Vorlesung am 12. Mai 2009

Serviceorientiertes eGovernment

Grundlagen verteilter Systeme

Dr. Frank Sarre
Lehrbeauftragter der LMU München

Was ist ein verteiltes System?

→ Softwaresystem, das aus mehreren Teilsystemen besteht, die mittels prozess- oder rechnerübergreifender Kommunikation zur Erfüllung einer Gesamtfunktionalität beitragen.

Ein verteiltes System kann auch auf einer einzelnen physischen Maschine laufen.

Beispiele:

- SAP R/3
- Jede Internet-Applikation, z.B. eBay
- Datenbanksysteme (z.B. Oracle 11g)
- Lufthansa-Flugbuchungssystem
- .....

Gründe für die Verteilung

Die Verteilung der „Einzelteile“ eines Softwaresystems kann verschiedene Gründe haben:

- Mehrere Clients nutzen eine **gemeinsame, zentrale Logik**
- Die Anwendung benötigt Funktionalitäten von **unterschiedlichen, bereits existierenden Systemen** (z.B. Nutzung von Web Services)
- Einzelne Komponenten sollen leicht **austauschbar** sein
- Hohe Anforderungen an die **Performance** erzwingen die Verwendung mehrerer Rechner (**Lastverteilung**)
- Hohe **Zuverlässigkeitsanforderungen** erfordern mehrfaches Vorhandensein von (evtl. sogar unterschiedlich implementierten) Systemen.

Kopplung von Systemen

Beteiligte in verteilten Systemen

- **Anwendungssysteme** mit speziellen Schnittstellen
- **Standardservices**, z.B. Webserver
- **Anwendungsservices**, z.B. Payment-Services

Kopplung von Systemen

- Weisen miteinander kommunizierende Systeme starke technische und zeitliche Abhängigkeiten auf, spricht man von „**enger Kopplung**“
- Sind die technischen und zeitlichen Abhängigkeiten der Systeme gering, spricht man von „**loser Kopplung**“

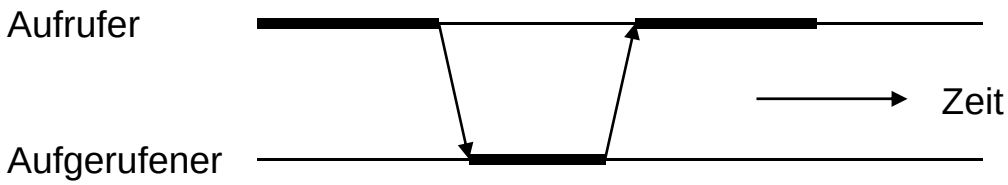
Enge Kopplung versus lose Kopplung

	Enge Kopplung	Lose Kopplung
Physische Verbindung	Punkt-zu-Punkt	Über Vermittler
Kommunikationsstil	synchron	asynchron
Datenmodell	komplexe gemeinsame Typen	nur einfache gemeinsame Typen
Typsystem	streng	schwach
Bindung	statisch	dynamisch
Plattformspezifika	stark / viel	schwach / wenig
Interaktionsmuster	über komplexe Objektbäume navigieren	datenzentrierte autonome Nachrichten
Transaktionssicherheit	2PC (Two-Phase-Commit)	Kompensation
Kontrolle fachlicher Logik	zentrale Kontrolle	verteilte Kontrolle
Deployment	gleichzeitig	zu verschiedenen Zeitpunkten
Versionierung	explizite Upgrades	implizite Upgrades

Quelle: Josuttis, SOA in der Praxis, S.48

Synchrone Kommunikation

- Bei der synchronen Kommunikation zwischen zwei Systemen stellt der Aufrufer eine Anfrage und **wartet auf die Antwort**. Er ist dabei so lange **blockiert**, bis die Antwort des Kommunikationspartners eintrifft.



