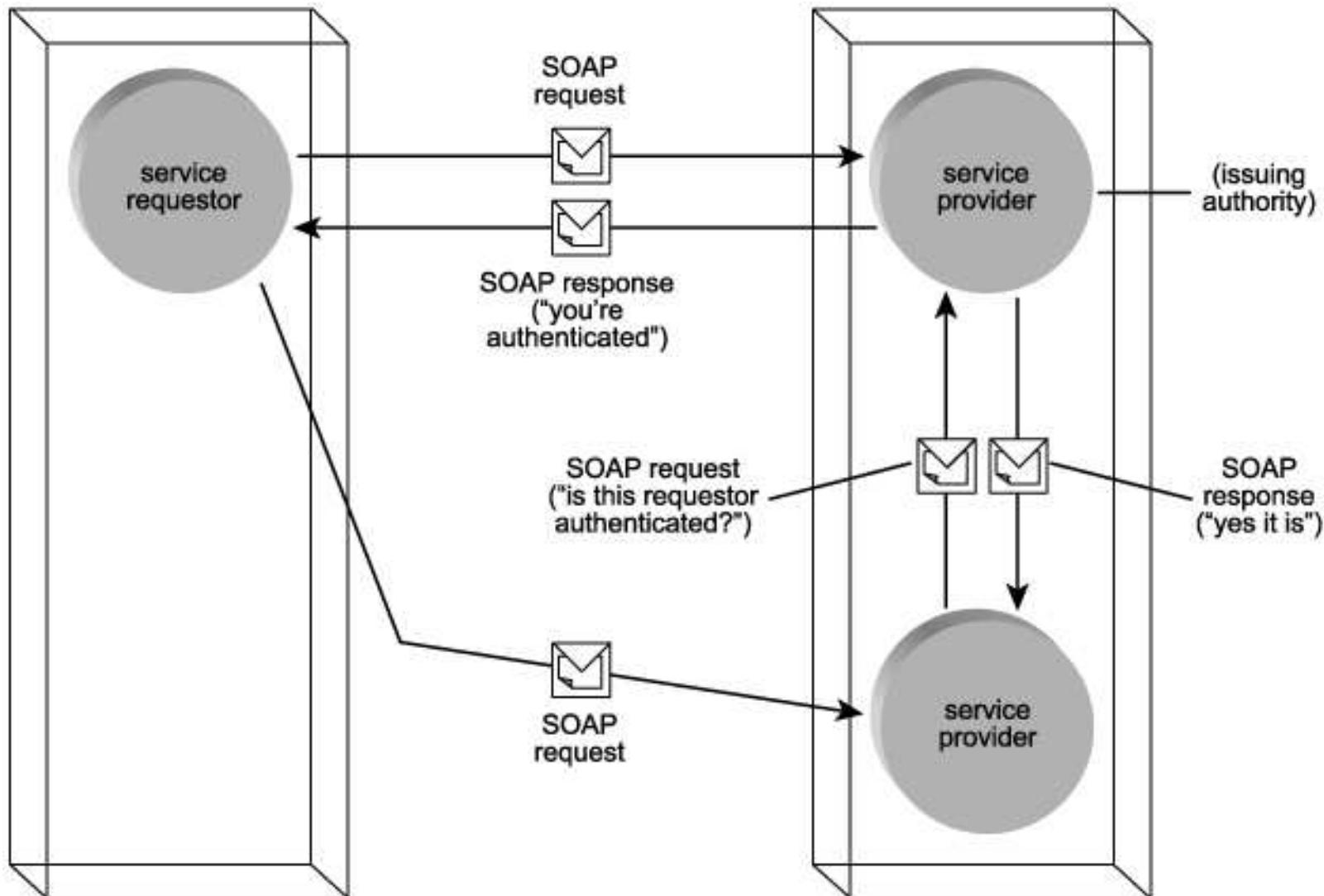
Eine Einmalanmeldung bzw. Single Sign-On (kurz SSO) bedeutet, dass ein Benutzer nach einer einmaligen Authentifizierung auf alle Rechner und Dienste, für die er berechtigt ist, zugreifen kann, ohne sich jedes Mal neu anmelden zu müssen.

### **SOA-typisches Szenario:**

Dienstenutzer eines externen Geschäftspartners

- nutzt verschiedene interne Dienste
- interne Dienste benötigen alle Authentifizierung
- Dienstenutzer authentifiziert sich nur einmal
- Typische Einmalauthentifizierung: Login mit Passwort, Smartcard oder biometrischer Prüfung

## Single Sign-On mit SAML



## **Sicherheit, WS-Policy und WS-SecurityPolicy**

- Die sicherheitstechnischen Anforderungen für die Nutzung eines Dienstes können mit WS-Policy formal spezifiziert werden
- WS-SecurityPolicy definiert die speziellen Policy-Elemente für die Sicherheitsanforderungen, z.B. Anmelde- und Authentifizierungsmöglichkeiten für die Nutzung eines Dienstes

## Beispiel für Richtlinie mit Sicherheitsanforderungen

Verlangt werden X.509-Zertifikat, digitale Signatur und 3DES-Verschlüsselung:

```
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy"
  xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext">
  <wsp:All>
    <wsse:SecurityToken wsp:Usage="wsp:Required">
      <wsse:TokenType>wsse:X509</wsse:TokenType>
    </wsse:SecurityToken>
    <wsse:Integrity wsp:Usage="wsp:Required">
      <wsse:Algorithm Type="wsse:AlgSignature"
        URI="http://www.w3.org/2000/09/xmlenc#aes" />
    </wsse:Integrity>
    <wsse:Confidentiality wsp:Usage="wsp:Required">
      <wsse:Algorithm Type="wsse:AlgEncryption"
        URI="http://www.w3.org/2001/04/xmlenc#3des-cbc"/>
      <MessageParts>
        wsp:GetNodesetForNode(wsp:GetBody(..))
      </MessageParts>
    </wsse:Confidentiality>
  </wsp:All>
</wsp:Policy>
```

## Anmerkungen zu einer Sicherheitsarchitektur

- Oft ist Firewall-Architektur sinnvoll (vertrauenswürdige Zone vom unsicheren Netz abschotten)
- Sicherheitsanforderungen als intermediäre Firewall-Dienste realisieren, z.B.
  - Dienst, der ausgehende Nachrichten signiert und ggf. chiffriert
  - Dienst für Sicherheitsfunktionen bei eingehenden Nachrichten
    - \* Dechiffrierung
    - \* Integritätsprüfung
    - \* Absenderauthentifizierung
    - \* Berechtigungsprüfung
    - \* Erzeugung eines Absende-Nachweises
- für Sicherheit zuständige Dienste sind Geschäftsprozess-unabhängig und gut wiederverwendbar
- Änderungen in den Sicherheitsrichtlinien betreffen nur den Sicherheitsdienst (Isolation)

- Sicherheits-Proxy für eingehende Nachrichten könnte sämtliche Sicherheitsheader vor der Weiterleitung entfernen. Sicherheitsaspekte hinter dem Proxy wären dann völlig versteckt. (Nutzen?)
- In komplexeren Anwendungsumgebungen wird PKI- und Sicherheitspolicy-Verwaltung benötigt.

## Fazit

- Sicherheitskonzepte auf unterschiedlichen Protokollschichten ergänzen sich
- Auf XML- und WS-Ebene gibt es sehr fortgeschrittene Sicherheitskonzepte für die unterschiedlichsten Anwendungsfälle
- Im Zusammenspiel mit WS-Policy ist eine zentrale Verwaltung von Sicherheitsrichtlinien möglich
- Sicherheitsaspekte lassen sich sauber von der Kernfunktionalität der Geschäftsprozesse trennen
- Die WS-Sicherheitsstandards werden derzeit noch intensiv weiterentwickelt.
- SOAP selbst hat keine („fest verdrahteten“) Sicherheitskonzepte
- Die Integration der WS-Sicherheitserweiterungen in die SOAP-Nachrichtenübermittlung zeigt die einfache Erweiterbarkeit des WS-Nachrichtenkonzepts

## Fallbeispiel

TLS hat für alle Geschäftsdokument-Nachrichten folgende Security-Richtlinie:

- Alle Geldbeträge innerhalb der Nachricht sind zu verschlüsseln
- Jede Rechnung an TLS mit einem Betrag von über \$30000 ist digital zu signieren

RailCO verwendet in seinem InvoiceSubmission-Dienst

- XML Encryption für die Verschlüsselung der Geldbeträge
- XML Signature für die digitale Unterschrift

## Fallbeispiel: Authentifizierung

```
<Envelope
  xmlns="http://www.w3.org/2003/05/soap-envelope">
  <Header>
    <wsse:Security xmlns:wsse=
      "http://schemas.xmlsoap.org/ws/2002/12/secext">
      <wsse:UsernameToken>
        <wsse:Username>
          rco-3342
        </wsse:Username>
        <wsse:Password Type="wsse:PasswordDigest">
          93292348347
        </wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </Header>
  <Body>      ...      </Body>
</Envelope>
```

## Fallbeispiel: Single Sign-On mit SAML

```
<Envelope ...>
  <Header>
    <wsse:Security xmlns:wsse=
      "http://schemas.xmlsoap.org/ws/2002/12/secext">
      <saml:Assertion xmlns:saml="..." ...>
        <saml:Conditions ...>
        <saml:AuthorizationDecisionStatement
          Decision="Permit"
          Resource="http://www.xmltc.com/tls/...">
          <saml:Actions>
            ...
            <saml:Action>
              Execute
            </saml:Action>
          </saml:Actions>
          ...
        </saml:AuthorizationDecisionStatement>
      </wsse:Security>
    </Header> ...
```

## Fallbeispiel: Geldbeträge mit und ohne Verschlüsselung

```
<InvoiceType>          <InvoiceType>
  <Number>            <Number>
    2322              2322
  </Number>           </Number>
  <Total>             <EncryptedData
    $32,322.73       xmlns="http://www.w3.org/2001/04/xmlenc#"
  </Total>             Type="http://www.w3.org/2001/04/xmlenc#Element">
  <Date>               <CipherData>
    07.16.05          <CipherValue>
  </Date>              R5J7UUI78
</InvoiceType>          </CipherValue>
                        </CipherData>
                        </EncryptedData>
                        <Date>
                          07.16.05
                        </Date>
</InvoiceType>
```

## Fallbeispiel: Digitale Signatur

```
<Envelope ...>
  <Header>
    <wsse:Security xmlns:wsse=
      "http://schemas.xmlsoap.org/ws/2002/12/secext">
      <Signature Id="RailCo333" xmlns=
        "http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
          <CanonicalizationMethod Algorithm=
            "http://www.w3.org/TR/2001/
              REC-xml-c14n-20010315"/>
          <SignatureMethod Algorithm=
            "http://www.w3.org/2000/09/
              xmldsig#dsa-sha1"/>
          <Reference URI=
            "http://www.w3.org/TR/2000/
              REC-xhtml1-20000126/">
          <DigestMethod Algorithm=
            "http://www.w3.org/2000/09/
              xmldsig#sha1"/>
```

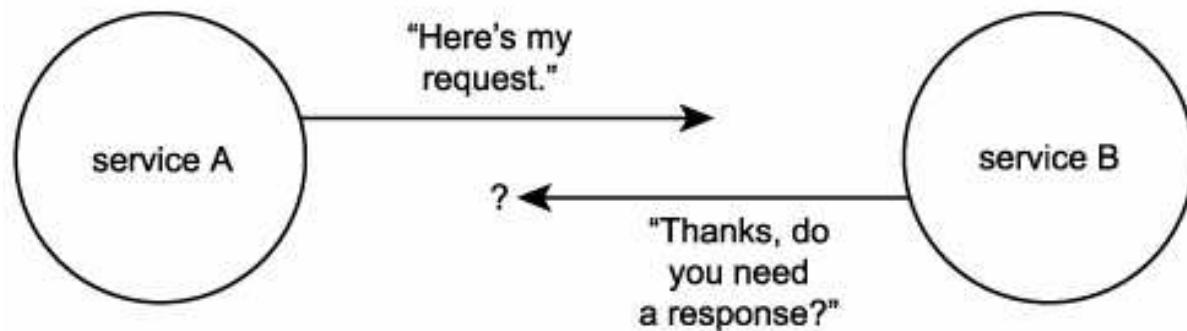
```
<DigestValue>
    LLSFK032093548=
</DigestValue>
</Reference>
</SignedInfo>
<SignatureValue>
    9879DFSS3=
</SignatureValue>
<KeyInfo>
    ...
</KeyInfo>
</Signature>
</wsse:Security>
</Header>
<Body>
    ... invoice document with total exceeding $30,000...
</Body>
</Envelope>
```

## **WS-\*-Standards zur Modellierung komplexer Geschäftsaktivitäten**

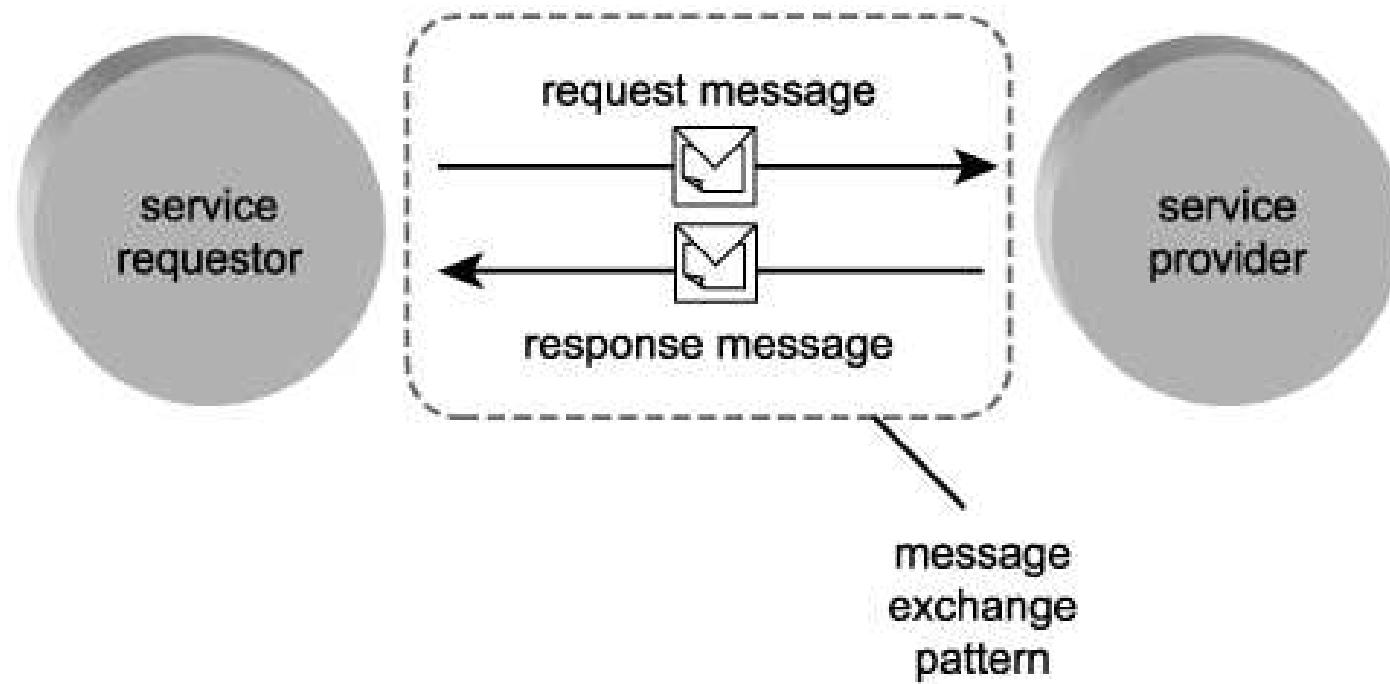
SOA – Technische Konzepte zur Modellierung auf unterschiedlichen Ebenen:

- Bezug zwischen Aktivitäten und Nachrichten
- Bezug zwischen Aktivitäten und Web Services
- Muster für den Nachrichtenaustausch
- Koordinationsmuster
- Transaktionen
- Orchestrierung von Diensten
- Choreographie
- Geschäftsprozesse

## Beispiel: MEPs – Message Exchange Patterns

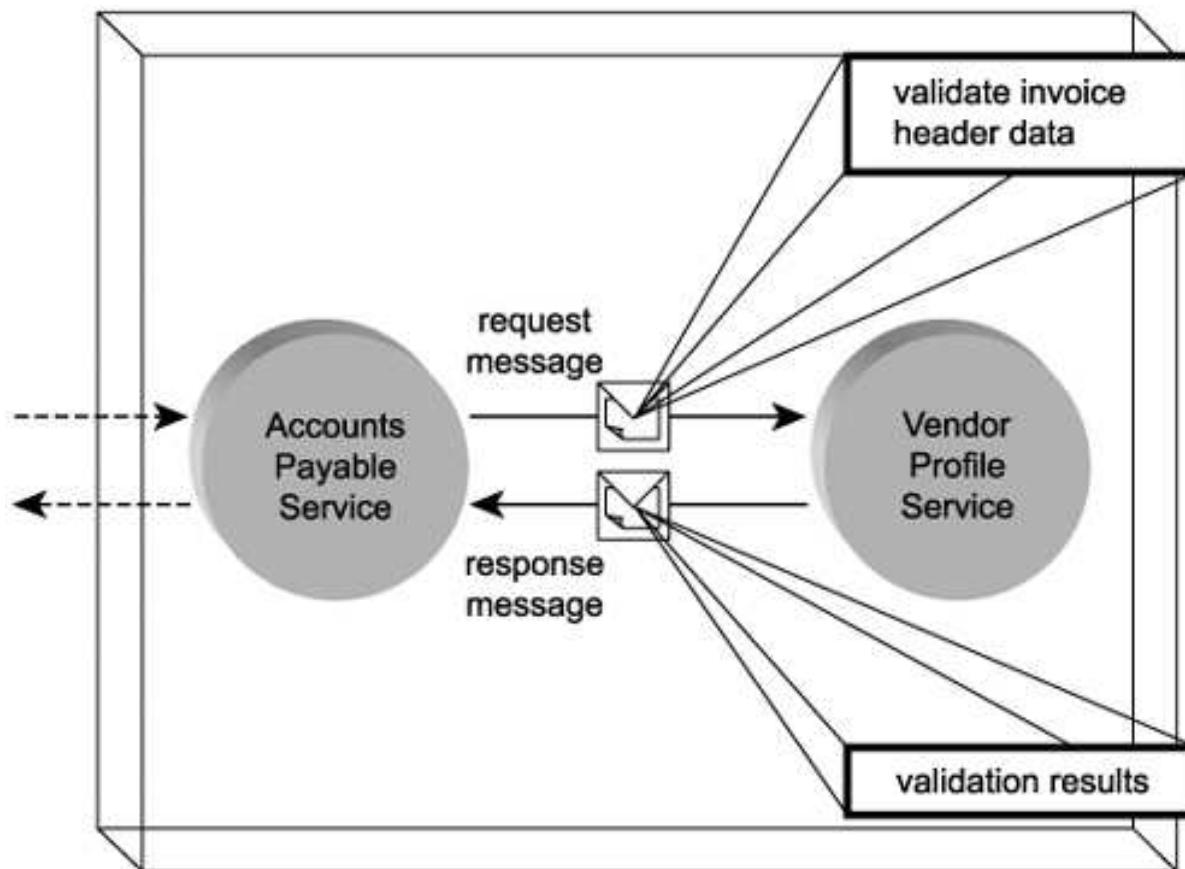


## Muster: Auftrag - Bestätigung (Request-Response)

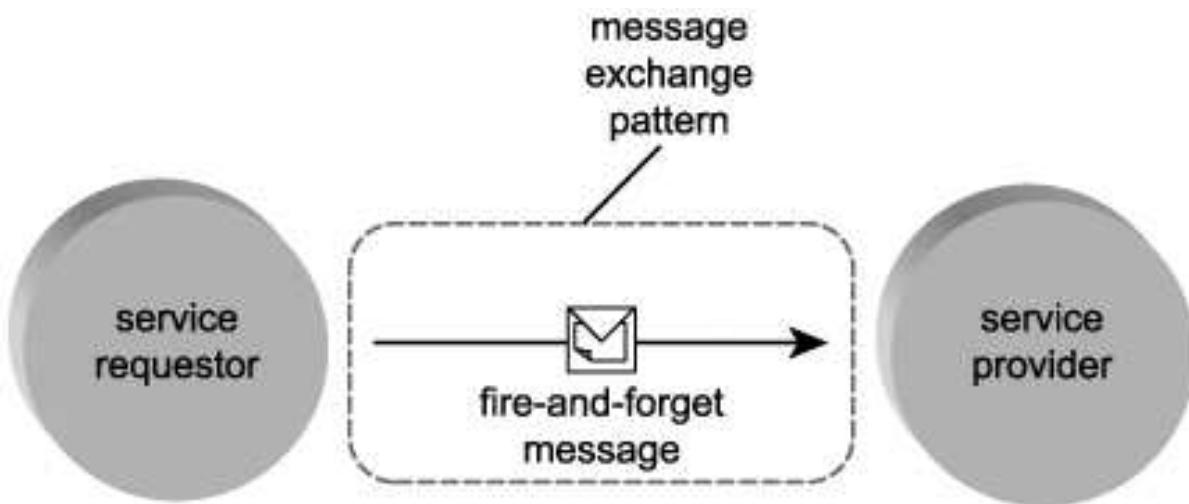


(Korrelationsinformation nötig!)

## Auftrag-Bestätigung-Muster im Fallbeispiel



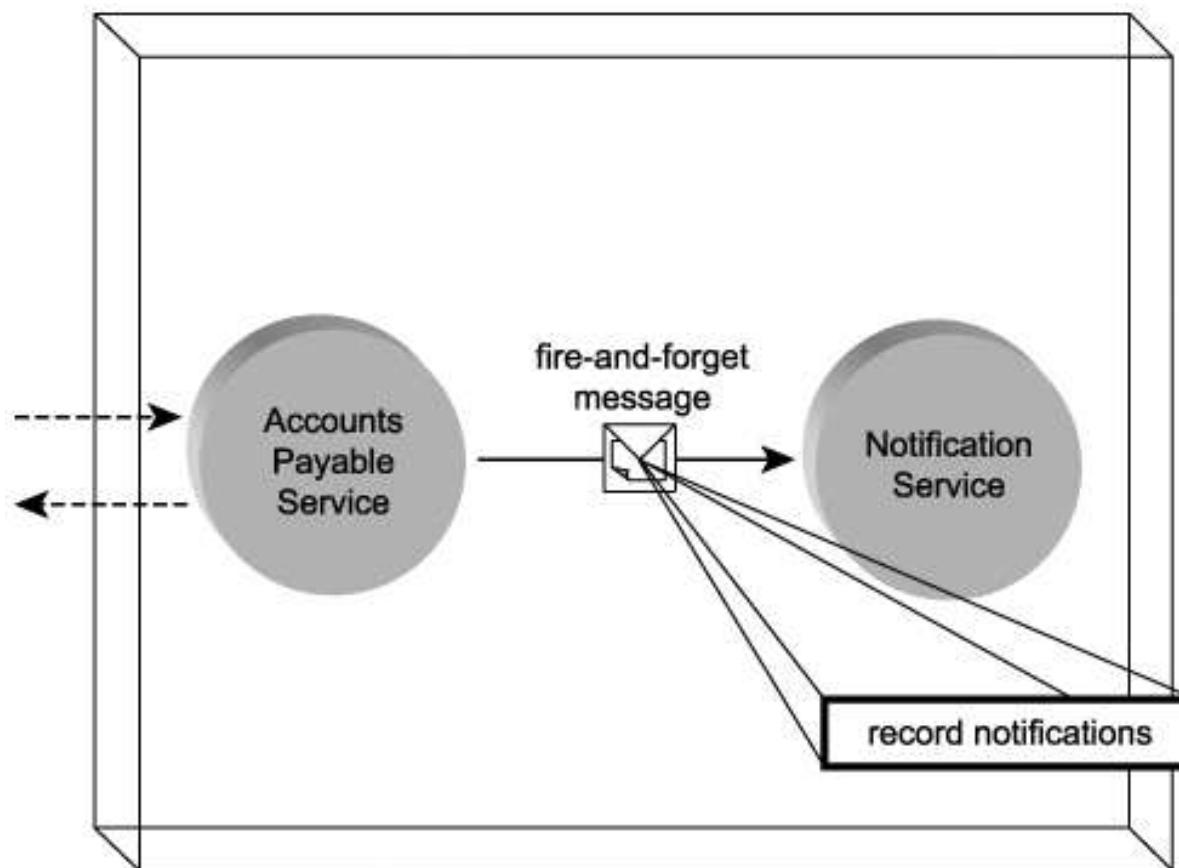
## Abschicken und vergessen (Fire and Forget)



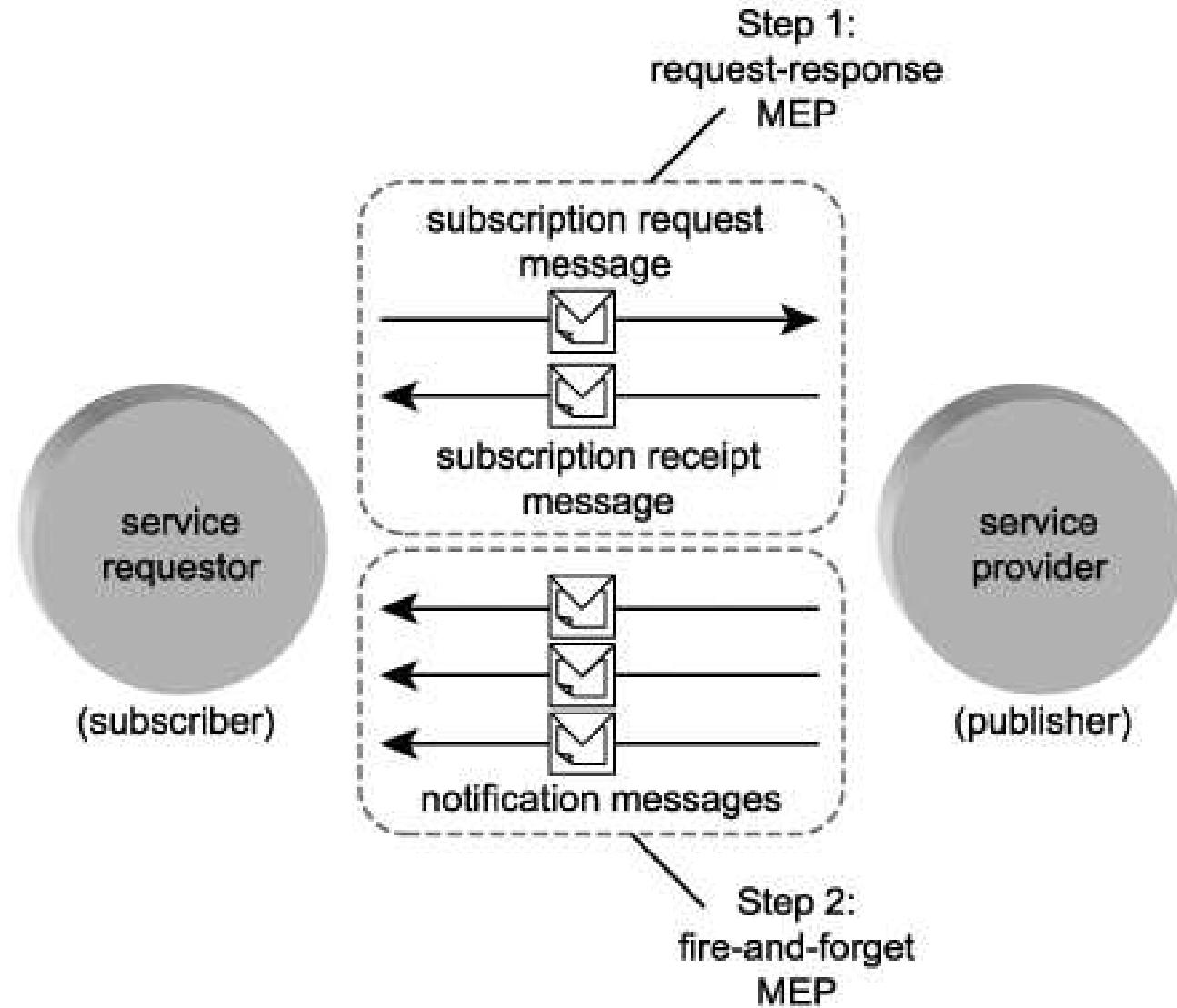
Varianten:

- Unicast, Multicast, Broadcast

## Fire-and-Forget im Fallbeispiel



## Komplexes Muster: Veröffentlichen/Abonnieren (Publish-Subscribe)



## **Beispiel für Bezug zwischen MEPs und WS-\*-Erweiterungen**

Für „publish/subscribe“ nutzbare WS-\*-Erweiterungen:

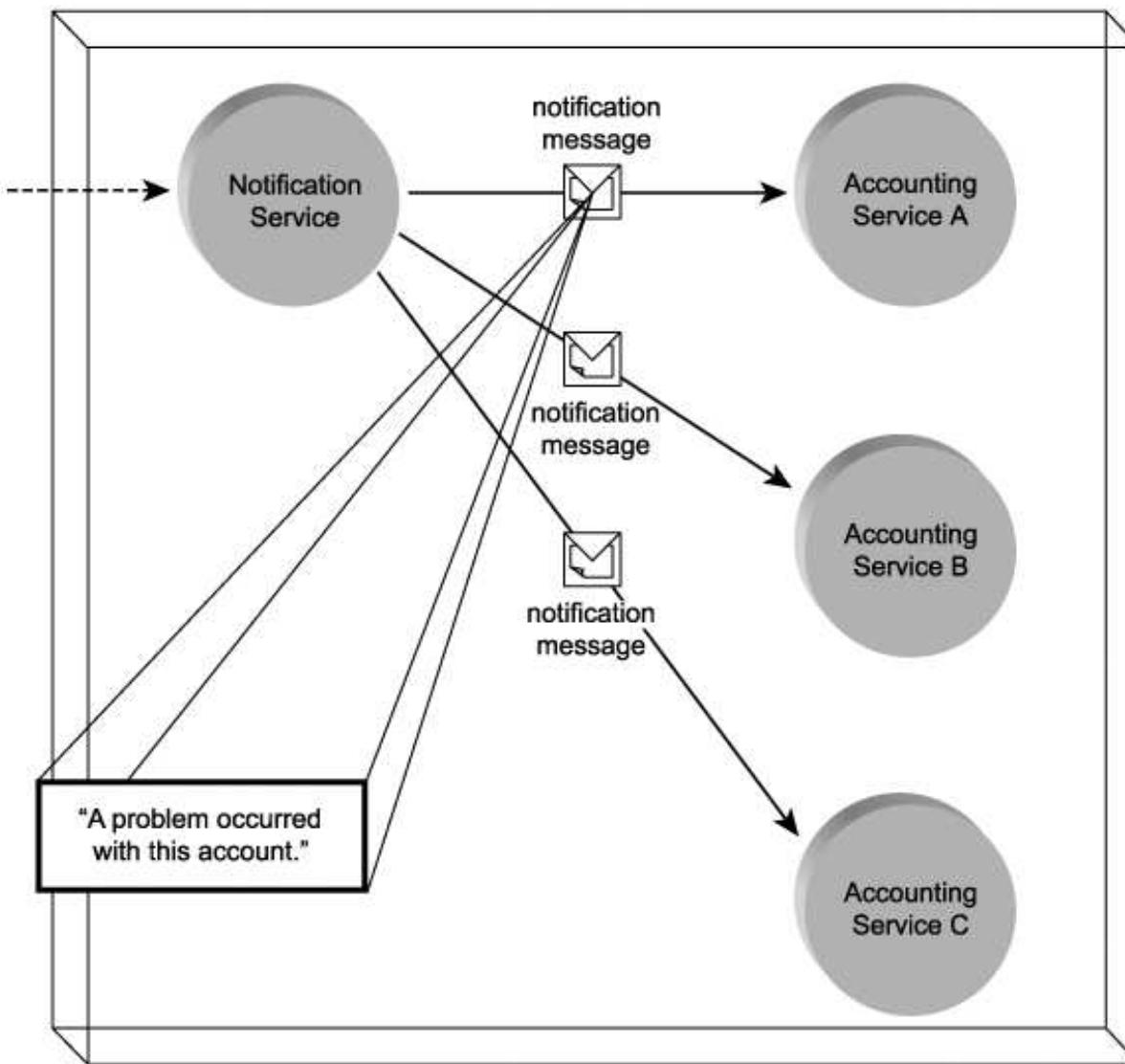
- WS-BaseNotification
- WS-BrokeredNotification
- WS-Topics
- WS-Eventing

## **Publisher-Subscriber-Muster im Fallbeispiel**

### TLS-Benachrichtigungsdienst

- erzeugt und verschickt Benachrichtigungen verschiedener Art
- sammelt z.B. Meldungen über Fehler bei Bearbeitung von Nachrichten externer Partner
- Aggregierte Informationen zu diesen Fehlern werden als Broadcast-Nachrichten veröffentlicht
- Buchführungsdienste abonnieren diese Nachrichten, um sie weiterzuverarbeiten

## Publisher-Subscriber-Muster im Fallbeispiel



## MEPs, SOAP und WSDL

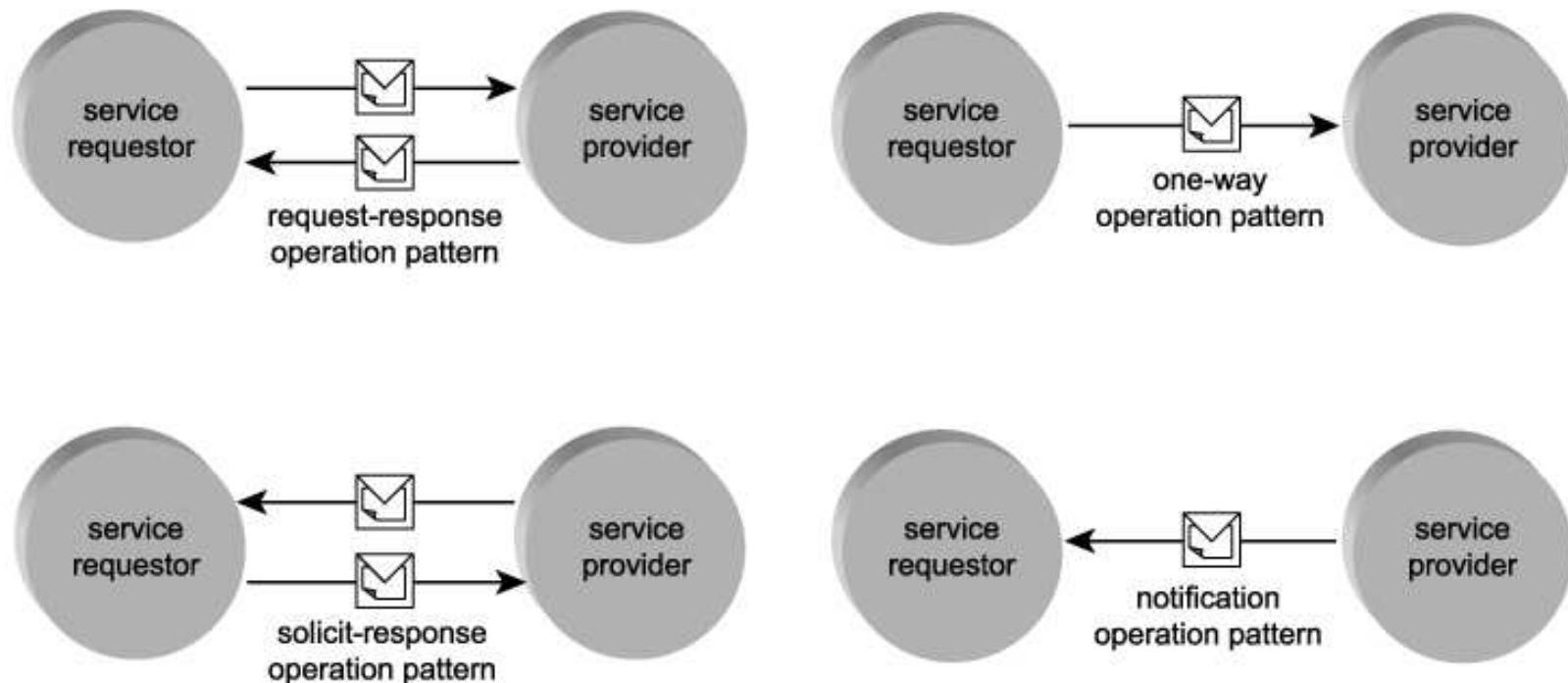
### MEPs und SOAP

- SOAP selbst sieht nur Einweg-Übertragung von Nachrichten vor
- Durch Unterstützung der WS-\*-Erweiterungen via SOAP-Header können **beliebige MEPs** realisiert werden

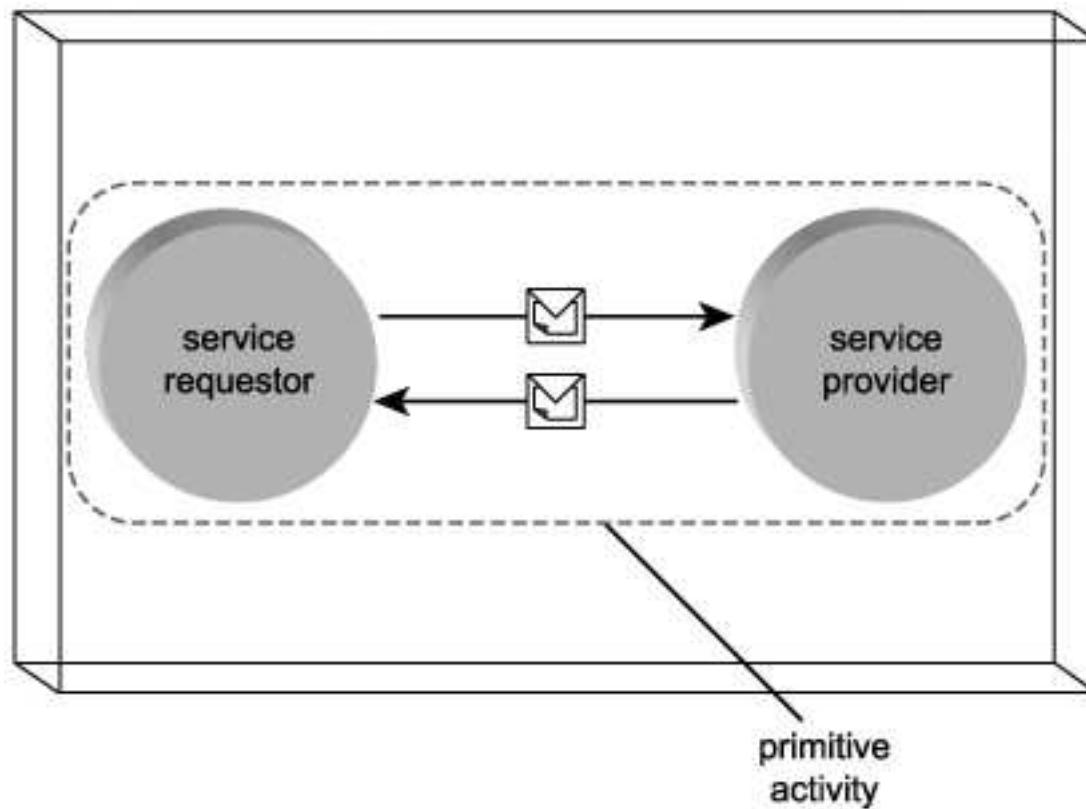
### MEPs und WSDL

- WSDL ordnet den Operationen einer Schnittstelle Input- und Output-Nachrichten zu
- Zusätzlich gibt es Fehler-Nachrichten für den Ausnahmefall
- In der neuen WSDL-Spezifikation 2.0 werden diverse MEPs unterstützt  
(in-out, out-in, in-only, out-only, robust-in-only, robust-out-only, in-optional-out, out-optional-in)