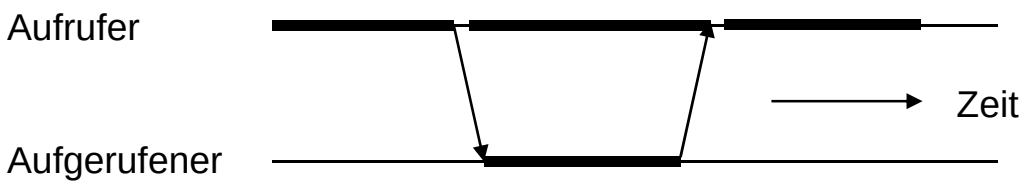
- **Vorteile / Nachteile**
  - Der Aufrufer kann einfach gestaltet werden, da das Ergebnis wieder direkt zurück geliefert wird.
  - Die Nachricht kann nicht unbemerkt verloren gehen.
  - Der Kommunikationspartner muss verfügbar sein und in endlicher Zeit antworten.
  - Wenn ein Server mehrere Kommunikationspartner gleichzeitig bedienen soll, muss der Server ausreichend schnell „dispatchen“ können.

Anwendungsbeispiel: Ein einfacher Java-Time-Server

Asynchrone Kommunikation

- Bei der asynchronen Kommunikation stellt der Aufrufer eine Anfrage und **fährt mit seiner Verarbeitung fort**, ohne auf die Antwort zu warten.

  Wenn eine Antwort zurück kommt, muss der Aufrufer im Regelfall seine Arbeit unterbrechen, um die Antwort zu verarbeiten.



- **Vorteile / Nachteile**
  - Der Aufrufer kann mit seiner Arbeit fortfahren, ohne auf den Kommunikationspartner warten zu müssen.
  - Wird ein Ergebnis erwartet, benötigt der Aufrufer einen Event-Mechanismus, der feststellt, ob und wann eine Antwort eintrifft.
  - Die Antwort muss der entsprechenden Anfrage zugeordnet werden.
  - Anfragen / Antworten könnten verloren gehen

Anwendungsbeispiel: Initiierung von Jobs unbekannter Abarbeitungszeit

Inkompatible Datentypen

- Idealerweise verwenden miteinander kommunizierende Systeme die **gleichen Datentypen**, praktisch tun sie das aber nur selten.
- Alle Systeme müssen die gleichen **fundamentalen** Datentypen verwenden, z.B. *String*, *Integer*, *etc*.
- **Komplexe Datentypen**, z.B. „*Kunde*“ müssen nicht in allen Systemen identisch sein!

  → Mapping von Daten ist praktisch immer erforderlich

Vermittler

Ohne Vermittler

Es muss eine Punkt-zu-Punkt-Verbindung genutzt werden, bei der der Aufrufer die genaue physische Adresse des Aufgerufenen kennt.

→ Enge Kopplung

Mit Vermittler

Zwei alternative Vermittlungsmethoden

1. Der Aufrufer fragt anhand eines symbolischen Namens bei einem **Name Server** oder **Broker** nach, wo er den Empfänger der Nachricht finden kann (z.B. DNS). Der echte Aufruf erfolgt dann wieder „**Punkt-zu-Punkt“** an die zurückgelieferte Adresse.
2. Der Aufrufer übergibt die Nachricht mit der symbolischen Adresse des Kommunikationspartners an eine **Kommunikationsinfrastruktur** (Netzwerk, Middleware, etc.), die die symbolische Adresse in eine physische Adresse umwandelt.

Transaktionen

- In einem verteilten System kann es erforderlich sein, dass mehrere Aktionen auf verschiedenen Systeme **entweder alle zusammen gültig werden oder keine davon**.
- Übliche Verfahren in verteilten Systemen sind **2PC** (Two-Phase-Commit) und **Kompensation**.
- Grundlegende Anforderung ist das ACID-Prinzip (**a**tomicity, **c**onsistency, **i**solation und **d**urability)

Das „ACID-Prinzip“

Atomicity:

Die Teilschritte einer Transaktion werden ganz oder gar nicht ausgeführt („alles oder nichts“).

Consistency:

Eine Transaktion führt von einem konsistenten Zustand in einen anderen.

Isolation:

Typischerweise muss ein „Isolation Level“ angegeben werden, der angibt, welche Daten wann für andere Transaktionen sichtbar sind.

Durability:

Sobald eine Transaktion beendet ist, ist garantiert, dass auch beim Auftreten von Fehlern (System, Hardware) das Ergebnis persistent (erhalten) bleibt.

2PC (Two-Phase-Commit) [Folie 1]

1. Für die Durchführung der Transaktion erhalten alle benötigten Ressourcen zunächst ein „prepare to commit“ von einem sog. „Transaction Manager“ (TM)

2. Alle beteiligten Systeme bereiten dann die jeweilige lokale Transaktion vor und **blockieren** die Ressource.

3. Die Beteiligten antworten dem TM mit „agreed“, wenn die Vorbereitung der Transaktion gelungen ist oder mit „abort“, wenn Fehler aufgetreten sind.