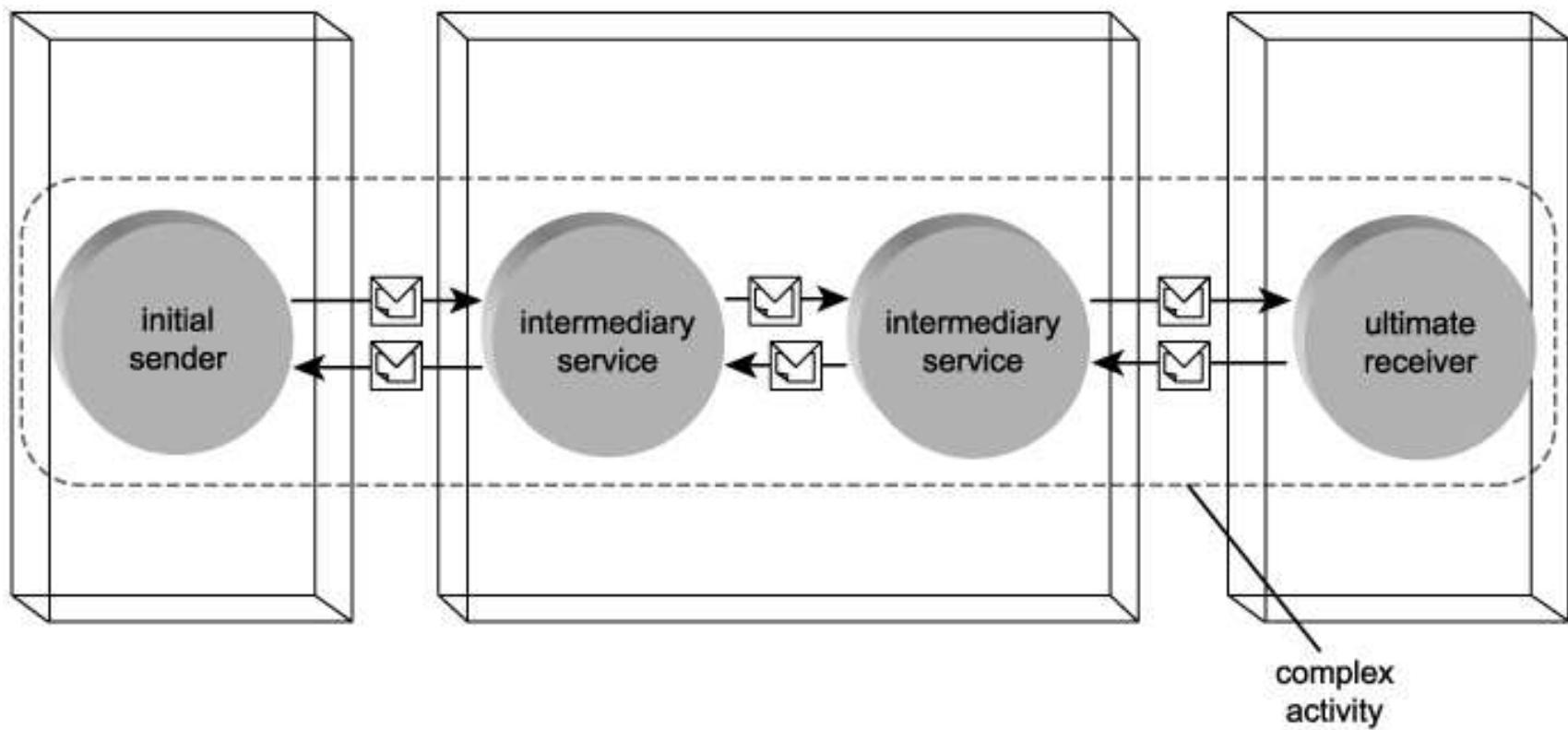
## WDSL 1.1 - Basismuster für Nachrichtenaustausch



## Einfache Aktivität



## Komplexe Aktivität

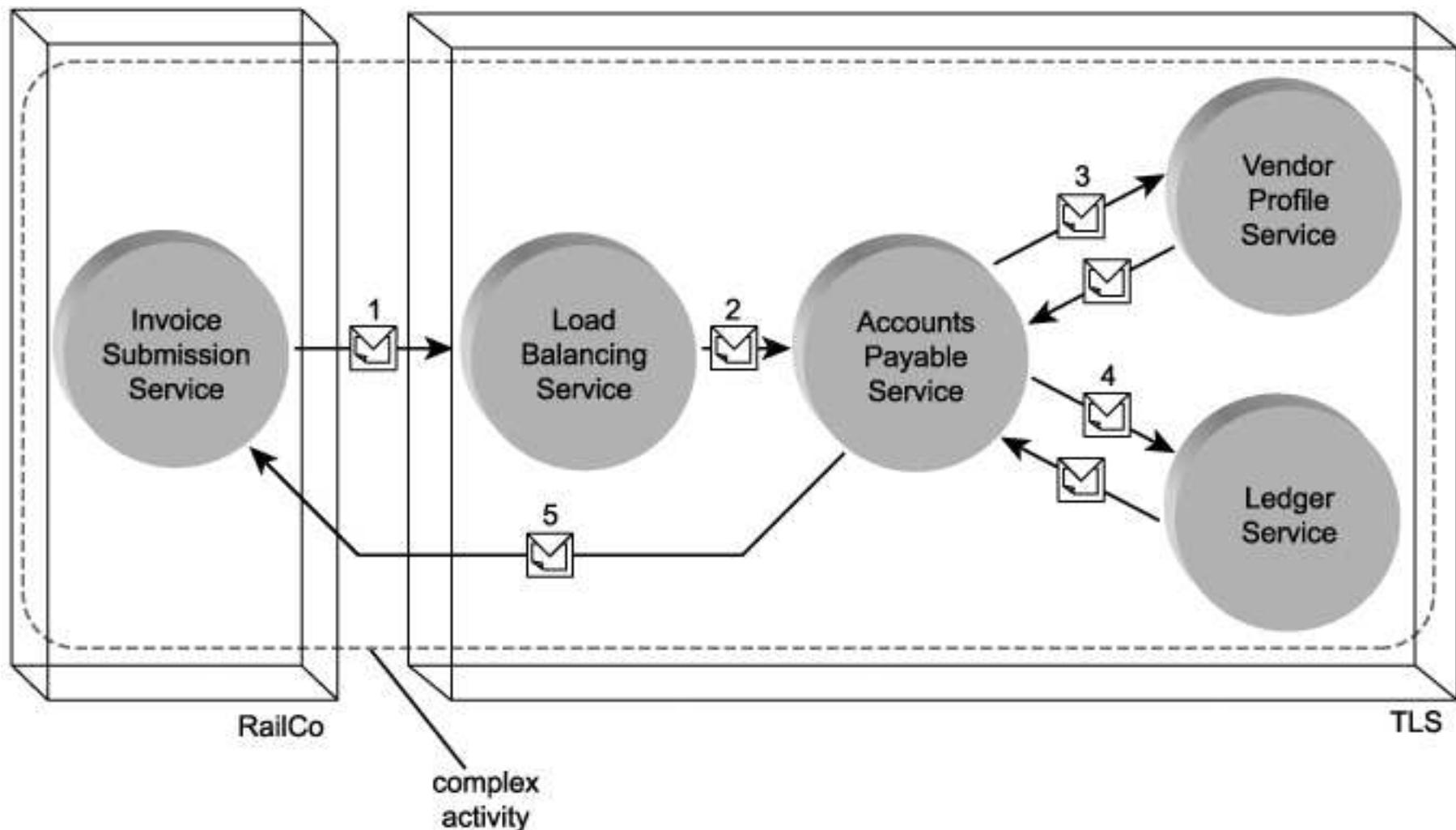


## Komplexe Aktivität im Fallbeispiel

Rechnungsstellung als komplexe Aktivität:

- InvoiceSubmission Service von RailCo schickt Rechnungsnachricht
- LoadBalancing Service von TLS empfängt Rechnung, leitet weiter an eine Instanz des Accounts Payable Service im Servercluster
- Accounts Payable Service
  - agiert als Controller in einer Komposition mit Lieferantenprofil-Service
  - verifiziert Lieferantendaten in der Rechnung und
  - verbucht Rechnung im Lieferantenkonto
- Accounts Payable Service sendet aggregierte Daten (Steuern, Versandkosten, Gesamtbetrag) an den Journal-Service, der diverse Journaltabellen aktualisiert
- Accounts Payable Service sendet eine Bestätigungsnachricht an RailCO und beendet damit die Aktivität

## Komplexe Aktivität im Fallbeispiel



## Gemeinsame Merkmale der Erweiterungen

- Die Erweiterungen werden durch SOAP-Nachrichtenheader genutzt
- Die Nutzung kann mittels „mustUnderstand“-Attribut als zwingend oder optional vereinbart werden

Beispiel:

```
<Header>
  <x:CorrelationID
    xmlns:x="http://www.xmltc.com/tls/headersample/"
    mustUnderstand="1">
    0131858580-JDJ903KD
  </x:CorrelationID>
</Header>
```

## WS-Addressing

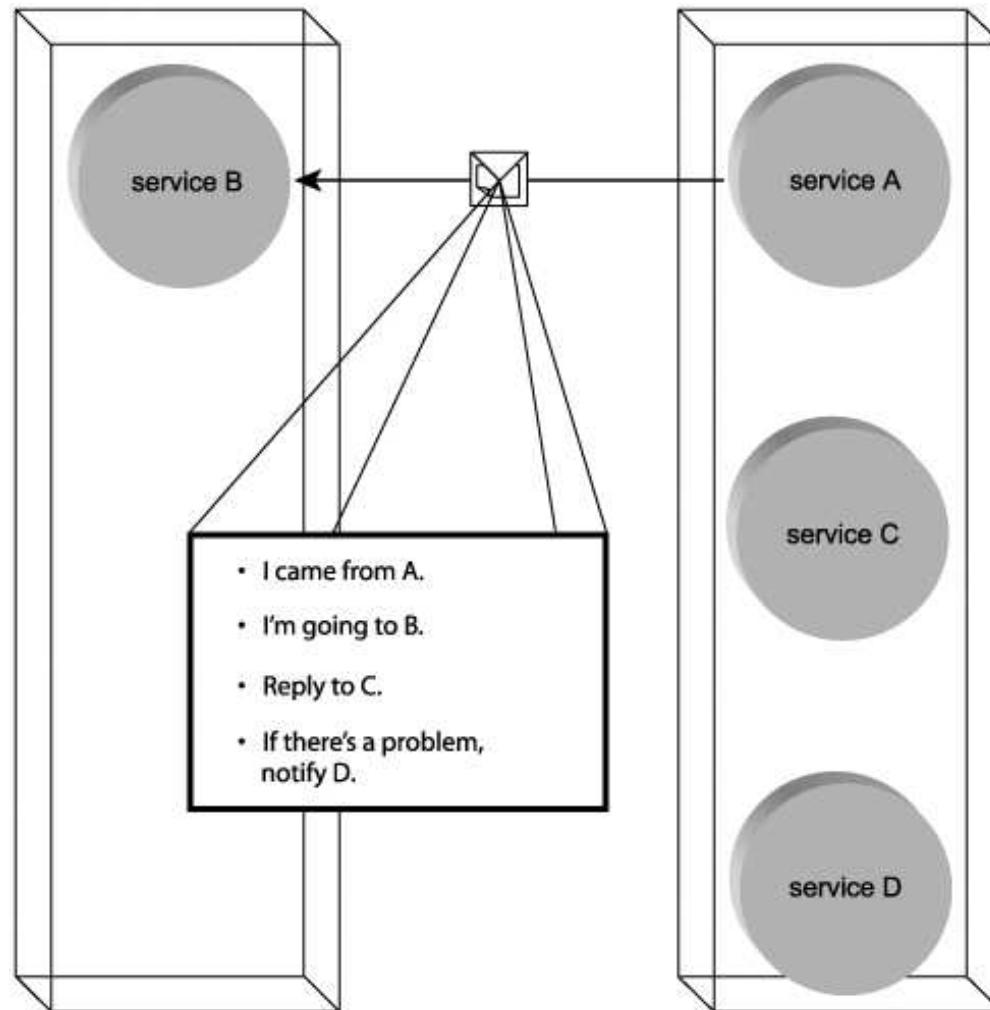
### Konzepte:

- Endpunkt-Referenz
  - Verweis auf bestimmte Instanz eines Dienstes
  - Von Absender-Instanz erzeugt
  - Als SOAP-Header mitversendet
- Message Information Header
  - Dynamische Beeinflussung des Nachrichtenaustausch-Musters
  - Attribute: Ziel, Absender-Endpunkt, Antwort-Endpunkt, Fehler-Endpunkt, Nachrichten-ID, Bezug, Aktion

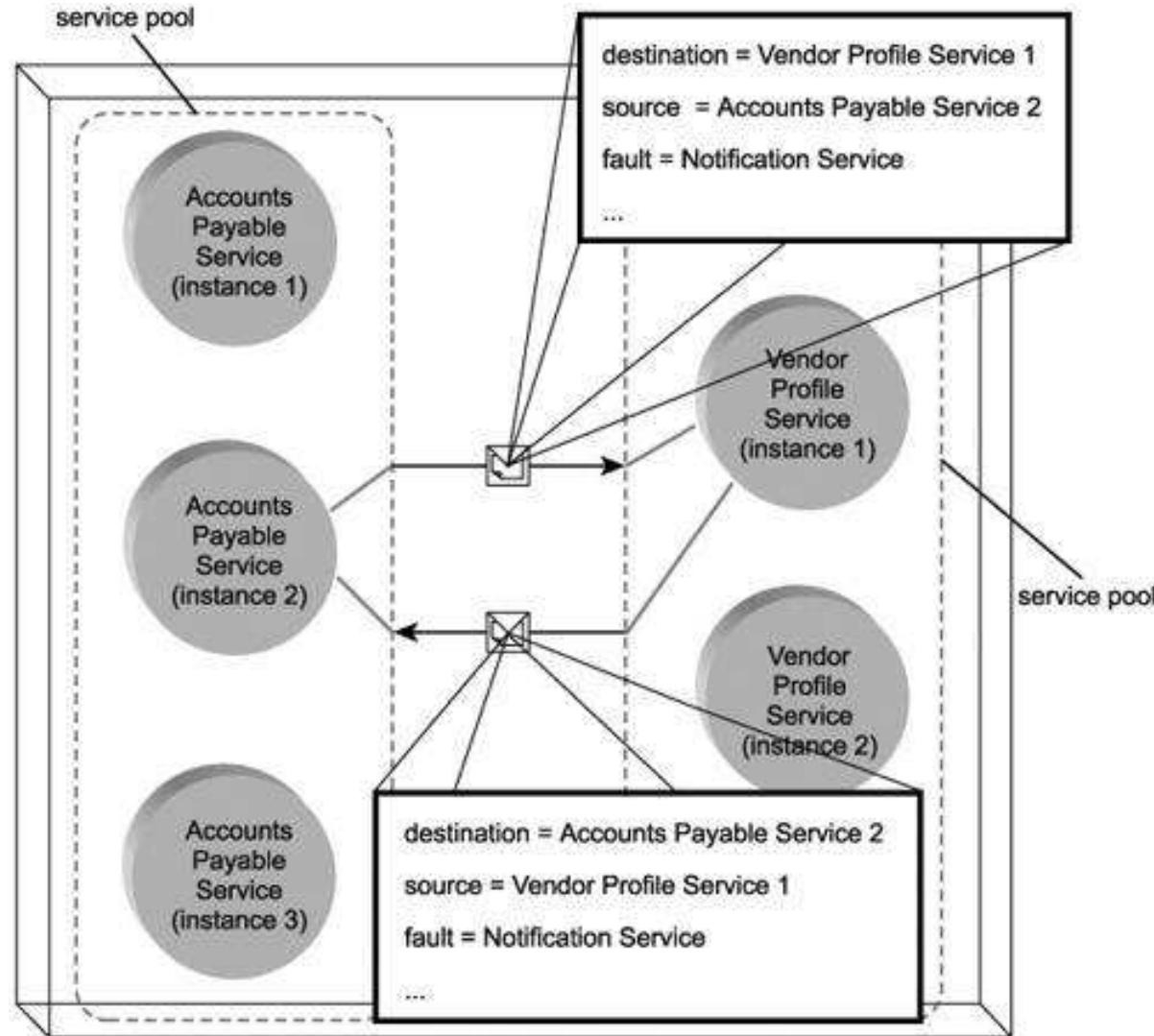
### Ziel: Nachrichtenautonomie

- Aufgabenspezifische Logik (MEP) wird zum Bestandteil der Nachricht
- Wiederverwendbarkeit der Dienste dadurch verbessert

## Message Information Header



## WS-Addressing im Fallbeispiel



## Beispiel für WS-Addressing-SOAP-Headers

```
<s12:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://www.w3.org/2005/08/addressing"
  xmlns:app="http://www.xmltc.com/railco/...">
  <s12:Header>
    <wsa:Action>
      http://www.xmltc.com/tls/vp/submit
    </wsa:Action>
    <wsa:To>
      http://www.xmltc.com/tls/vp/...
    </wsa:To>
```

```
<wsa:From>
  <wsa:Address>
    http://www.xmltc.com/railco/ap1/...
  </wsa:Address>
  <wsa:ReferenceProperties>
    <app:id>
      unn:AFJK3231lws
    </app:id>
  </wsa:ReferenceProperties>
  <wsa:ReferenceParameters>
    <app:sesno>
      22322447
    </app:sesno>
  </wsa:ReferenceParameters>
</wsa:From>
<wsa:MessageID>
  uuid:243234234-43gf433
</wsa:MessageID>
```

```
<wsa:ReplyTo>
  <wsa:Address>
    http://www.xmltc.com/railco/ap2/
  </wsa:Address>
  <wsa:ReferenceProperties>
    <app:id>
      unn:AFJK3231lws
    </app:id>
  </wsa:ReferenceProperties>
  <wsa:ReferenceParameters>
    <app:sesno>
      22322447
    </app:sesno>
  </wsa:ReferenceParameters>
</wsa:ReplyTo>
```

```
<wsa:FaultTo>
  <wsa:Address>
    http://www.xmltc.com/railco/ap-err/
  </wsa:Address>
  <wsa:ReferenceProperties>
    <app:id>
      unn:AFJK323llws
    </app:id>
  </wsa:ReferenceProperties>
  <wsa:ReferenceParameters>
    <app:sesno>
      22322447
    </app:sesno>
  </wsa:ReferenceParameters>
</wsa:FaultTo>
</s12:Header>
<s12:Body>
  ...
</s12:Body>
</Envelope>
```

## Nutzung der MessageID in Antwortnachricht

Ursprüngliche Nachricht: ID im Header

```
<wsa:MessageID>
    uuid:243234234-43gf433
</wsa:MessageID>
```

Antwortnachricht: ID und Verweis auf ursprüngliche Nachricht

```
<wsa:MessageID>
    ...
</wsa:MessageID>

<wsa:RelatesTo>
    uuid:243234234-43gf433
</wsa:RelatesTo>
```

## **WS-ReliableMessaging**

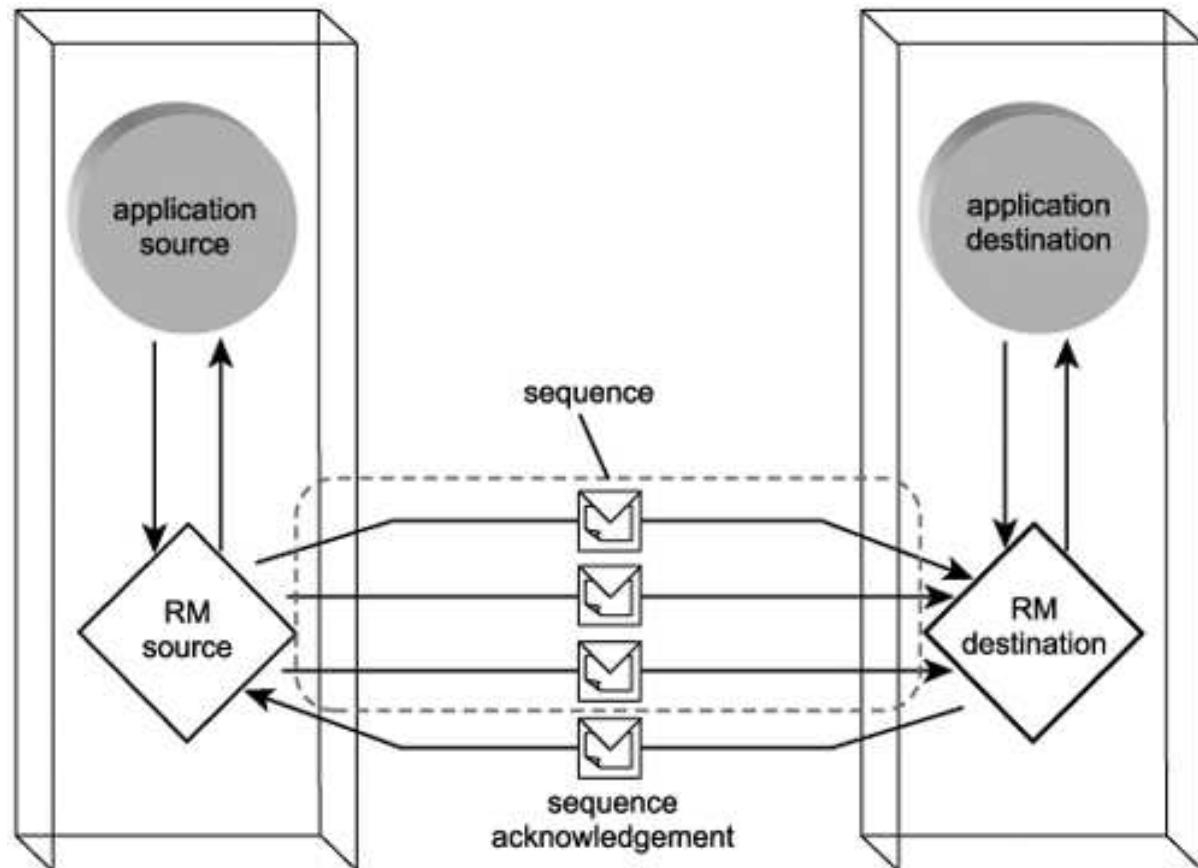
### **Konzepte:**

- Empfangsbestätigungen
- Unterstützung für Nachrichtensequenzen
- Zustellungsgarantie mit Varianten: AtMostOnce, AtLeastOnce, ExactlyOnce, InOrder

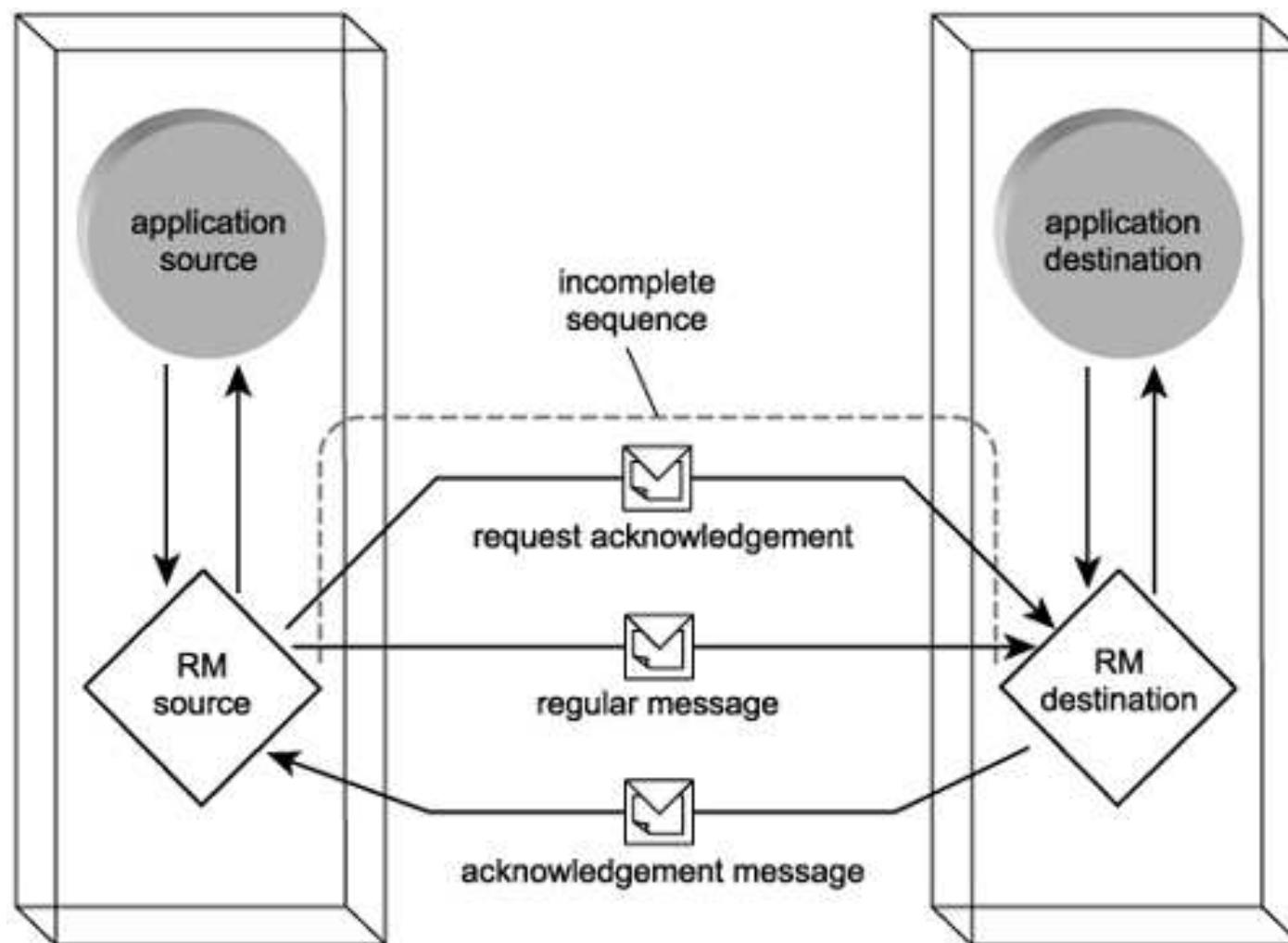
### **Ziel:**

- Robustheit durch Zuverlässigkeitssmerkmale sicherstellen

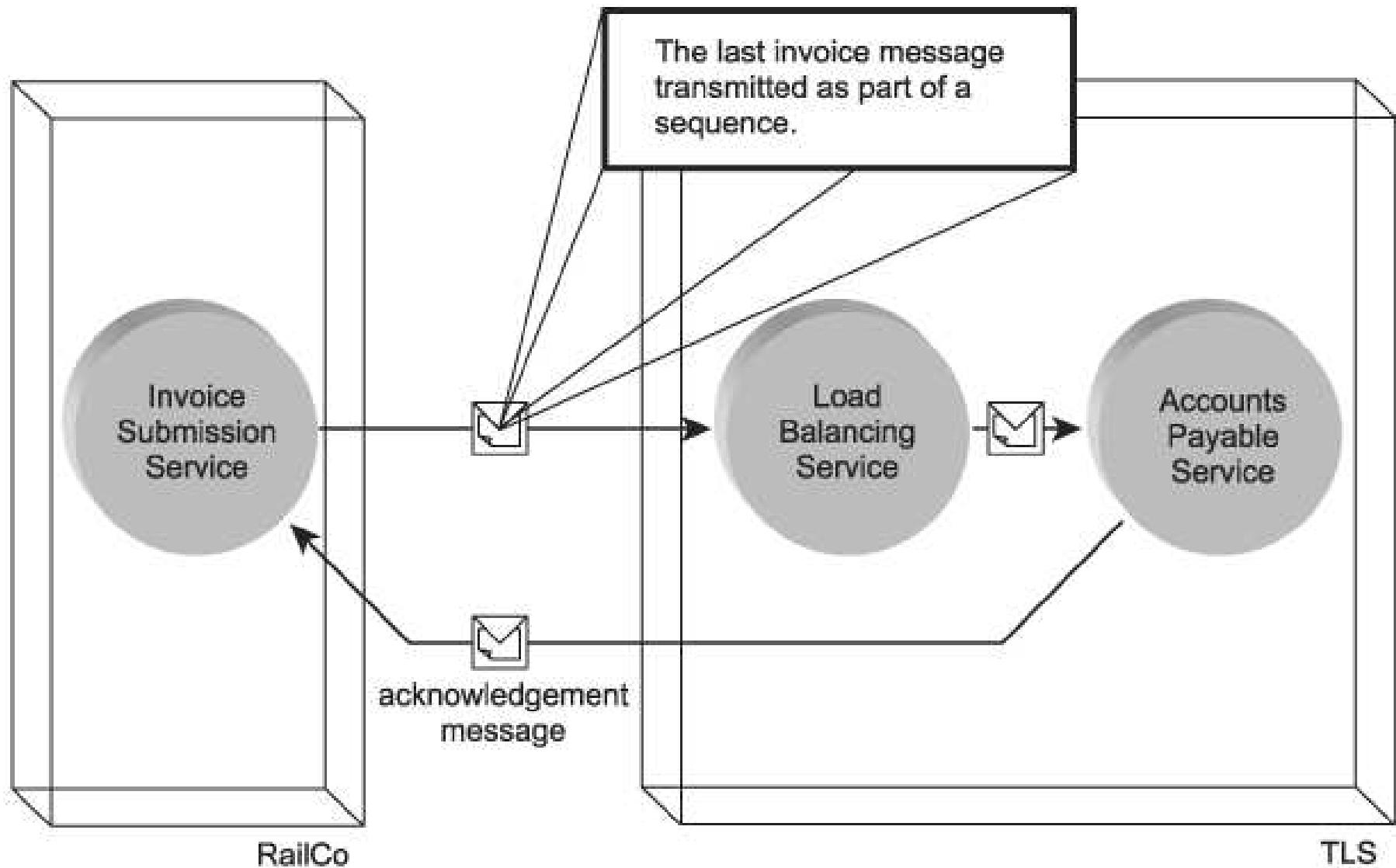
## Nachrichtensequenzen



## Explizite Anforderung einer Empfangsbestätigung

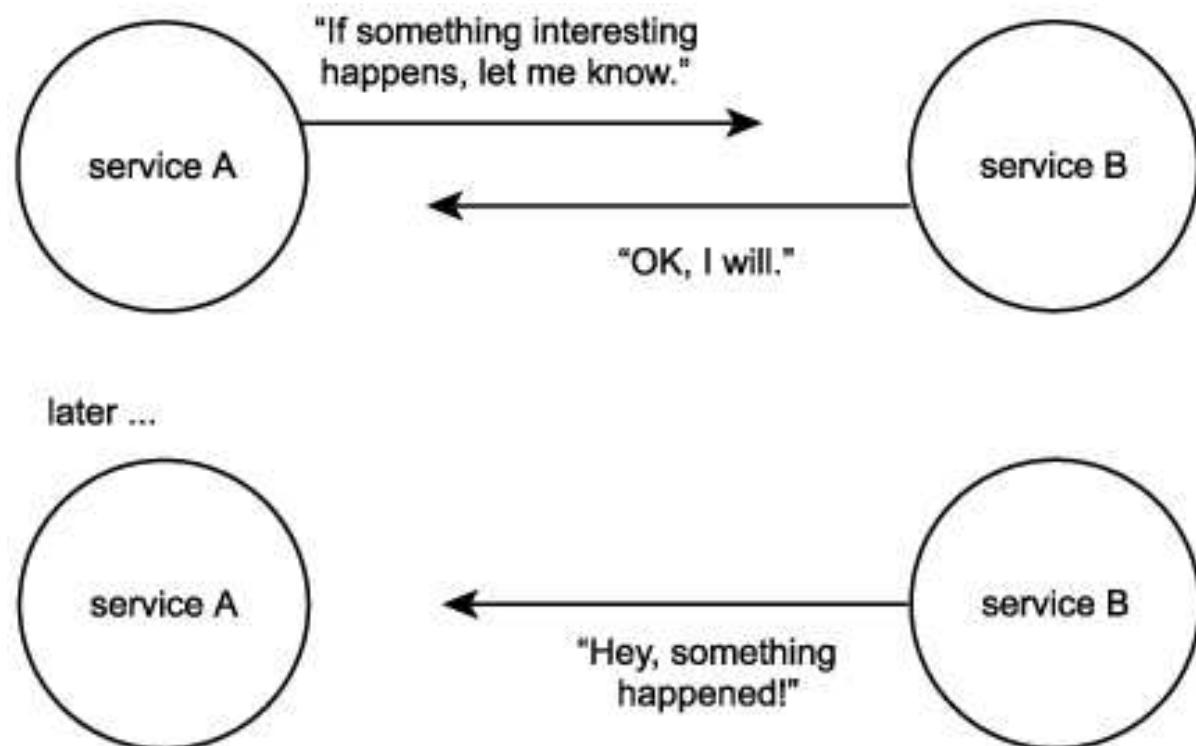


## WS-ReliableMessaging im Fallbeispiel

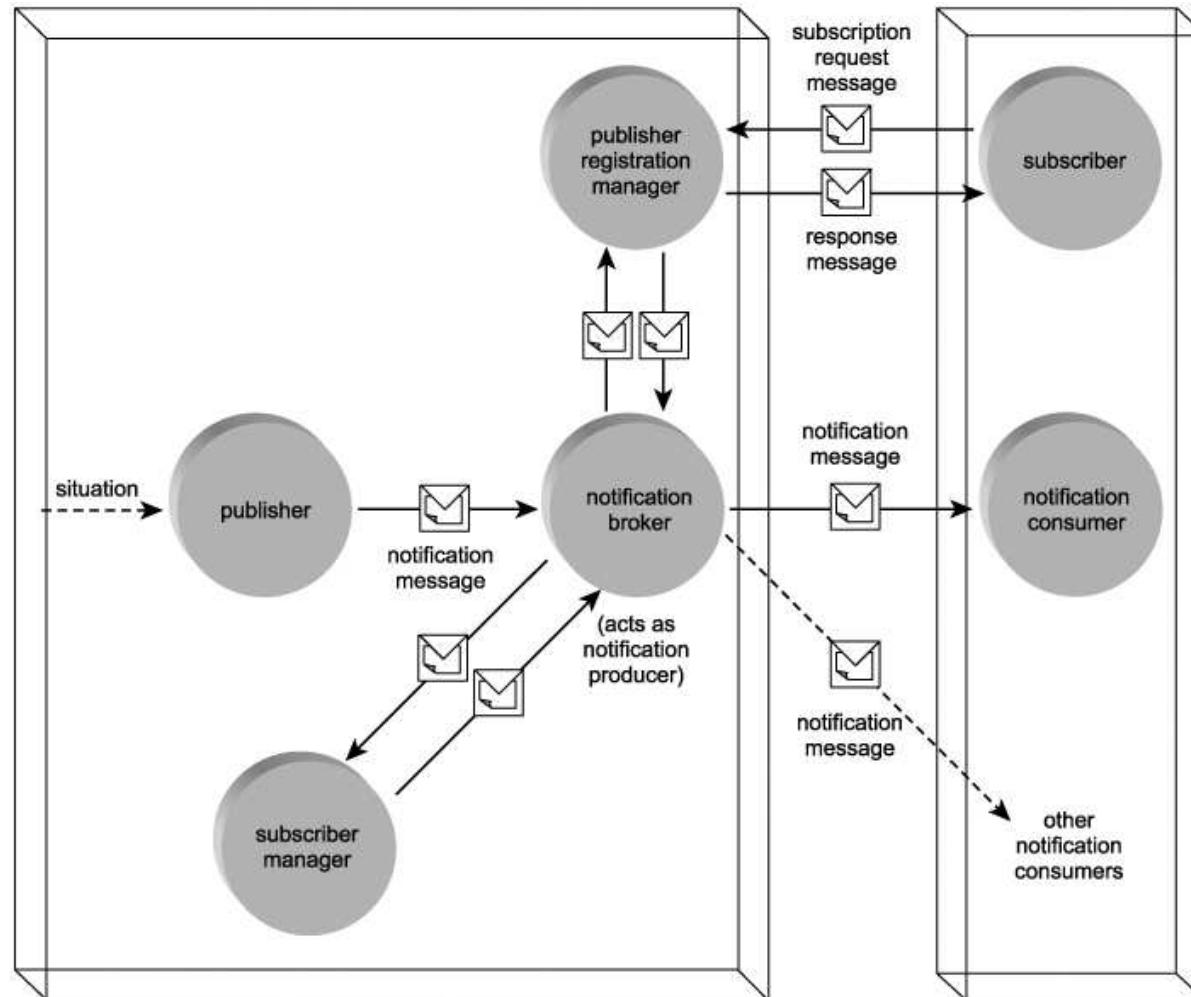


## WS-Notification und WS-Eventing

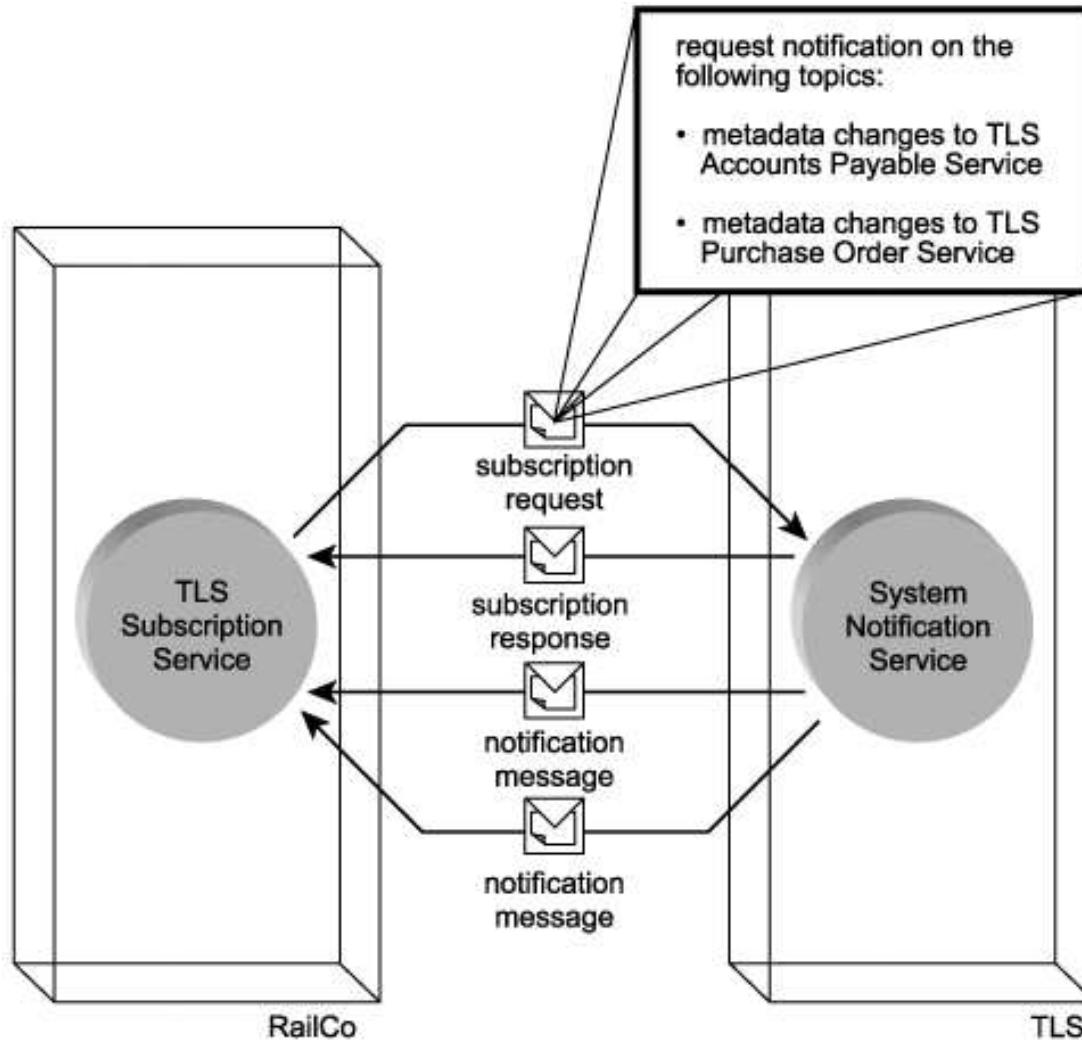
- Konkurrierende Frameworks für Ereignisverwaltung
- Realisierung des Publish-Subscribe-Musters



# Komplexes WS-Notification Benachrichtigungs-Szenario



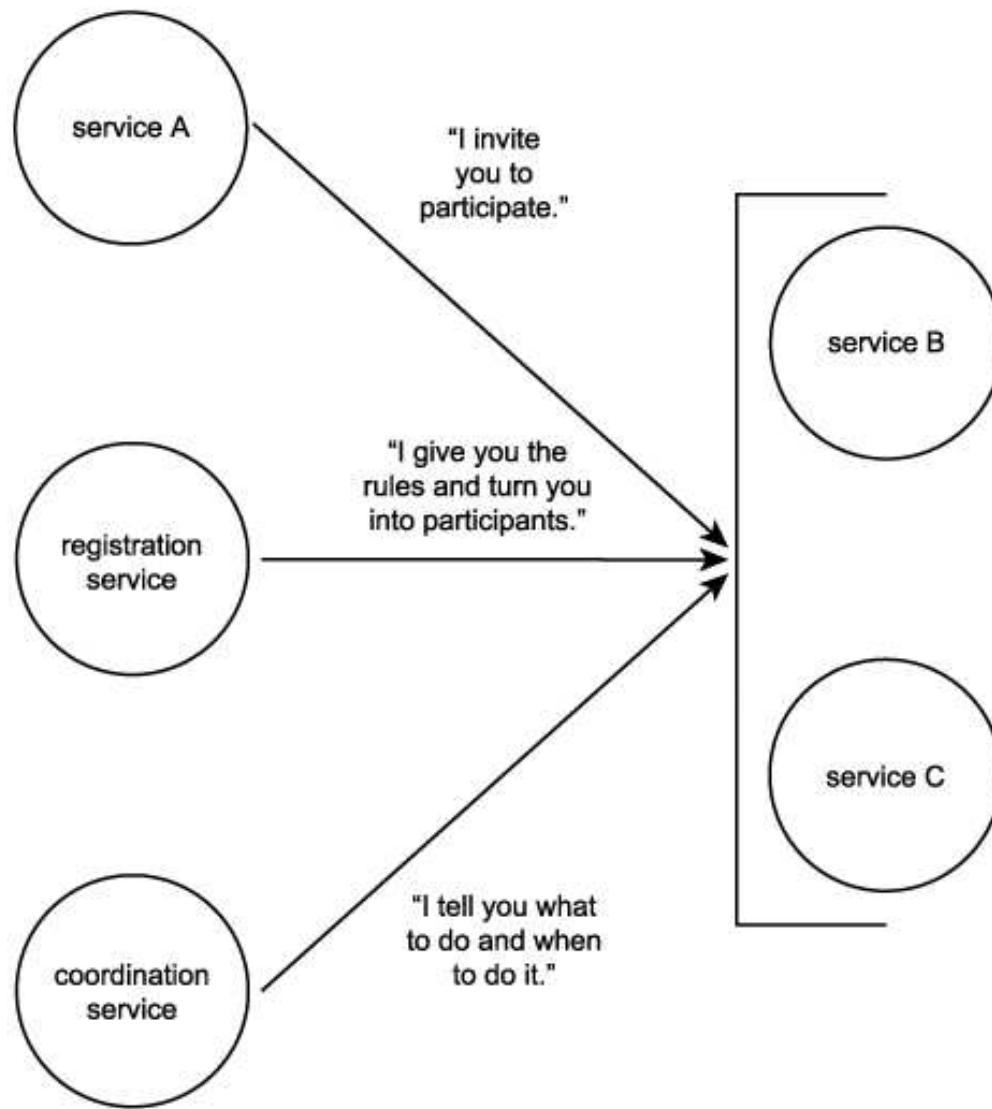
## Benachrichtigungs-Szenario im Fallbeispiel



## WS-Coordination: Standard für Koordinationsmuster

- Komplexe Aktivitäten beinhalten Abhängigkeiten zwischen den beteiligten Subaktivitäten
- Service-übergreifende Informationen werden zur Steuerung benötigt  
Aus Service-Sicht sind dies **Kontextinformationen**
- Ein Koordinator verwaltet Kontextinformationen und steuert die komplexe Aktivität
- WS-Coordination ist ein Framework, dass dieses Modell (coordinator service model) unterstützt.

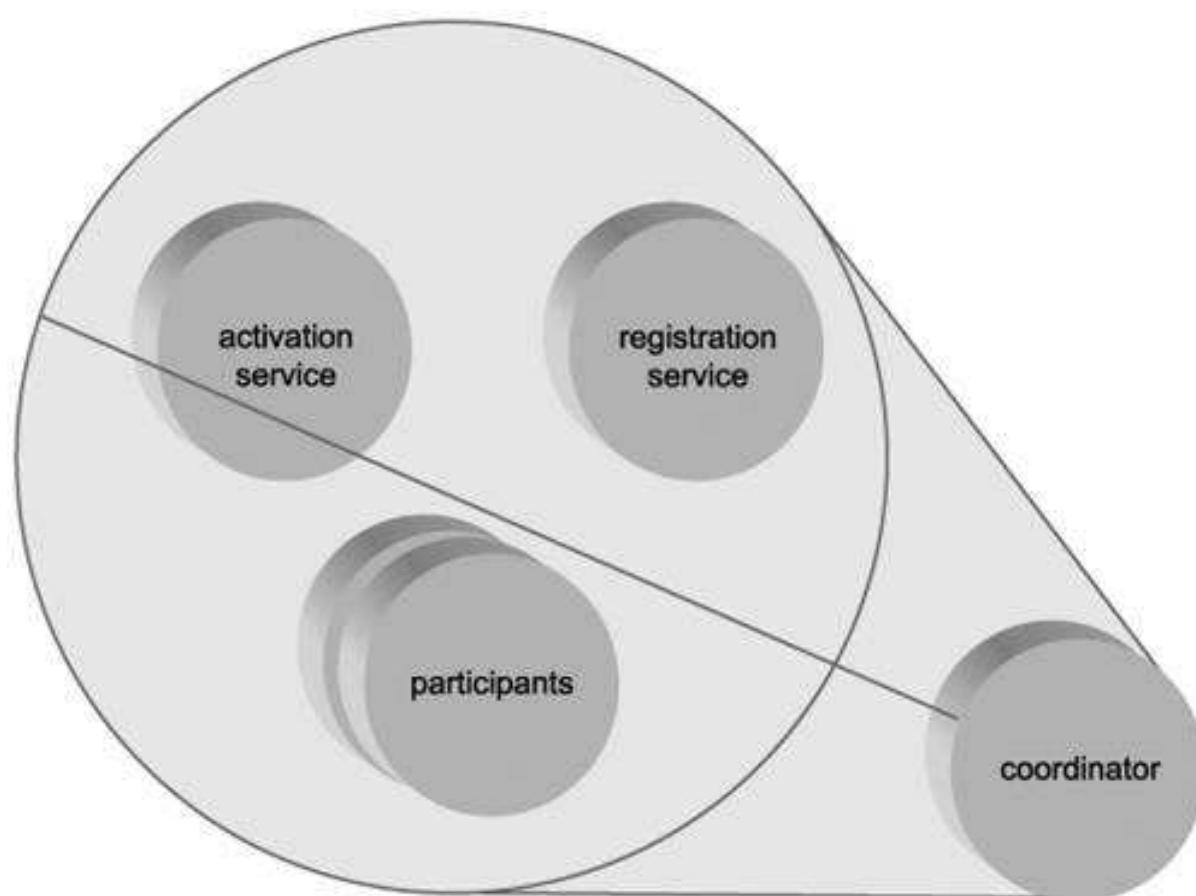
# Koordination



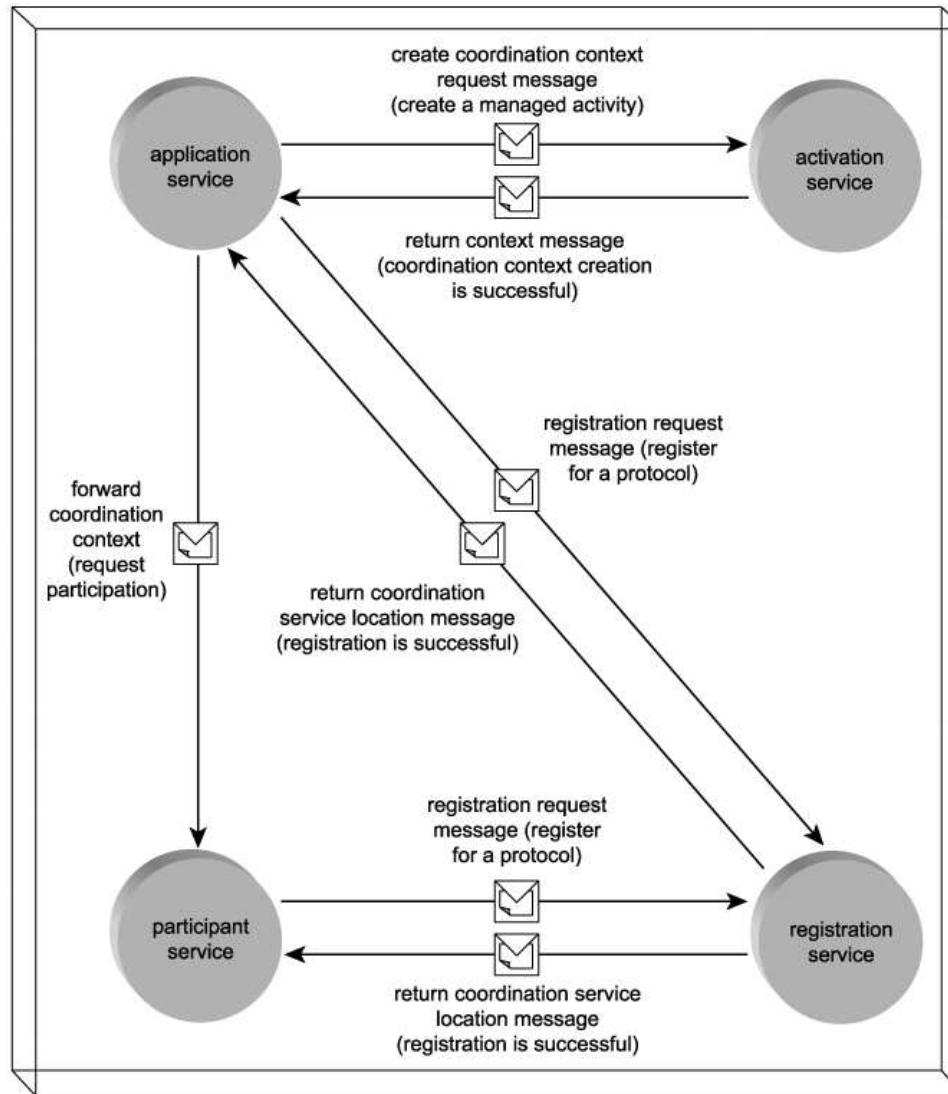
## **Coordinator Service Model**

- Koordinationsdienst
  - kontrolliert Komposition
- Aktivierungsdienst
  - erzeugt und verwaltet Aktivitätskontext (Aktivitäts-ID, Koordinationstyp u.a.)
  - Koordinationstyp = fest definiertes Protokoll für Kooperation  
z.B. Transaktion, Geschäftsaktivität
- Registrierungsdienst
  - registriert Teilnehmer, macht Koordinationsdienst und Teilnehmer „miteinander bekannt“
- Protokoll-spezifische Dienste (Anwendungsdienste)
  - initiieren das Zusammenwirken
  - können andere Anwendungsdienste zum Mitmachen einladen

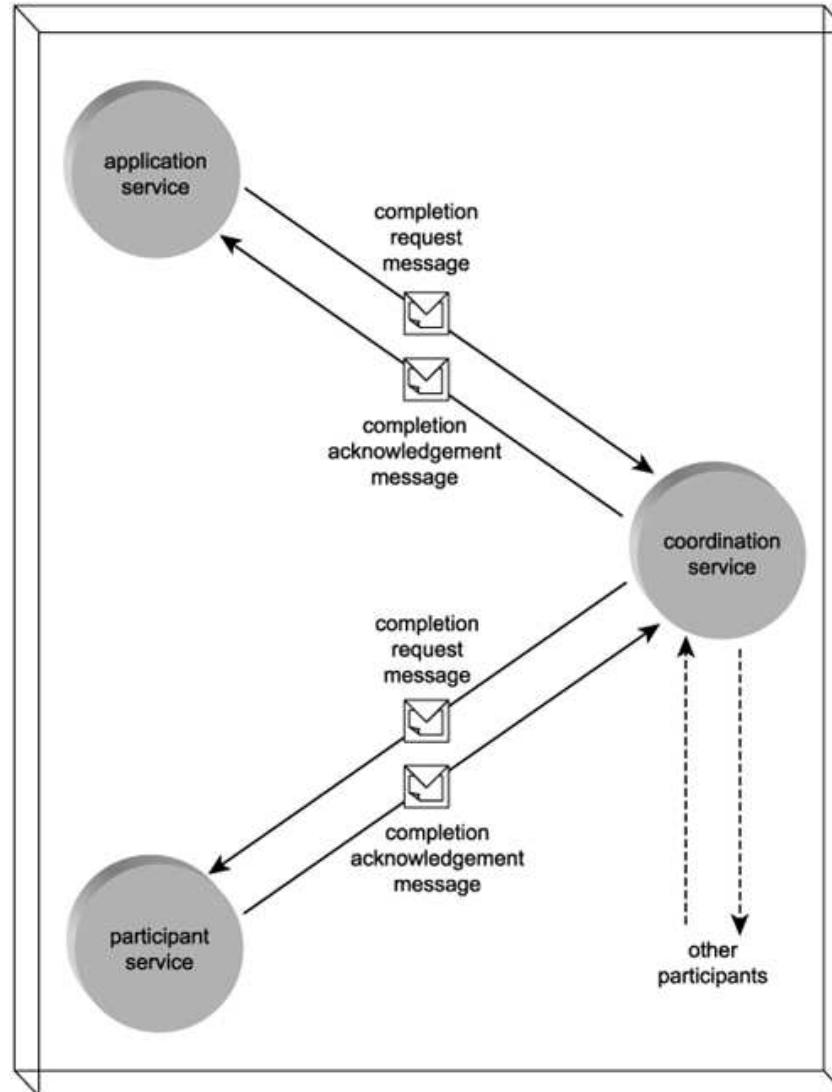
## Coordinator Service Model - Teilnehmer



# Coordinator Service Model - Initialisierung



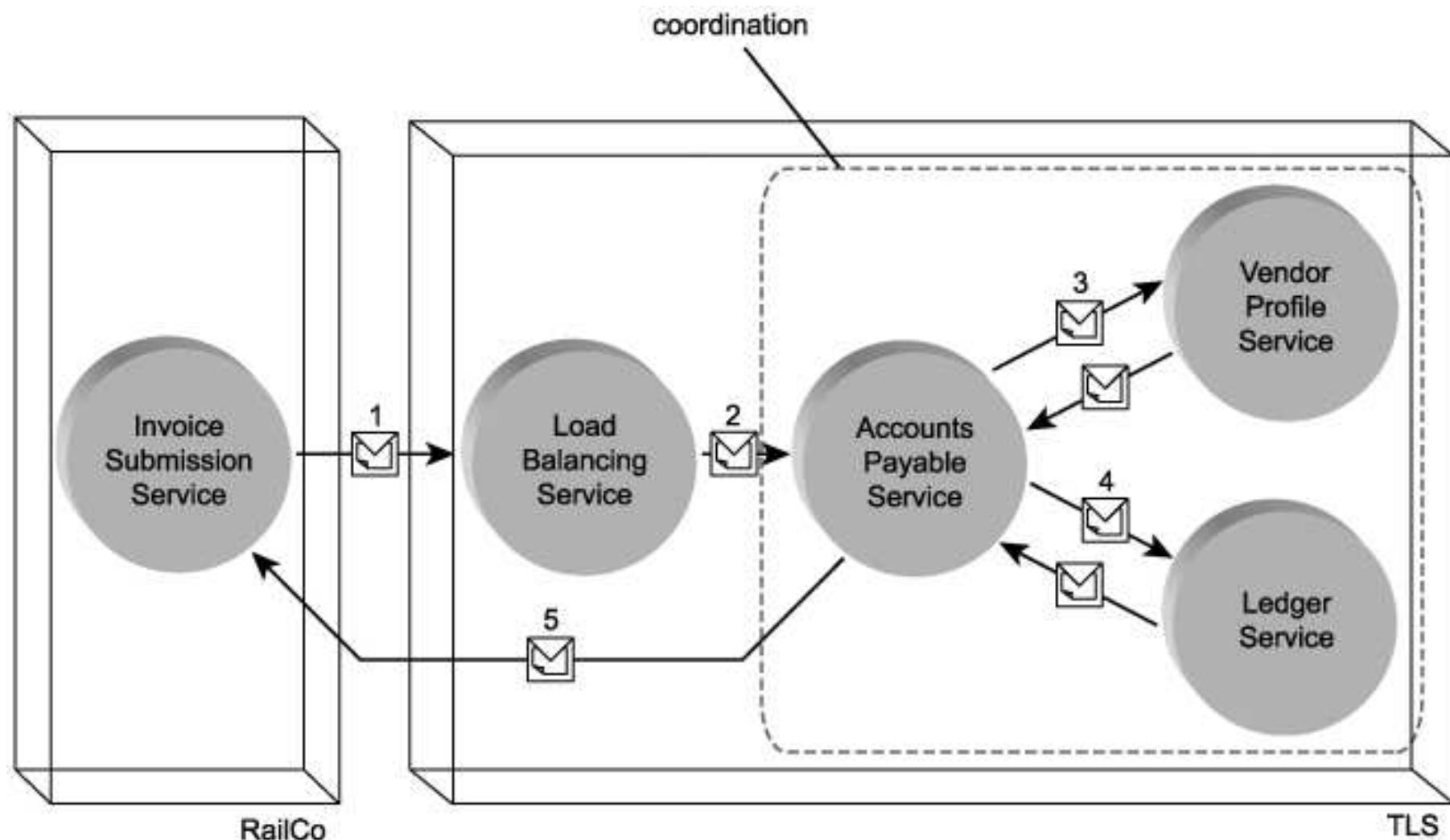
## Coordinator Service Model - Beenden der Aktivität



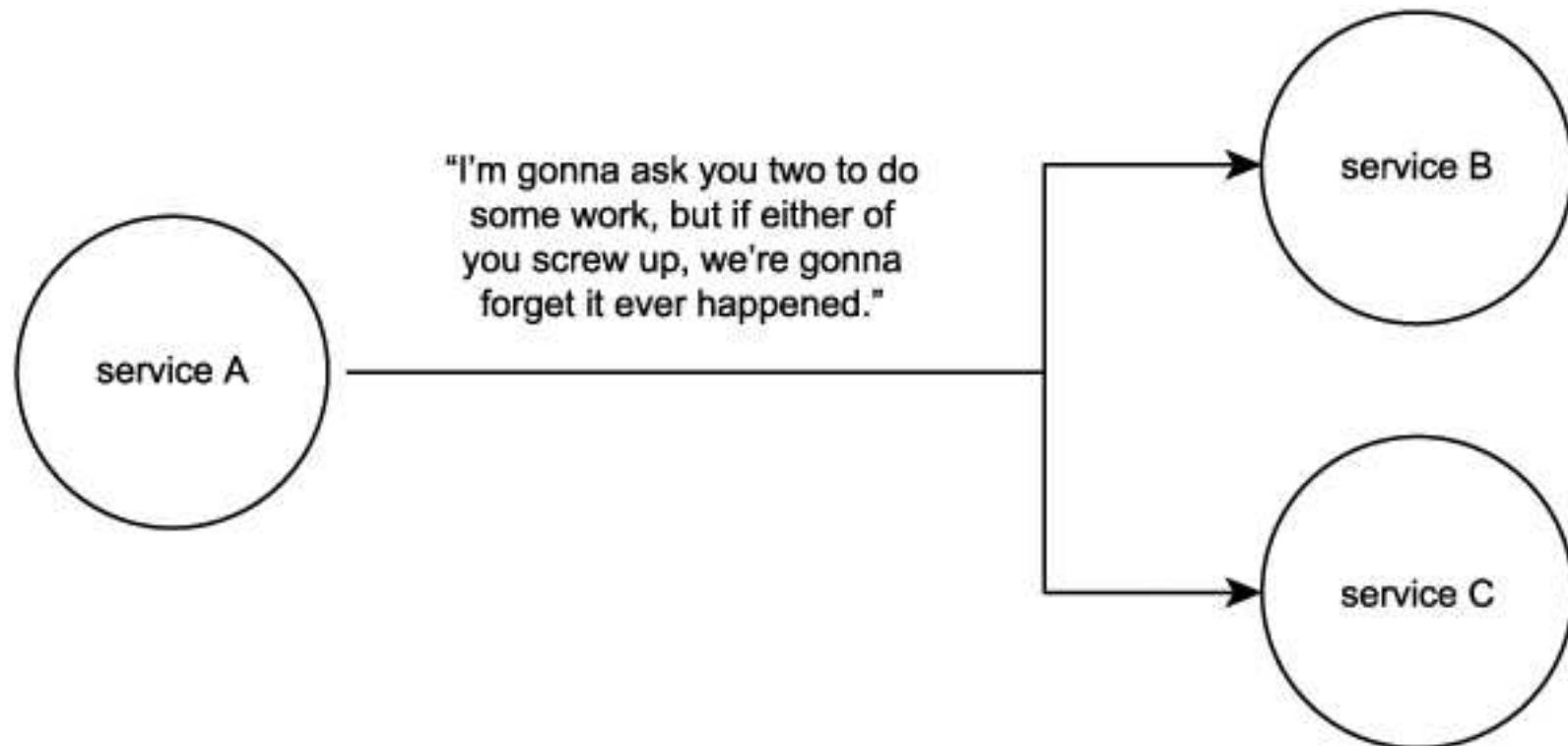
## Ziel des Frameworks

- Ansatz zur Verwaltung von Kontextinformation
- für verschiedene Kooperationsmuster verwendbar, allgemeiner als etwa reiner Transaktions-Koordinator-Dienst
- Zustandslosigkeit der Anwendungsdienste durch Kontextmanagement

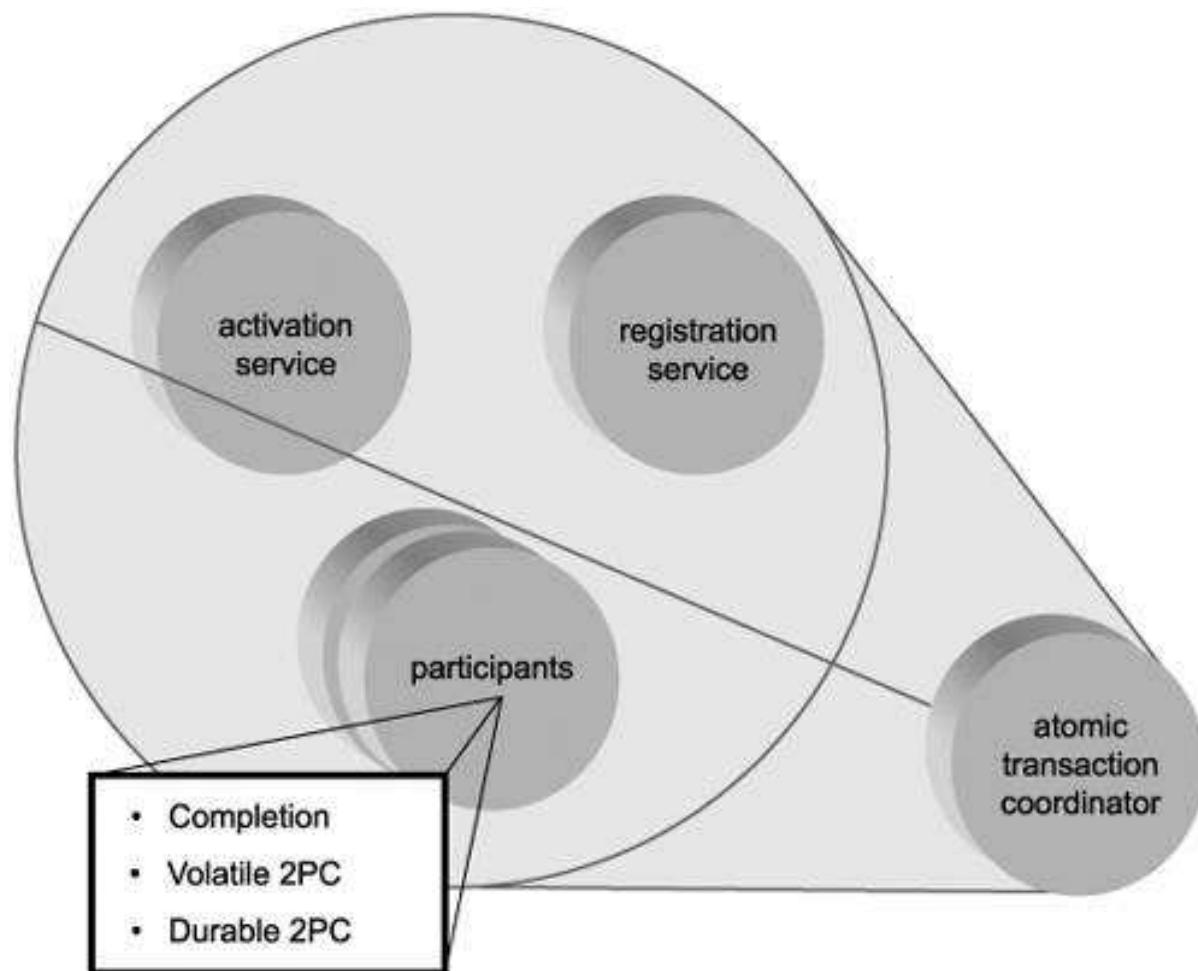
## Koordination im Fallbeispiel, Koordinationstyp: Transaktion



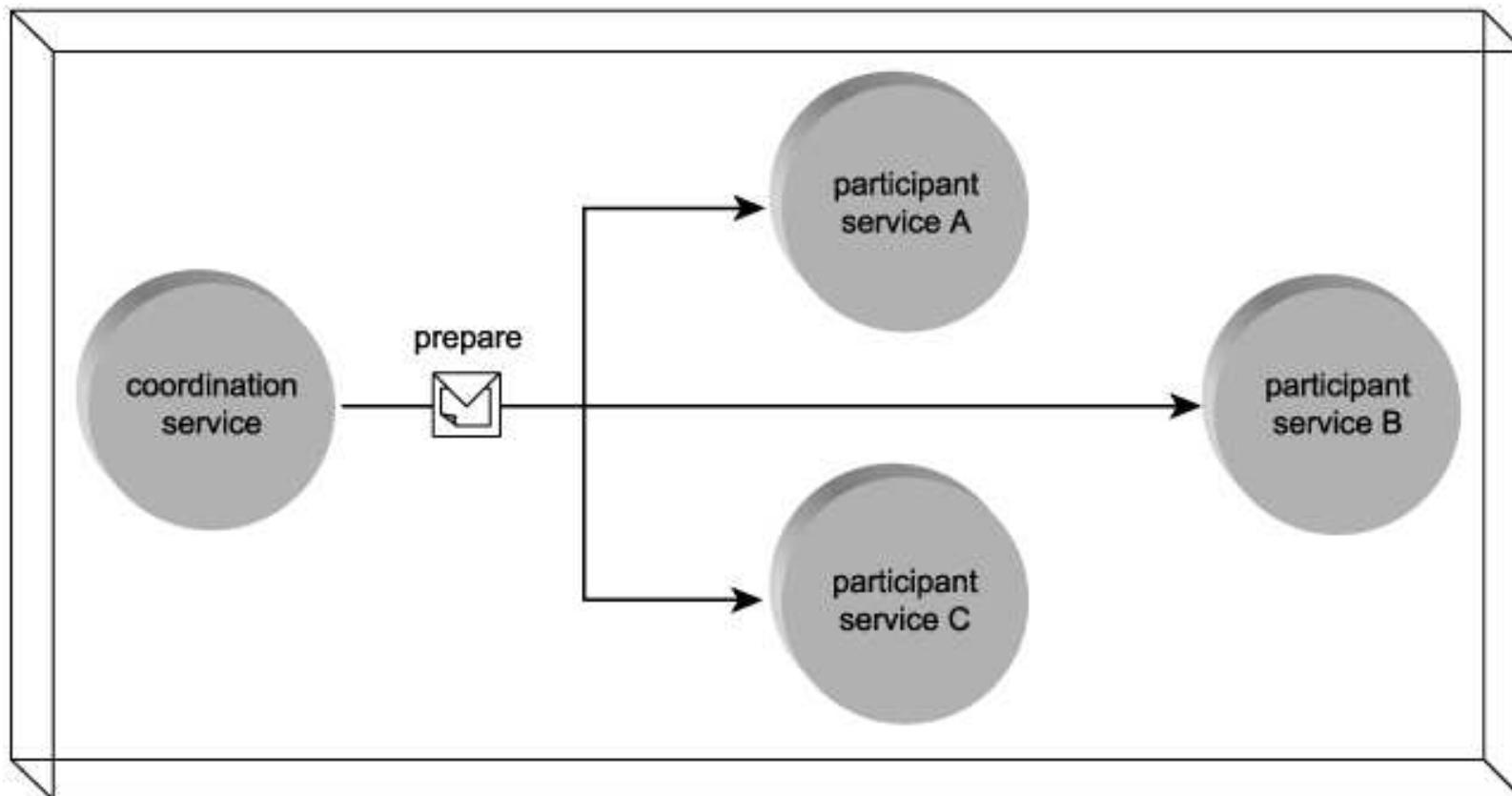
## Koordinationstyp: Transaktion



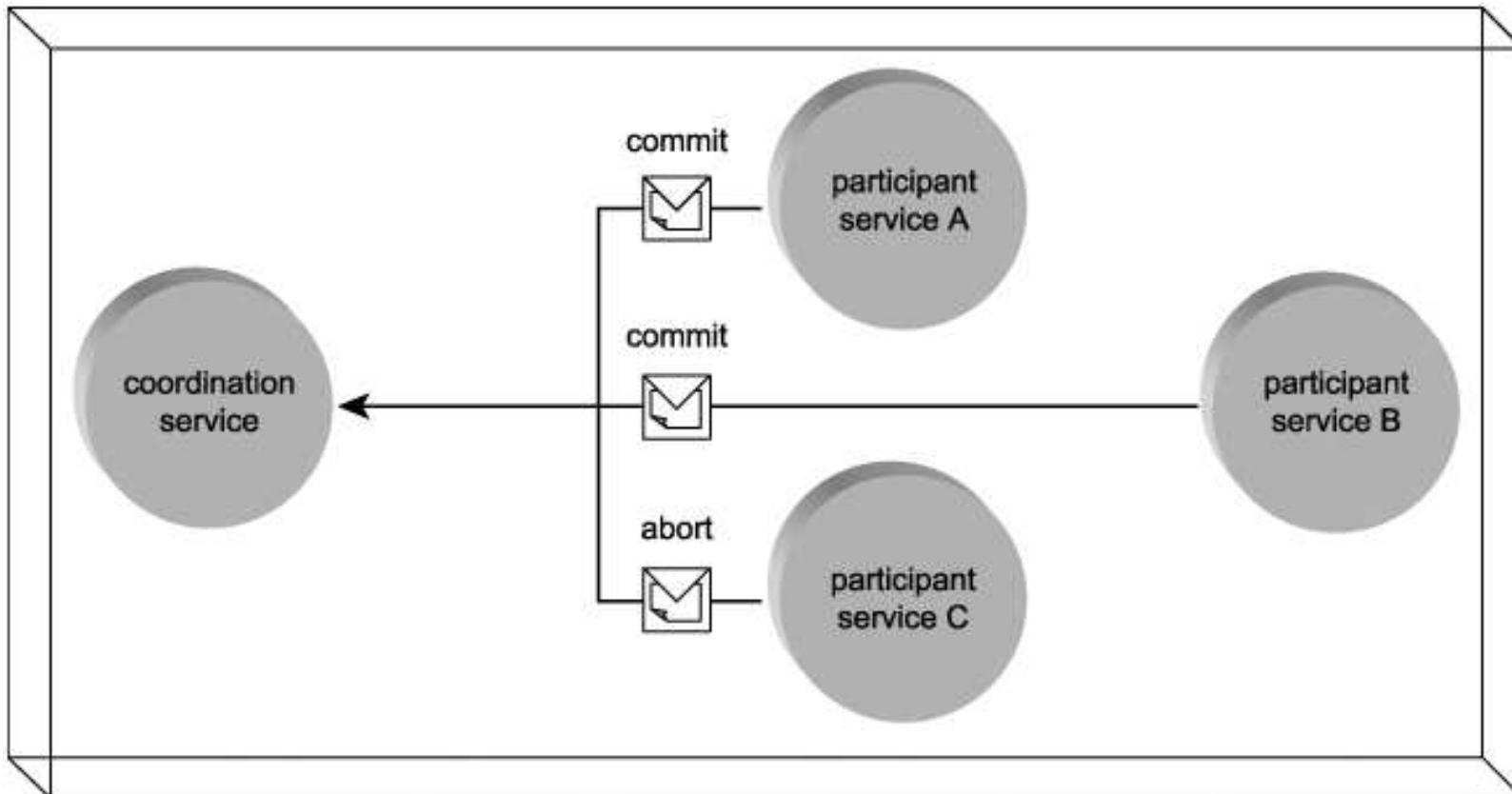
## Koordinationstyp: Transaktion



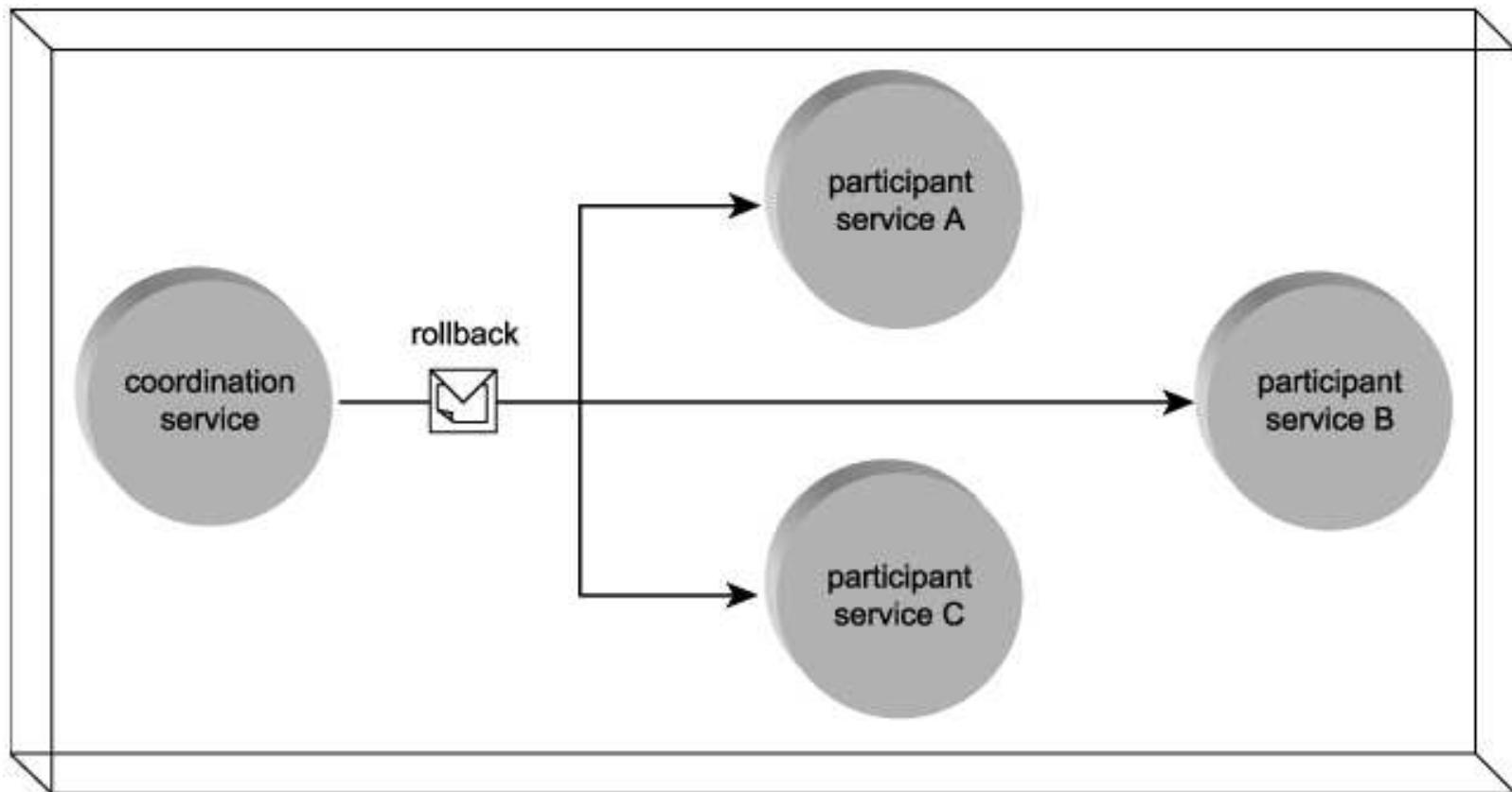
## Koordinationstyp: Transaktion



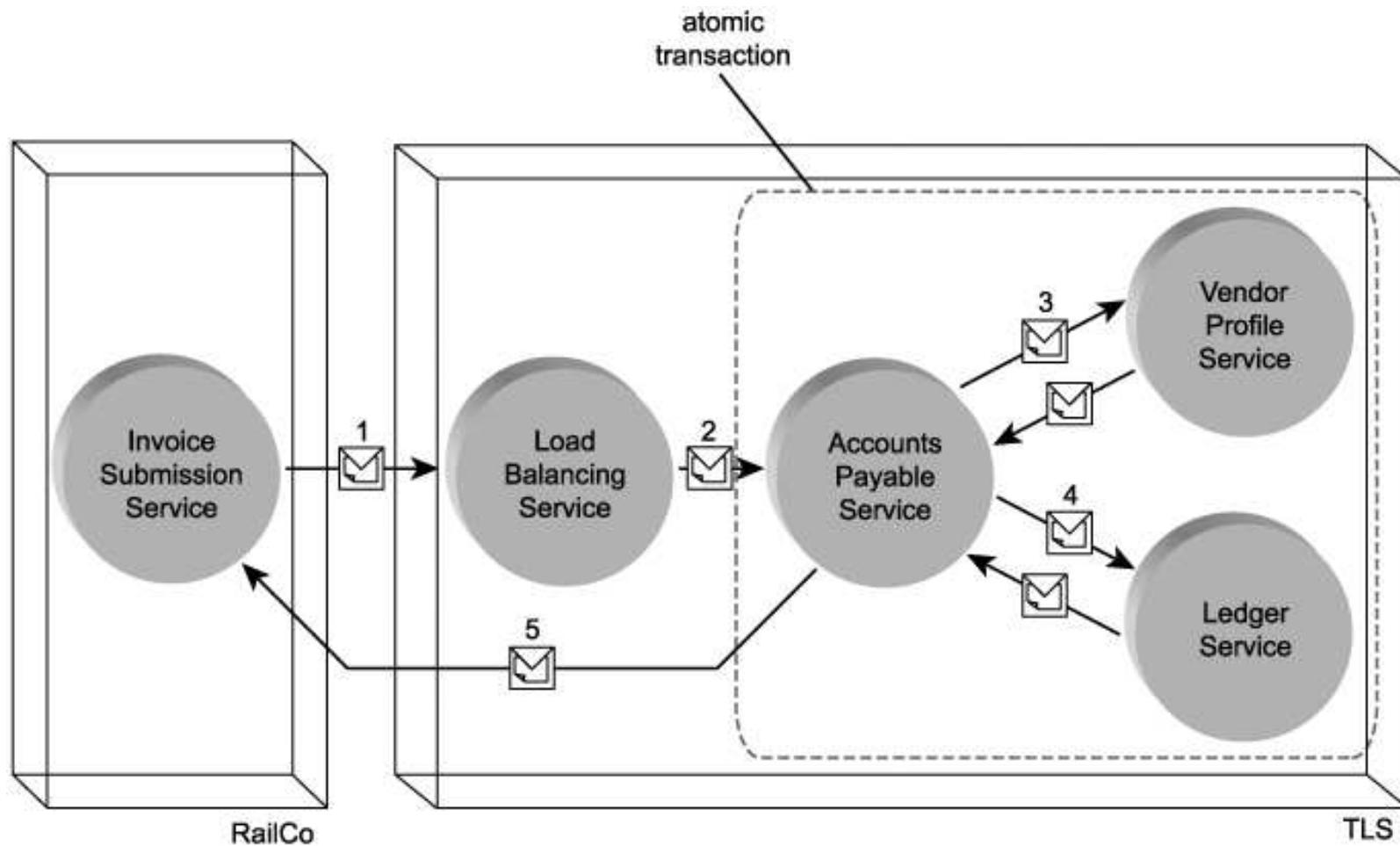
## Koordinationstyp: Transaktion



## Koordinationstyp: Transaktion



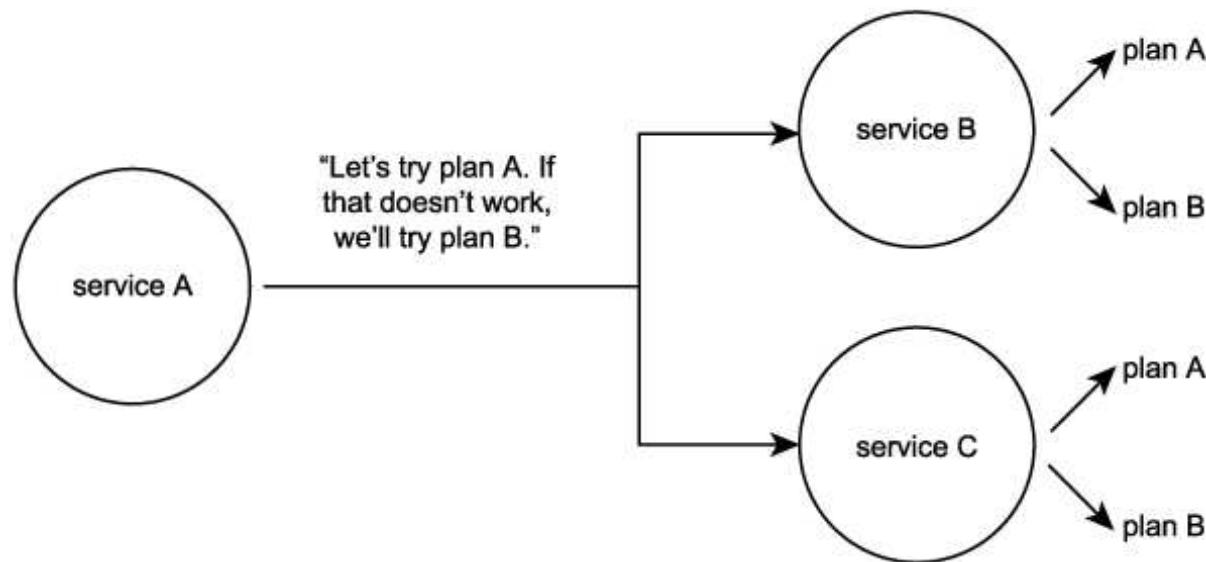
## Koordination im Fallbeispiel, Koordinationstyp: Transaktion