4. Sind alle „agreed“, sendet der TM „commit“ an die Ressourcen für die Festschreibung der Transaktion.

5. Tritt in der query-Phase ein einziges „abort“ auf, sendet der TM ein „rollback“ an alle Ressourcen.

2PC (Two-Phase-Commit) [Folie 2]

Vorteile

- Standardisierter Automatismus
- ACID wird gewährleistet

Nachteile

- Synchrones Protokoll

  → Enge Kopplung

- Die Transaktion kann erst durchgeführt werden, wenn alle Ressourcen verfügbar sind.

  → Es kann unter Umständen zu Deadlocks kommen

- Evtl. unterstützen nicht alle beteiligten Systeme dieses Verfahren

  → 2PC kann nicht immer eingesetzt werden

Kompensation

Beispiel

- Die erste Einzeltransaktion gelingt, z.B. eine Flugbuchung
- Die zweite Einzeltransaktion schlägt fehl, z.B. die Hotelbuchung
  → Die erste Transaktion muss wieder rückgängig gemacht werden

Vorgehensweise

- Die Transaktion wird nicht durch einen Rollback rückgängig gemacht; stattdessen wird eine **inverse Transaktion** durchgeführt, die den Vorgang inhaltlich wieder korrigiert.

Vorteile

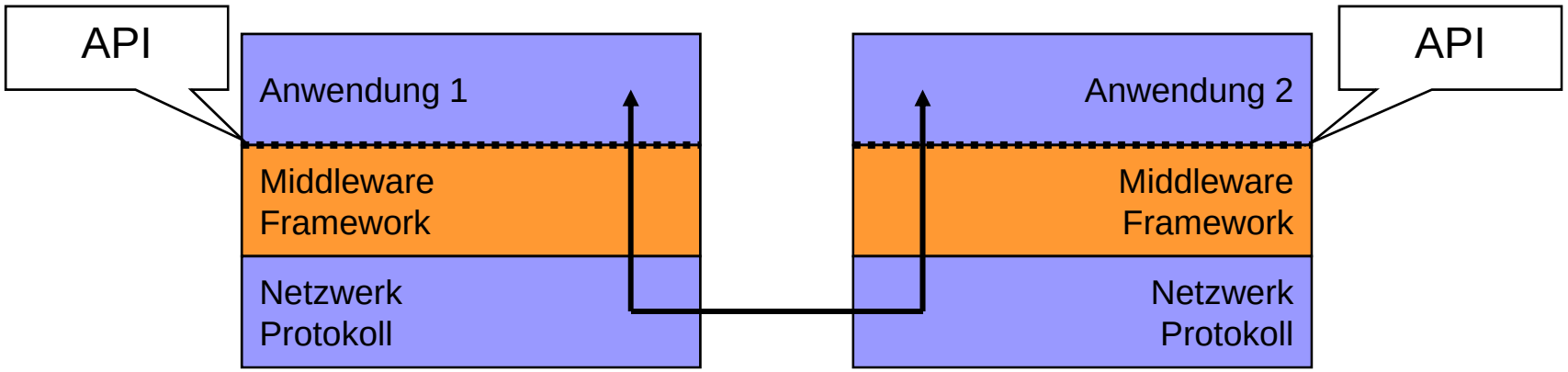
- Es müssen nicht alle beteiligten Systeme synchron arbeiten.
  → Lose Kopplung möglich / typisch

Nachteile

- Die inverse Transaktion muss **explizit implementiert** werden.
- Der Ablauf muss in der Anwendung ausprogrammiert werden.
- Es werden zusätzliche Daten erzeugt.

Middleware

Ein Middleware-Framework bietet eine Umgebung, die es Anwendungen ermöglicht, Verbindungen aufzubauen und Daten auszutauschen.