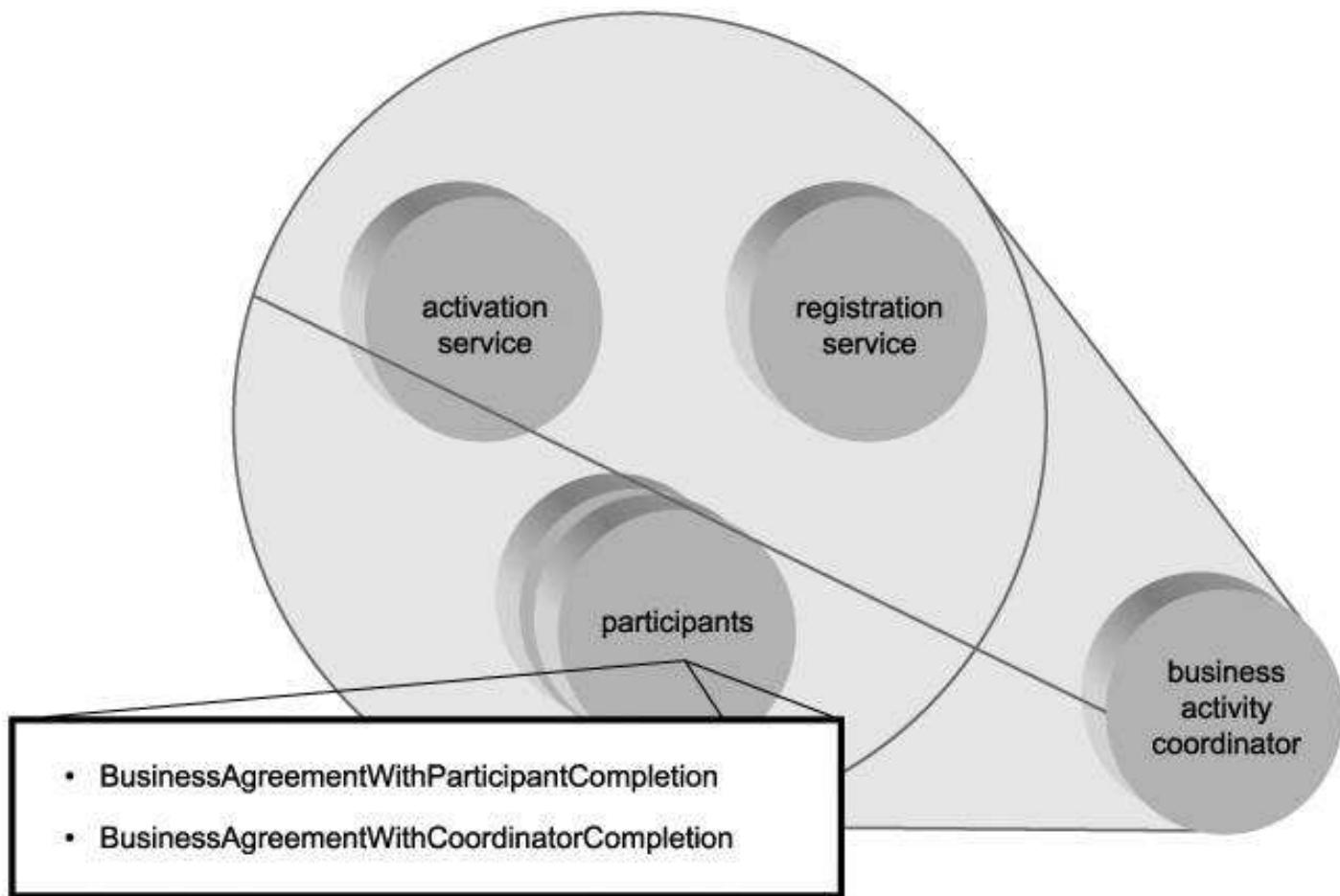
## Koordinationstyp: Geschäftsaktivität

WS-BusinessActivity definiert ein Kooperationsmuster

- komplexe, ggf. lange andauernde Aktivitäten
- Transaktionskonzept wegen Dauer nicht adäquat
- Stattdessen: „Plan B“, falls ein Problem auftritt



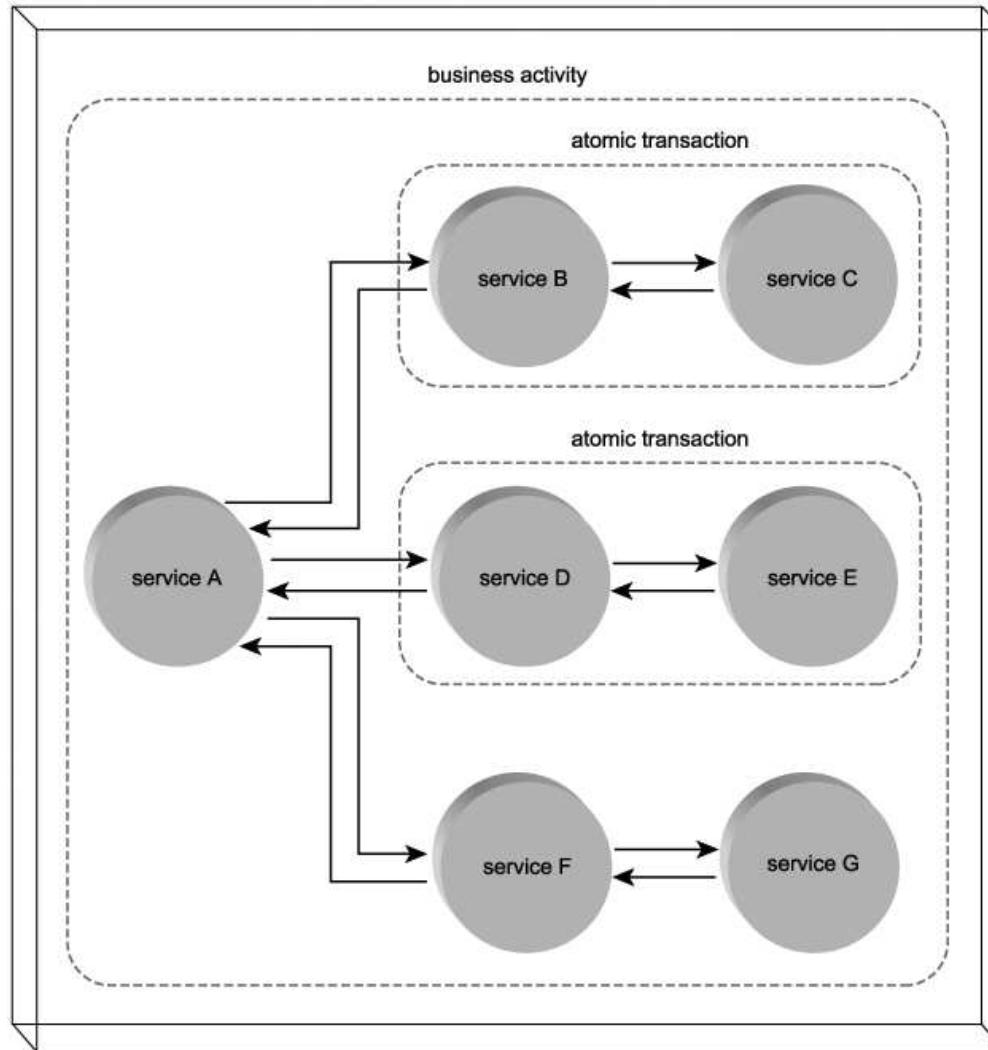
# Geschäftsaktivität



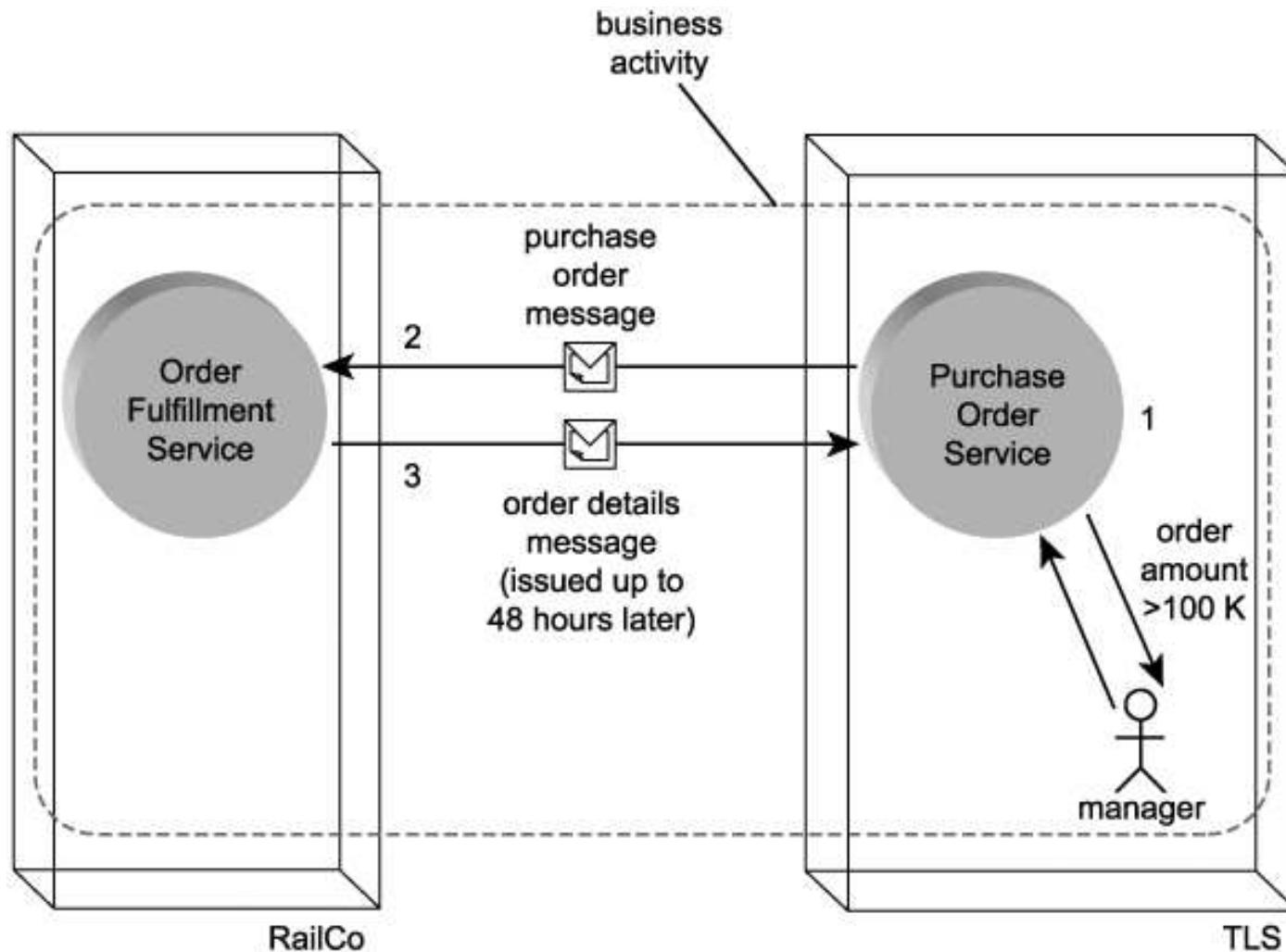
## Geschäftsaktivität

- Zustände der Teilnehmer werden durch Koordinator verwaltet:
  - active state
  - completed state
  - compensation state
  - cancelled state
  - exit state
- Teilnehmer können der Komposition beitreten und diese wieder verlassen
- Bei Zustandsänderungen werden andere Teilnehmer über den Koordinator benachrichtigt

## Geschäftsaktivität mit Transaktions-Subaktivitäten



## Geschäftsaktivität im Fallbeispiel



## Orchestrierung und SOA

- Orchestrierung modelliert komplette Geschäftsprozesse
- Standardisierte Spezifikation der Prozesse
- Orchestrierung ist SOA-Kernkonzept:
  - Komposition
  - Wiederverwendbarkeit
  - Erweiterbarkeit
  - Schichtenarchitektur
  - Interoperabilität
  - Anbieterunabhängigkeit
- Orchestrierung selbst kann als Service implementiert werden!
- Standard: WS-BPEL = Business Process Execution Language

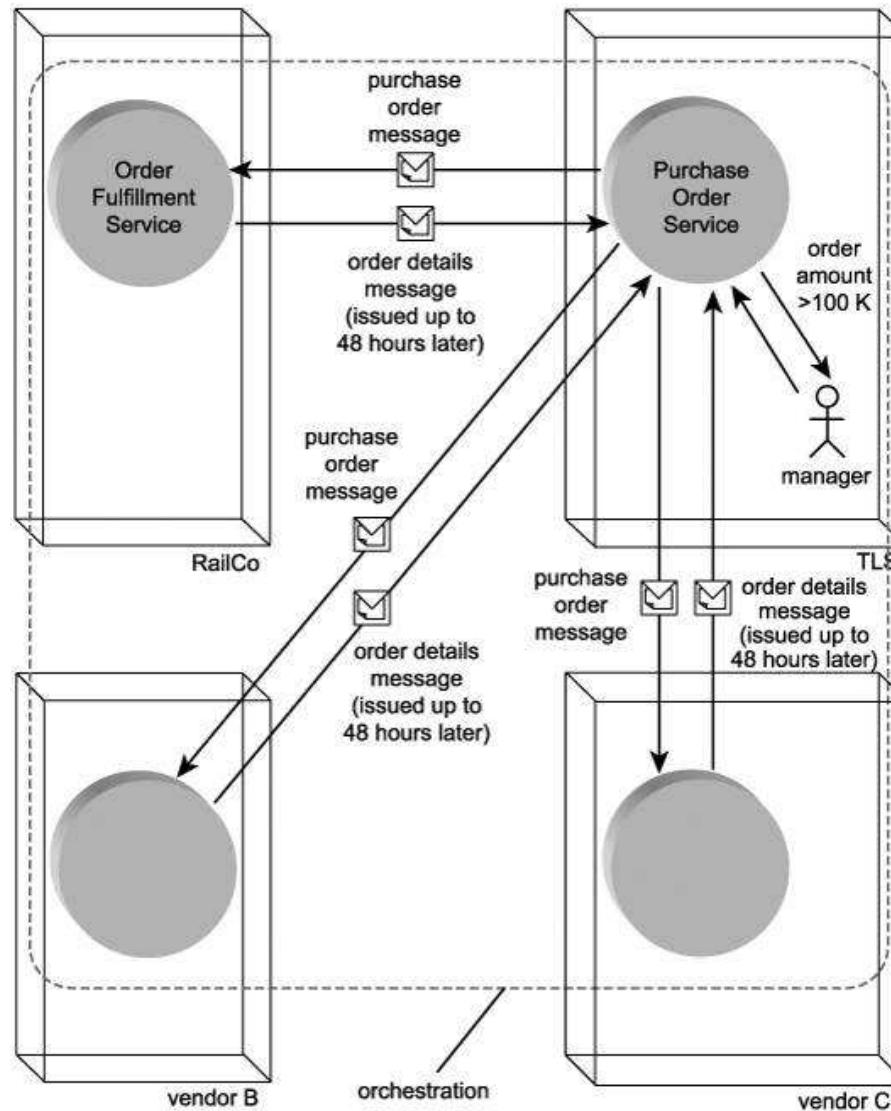
## Orchestrierung im Fallbeispiel

TLS verwendet Orchestrierung um den Geschäftsprozess der Bestellung zu modellieren:

- Auftrag an einen Lieferanten versenden
- Warten auf Bestätigung der Lieferbarkeit
- Falls nicht lieferbar, Auftrag an alternativen Lieferanten

Orchestrierung ist wesentlich für die Modellierung des Zusammenhangs zwischen den Aufträgen!

# Orchestrierung im Fallbeispiel



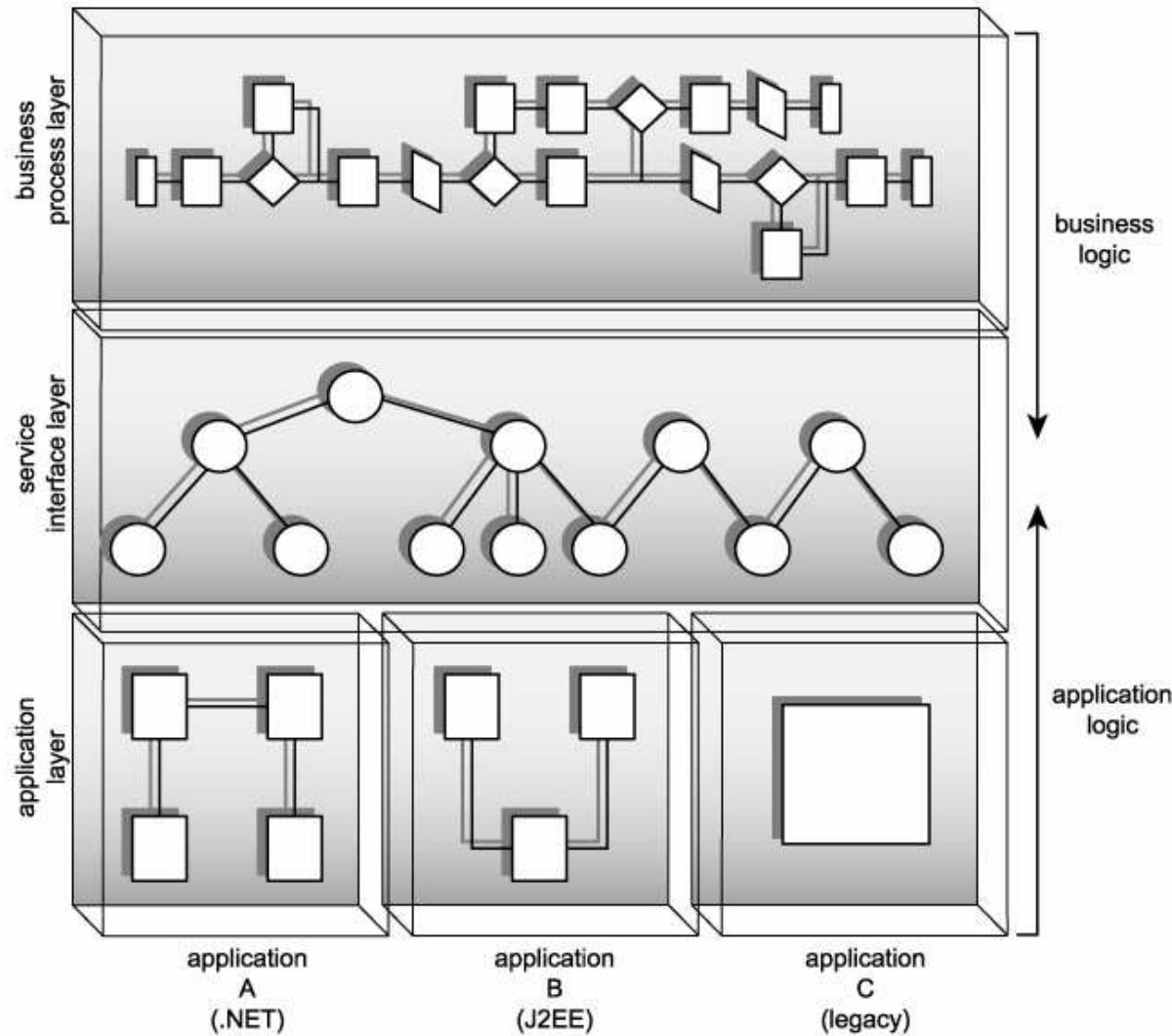
## Choreographie

- Konzept für Unternehmens-übergreifende Kollaboration
- Keine zentralisierte, sondern verteilte Kontrolle
- Definition von (1:1-)Relationen und Kommunikationskanälen
- Interaktionen gemäß formal spezifizierten Regeln und Einschränkungen („work units“)
- Module

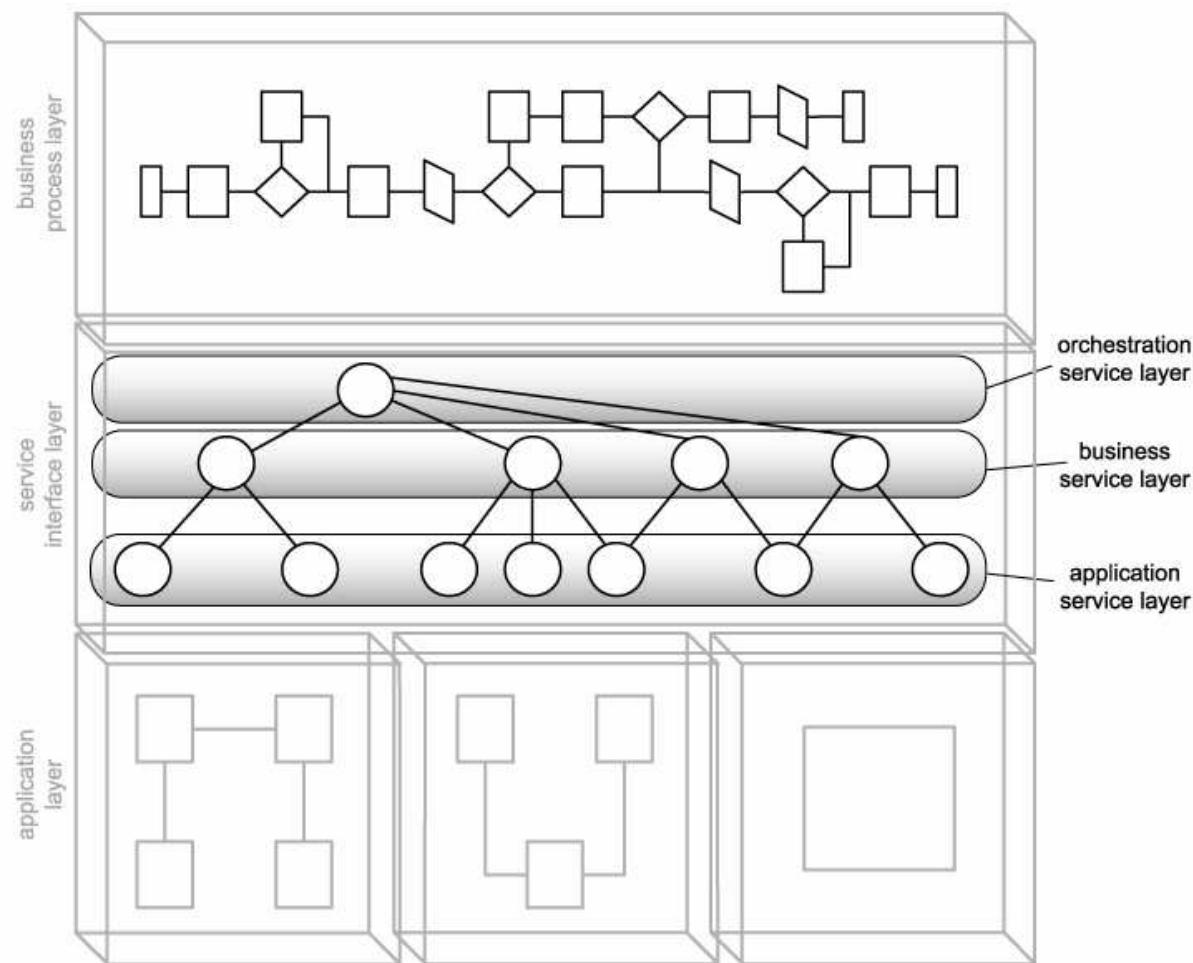
## Choreographie im Fallbeispiel

- TLS hat den Metallbauer Sampson Steel übernommen
- Ein bedeutender Kunde benötigt spezifische, technisch komplexe Stahlprodukte
- Maschinenbau-Ingenieure des Kunden müssen mit den Stahlspezialisten von TLS über längere Zeit in der Produktspezifikation zusammenarbeiten
- In dieser Zeit müssen viele Entwürfe technischer Spezifikationen ausgetauscht werden.
- TLS und der Kunde überbrücken ihre IT-Systeme mit einer Service-Choreographie, die dem Dokumentenaustausch dient.

## Dienste als Schnittstellen-Schicht in der SOA



# SOA-Schichtenarchitektur



## Dienste der Applikationsschicht

- Schnittstelle zu Applikationen
- möglichst Geschäftsprozess-unabhängige, wiederverwendbare Logik
- dienen als Integrationsmechanismus
- plattformabhängig
- Hilfsdienste (utility services)
- Umschlagdienste (wrapper services)

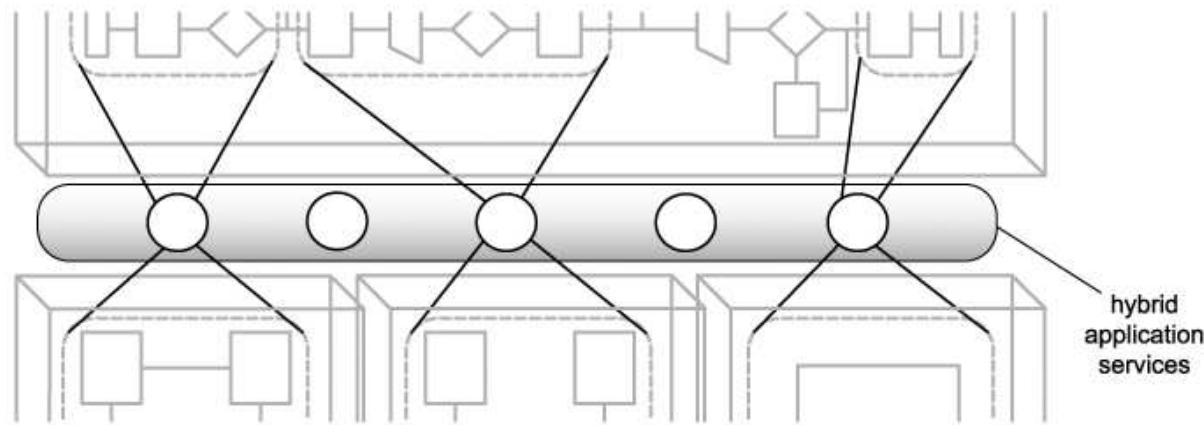
## Dienste der Geschäftsschicht

- Applikations-unabhängige Repräsentation von Geschäftslogik
- Dienste agieren idealerweise als Controller für andere Dienste
  - Anwendungsdienste
  - andere Geschäftsschicht-Dienste
- Dienste dieser Schicht modellieren
  - Abläufe innerhalb von Geschäftsprozessen (prozessorientiert) oder
  - Entitäten (entitätsorientiert)
- *Hybriddienste* realisieren Geschäftslogik **anwendungsorientiert**
  - softwaretechnisch nicht wünschenswert
  - in der Praxis häufig

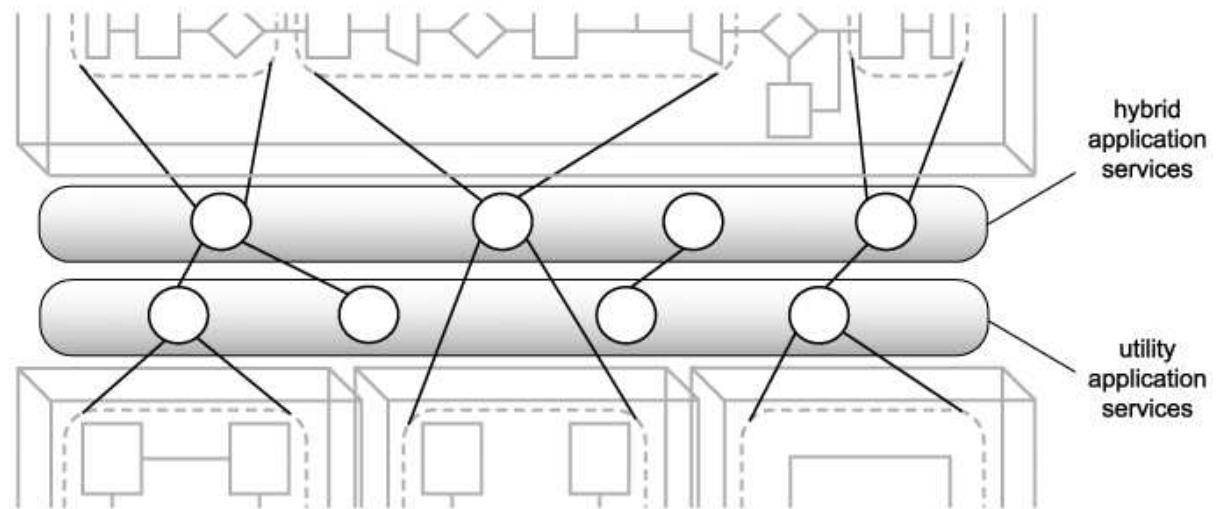
## Orchestrierungsschicht

- Separate Schicht zur Geschäftsprozesssteuerung
- Workflow-Konzept zur Spezifikation der Interaktion zwischen Diensten der Geschäftsschicht
- Ziele:
  - Verbesserte Wiederverwendbarkeit der Geschäftsschicht-Dienste
  - Einfachere Änderbarkeit der Geschäftsprozesse

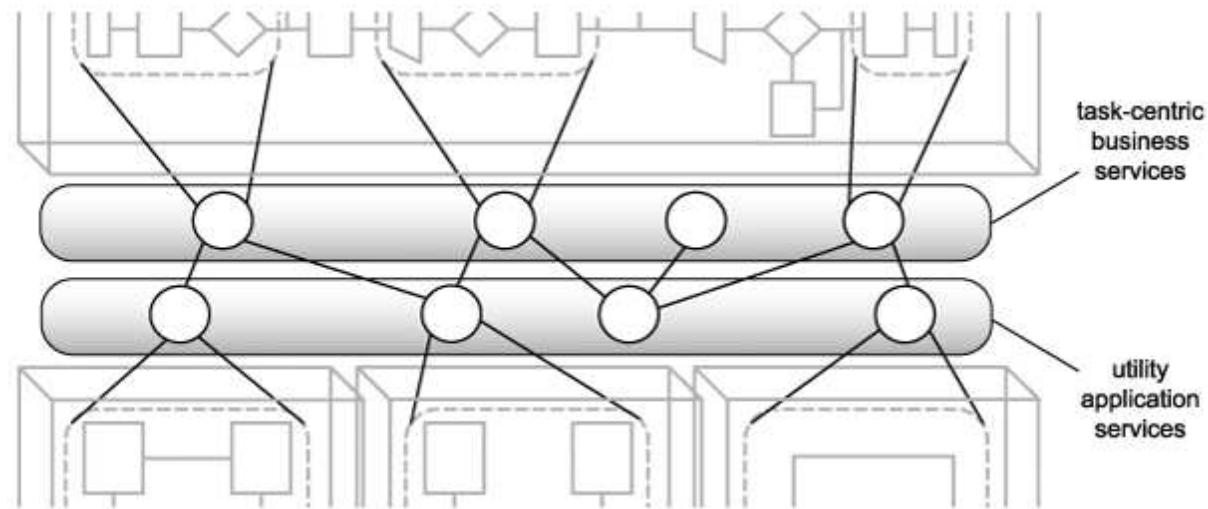
## Varianten serviceorientierter Architekturen: SOA-Szenario 1



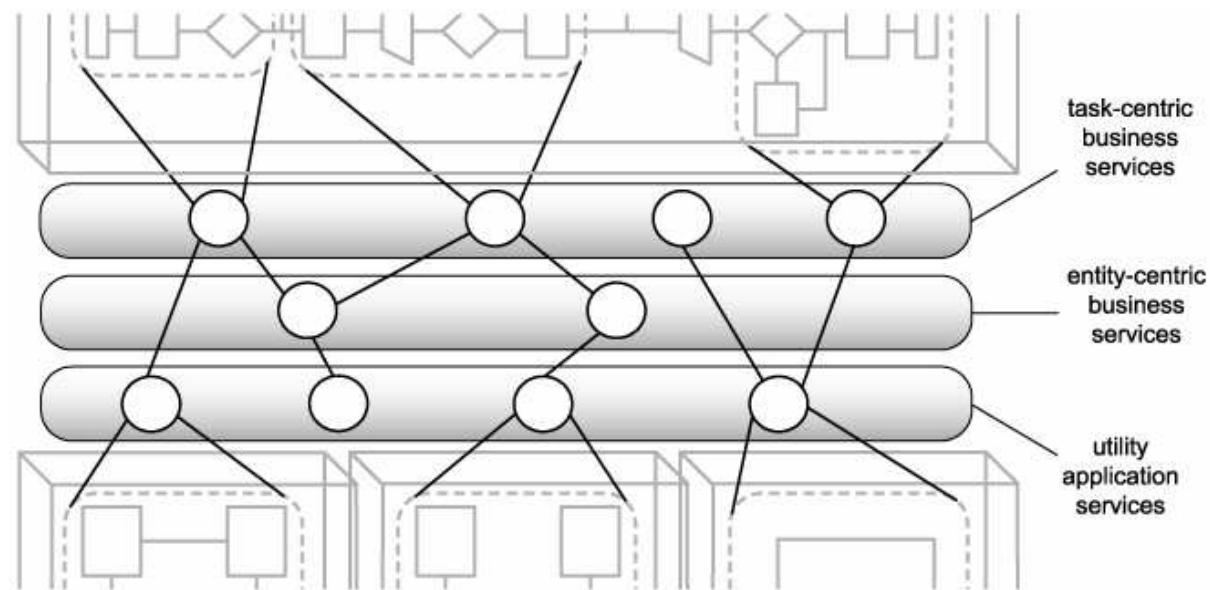
## SOA-Szenario 2



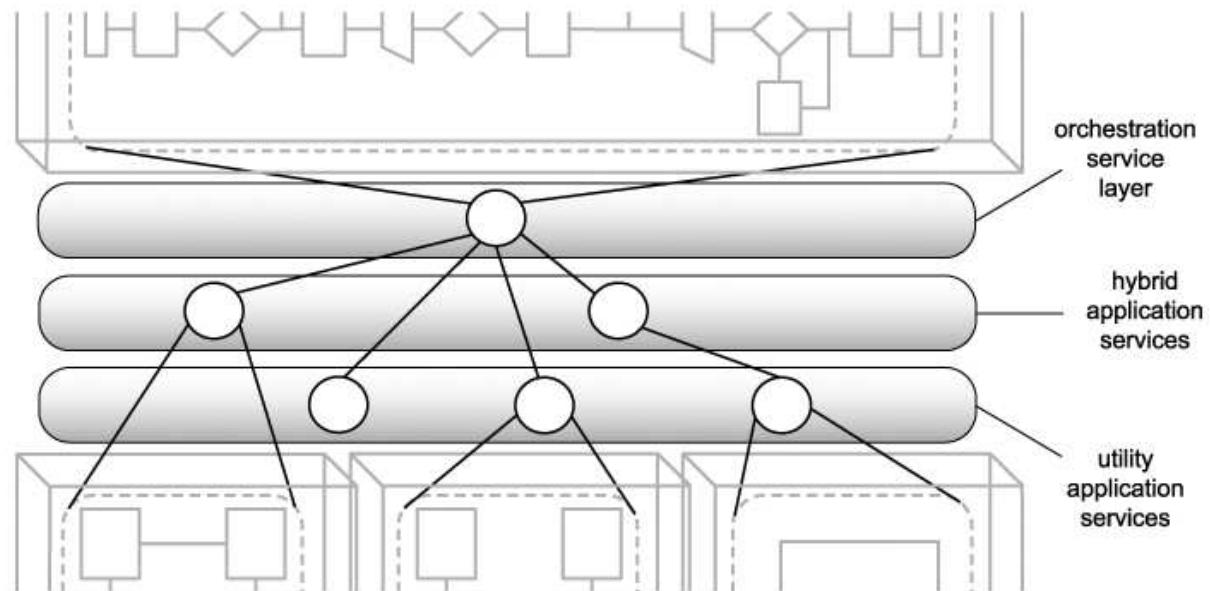
## SOA-Szenario 3



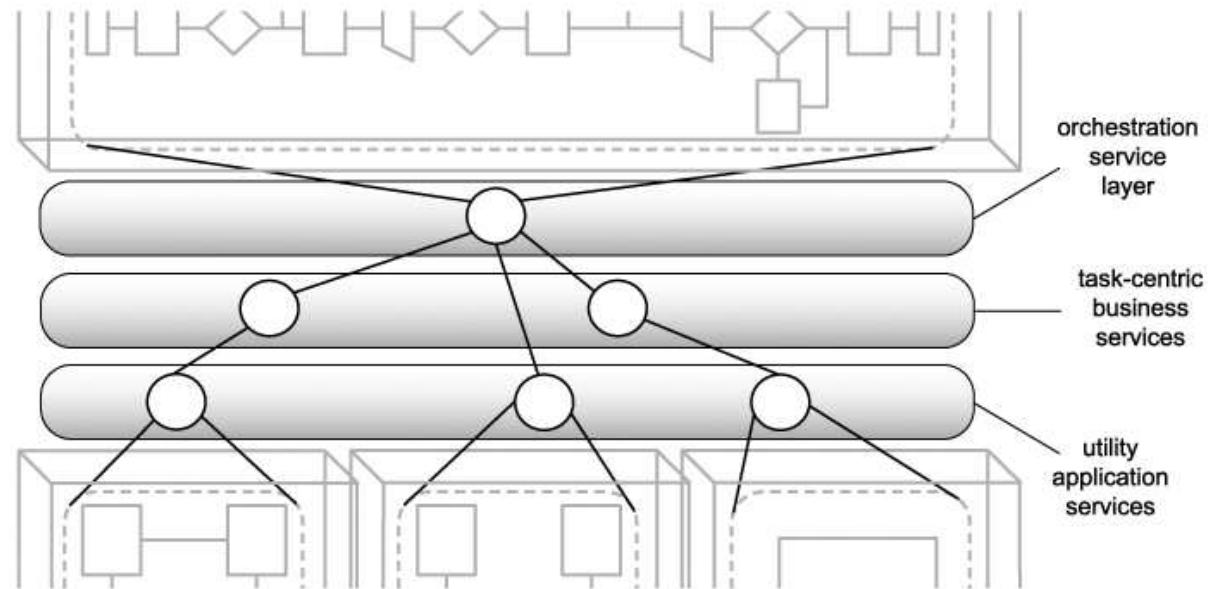
## SOA-Szenario 4



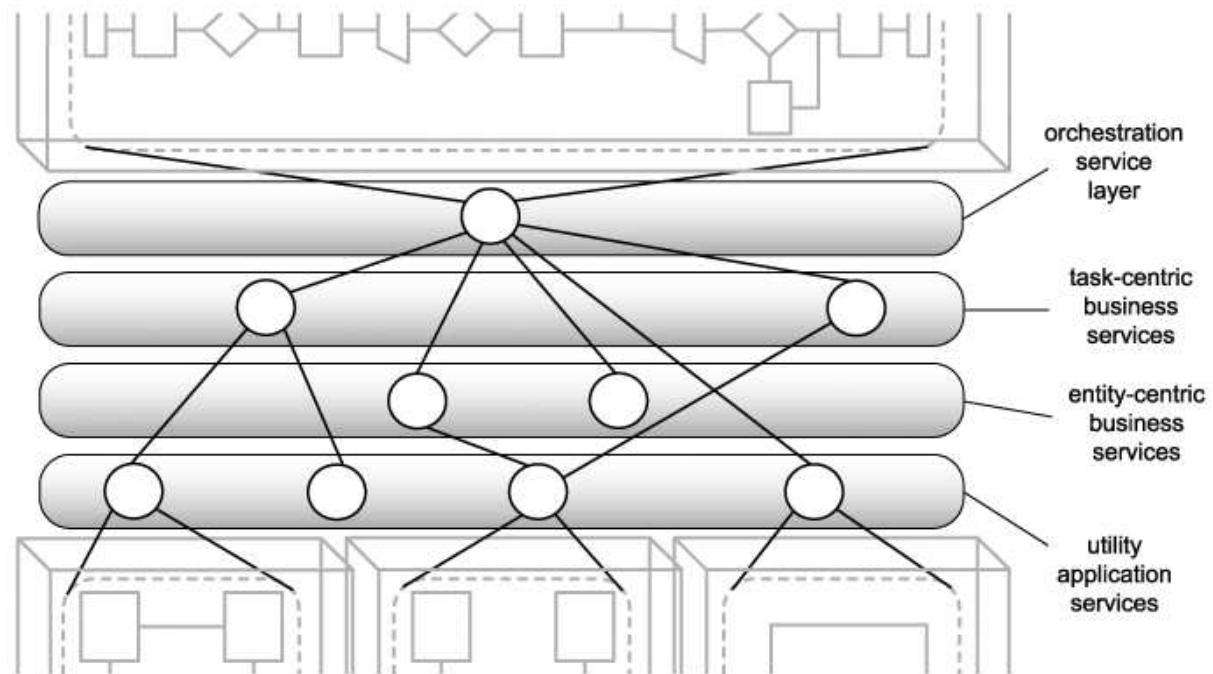
## SOA-Szenario 5



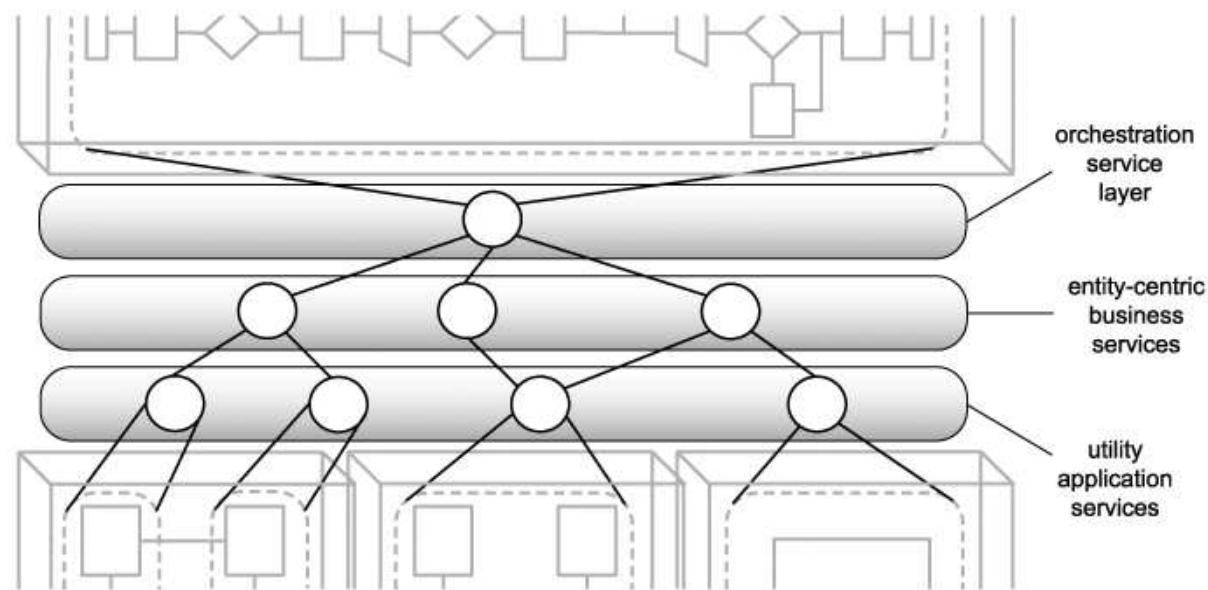
## SOA-Szenario 6



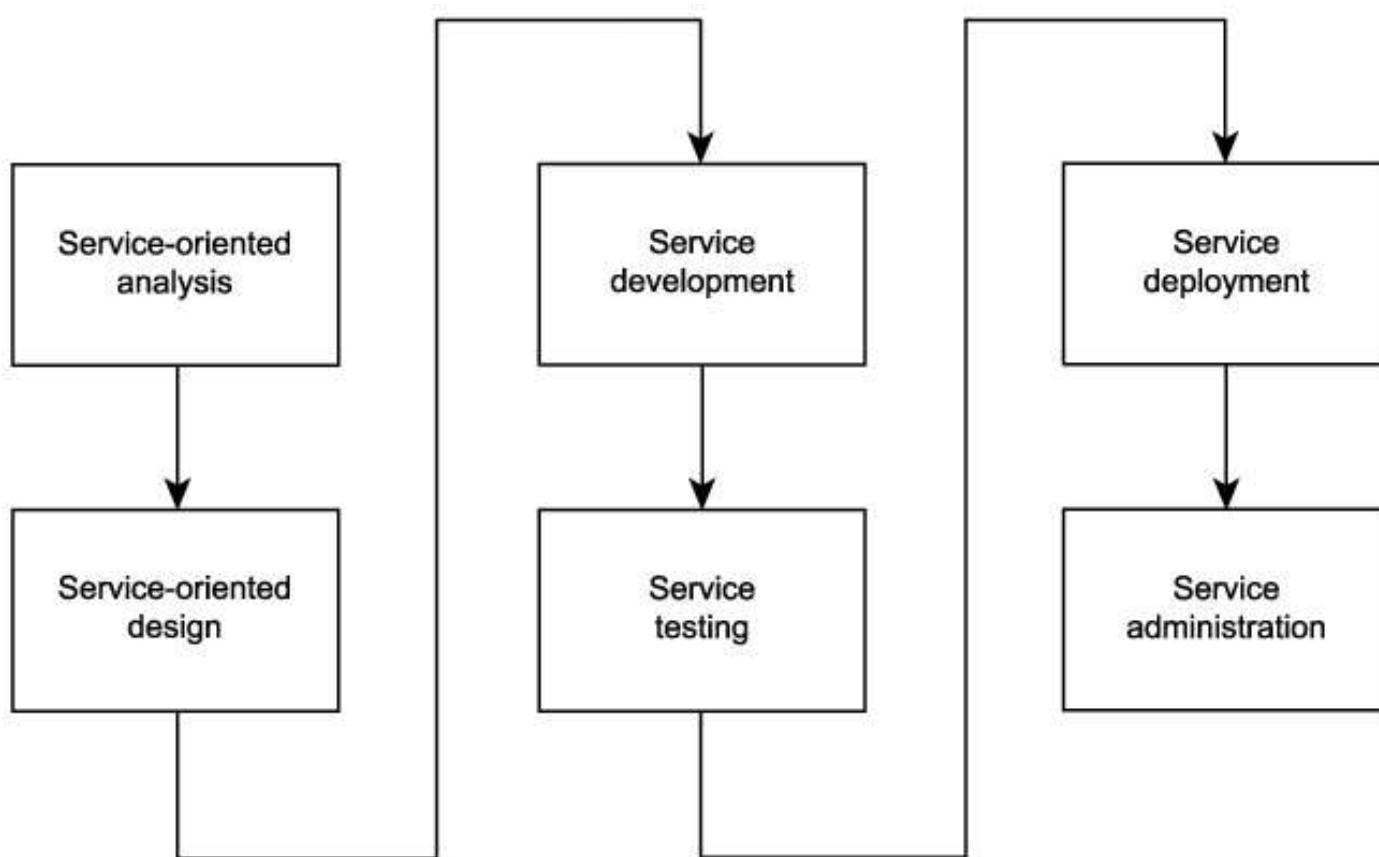
## SOA-Szenario 7



## SOA-Szenario 8

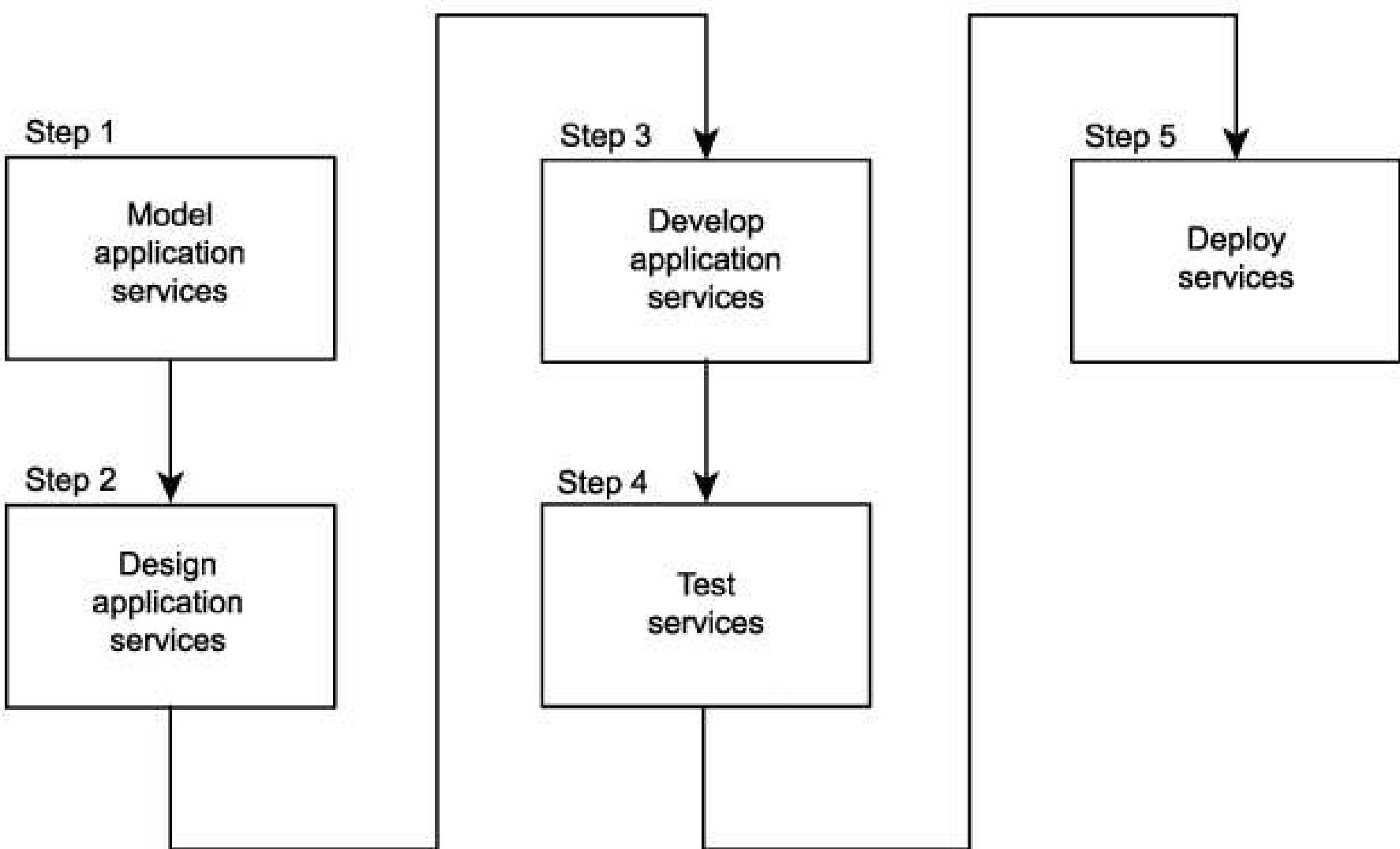


## „Top Down“-SOA-Entwicklung



Problem: sehr hoher Aufwand

## Bottom-Up-Strategie



## Bottom-Up-Strategie

- **Typische Ausgangssituationen:**

- Mehrere Anwendungen müssen integriert werden.  
*Mit welchen Services lässt sich die Integration durchführen?*
- Vorhandene Middleware-Systeme sollen durch Webservices ersetzt werden.  
*Wie können vorhandene Komponenten in serviceorientierte transformiert oder durch serviceorientierte ersetzt werden?*

- **Ergebnis:** 2-schichtige SOA

- Applikationsschicht mit Hilfsdiensten
- Hybriddienste für die Geschäftsprozesse

- **Bewertung**

- Meistbenutzter Ansatz
- Führt nicht zu besonders guter Architektur
- meist keine Strategie, sondern eher „Fehlen jeglicher Strategie“

## Vorgehensweise zur SOA-Entwicklung im Fallbeispiel: TLS

- TLS benutzte Top-Down-Ansatz, um SOA einzuführen
- Ergebnis:
  - Architekturkonzept erfüllt bislang die Erwartungen
  - Aufwand der Top-Down-Entwicklung war höher als erwartet
- TLS benutzt Top-Down-Ansatz aus Kostengründen *nicht mehr* für die Weiterentwicklung der SOA

## Vorgehensweise zur SOA-Entwicklung im Fallbeispiel: RailCO

- RailCO benutzt Bottom-Up-Strategie (ohne dies zu wissen):
  - TLS definiert Service-Schnittstellen für E-Business
  - RailCO entwickelt Dienste, um den Anforderungen des Kunden zu genügen
- Ergebnis:
  - kostengünstige, schnelle Entwicklung
  - sehr geringes Wiederverwendungspotenzial
  - schlechte Adaptierbarkeit für den Fall geänderter Rahmenbedingungen

Beispiel:

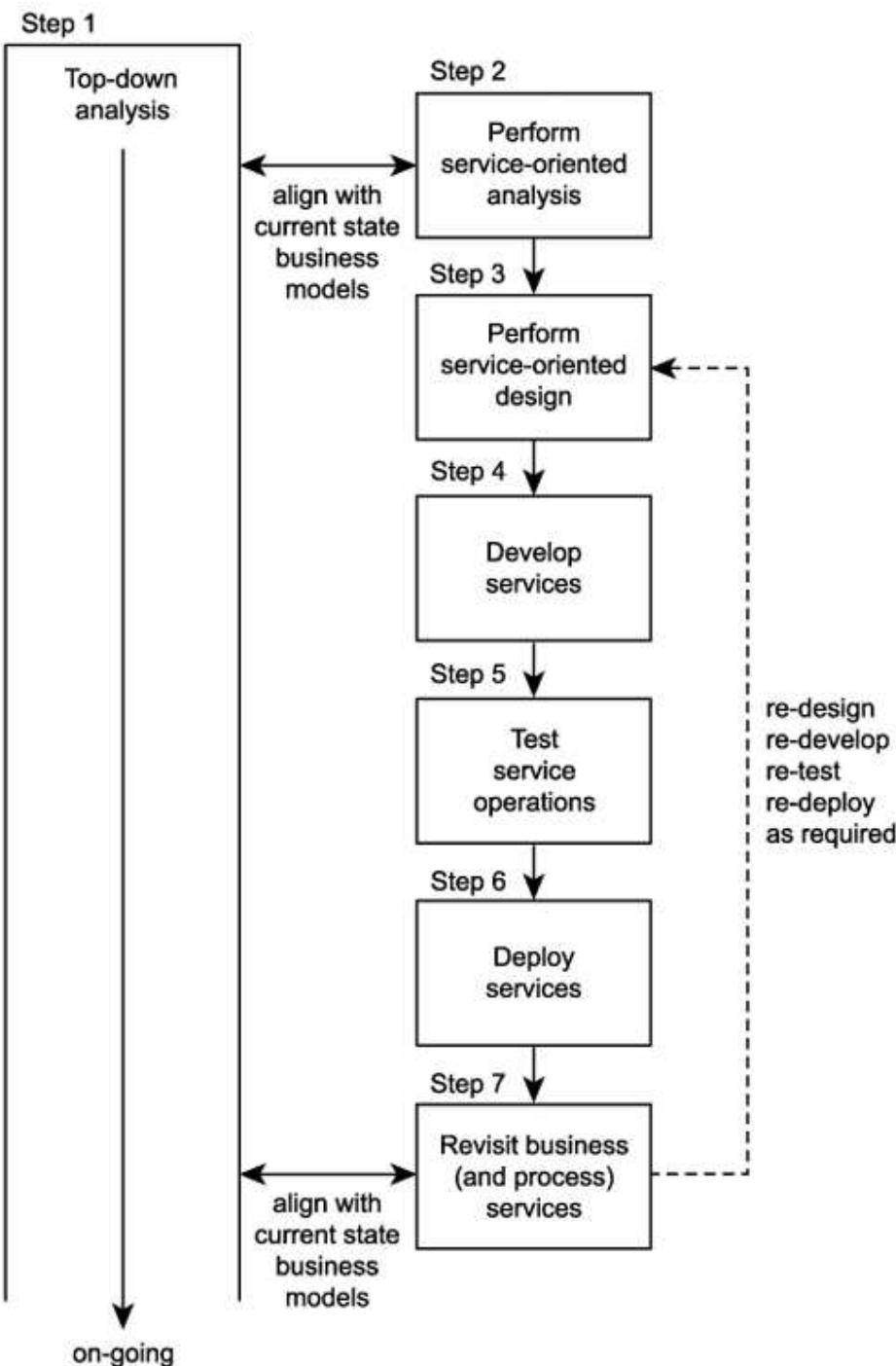
Kleine technische Änderung bezüglich Rechnungseinreichung durch TLS, Geschäftsprozess ist nicht betroffen.

Wegen fehlende Schichtentrennung bei RailCO dort Änderungen im sehr komplexen InvoiceSubmission-Dienst mit aufwändiger Testphase.

## Ein Kompromiss: Agile Strategie

- Inkrementelle, fortlaufende Top-Down-Analyse mit anfänglicher Konzentration auf die wichtigsten Probleme
- nach genügend Vorlauf bzgl. Analyse Beginn des Service-Designs
- Gemäß Fortschritt der Top-Down-Analyse werden Design-Entscheidungen abgeglichen

Problem: Was ist genügend Vorlauf?



## Serviceorientierte Analyse

- Vom Umfang des Projekts und gewählter Strategie abhängig, z.B.:
  - Umfassende Analyse als Teil eines unternehmensweiten Plans zur SOA-Einführung
  - Beschränkte Analyse zur Entwicklung einiger weniger Dienste
- Wichtige Fragen:
  - Fachliche Anforderungen festlegen, Dokumentation der Geschäftsprozesse
  - Gesamtmodell für Architektur festlegen (z.B. Entscheidung über Orchestrierungsschicht)
  - Plan zur Integration vorhandener Anwendungen
  - Service-Kandidaten modellieren

## Sicherstellung der SOA-Service-Eigenschaften

Bei Festlegung der Service-Kandidaten achten auf:

- Wiederverwendbarkeit
- Autonomie

Beim Service-Design wichtig:

- Zustandslosigkeit
- Entdeckbarkeit

Sonstige Service-Eigenschaften durch Verwendung von Webservices sichergestellt

## Fallbeispiel: RailCO

Situation / Geschäftsmodell / Grundsatzentscheidungen:

- Durch entwickelte Services kann TLS weiter beliefert werden, aber: Konkurrierender Bremsenhersteller bleibt Hauptlieferant von TLS
- Aquisition neuer Kunden notwendig, um Produktionskapazität auszulasten
- Defizite der vorhandenen Softwarearchitektur:
  - sehr TLS-spezifisch
  - schlecht änderbar
- Ziel: SOA-Neuentwicklung nach agiler Strategie

## RailCo - Erste Analyseergebnisse

- Verzicht auf Orchestrierungsschicht: zu aufwändig, Middleware zu teuer
- Separate Applikationsschicht, Integration bestehender Anwendungen in diese Service-Schicht
- Plattform-unabhängige Geschäftsschicht mit wenigen Diensten  
(konkretes Ziel: vereinfachte Austauschbarkeit der derzeit benutzten Applikationen)
- Neuer Service für CRM (Kundenkontakte)

## **Problem:** Wie identifiziert man die Dienste der Geschäftsschicht?

Alternative Ansätze:

- Geschäftsprozess-Modellierung (einfacher, weniger flexible Dienste)

Kandidaten für Prozess-orientierte Services aus unterschiedlichen Quellen:

- Verschiedene Methoden zur Spezifikation und Verwaltung von Geschäftsprozessen (BPM = „business process management“)
- Analyse der Anwendungsfälle („use cases“)

Beispiele: *VerifiziereRechnung*, *BearbeiteAuftrag*

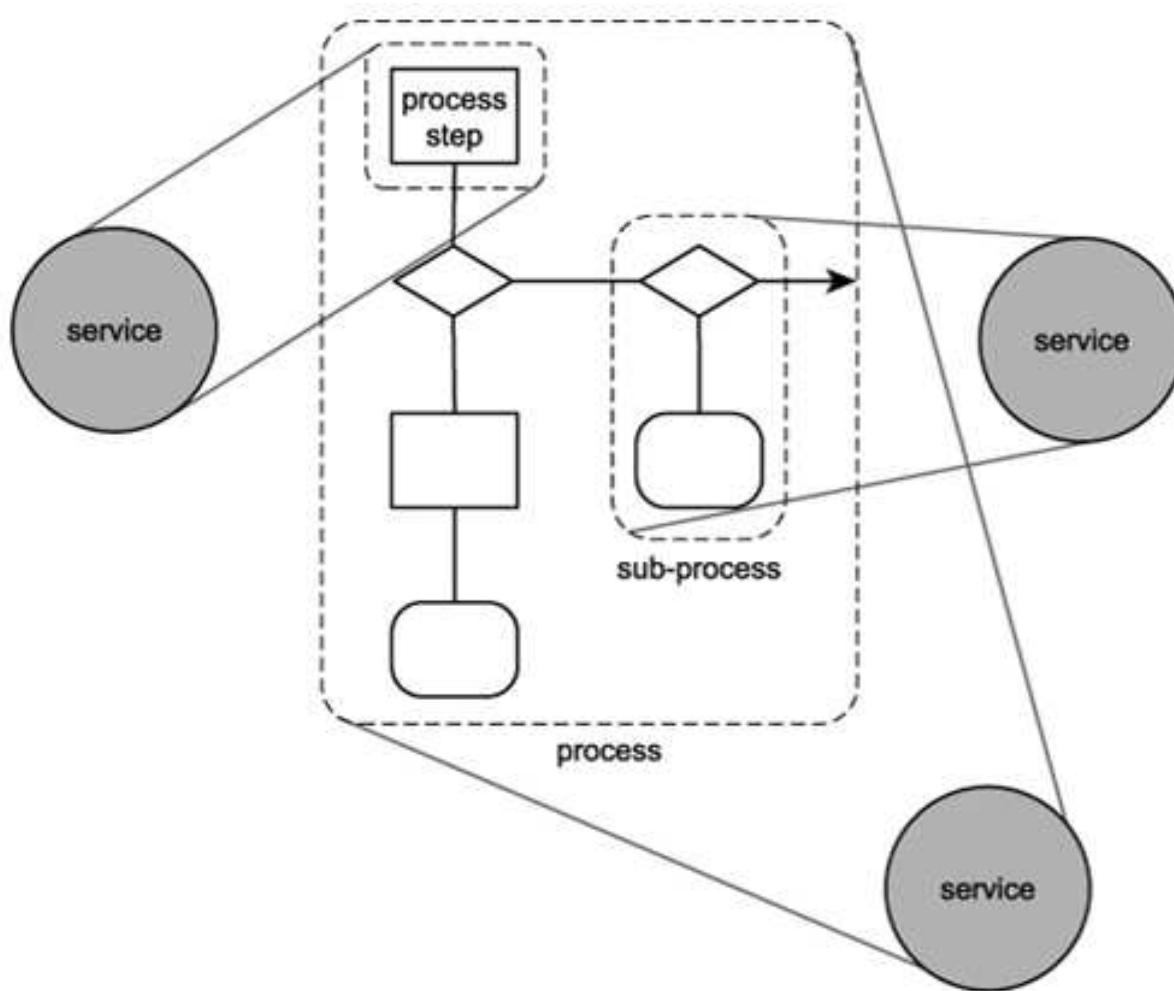
Fallbeispiel: alle RailCO-Dienste sind Prozess-orientiert

- Entitäts-Modelle (Analyse/Design aufwändiger, dafür tendenziell flexiblere Komponenten)

Kandidaten für Entitäts-orientierte Services lassen sich aus Entity-Relationship-Modellen entnehmen

Beispiele: *Rechnung*, *Auftrag*, *Kunde*

## Geschäftsprozess-orientierte Identifikation der Service-Kandidaten



## Analyse im Fallbeispiel: Rechnungsstellung bei RailCO (Geschäftsprozess)

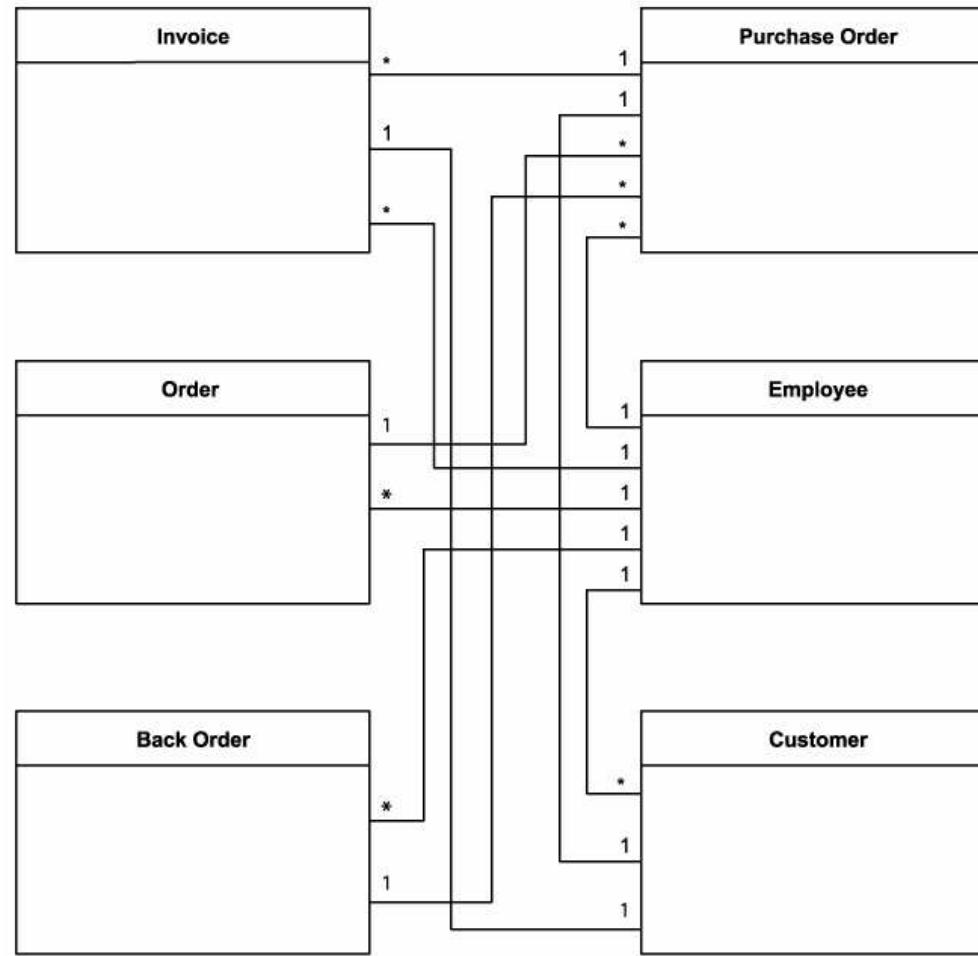
1. Buchhalter erstellt Rechnung mit Buchführungsprogramm
2. Einbuchung der Rechnung stößt ein von RailCO entwickeltes Skript an, dass eine elektronische Rechnung in einem Rechnungsverzeichnis auf einem Netzwerkserver erzeugt
3. Eine von RailCO entwickelte Komponente prüft alle 10 Minuten das Rechnungsverzeichnis, erzeugt aus jeder neuen Rechnung ein XML-Dokument
4. XML-Rechnung wird validiert und an den Rechnungseinreichungs-Service geleitet. Prozess endet, falls Validierung fehlschlägt.
5. Rechnungseinreichungs-Service führt zeitgesteuert Metadaten-Abfragen bei Kunden aus.
6. Falls Metadaten noch aktuell, wird Rechnung an TLS verschickt, „exactly once“-Semantik wird verwendet.

Bei Metadaten-Änderung endet der Prozess

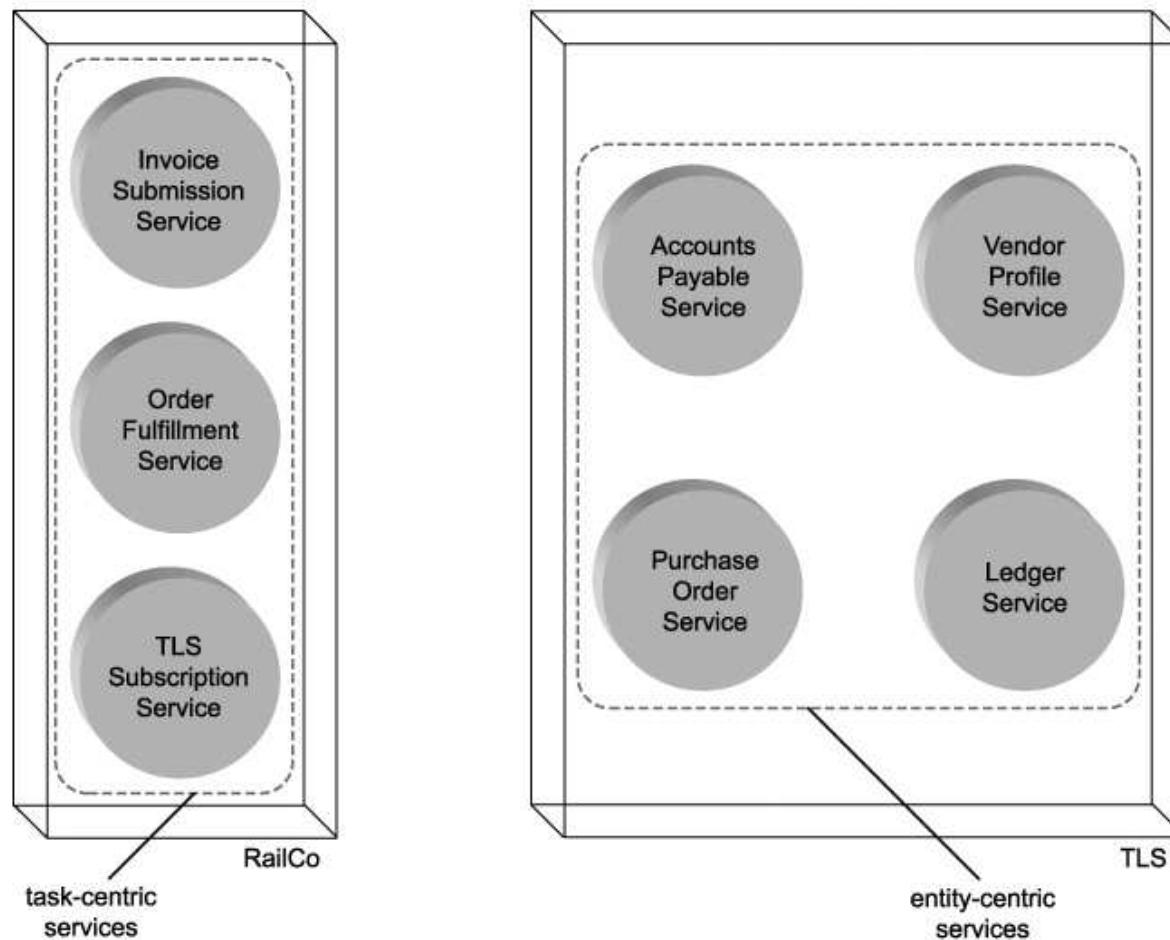
## Analyse im Fallbeispiel: Auftragsbearbeitung bei RailCO (Geschäftsprozess)

1. Empfang einer SOAP-Nachricht von TLS mit Auftrag als Nutzlast
2. Validierung
  - bei Fehler: Benachrichtigung von TLS
  - bei Erfolg: Weiterleitung an Transformationsskript
3. Transformation vom XML-Format in das interne Format der Buchführungssoftware
4. Import in die Buchführungssoftware
5. Auftrag in den Auftragsstapel des zuständigen Sachbearbeiters einfügen

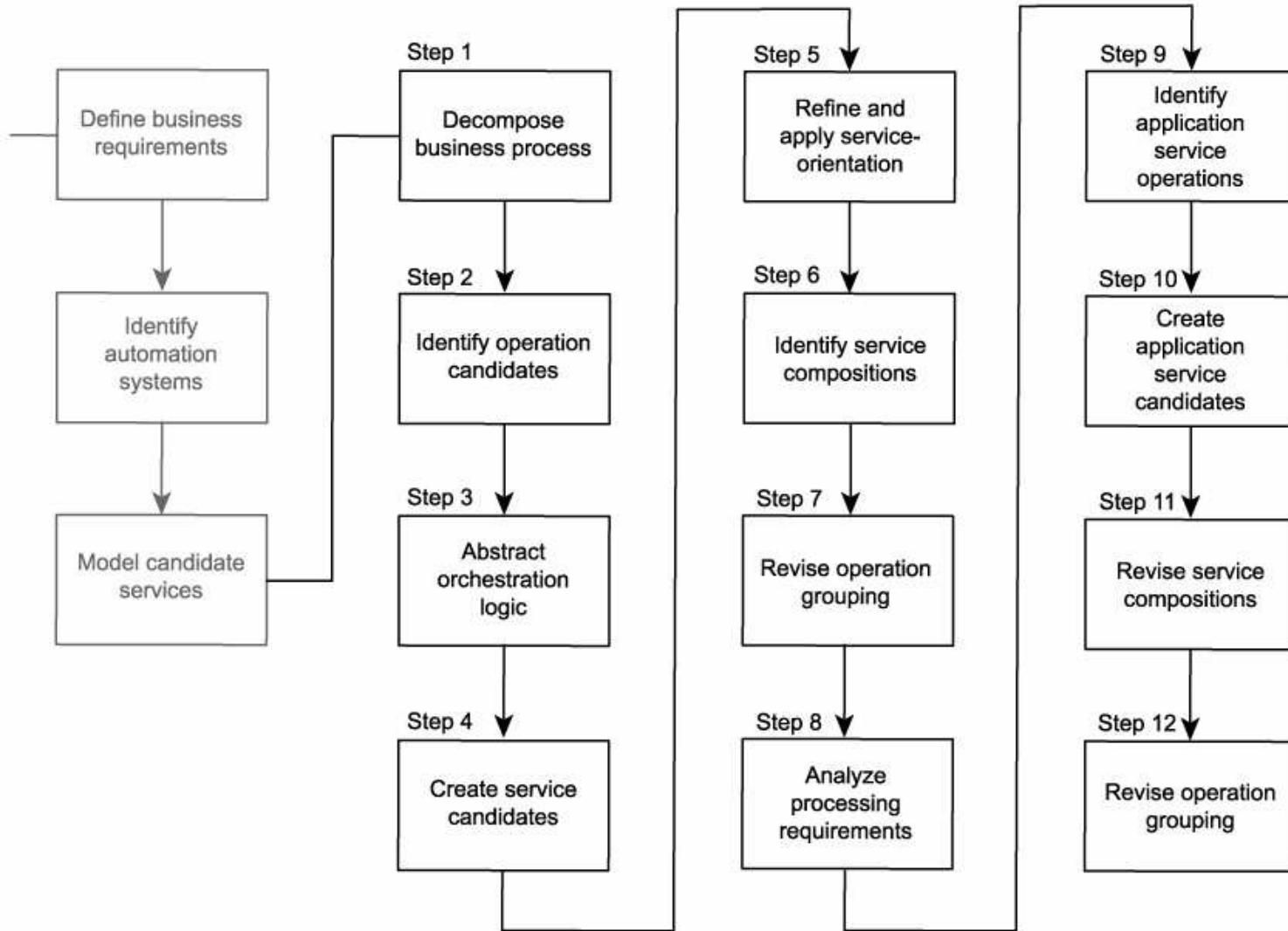
## Fallbeispiel: RailCO Entitätenmodell



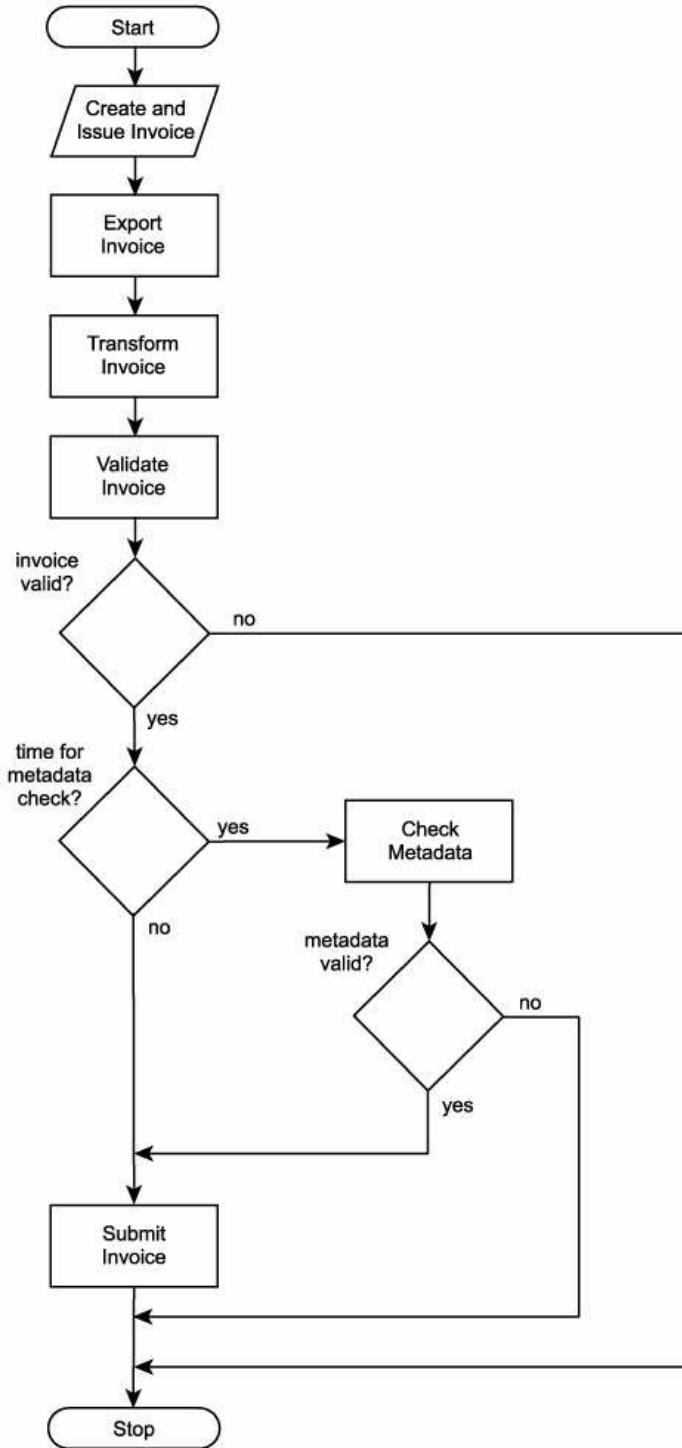
## Prozessbezogene und Entitäts-bezogene Services