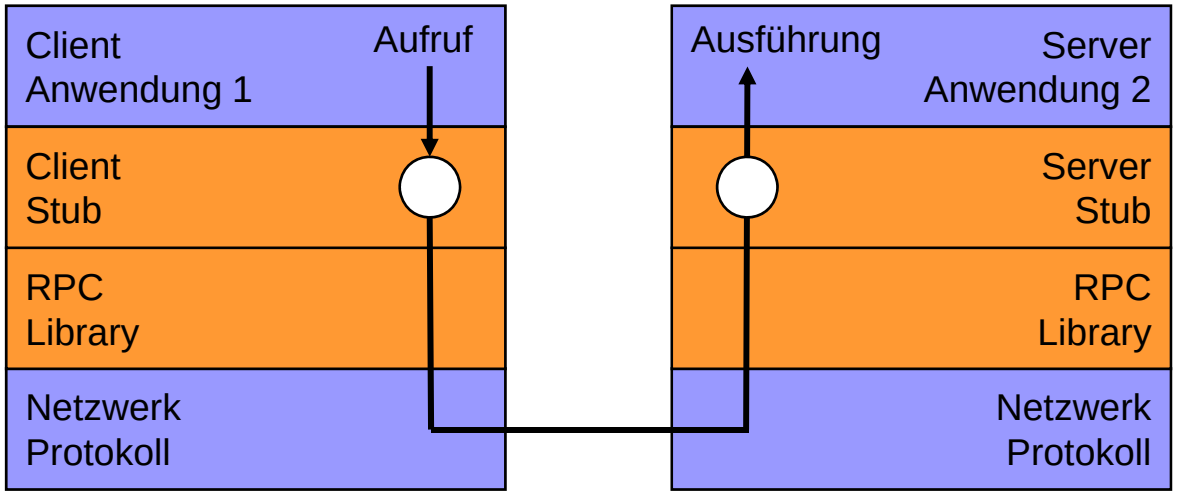
Die Middleware verbirgt technische Details:

- Betriebssystem
- Netzwerkprotokolle
- Details des Verschlüsselungsverfahrens
- …

Remote Procedure Call (RPC)

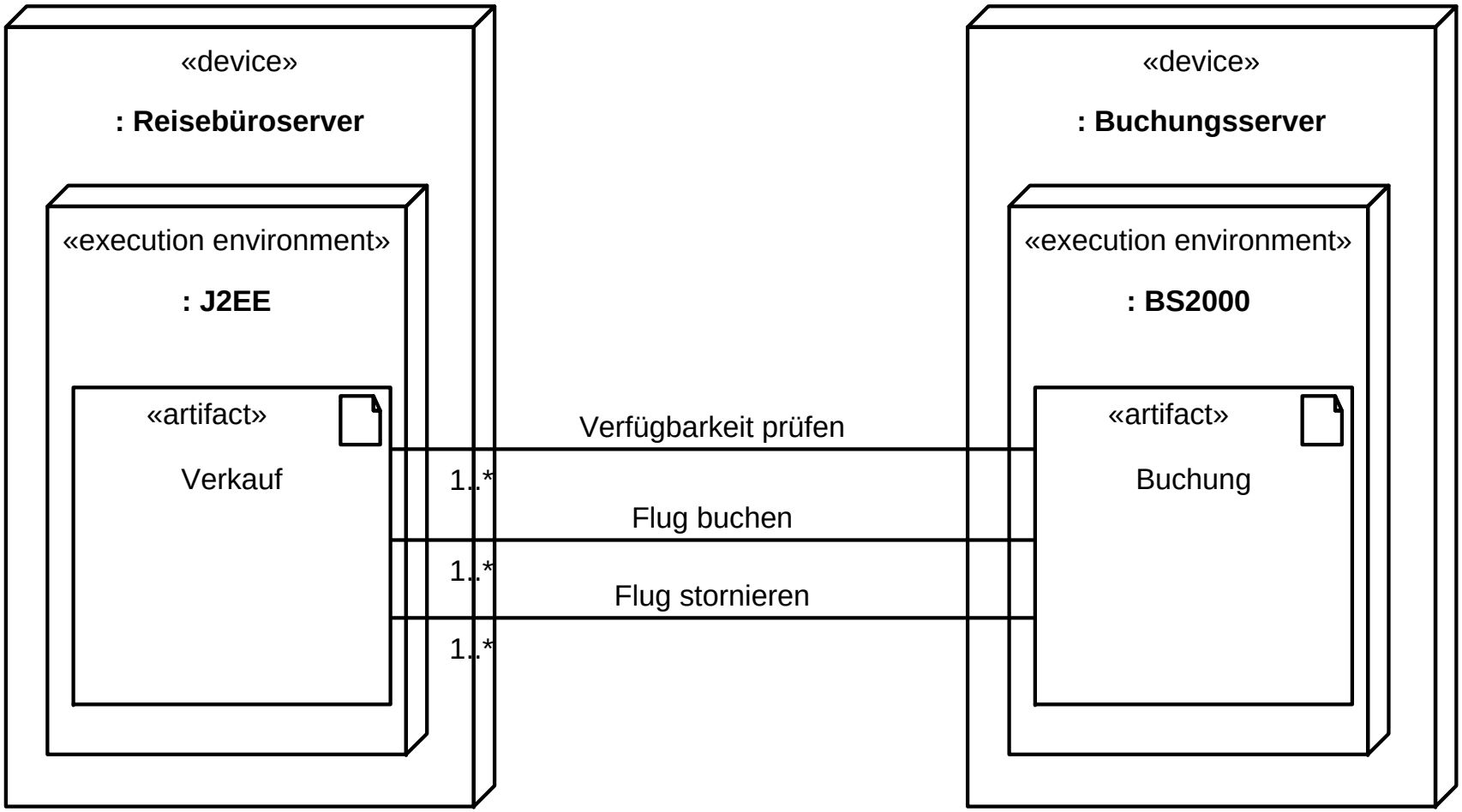
- Auf vielen **Plattformen** verfügbar
- Ursprünglich von der Fa. SUN entwickelt (SUN-RPC)
- Geringe Unterstützung für die Übertragung von **Parametern**
- Verschiedene Varianten von RPCs, z.B. DCE-RPC
- **Kein objekt-orientierter Ansatz**
- Grundsätzlich **synchrone** Kommunikation,
  bei einigen Varianten aber auch asynchrone Kommunikation möglich.