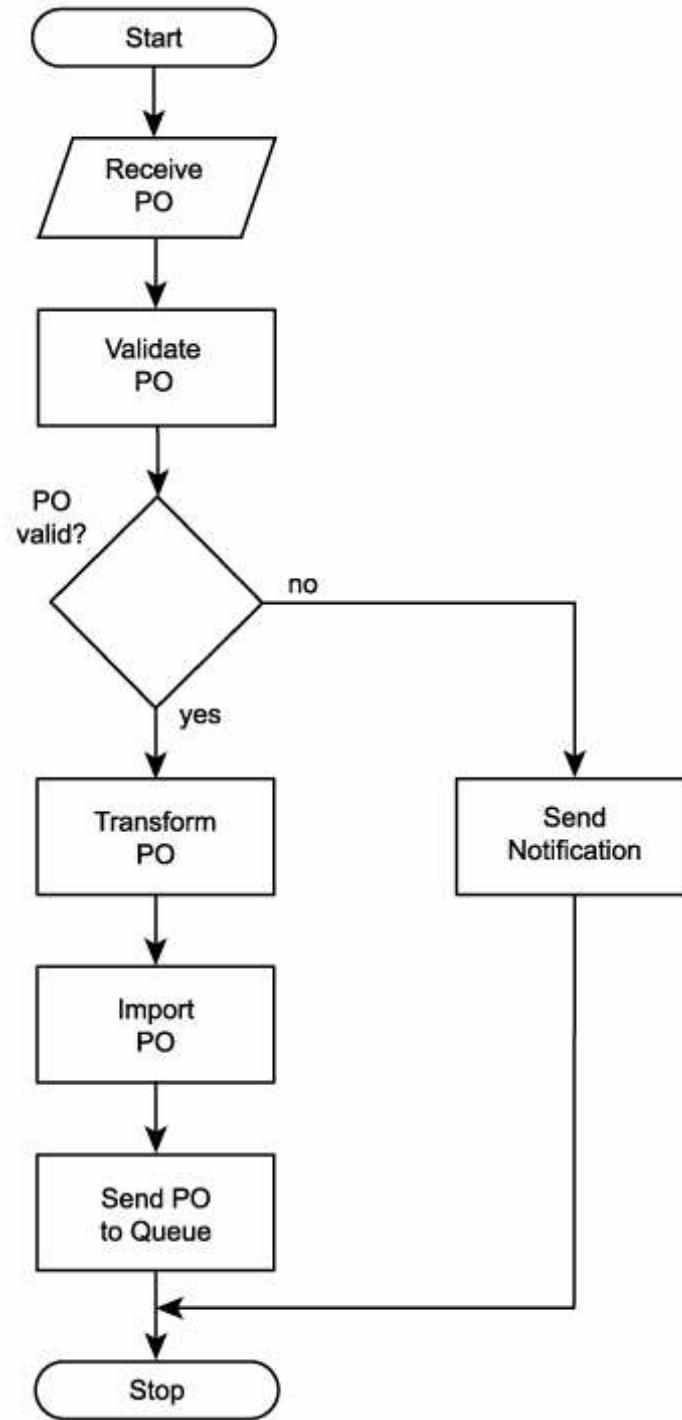
# Modellierung der Service-Kandidaten



## **Schritt 1 - Geschäftsprozess zerlegen: RailCO Rechnungserstellung**



# RailCO Auftragsbearbeitung



## **Schritt 2 - Business-Service-Operations-Kandidaten suchen im RailCO Rechnungserstellungsprozess**

### **Welche Schritte sind Business-Service-Operations-Kandidaten?**

- Erfasse Daten für elektronische Rechnung
- Verbuche elektronische Rechnung
- Exportiere Rechnung in Netzwerkverzeichnis
- Prüfe regelmäßig Netzwerkverzeichnis
- Hole Rechnung aus Netzwerkverzeichnis
- Erzeuge XML-Format
- Validiere XML-Rechnung
- Prüfe, ob Metadatenabgleich notwendig
- Falls notwendig, führe Metadatenabgleich durch

## **Service-Operations-Kandidaten suchen im RailCO Auftragsbearbeitungsprozess**

### **Welche Schritte sind Service-Operations-Kandidaten?**

- Rechnung empfangen
- Rechnung validieren
- Falls Validierung fehlschlägt, Benachrichtigung schicken und Prozess beenden
- XML-Auftragsformat in internes Format umwandeln
- Import des Auftrags in die Buchführungssoftware
- Auftrag in die Bearbeitungs-Warteschlange des Buchhalters stellen

## Schritt 3: Konzept für Orchestrierungslogik

- Erstelle Workflow-Konzept für das Zusammenwirken der potentiellen Services im Geschäftsprozess

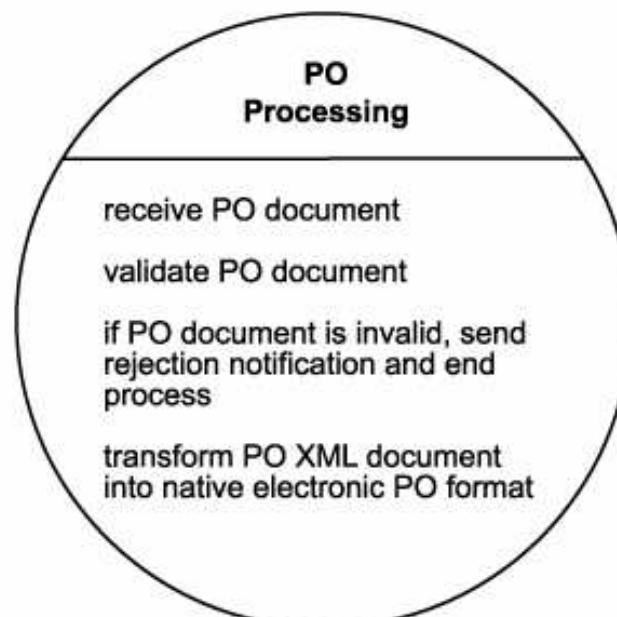
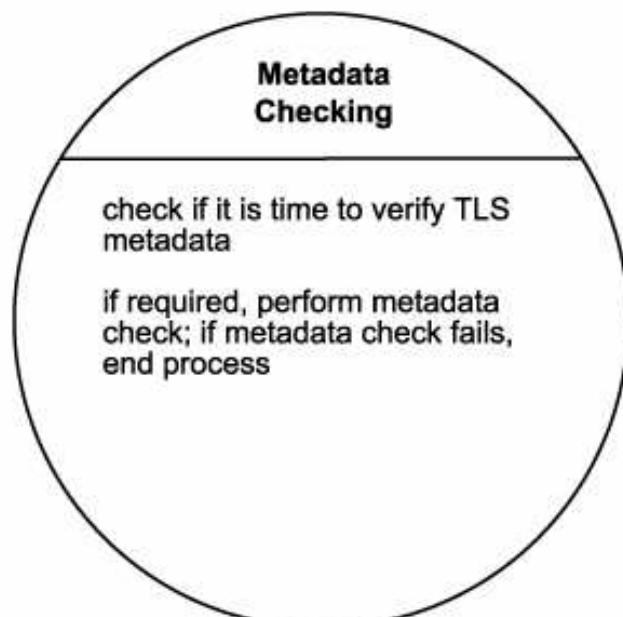
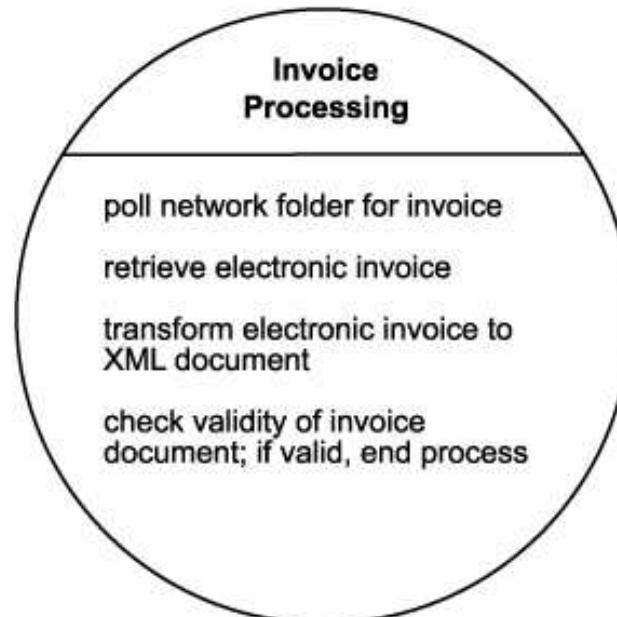
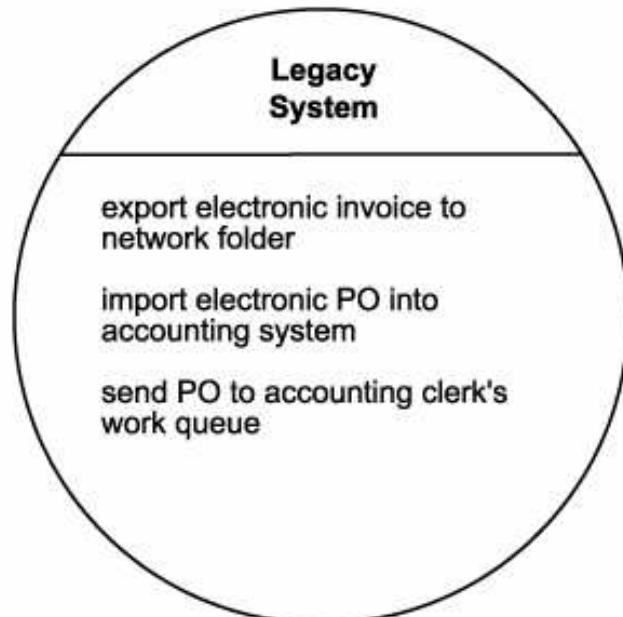
**Fallbeispiel:** Workflow in den RailCO-Geschäftsprozessen:

- Falls Rechnungsdokument validiert, weiter mit Metadatenprüfung  
ansonsten, Prozess abbrechen
- Falls Metadatenabgleich zu lange her, neuer Metadatenabgleich  
ansonsten weiter mit nächstem Schritt
- Falls Auftragsdokument validiert, weiter mit Transformationsschritt  
ansonsten Prozessabbruch

(RailCO entwickelt keine Orchestrierungsschicht)

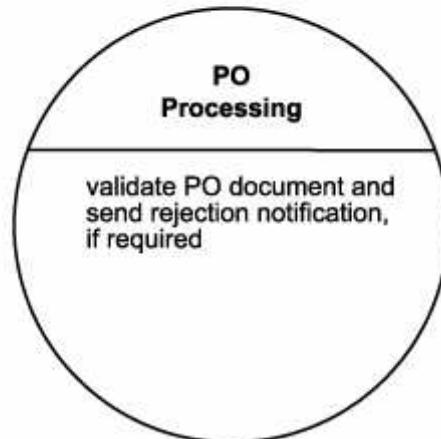
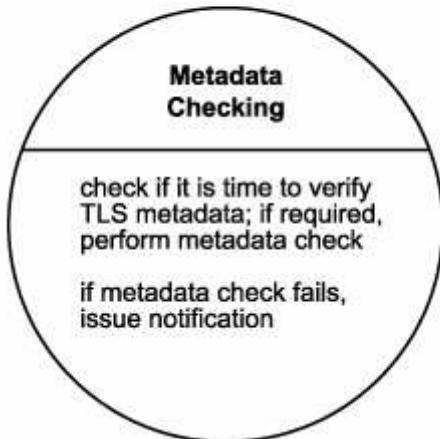
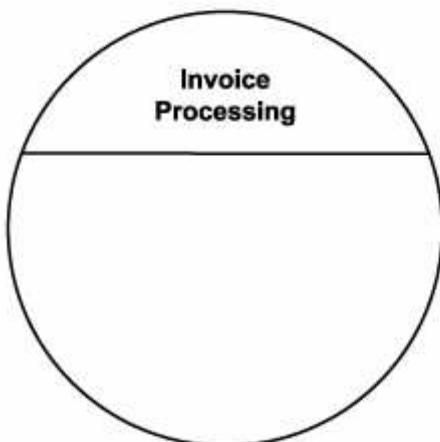
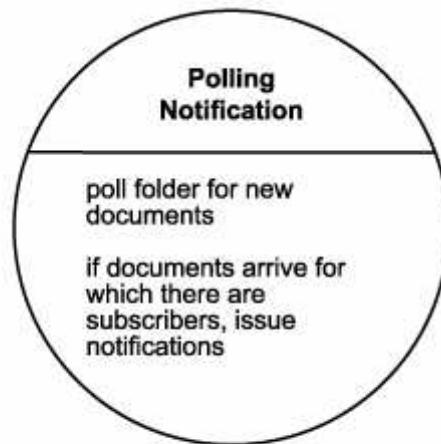
## **Schritt 4: Service-Kandidaten bestimmen**

- Wie gruppiertert man die Operationen?
- Entitäts-bezogene Gruppierungen
- Schichtentrennung
- Gruppierung nach Prozesskontext
- ggf. neue Operationen hinzufügen, um die Wiederverwendbarkeit generischer Dienste zu verbessern

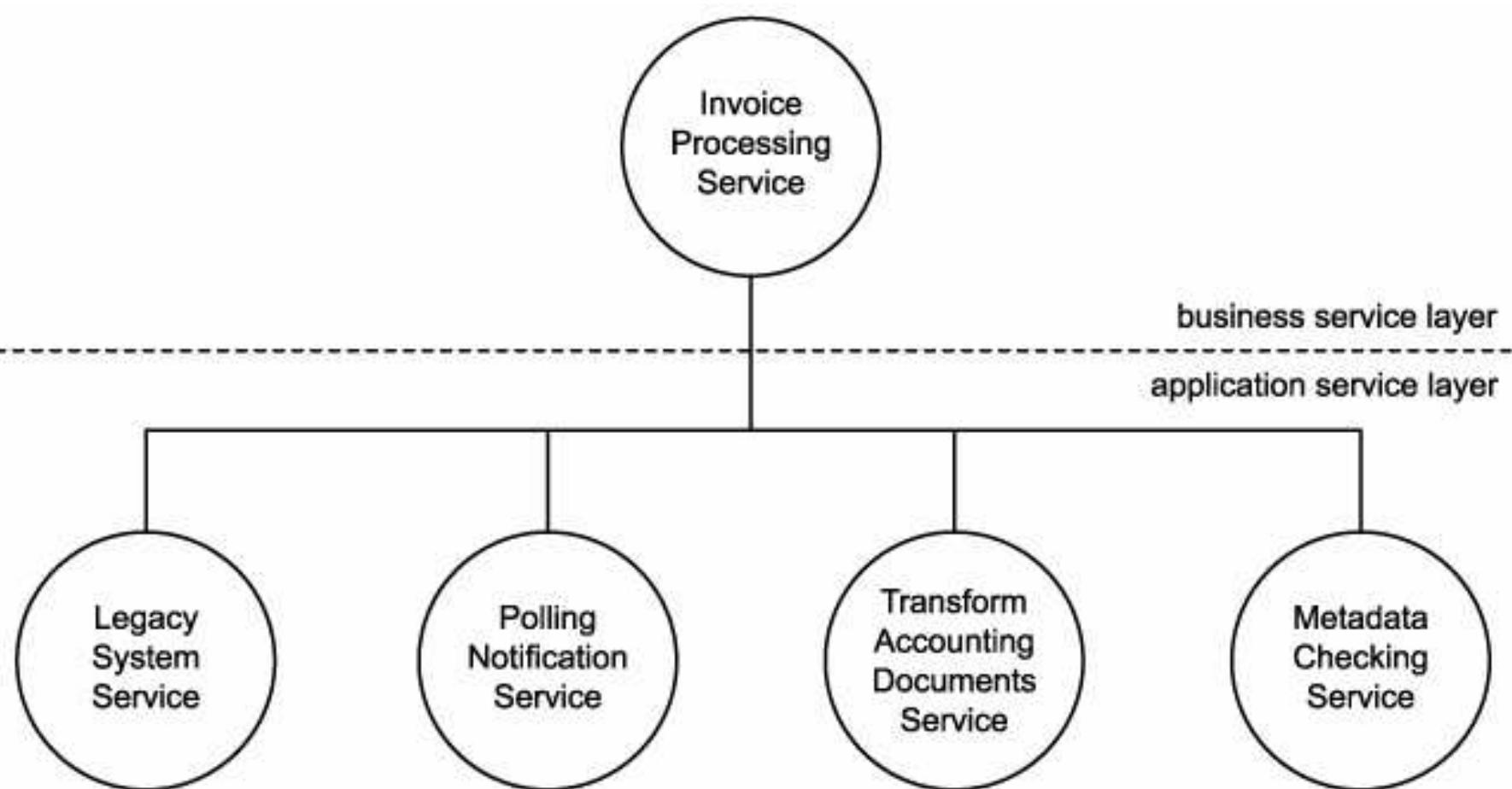


## **Schritt 5: SOA-Prinzipien prüfen, speziell: Wiederverwendbarkeit und Autonomie sicherstellen**

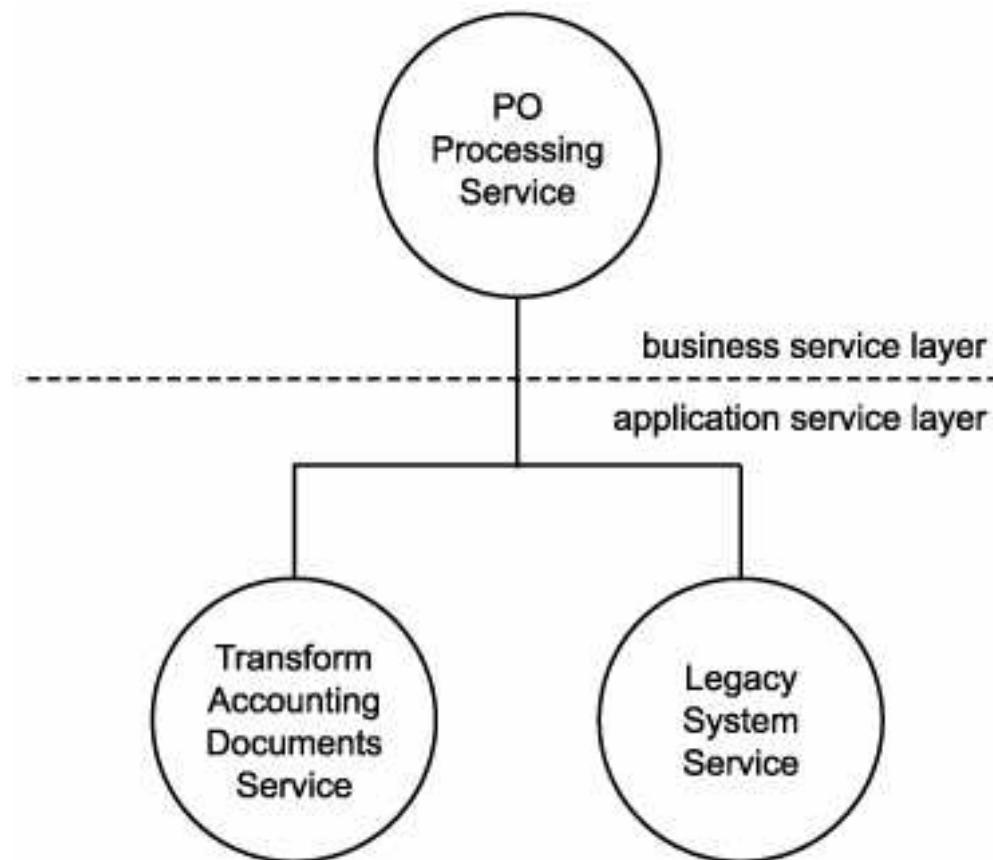
- Bei direkten Abhängigkeiten Operationen ggf. zusammenfassen  
Beispiel: Rechnung importieren und zur Bearbeitung weiterleiten
- Verallgemeinern  
Beispiel: Dokument importieren und zur Bearbeitung weiterleiten, Dokumentenverzeichnis auf neue Dokumente überprüfen



## Schritt 6: Kompositionen identifizieren: Rechnungserstellungsprozess



## Schritt 6: Kompositionen im Auftragsbearbeitungsprozess



## **Schritt 7:** Operationsgruppierungen ggf. revidieren

- Kompositionskonzepte in Schritt 6 können zu Änderungen führen
- Im Fallbeispiel kein Änderungsbedarf

## **Schritt 8:** Applikationsanforderungen berücksichtigen

- Schritte 1-7 betrachten nur die Geschäftsebene
- Bei komplexen Umgebungen mit vielen vorhandenen Applikationen muss deren Integration schon in der Analysephase bedacht werden
  - Welche Prozesse/Teilprozesse sind schon realisiert?
  - Müssen vorhandene Applikationen integriert werden, um einen Service zu implementieren?
- Im Fallbeispiel kein Handlungsbedarf

**Schritt 9:** Dienste der Applikationsschicht: Kandidaten für Operationen bestimmen

**Schritt 10:** Dienste der Applikationsschicht: Dienst-Kandidaten bestimmen

**Schritt 11:** Kompositionskonzept überarbeiten:  
Einbeziehung der Applikationsdienst-Kandidaten

**Schritt 12:** Gruppierung der Operationen überarbeiten

## Richtlinien zur Modellierung der Services

- intensive Prüfung auf Wiederverwendungsmöglichkeiten für prozessbezogene Service-Kandidaten
  - innerhalb eines Geschäftsprozesses
  - Geschäftsprozess-übergreifend

*Nicht zu feinkörnig modellieren!*

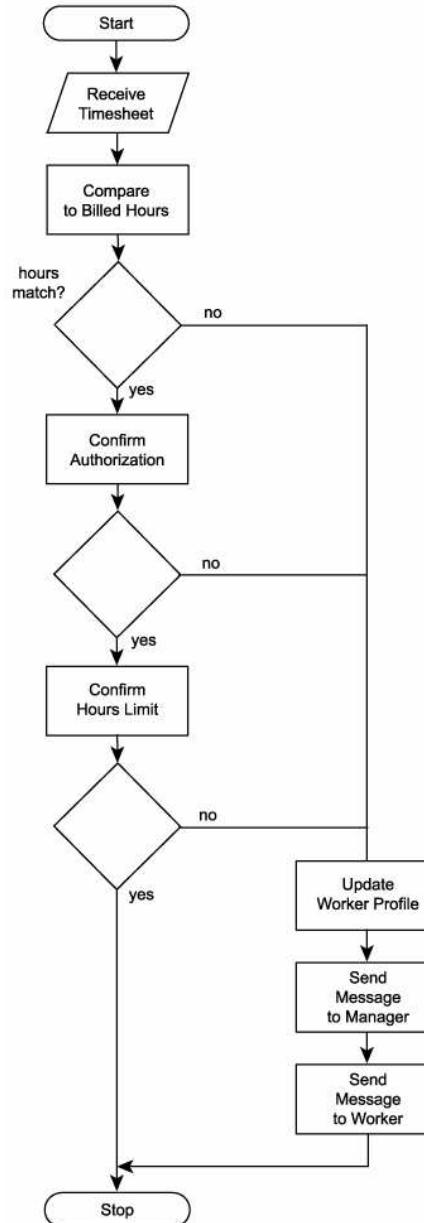
- Genaue Kenntnis *aller* wesentlichen Geschäftsprozesse des Unternehmens nötig

*Service-Modellierung ist nicht auf IT-Personal beschränkt*

⇒ Modellierungsressourcen bereitstellen
- Applikationsdienst-Kandidaten generisch, d.h. unabhängig vom Geschäftsprozess modellieren
- Dekomposition von Services prüfen
- Funktionale Abgrenzbarkeit der Services überprüfen

- Prozess-Services definieren, falls keine Orchestrierungsschicht
- Services klassifizieren nach Rolle und Anwendungsbereich
- Modellierungstandards definieren und veröffentlichen

## Fallbeispiel TLS: Geschäftsprozess Zeiterfassung - Ablauf



## Geschäftsprozess Zeiterfassung: Gruppierung der Aktivitäten

Timesheet Verification Tasks	Timesheet Rejection Tasks
Compare Billed Hours	Update Profile
Confirm Authorization	Send Message to Manager
Confirm Hours Limit	Send Message to Worker

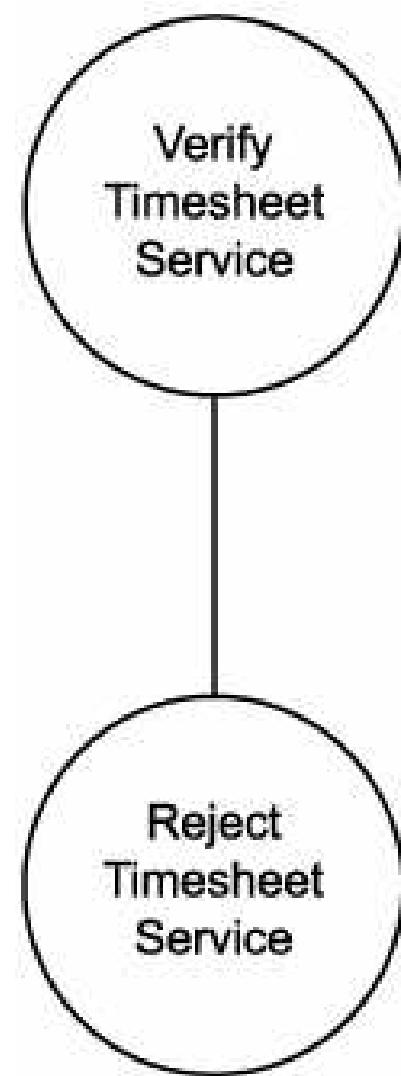
## Service-Kandidat: Prüfung der Zeiterfassungbögen



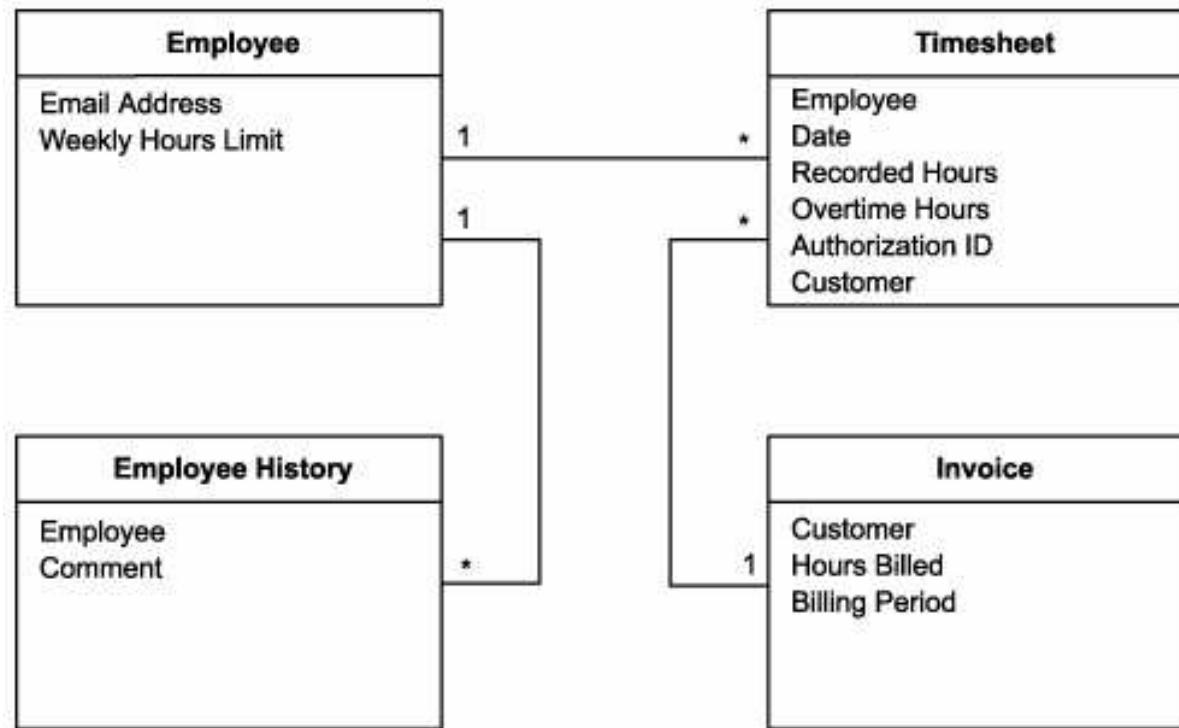
## Service-Kandidat: Ablehnung fehlerhafter Zeiterfassungbögen



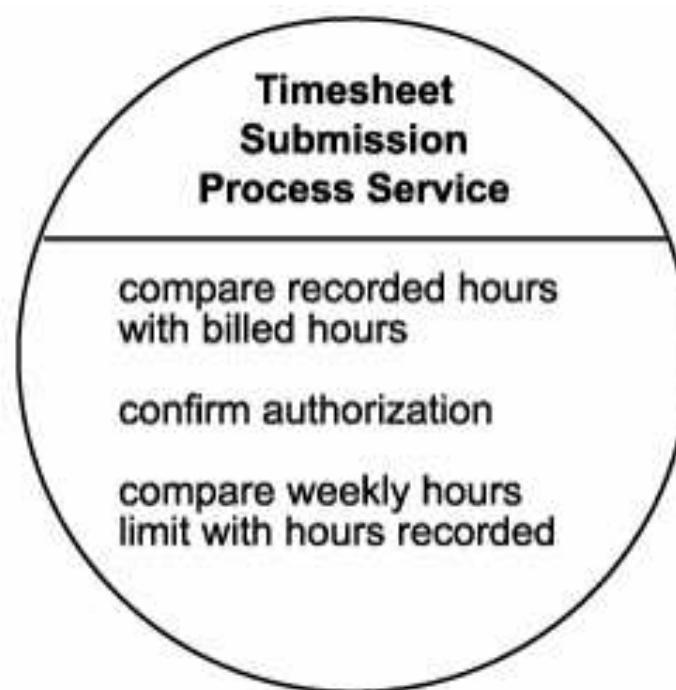
## Ergebnis der prozessorientierten Analyse: Service-Kandidaten



## Entitäts-bezogene Analyse der Zeiterfassung: Entitätsmodell



## Entitäts-bezogene Analyse: Prozesssteuerungs-Service



## Entitäts-bezogene Analyse: Service-Kandidaten



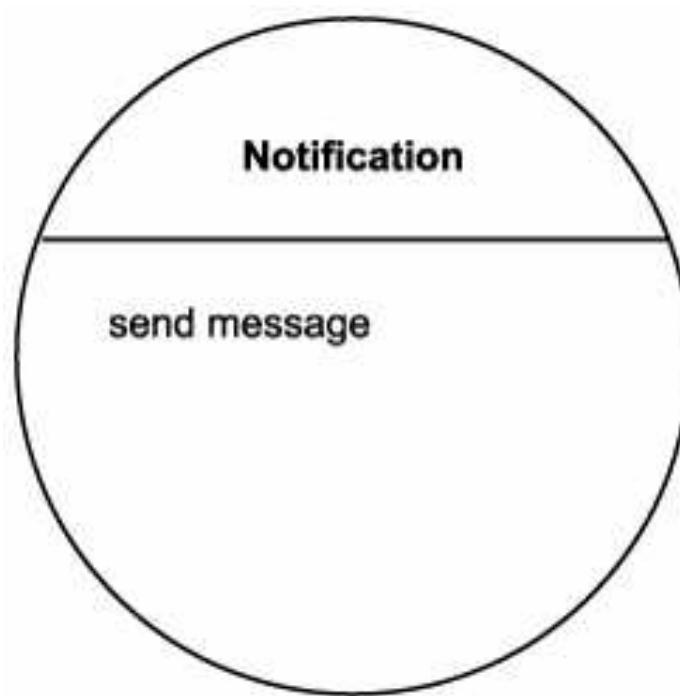
## Entitäts-bezogene Analyse: Service-Kandidaten



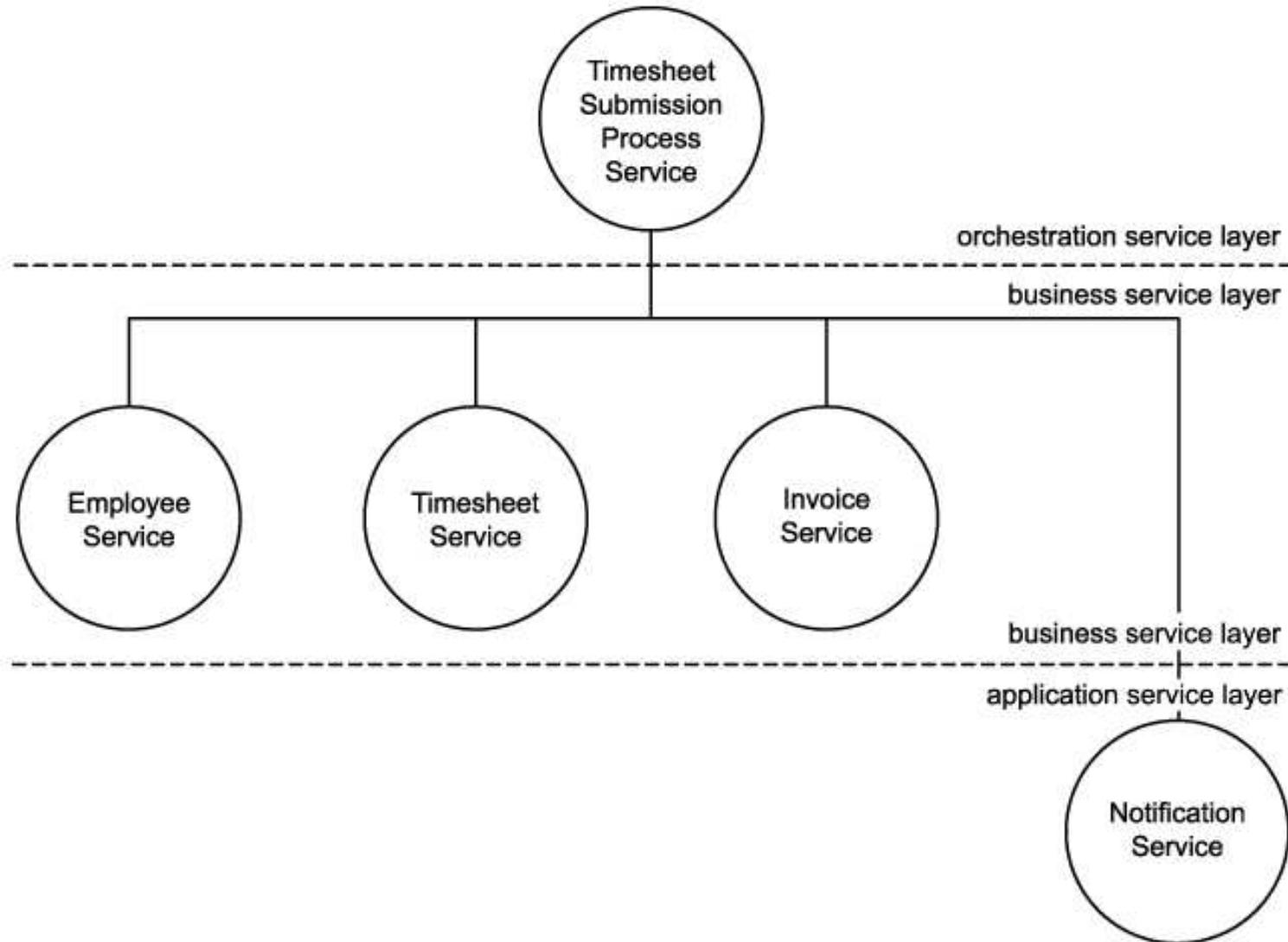
## Entitäts-bezogene Analyse: Service-Kandidaten



## Entitäts-bezogene Analyse: Hilfsdienst



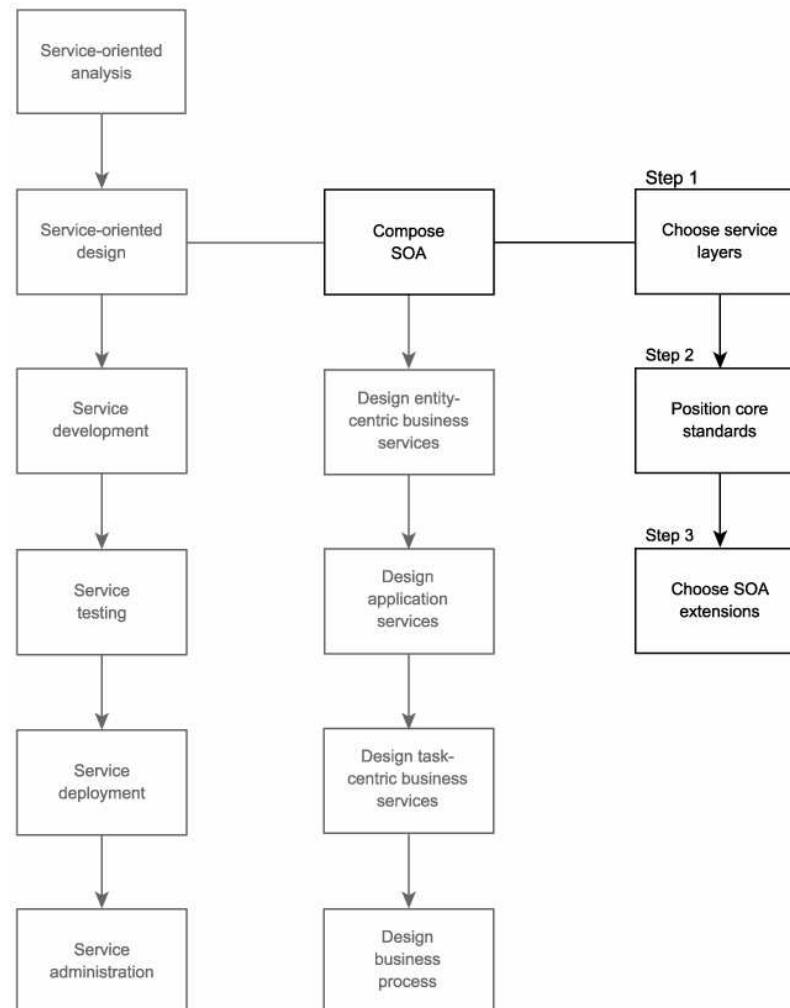
## Entitäts-bezogene Analyse: Koordinationsmodell



## Serviceorientiertes Design

- Service-übergreifende Designaufgaben
  - Schichtenarchitektur festlegen
  - Einsatz der WS-Basistechnologie festlegen: XML, SOAP, WSDL, UDDI
  - Einsatz von WS-Erweiterungen festlegen
- Service-Design
  - Ziel: WSDL-Schnittstellen entwickeln
  - Strategie bzgl. Reihenfolge?
  - Unterschiedliches Vorgehen für einzelne Servicetypen

# Designschritte



## Allgemeine Designfragen

- Schichtenarchitektur: Welche Variante? Wie?

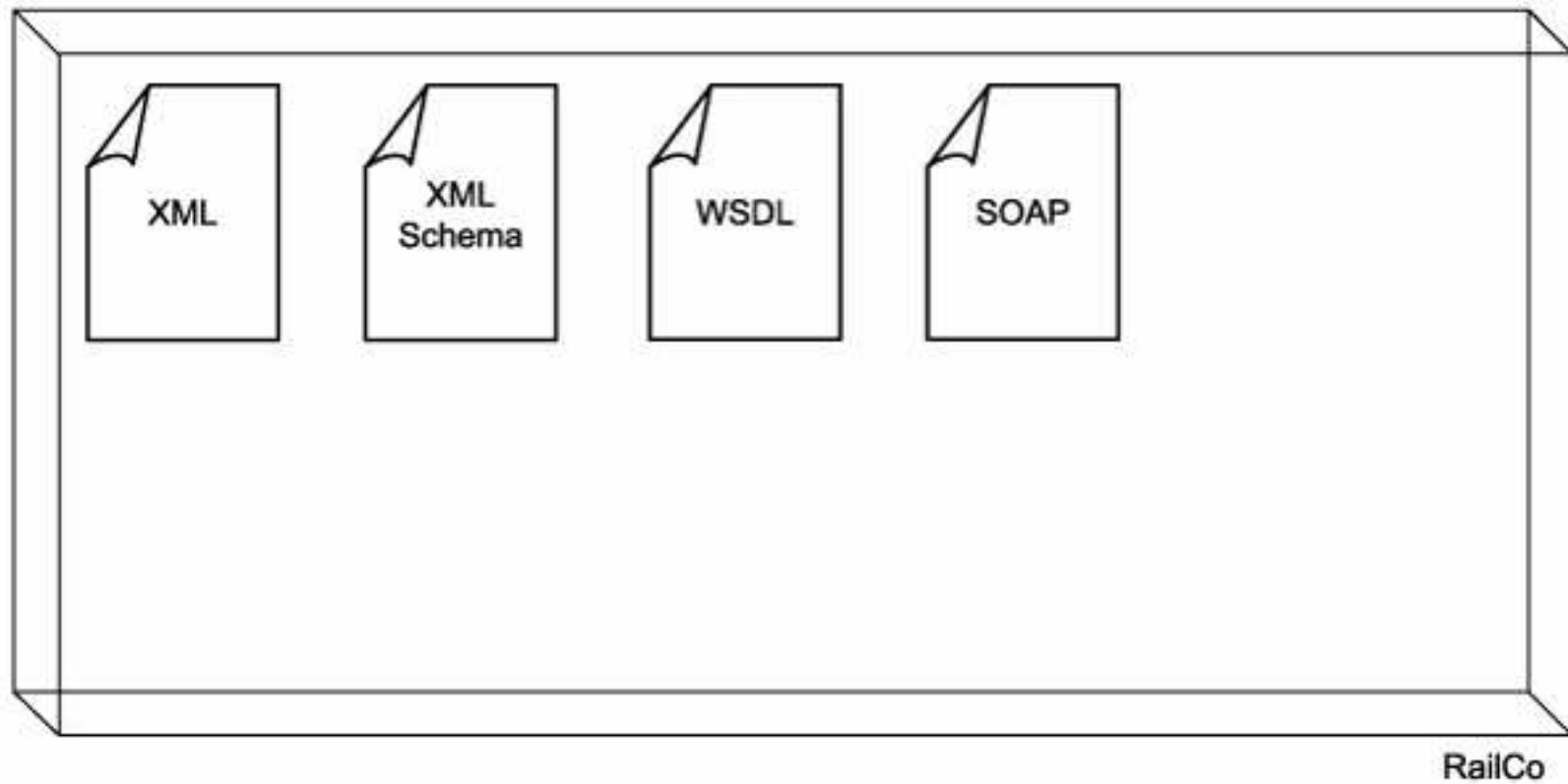
Existierende Schichten, Firmenstandards, Perfomanzprobleme durch Komposition, Perfomanzprobleme durch Zentralisierung von wiederverwendbaren Diensten, Versierung wiederverwendbarer Dienste

- Einbindung existierender XSD-Spezifikationen möglich/sinnvoll?
- „WSDL first“-Strategie
- Planung der Wartung der Dienste bei agiler Strategie
- Einsatz automatisch generierter XML-Dokumente?

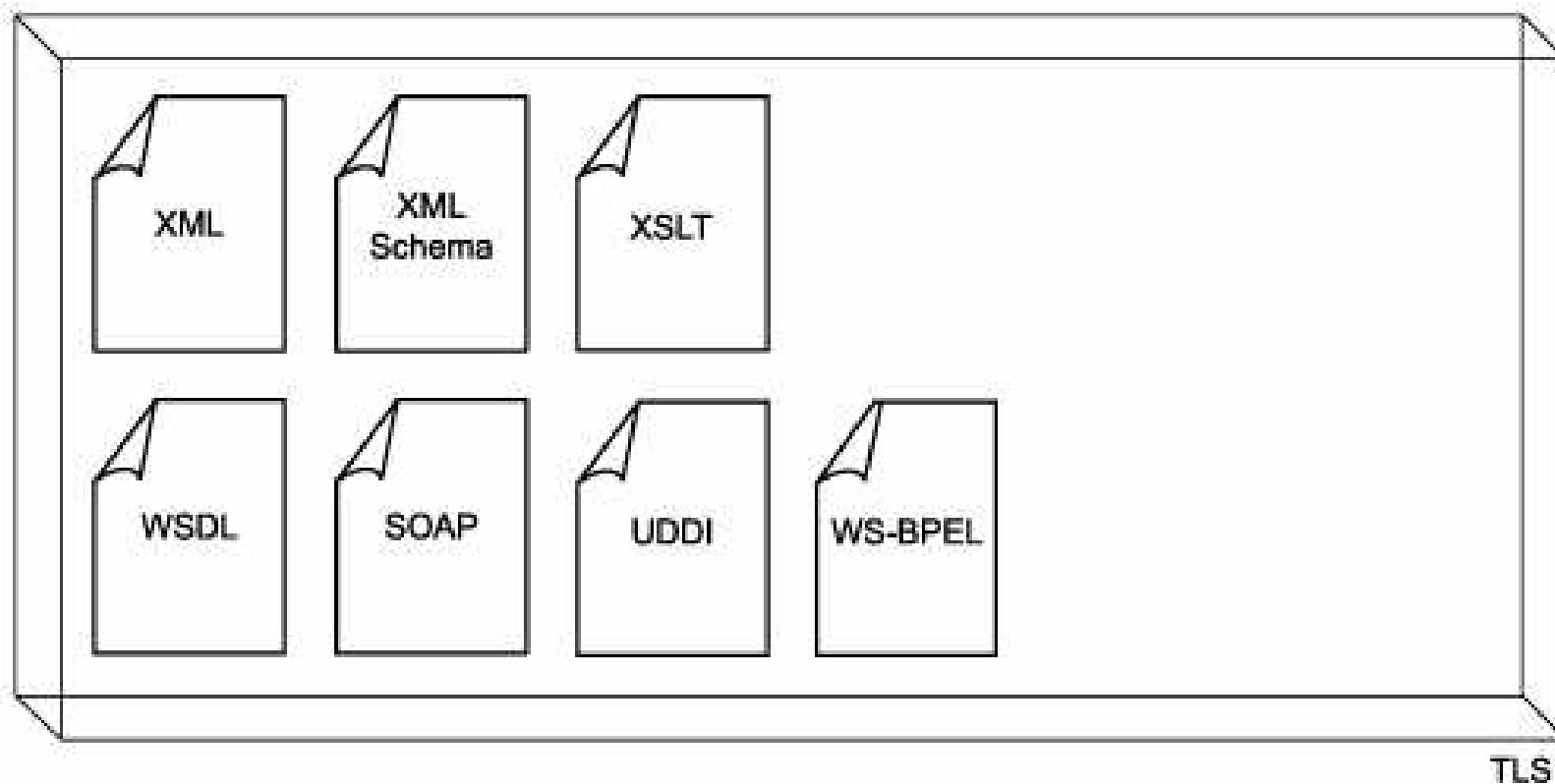
Probleme bzgl. Interoperabilität und Standardkonformität

- Standardisierung
  - Welche Industriestandards?  
SOAP-Nachrichtentyp, WS-I-Basisprofil verwenden (Version?)
  - Firmenstandards berücksichtigen/neu entwickeln/weiterentwickeln:  
z.B.: Namespaces, Modularisierung der Dokumente
- WS-Erweiterungen: Standardkonformität, Stabilität, Unterstützung durch vorhandene Middleware

## Fallbeispiel: Designgrundlagen bei RailCO

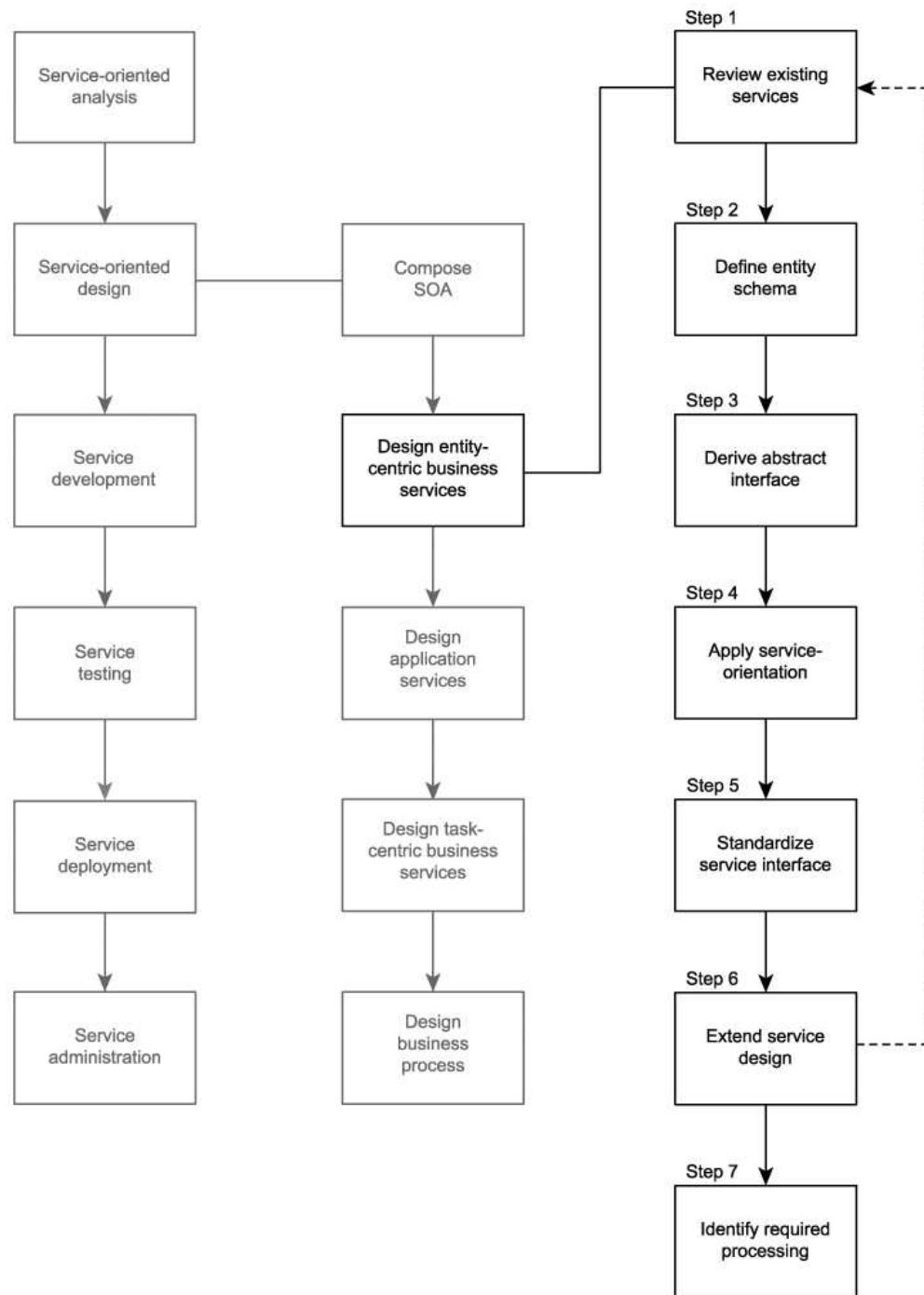


## Fallbeispiel: Designgrundlagen bei TLS

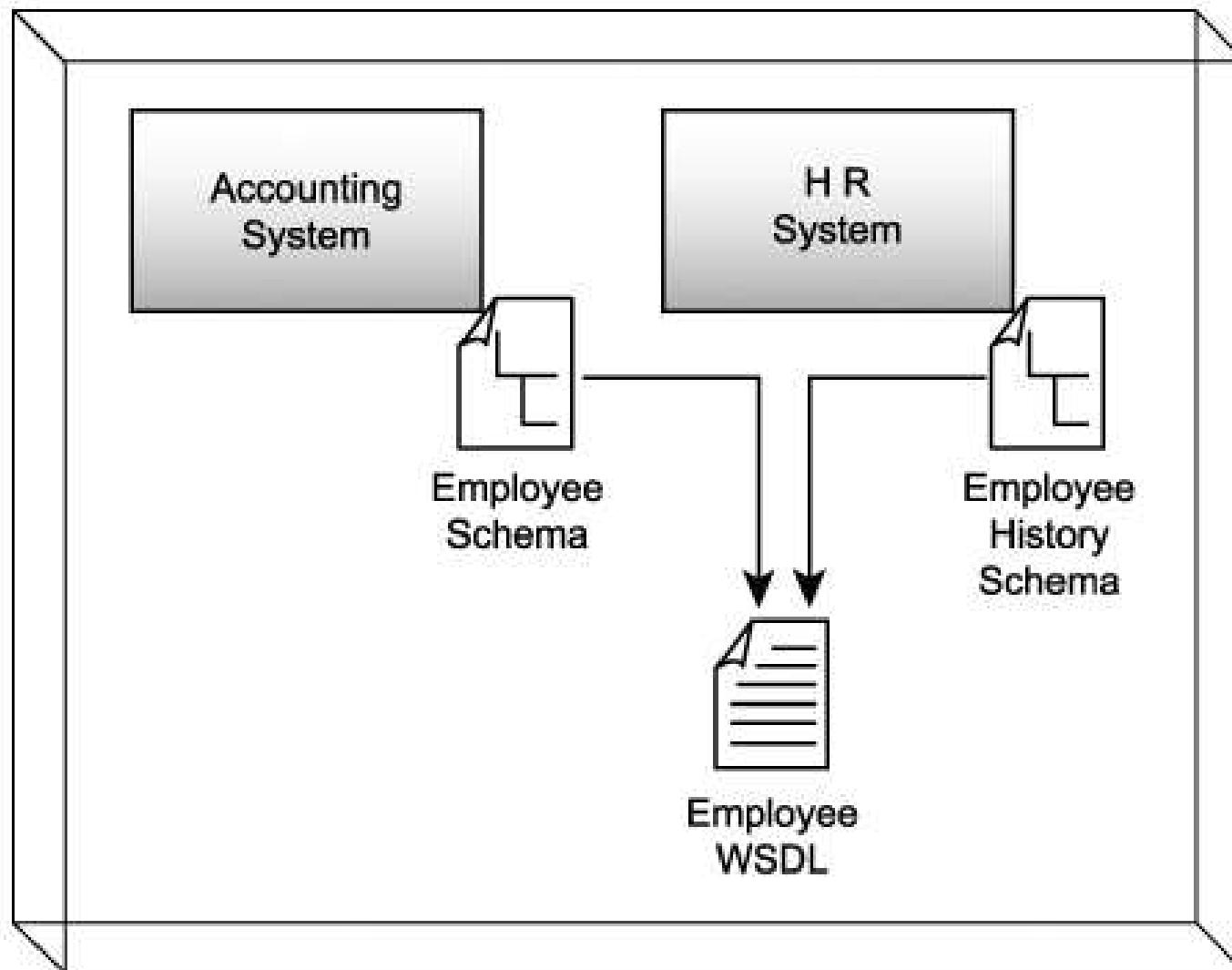


## Design Entitäts-bezogener Services

- Ziel: wiederverwendbare Geschäftsprozess-unabhängige Dienste
- Anforderungen vorhandener Gechäftsprozesse berücksichtigen
- Anforderungen noch nicht vorhandener Geschäftsprozesse? Eventuell generische Operationen für zukünftige Geschäftsprozesse vorsehen (GET/SET/ADD/DELETE/UPDATE)
- Reihenfolge:
  - Entitäten modellieren (separate Datei: XSD)
  - XSD in WSDL importieren
  - Nachrichten spezifizieren
  - Operationen spezifizieren
- Anforderungen an die Applikationsebene festlegen
- Automatische WSDL-Generierung problematisch



## Fallbeispiel: TLS Employee Service



## Probleme:

- Für das HR-System („human resources“) existieren keine XML-Schemata
- Für „Accounting“ existiert Schema, aber: viel zu komplex

daher: komplette Neudefinition

## Schema-Beispiel: EmployeeHistory.xsd

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.xmltc.com/tls/employee/schema/hr/">
  <xsd:element name="EmployeeUpdateHistoryRequestType">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="ID" type="xsd:integer"/>
        <xsd:element name="Comment" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="EmployeeUpdateHistoryResponseType">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="ResponseCode"
          type="xsd:byte"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

## Import von EmployeeHistory.xsd in die WSDL-Definition

```
<types>
  <xsd:schema targetNamespace=
    "http://www.xmltc.com/tls/employee/schema/">
    <xsd:import namespace=
      "http://www.xmltc.com/tls/employee/schema/accounting/"
      schemaLocation="Employee.xsd"/>
    <xsd:import namespace=
      "http://www.xmltc.com/tls/employee/schema/hr/"
      schemaLocation="EmployeeHistory.xsd"/>
  </xsd:schema>
</types>
```

## Abstrakte Service-Schnittstelle

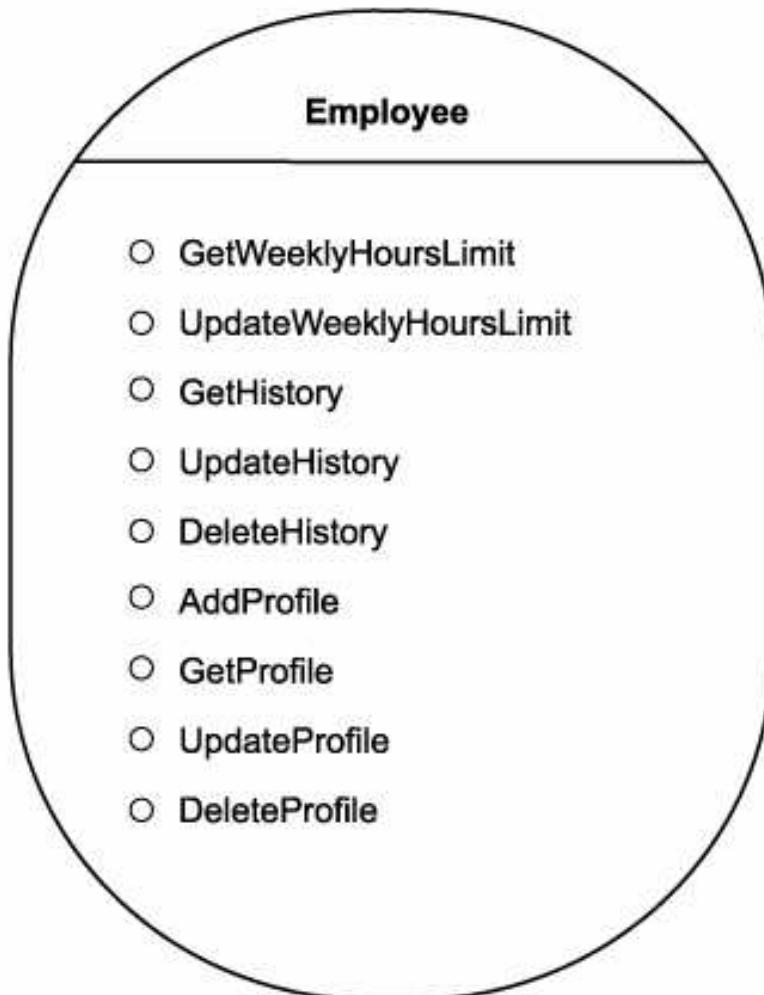
```
<message name="getEmployeeWeeklyHoursRequestMessage">
    <part name="RequestParameter"
        element="act:EmployeeHoursRequestType"/>
</message>
<message name="getEmployeeWeeklyHoursResponseMessage">
    <part name="ResponseParameter"
        element="act:EmployeeHoursResponseType"/>
</message>
<message name="updateEmployeeHistoryRequestMessage">
    <part name="RequestParameter"
        element="hr:EmployeeUpdateHistoryRequestType"/>
</message>
<message name="updateEmployeeHistoryResponseMessage">
    <part name="ResponseParameter"
        element="hr:EmployeeUpdateHistoryResponseType"/>
</message>
```

```
<portType name="EmployeeInterface">
  <operation name="GetEmployeeWeeklyHoursLimit">
    <input message=
      "tns:getEmployeeWeeklyHoursRequestMessage"/>
    <output message=
      "tns:getEmployeeWeeklyHoursResponseMessage"/>
  </operation>
  <operation name="UpdateEmployeeHistory">
    <input message=
      "tns:updateEmployeeHistoryRequestMessage"/>
    <output message=
      "tns:updateEmployeeHistoryResponseMessage"/>
  </operation>
</portType>
```

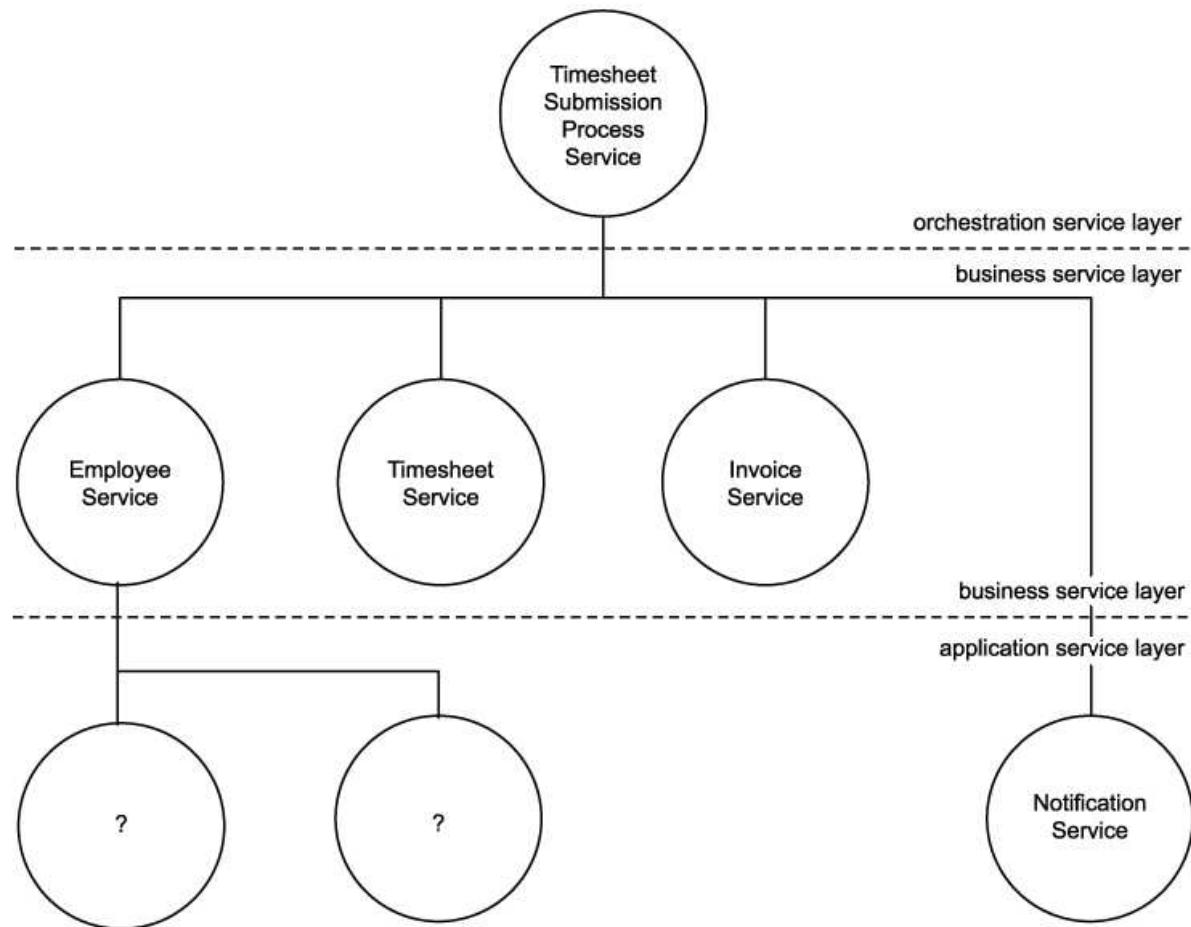
## Entdeckbarkeit verbessern: Abstrakte Service-Schnittstelle mit Kurzbeschreibung

```
<message ...>
...
<portType name="EmployeeInterface">
  <documentation>
    GetEmployeeWeeklyHoursLimit uses the Employee
    ID value to retrieve the WeeklyHoursLimit value.
    UpdateEmployeeHistory uses the Employee ID value
    to update the Comment value of the EmployeeHistory.
  </documentation>
  <operation name="GetEmployeeWeeklyHoursLimit">
    <input message="tns:getEmployeeWeeklyHoursRequestMessage"/>
    <output message="tns:getEmployeeWeeklyHoursResponseMessage"/>
  </operation>
  <operation name="UpdateEmployeeHistory">
    <input message="tns:updateEmployeeHistoryRequestMessage"/>
    <output message="tns:updateEmployeeHistoryResponseMessage"/>
  </operation>
</portType>
```

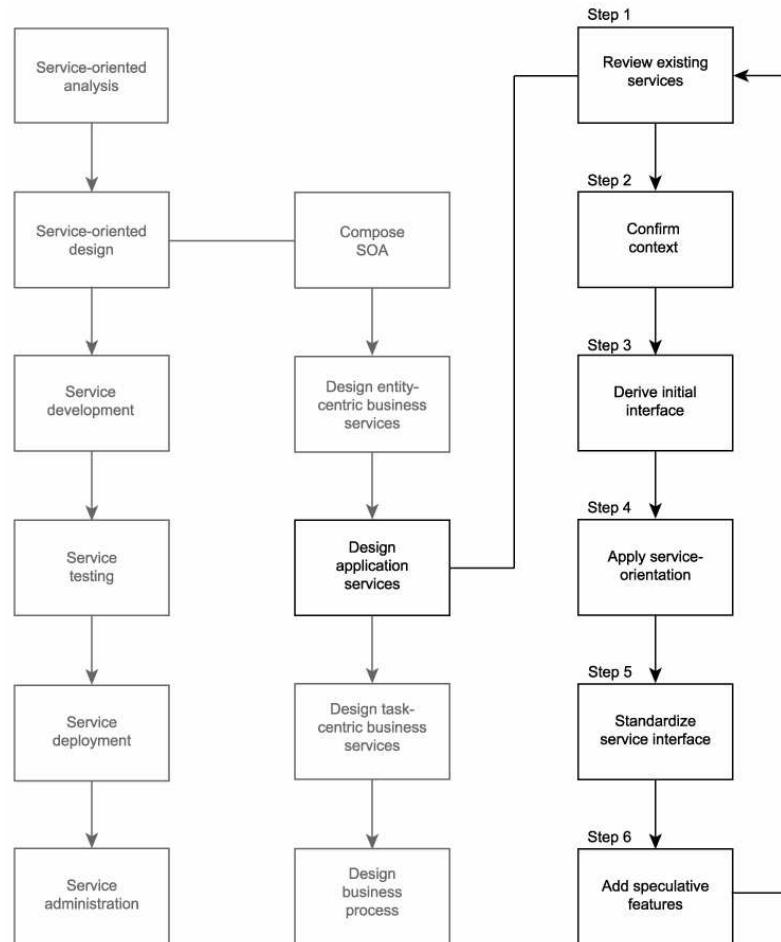
## Wiederverwendbarkeit verbessern: Erweiterung des TLS Employee Service



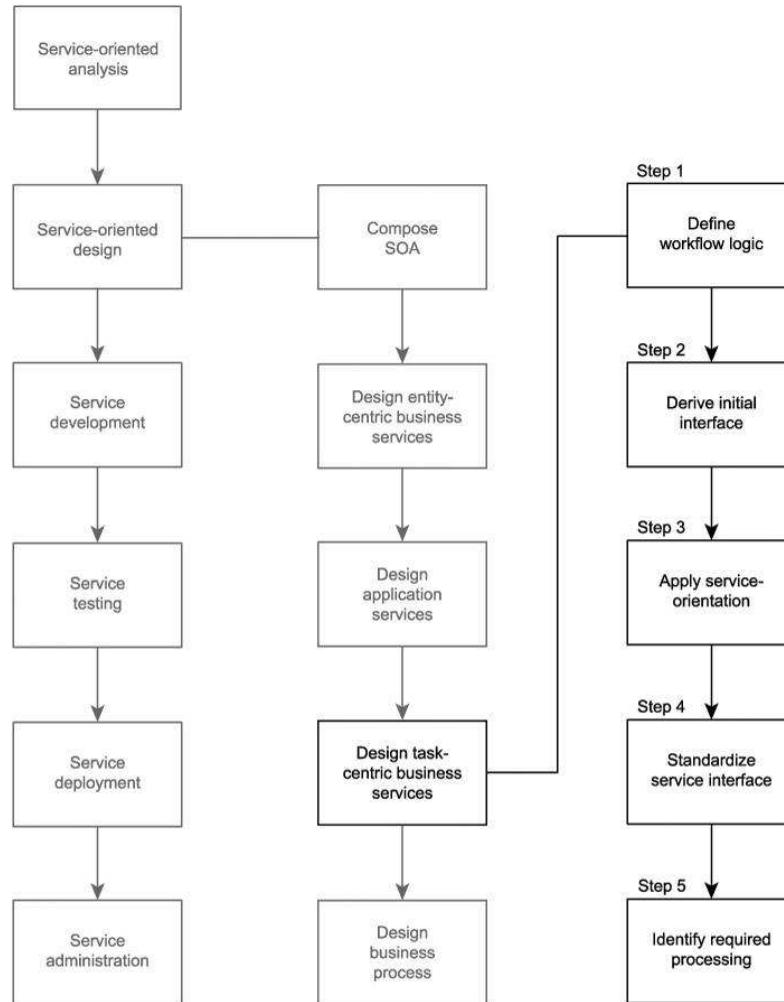
## Fallbeispiel: Employee im TLS-Schichtenmodell



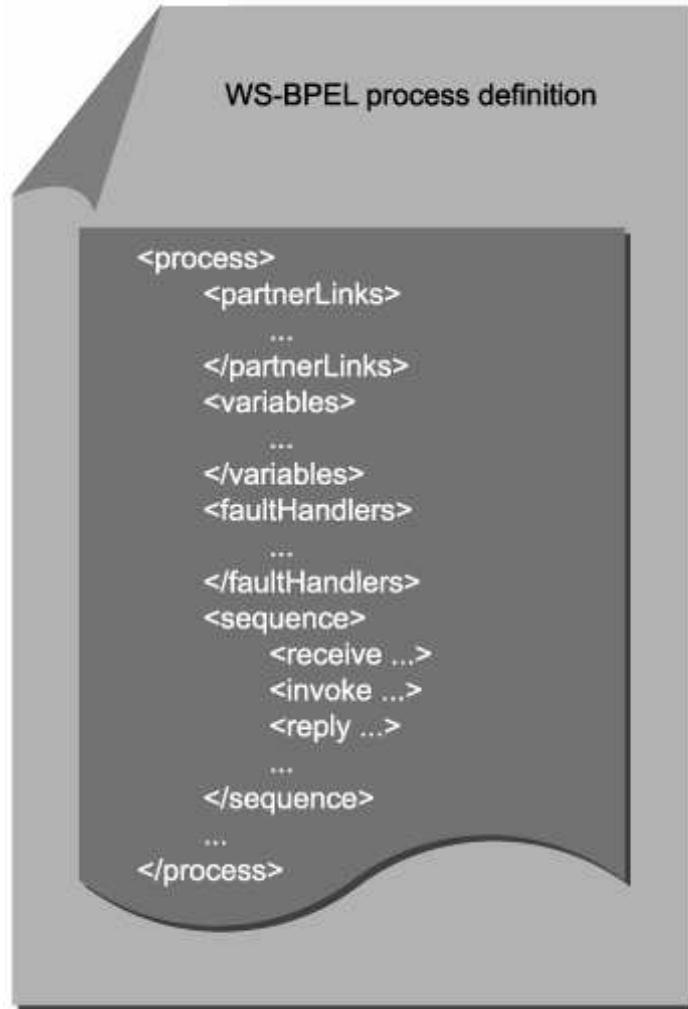
# Designschritte für Applikationsdienste



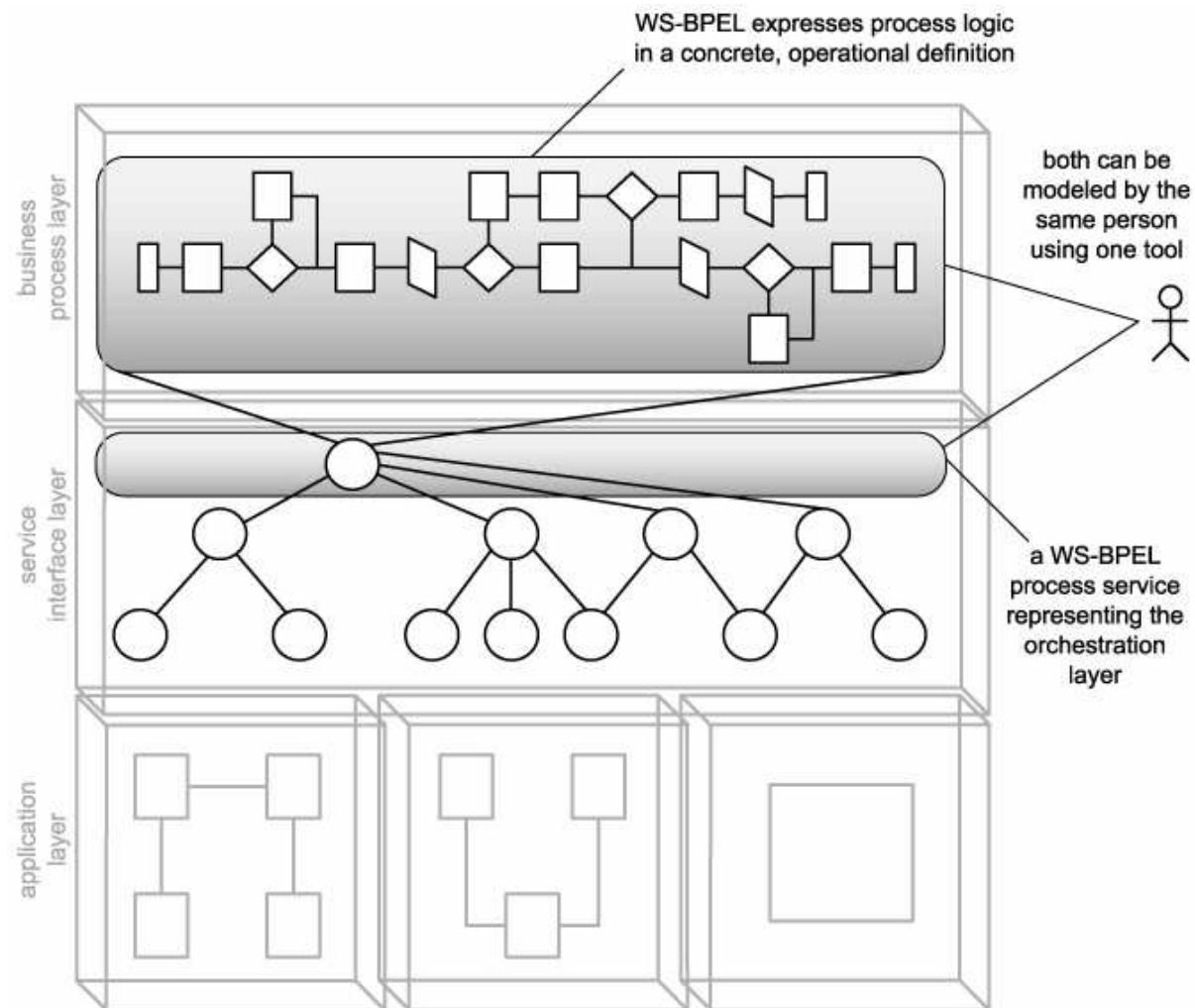
# Designschritte für prozessbezogene Dienste der Geschäftsebene



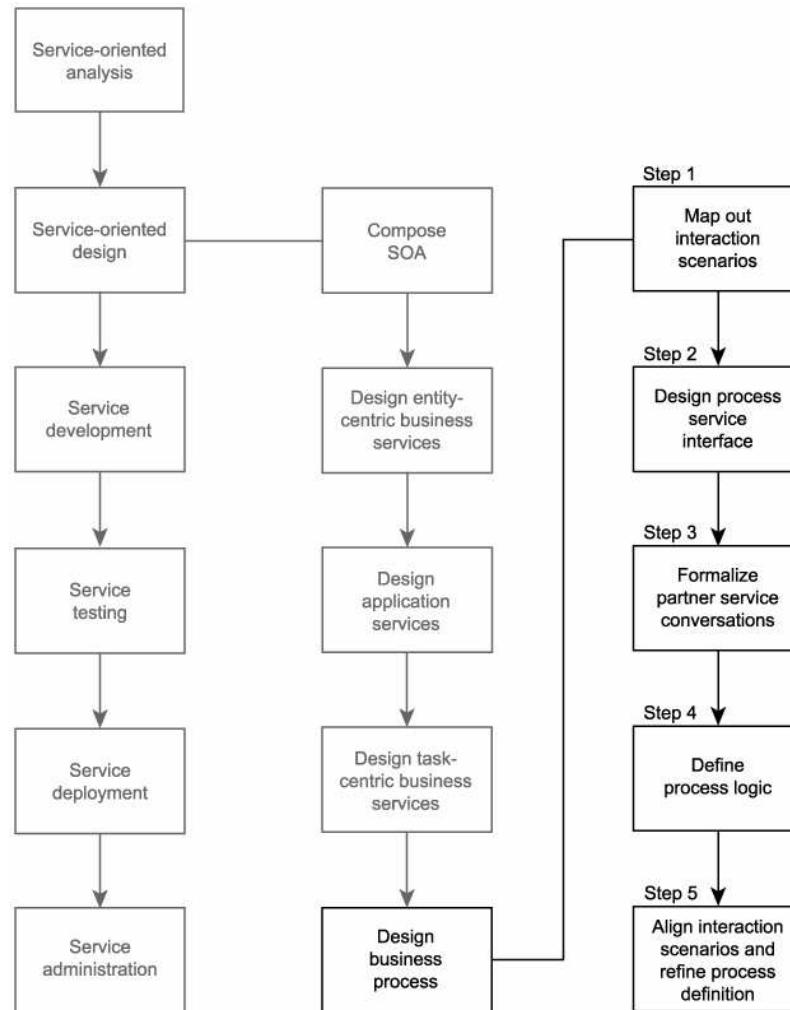
# BPEL-Prozessspezifikation



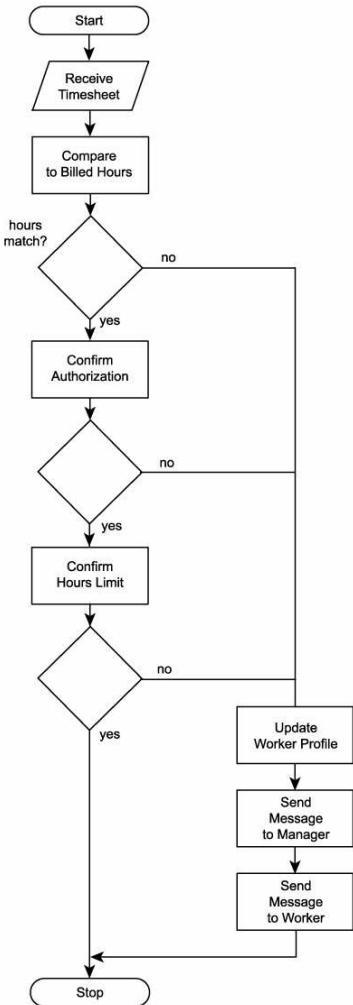
# Erzeugung der Service-Spezifikation aus dem BPEL-Geschäftsprozess-Modell