RPC-Ablaufschema

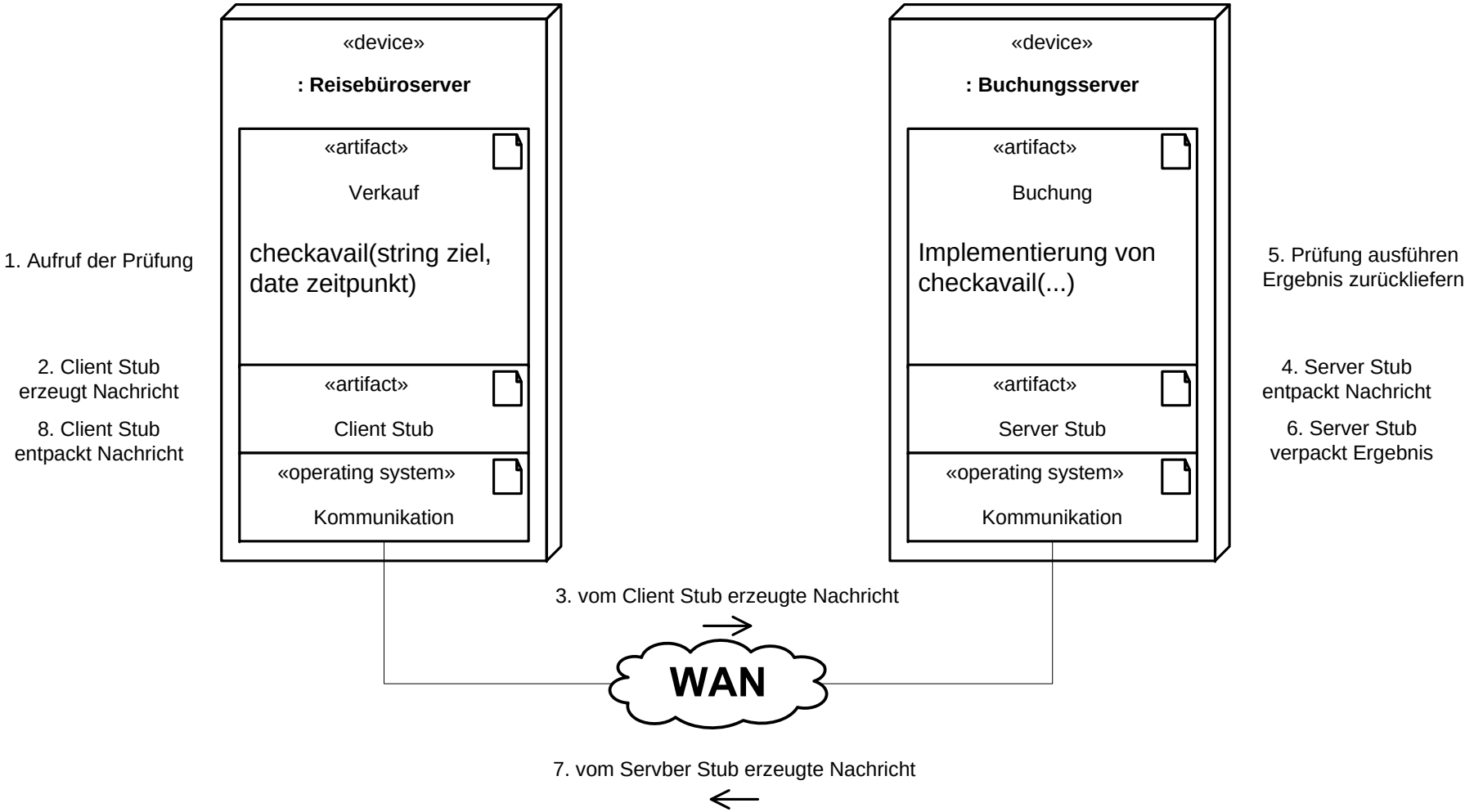


- Funktionsaufruf sieht für den Client wie ein lokaler Aufruf aus
- Der Aufruf wird zum Server geroutet, dort ausgeführt,
  und dann wird das Ergebnis zurück geliefert.
- Die Stubs kapseln die technischen Eigenschaften der Übertragung gegenüber den Anwendungen.
- Verwendung eines Request/Reply-Schemas:
  Der Aufrufer wird so lange blockiert, bis das Ergebnis vorliegt

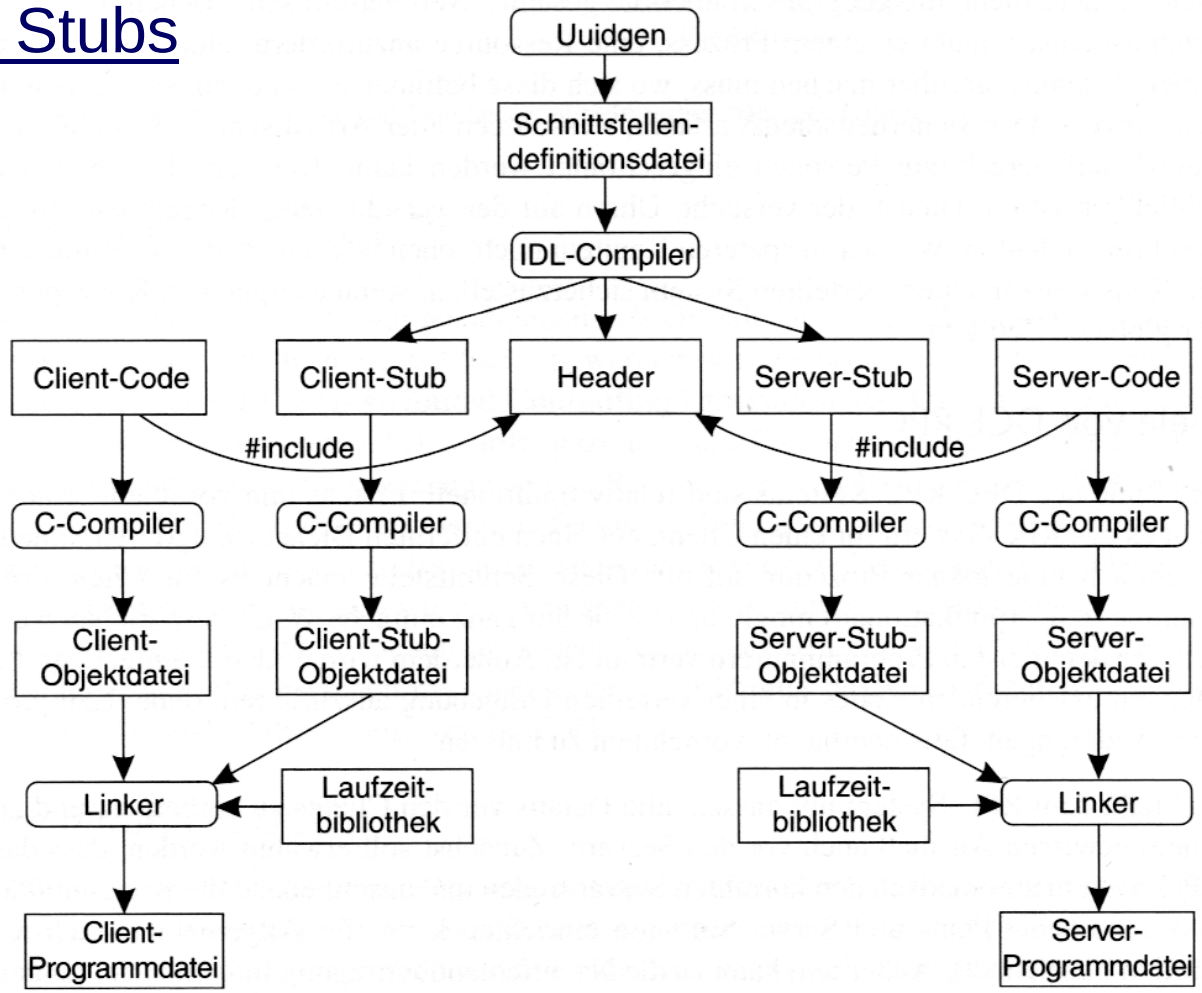
Beispiel: Flugbuchung



Beispiel: Flugbuchung mit RPC



Erzeugung der Stubs



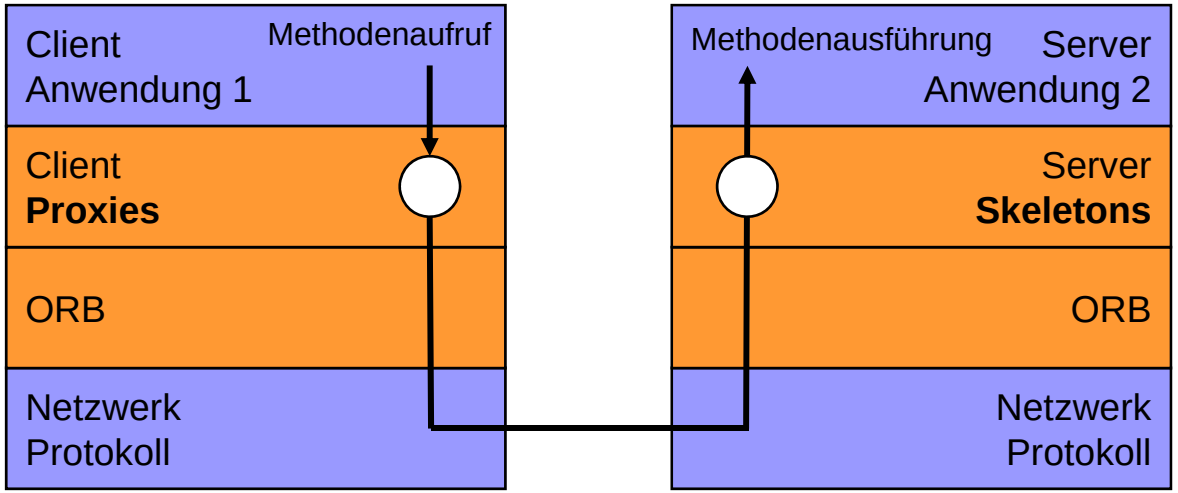
[Grafik: Tannenbaum, Verteilte Systeme]