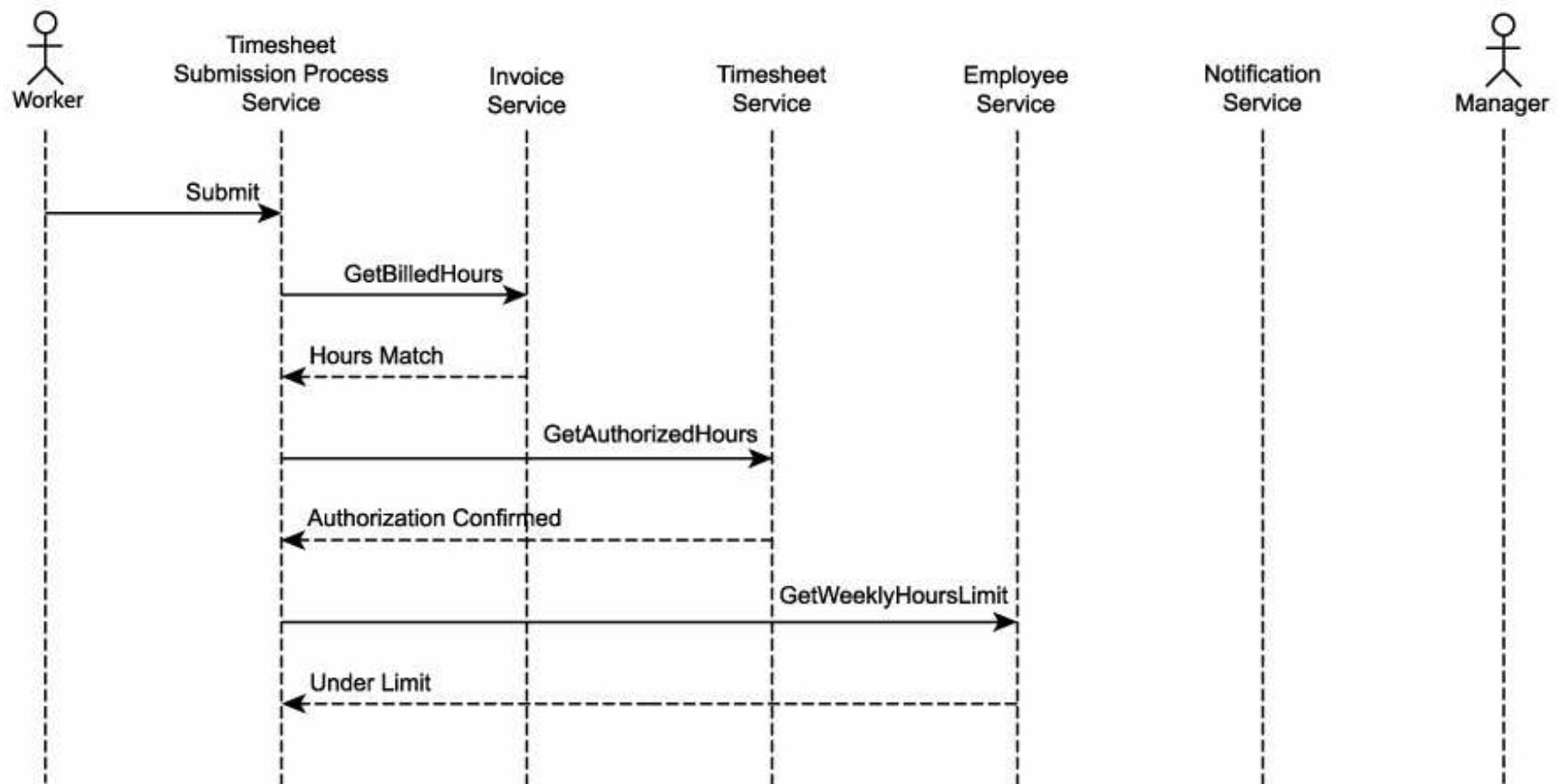
# Designschritte für Geschäftsprozesse



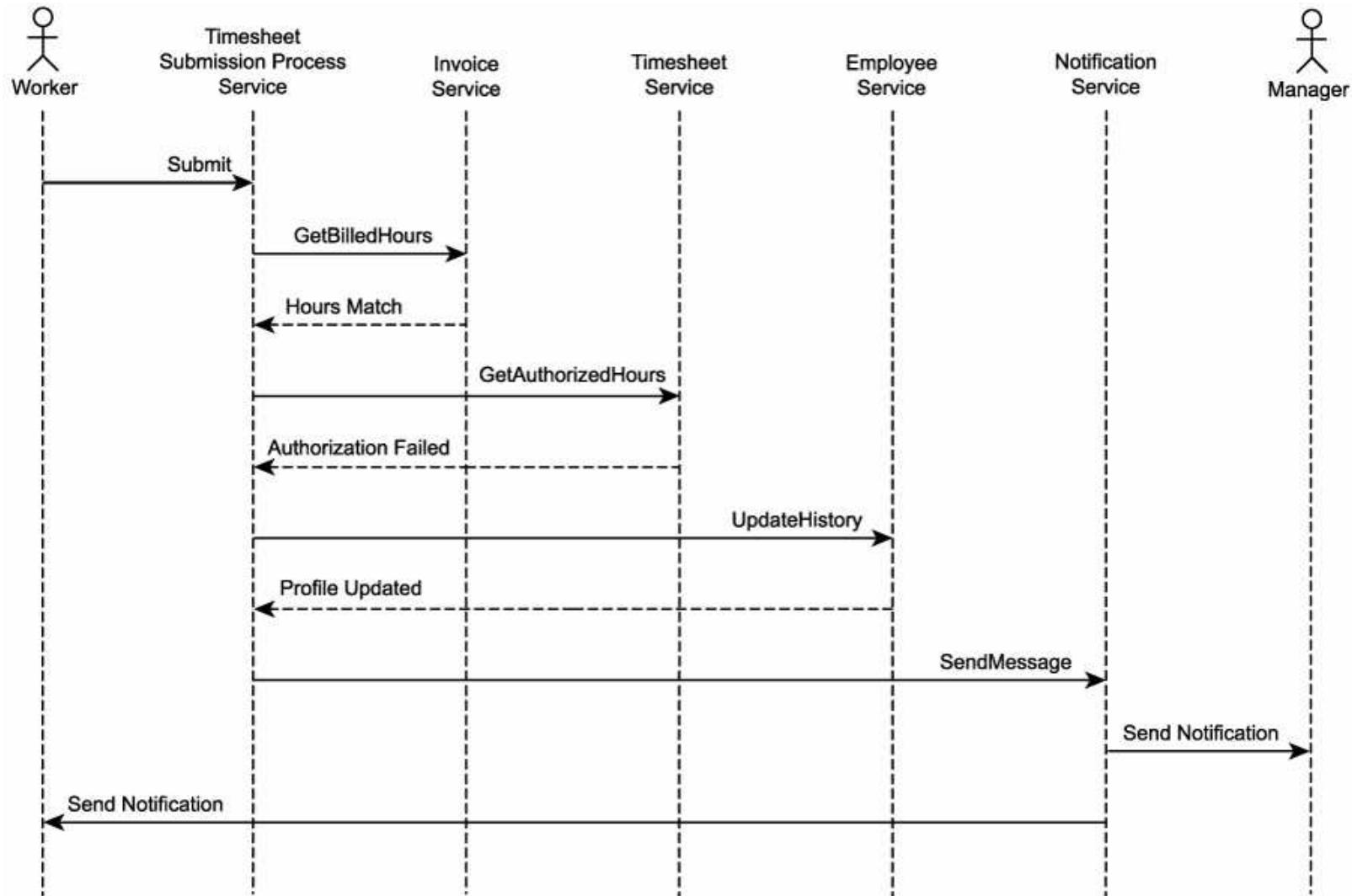
# Fallbeispiel: Zeiterfassungsprozess



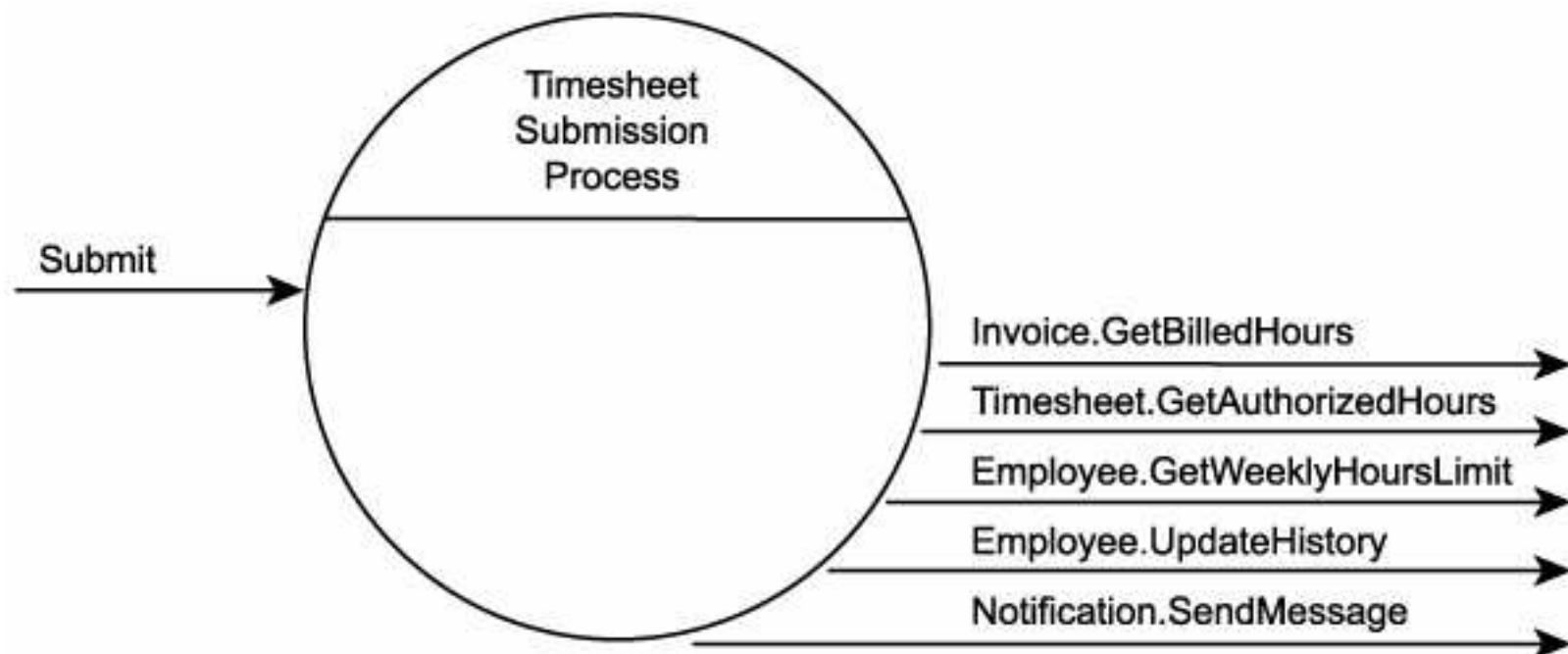
## Interaktionsdiagramm bei erfolgreicher Prüfung



# Interaktionsdiagramm bei nicht erfolgreicher Prüfung



## Geschäftsprozess-Service und Nachrichten



## WSDL: Abstrakte Schnittstelle

```
<definitions name="TimesheetSubmission"
  targetNamespace="http://www.xmltc.com/tls/process/wsdl/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:ts="http://www.xmltc.com/tls/timesheet/schema/"
  xmlns:tsd="http://www.xmltc.com/tls/timesheetservice/schema/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.xmltc.com/tls/timesheet/wsdl/"
  xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
```

```
<types>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.xmltc.com/tls/timesheetsubmissionservice/schema/">
    <xsd:import namespace="http://www.xmltc.com/tls/timesheet/schema/"
      schemaLocation="Timesheet.xsd"/>
    <xsd:element name="Submit">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="ContextID"
            type="xsd:integer"/>
          <xsd:element name="TimesheetDocument"
            type="ts:TimesheetType"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
</types>
```

```
<message name="receiveSubmitMessage">
    <part name="Payload" element="tsd:TimesheetType"/>
</message>

<portType name="TimesheetSubmissionInterface">
    <documentation>
        Initiates the Timesheet Submission Process.
    </documentation>
    <operation name="Submit">
        <input message="tns:receiveSubmitMessage"/>
    </operation>
</portType>

<plnk:partnerLinkType name="TimesheetSubmissionType">
    <plnk:role name="TimesheetSubmissionService">
        <plnk:portType name="tns:TimesheetSubmissionInterface"/>
    </plnk:role>
</plnk:partnerLinkType>

</definitions>
```

## Erweiterung des Employee-Service für die Teilnahme am BPEL-Workflow

```
<definitions
  name="Employee"
  targetNamespace="http://www.xmltc.com/tls/employee/wsdl/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:act=
    "http://www.xmltc.com/tls/employee/schema/accounting/"
  xmlns:hr="http://www.xmltc.com/tls/employee/schema/hr/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.xmltc.com/tls/employee/wsdl/"
  xmlns:plnk=
    "http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
  ...
<plnk:partnerLinkType name="EmployeeType">
  <plnk:role name="EmployeeService">
    <plnk:portType name="tns:EmployeeInterface"/>
  </plnk:role>
</plnk:partnerLinkType>
</definitions>
```

## Erweiterung für alle Workflow-Teilnehmer: BPEL-Partnerliste

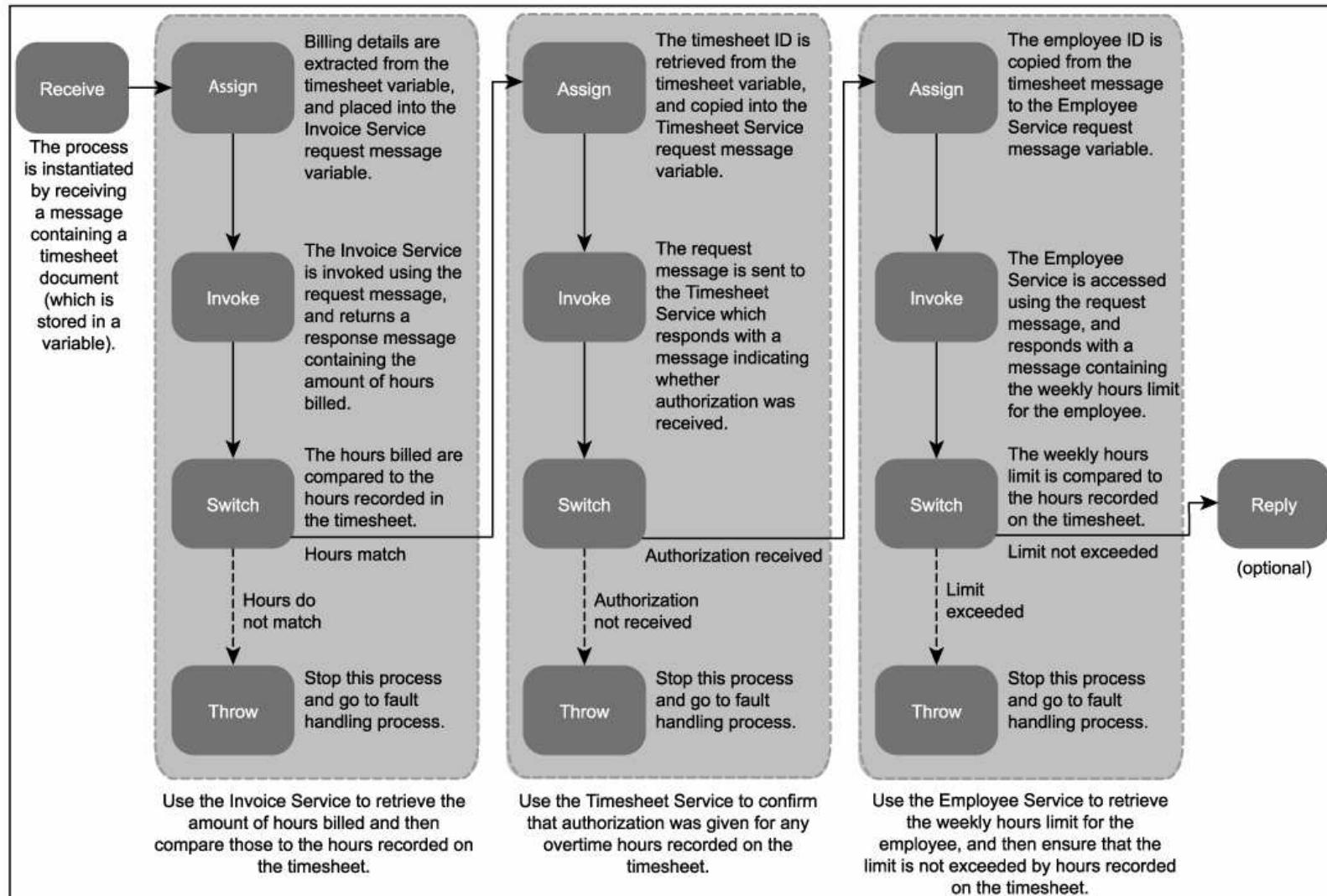
```
<partnerLinks>
  <partnerLink name="client"
    partnerLinkType="bpl:TimesheetSubmissionProcessType"
    myRole="TimesheetSubmissionProcessServiceProvider"/>
  <partnerLink name="Invoice"
    partnerLinkType="inv:InvoiceType"
    partnerRole="InvoiceServiceProvider"/>
  <partnerLink name="Timesheet"
    partnerLinkType="tst:TimesheetType"
    partnerRole="TimesheetServiceProvider"/>
  <partnerLink name="Employee"
    partnerLinkType="emp:EmployeeType"
    partnerRole="EmployeeServiceProvider"/>
  <partnerLink name="Notification"
    partnerLinkType="not:NotificationType"
    partnerRole="NotificationServiceProvider"/>
</partnerLinks>
```

## Variablendefinition für die Input-/Output-Nachrichten aller Nachrichten

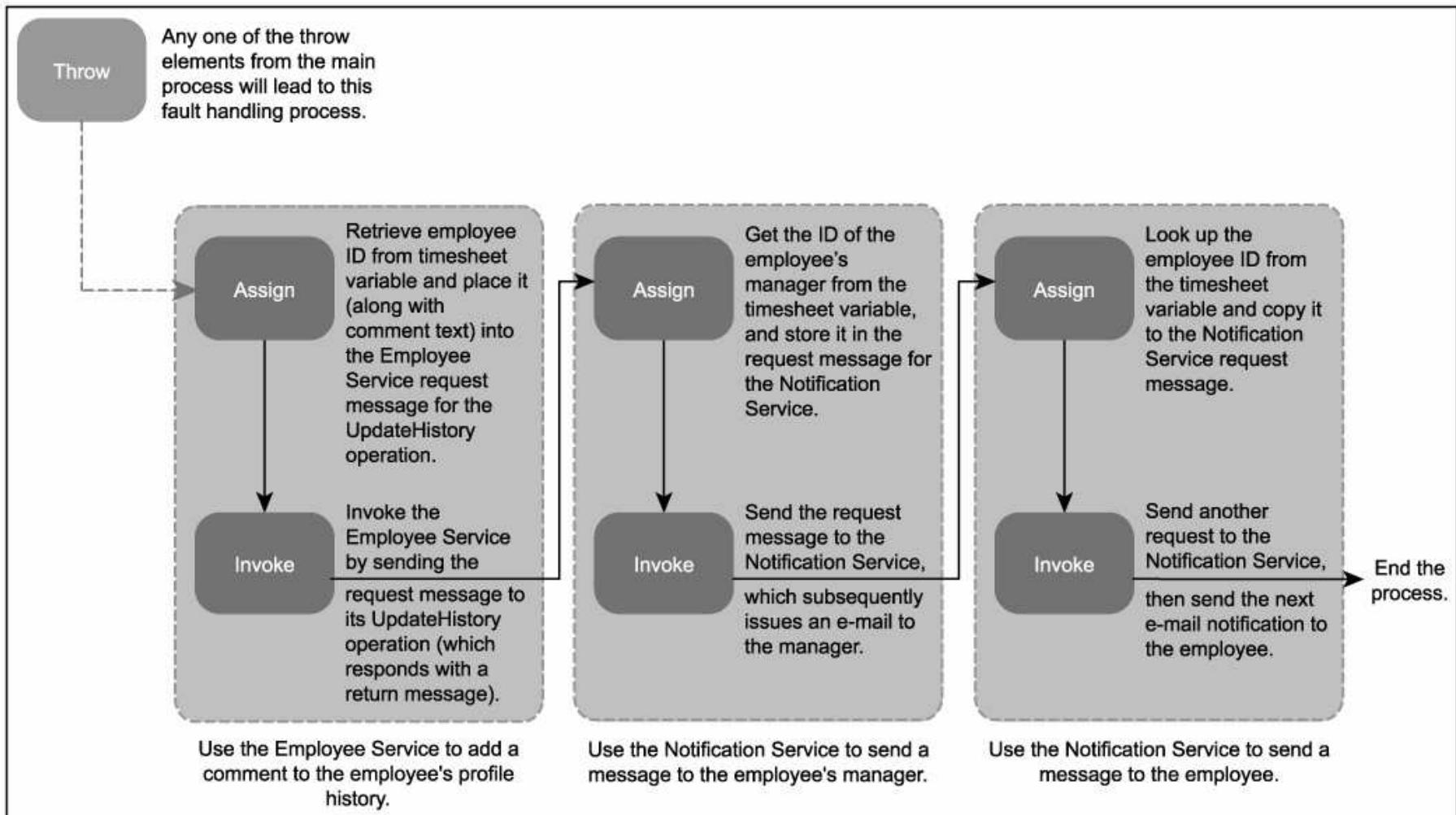
```
<variables>
  <variable name="ClientSubmission"
    messageType="bpl:receiveSubmitMessage"/>
  <variable name="EmployeeHoursRequest"
    messageType="emp:getWeeklyHoursRequestMessage"/>
  <variable name="EmployeeHoursResponse"
    messageType="emp:getWeeklyHoursResponseMessage"/>
  <variable name="EmployeeHistoryRequest"
    messageType="emp:updateHistoryRequestMessage"/>
  <variable name="EmployeeHistoryResponse"
    messageType="emp:updateHistoryResponseMessage"/>
  <variable name="InvoiceHoursRequest"
    messageType="inv:getBilledHoursRequestMessage"/>
  <variable name="InvoiceHoursResponse"
    messageType="inv:getBilledHoursResponseMessage"/>
  <variable name="TimesheetAuthorizationRequest"
    messageType="tst:getAuthorizedHoursRequestMessage"/>
  <variable name="TimesheetAuthorizationResponse"
    messageType="tst:getAuthorizedHoursResponseMessage"/>
```

```
<variable name="NotificationRequest"  
    messageType="not:sendMessage"/>  
</variables>
```

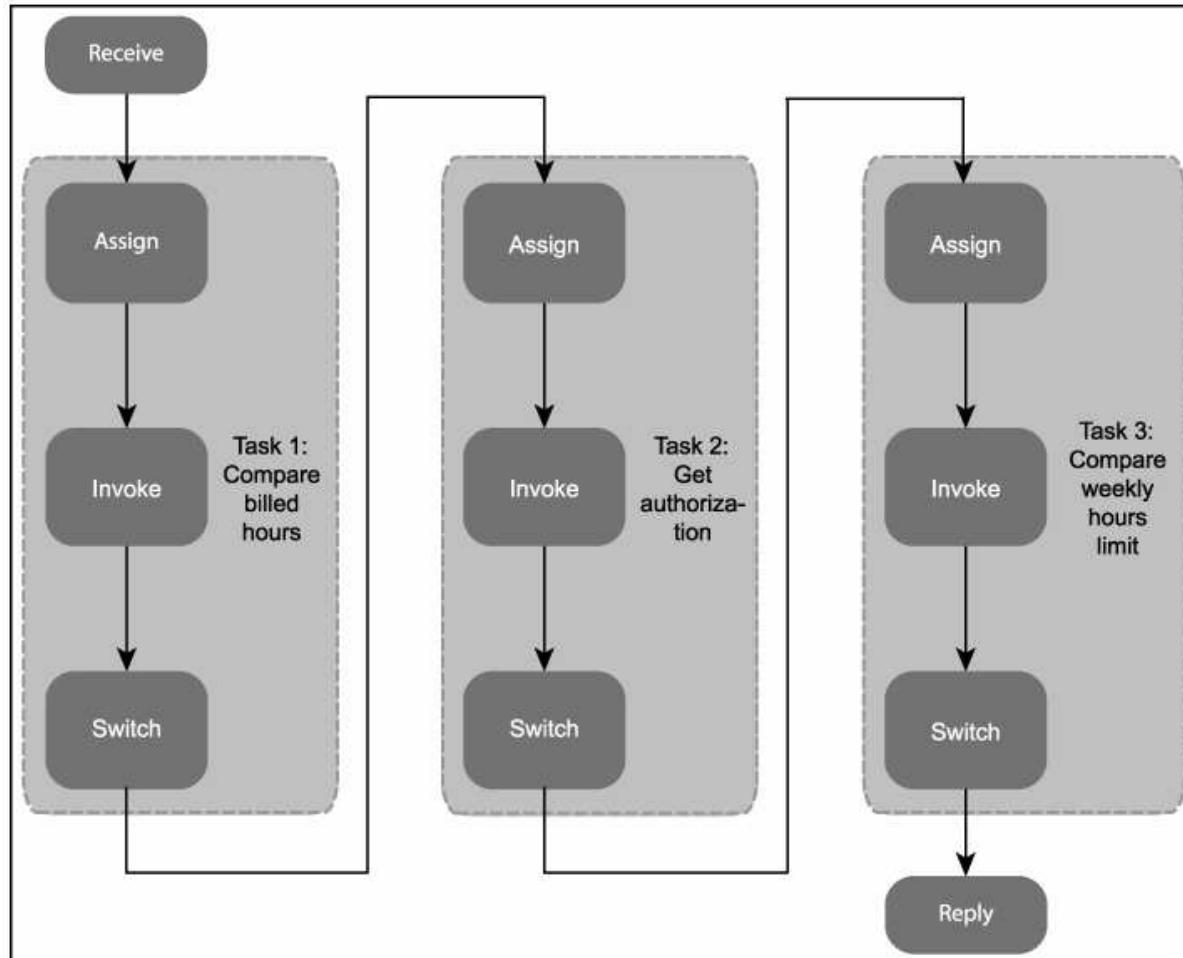
# Workflow



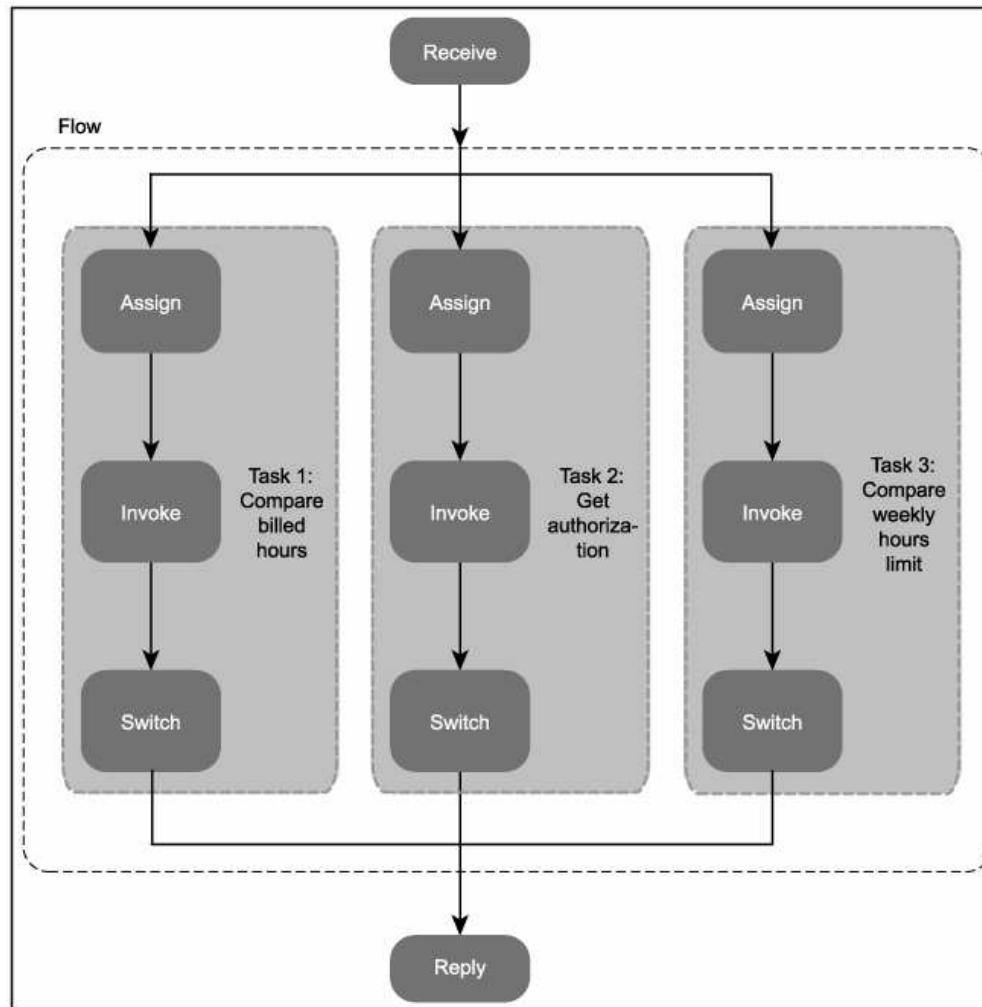
# Fehlerbedingungen



## Sequenz von Teilprozessen



## Flow - Nebenläufige Teilprozesse



## WS-BPEL-Prozessspezifikation

```
<receive xmlns=
  "http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  name="receiveInput"
  partnerLink="client"
  portType="tns:TimesheetSubmissionInterface"
  operation="Submit"
  variable="ClientSubmission"
  createInstance="yes"/>

<assign name="GetInvoiceID">
  <copy>
    <from variable="ClientSubmission" part="payload"
      query="/TimesheetType/BillingInfo"/>
    <to variable="InvoiceHoursRequest"
      part="RequestParameter"/>
  </copy>
</assign>
```

```

<invoke name="ValidateInvoiceHours"
    partnerLink="Invoice"
    operation="GetBilledHours"
    inputVariable="InvoiceHoursRequest"
    outputVariable="InvoiceHoursResponse"
    portType="inv:InvoiceInterface"/>

<switch name="BilledHoursMatch">
    <case condition=
        "getVariableData('InvoiceHoursResponse',
                      'ResponseParameter') != 
        getVariableData('input','payload',
                      '/tns:TimesheetType/Hours/...')">
        <throw name="ValidationFailed"
            faultName="ValidateInvoiceHoursFailed"/>
    </case>
</switch>

<assign name="SetEmployeeMessage">
    <copy>
        <from variable="ClientSubmission" .../>
        <to variable="EmployeeHistoryRequest" .../>

```

```
</copy>
<copy>
  <from expression="..."/>
  <to variable="EmployeeHistoryRequest" .../>
</copy>
</assign>

<invoke name="updateHistory"
  partnerLink="Employee"
  portType="emp:EmployeeInterface"
  operation="updateHistory"
  inputVariable="EmployeeHistoryRequest"
  outputVariable="EmployeeHistoryResponse"/>

<assign name="GetManagerID">
  <copy>
    <from expression="getVariableData(...)" />
    <to variable="NotificationRequest" .../>
  </copy>
</assign>
```

```
<invoke name="SendNotification"
    partnerLink="Notification"
    portType="not:NotificationInterface"
    operation="SendMessage"
    inputVariable="NotificationRequest"/>

<assign name="GetEmployeeID">
    <copy>
        <from expression="getVariableData(...)" />
        <to variable="NotificationRequest" . . . />
    </copy>
</assign>

<invoke name="SendNotification"
    partnerLink="Notification"
    portType="not:NotificationInterface"
    operation="SendMessage"
    inputVariable="NotificationRequest"/>

<terminate name="EndTimesheetSubmissionProcess"/>
```

## **Open Source Java-Toolkits für die Entwicklung von Webservices**

- JAX-WS / Metro
  - Referenzimplementierung von Sun Microsystems
  - Kern des „Metro“-Stack
  - Nachfolger von JAX-RPC
  - basiert auf JAXB (Java API für XML)
- Apache Axis 2
  - Nachfolger von Axis 1
  - JAX-WS kompatibel
  - JEE-kompatibel
  - Spring-kompatibel
- Apache CXF
  - basiert auf Spring Framework
  - JAX-WS kompatibel
- Spring WS
  - unterstützt Erzeugung von Java und WSDL aus XML-Schema

## Herausforderungen für Java-WS-Toolkits

- Performance: Einsatz von StAX-XML-Parsern (anstelle von DOM/SAX)
- Unterstützung komplexer Nachrichtenaustauschmuster
- Annotations-basierte Deployment-/Runtime-Konfiguration (JAX-WS)
- Unterstützung aktueller WS-\*-Erweiterungen
- Unterstützung für „WSDL-First“-Entwicklungsmuster

## Unterstützung von WS-\*

	<b>Axis2</b>	<b>CXF</b>	<b>Metro</b>
WS-Addressing	X	X	X
WS-Coordination	X <sup>(2)</sup>		X
WS-MetadataExchange			X
WS-Policy	X	X	X
WS-ReliableMessaging	X <sup>(3)</sup>	X	X
Web Services Security	X <sup>(1)</sup>	X <sup>(4)</sup>	X
WS-SecureConversation	X <sup>(1)</sup>		X
WS-SecurityPolicy			X
WS-Transaction	X <sup>(2)</sup>		X
WS-Trust	X		X
WS-Federation			

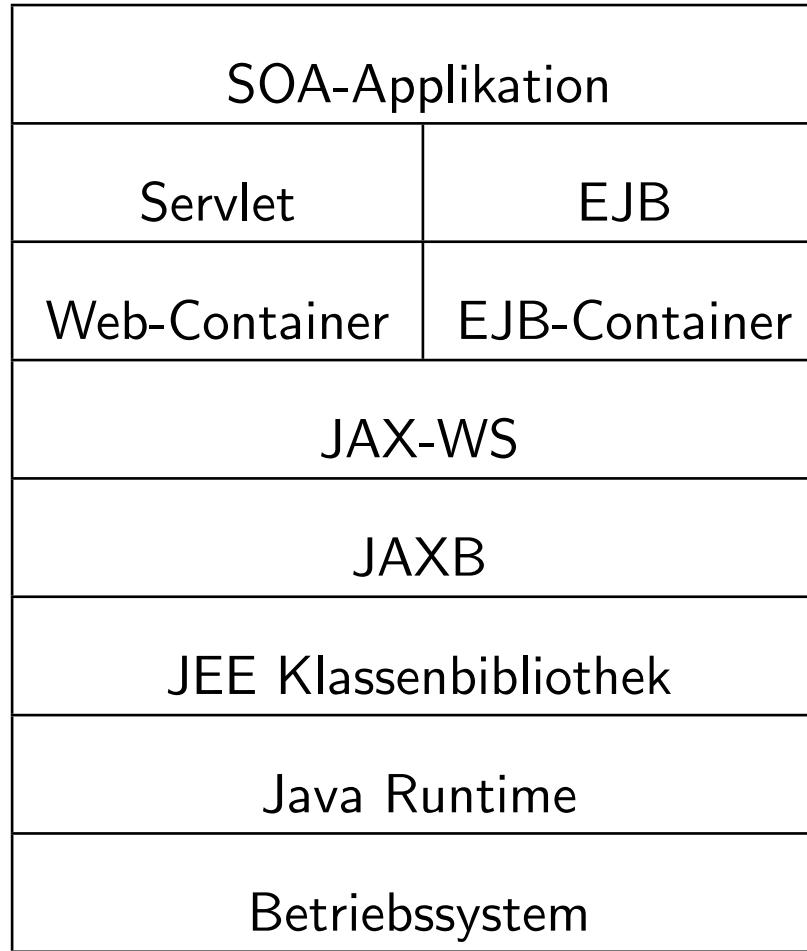
(1) Zusatzmodul Apache Rampart

(2) Zusatzmodul Apache Kandula2

(3) Zusatzmodul Apache Sandesha2

(4) Zusatzmodul Apache WSS4J Interceptor

## Schichtenmodell der Referenzimplementierung



## Aufgaben einer WS-Plattform

- Aufrufmechanismus
  - Serverseite: eingehende SOAP-Nachricht → Methodenaufruf, Antwort schicken
  - Clientseite: Methodenaufruf → SOAP-Nachricht schicken, Antwort lesen
- Serialisierungssystem
  - Serialisierung: Java-Objekte → SOAP-Nachrichten
  - Deserialisierung: SOAP-Nachrichten → Java-Objekte
- Deployment-Mechanismus
  - Java Service-Repräsentanten (SR) installieren (EJBs, Servlets)
  - Jeder WSDL-Operation einen Java SR zuordnen
  - Serialisierung konfigurieren (Serialisierung-Kontext festlegen)
  - WSDL-Dokument verfügbar machen
  - SOAP-Handler konfigurieren (Prä-/Postprozessoren für WS-\*SOAP-Header)
  - Endpunkt-“Listener“ konfigurieren

## Aufrufmechanismus Server

- SOAP-Nachricht vom Transportdienst entgegennehmen
- „SOAP-Handler“ zur Vorbehandlung von SOAP-Headern aktivieren
- Dispatching: zuständige WSDL-Operation und Java-Zielklasse bestimmen
- Nutzlast der SOAP-Nachricht an das Serialisierungssystem zum Deserialisieren übergeben
- Java SR-Methode aufrufen, Resultat entgegennehmen
- Resultat zum Serialisieren an Serialisierungssystem übergeben
- SOAP-Antwortnachricht erzeugen
- Antwortnachricht an Transportdienst übergeben

## Aufrufmechanismus Client

- „Service Endpoint Interface“ (SEI) erzeugen  
(repräsentiert entfernten Dienst)
- Parameter serialisieren (XML erzeugen)
- SOAP-Nachricht erzeugen
- SOAP-Handler zur Nachbehandlung aufrufen (WS-\* SOAP-Header generieren)
- Nachricht an Transportdienst übergeben
- Antwortnachricht vom Transportdienst entgegennehmen
- . . .

## Serialisierungssystem

- Ein XML-Dokument referenziert meist mehrere XML-Schemata
- (De-)Serialisierung erfordert Laden aller benötigten Schema-Dokumente
- XML-Schema-Typen müssen auf Java-Klassen abgebildet werden und umgekehrt
- Problem:
  - Java- und XML-Schema-Typsysteme sind komplex:
  - eindeutige „natürliche“ Zuordnung existiert meist nicht
- Lösungsansatz: Konfigurierbare „Mapping-Strategie“ für die Typzuordnung
- Mapping-Strategie ist WS-Plattform-spezifisch, z.B.
  - Definition einer Standard-Abbildung (JAXB)
  - Standard-Abbildung parameterisierbar durch Annotationen (JAXB)
  - Benutzer-definierte Serialisierer/Deserialisierer (z.B. JAX-RPC, Axis)
  - Regelbasierte Definition der Mapping-Strategie (CASTOR, SOA-J)

## Entwicklungsmodelle

### 1. „Start from Java“

- Entwickler erzeugt WSDL und Schema-Dokumente aus Java-Klasse
- gut unterstützt durch Toolkits
- Tool: Schemagenerator

### 2. „Start from WSDL“

- Entwickler erzeugt Java-Interfaces (SEIs) aus WSDL und Schema-Dokumenten
- Implementierung der Interfaces
- Tool: Schemacompiler

### 3. „Start from Java and WSDL“

- Typisch für EAI-Szenarien
- Entwickler muss existierende WSDL-Operationen in Java implementieren
- Es gibt eine Java-Applikation, die die Funktionalität der Service-Operationen im Prinzip schon enthält
- Lösungsmöglichkeiten:
  - Mapping so konfigurieren, dass Java-Klassen und WSDL zueinander passen (Oft schwierig bzw. unmöglich!)
  - Java-SEIs erzeugen und durch Adapterklasse implementieren
- Softwaretechnisch einfach:
  - WSDL-Design und entfällt
  - Java Klassen-Design entfällt ebenfalls

## JAXB 2.0

- Sprach-Mapping zwischen XML-Schema und Java
- komplex (Standard ca. 370 Seiten)
- nicht WS-spezifisch
- JAXB als Alternative zu JAXP  
(JAXP = Java API for XML Processing: DOM, SAX, XPath usw. ):  
„High level“-API zur Verarbeitung von XML-Dokumenten:
  - Java-Klasse repräsentiert XML-Schema
  - Java-Objekt repräsentiert XML-Dokument
  - Entwickler kommt mit XML-Syntax und -Typsystem nicht mehr in Berührung

## Typbindungen in JAXB

- Eine Typbindung ist eine eindeutige Zuordnung zwischen einem XML-Schema und einer Java-Klasse („value class“)
- Jede Typbindung wird implementiert durch
  - Serialisierer und
  - Deserialisierer

## Die Bindung Java→XML

- Jeder Java-Typ hat eine Standard-Bindung an einen XML-Schema-Typ
- Bindung konfigurierbar
  - durch Java-Annotationen
  - durch benutzerdefinierte Bindungsklassen
- Basis für Generierung der Schema-Dokumente, die im WSDL benötigt werden
- Basis für Generierung des Serialisierers

## Die Bindung XML→Java

- Jeder XML-Schema-Typ hat eine Standard-Bindung an eine Java-Klasse
- Bindung konfigurierbar durch XML-Annotationen:  
JAXB „Binding Language“
- Basis für Generierung der Java-Value-Klassen:
  - pro Element/Attribut eine Bean-Property mit „get-“ und „set-“ Methode
  - Generierung von Bindungs-Annotationen im Java-Code
- Basis für Generierung des Deserialisierers
- Die „Binding Language“ erlaubt DOM-Repräsentanten anstelle von Java-Value-Klassen
  - beliebige Deserialisierer auf DOM-Basis implementierbar

## JAXB im Überblick

- Standardbindung Java → XML und XML → Java
- Java-Annotationen zur Konfiguration der Schema-Erzeugung
- XML-Bindungssprache zur Konfiguration der Value-Klassen-Erzeugung
- Schema-Compiler
- Schema-Generator
- MTOM/SAAJ-Unterstützung (binär codierte Daten)

MTOM = SOAP Message Transmission Optimization Mechanism (W3C-Standard)

SAAJ = SOAP with Attachments API for Java

- Konfigurierbare Validierung mit Fehlertoleranz
- Partielle Bindungen (z.B. nur für Nutzlast, nicht aber für SOAP-Header)