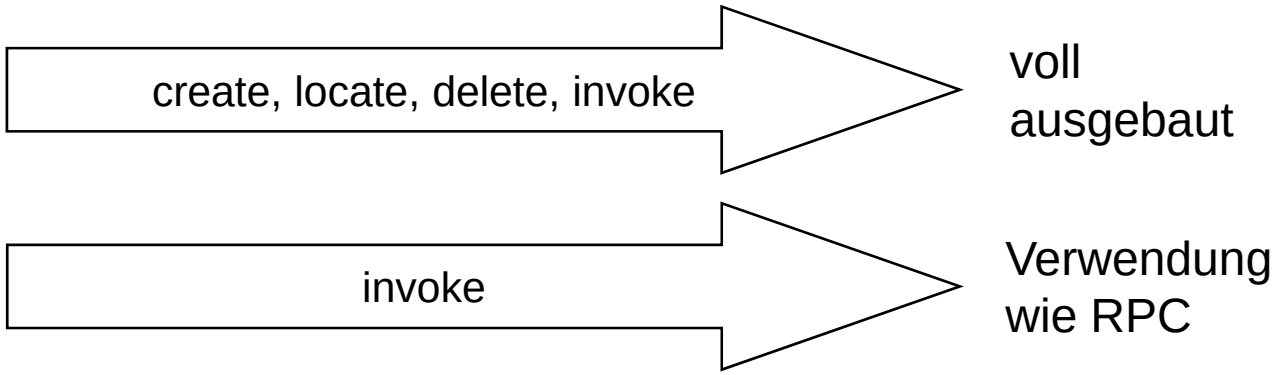
Verteilte Objekte

- Seit der Verfügbarkeit objektorientierter Sprachen werden **objektorientierte Programmierparadigmen** auch auf verteilte Systeme angewendet.
- Kommunikation und Datenaustausch zwischen verteilten Objekten wird typischerweise durch einen **Object Request Broker** (ORB) gesteuert.

  Der ORB unterstützt Erzeugung, Lokation, Aufruf und Löschung von verteilten Objekten.
- Weitverbreitete Implementierungen:
  - COM/DCOM (Microsoft)
  - RMI (Java)
  - CORBA (plattformunabhängig)
- Wie in objektorientierten Anwendungen üblich, werden viele Nachrichten zwischen Objekten ausgetauscht, die aber (meist) nur geringen Umfang haben.

Ablaufschema für verteilte Objekte





MOM - Message-Oriented Middleware

Schlüsselkonzepte

- Message
- Queue

Message

- Header (technische Informationen und Adressierung)
- „Payload“ (Anwendungsdaten)

Queue

- Speichert und verteilt Messages
- Entkopplung von Sender und Empfänger

Beispiel E-Mail

- E-Mail-Server …
  - entkoppeln Sender und Empfänger
  - speichern die Nachricht, bis sie ausgeliefert bzw. abgeholt werden kann (Persistenz!)
- Der E-Mail-Header enthält Adressierungsinformationen (<from>, <to>, <cc>), die das Routing der Nachricht ermöglichen
- Der E-Mail-Body enthält die eigentliche Information der Nachricht.

MOM - Verbindungsarten

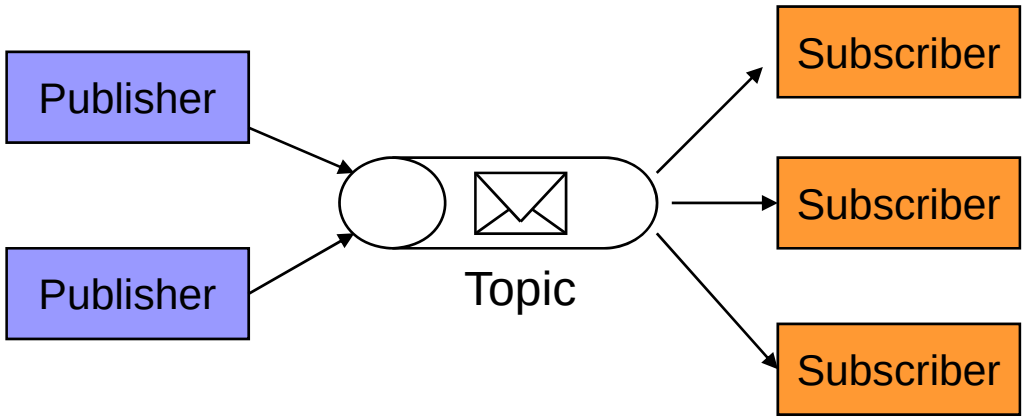
Verbindung von Kommunikationspartnern

- one-to-one (1:1)
- one-to-many (1:n)
- many-to-many (m:n)

Technische Konzepte

- Point-to-Point (1:1)
  → Ein Sender ist mit einem Empfänger über eine Queue verbunden
- Publish / subscribe (1:n und m:n)

Publisher versenden eine Nachricht mit einem Topic



Alle Subscriber für ein bestimmtes Topic erhalten diese Nachrichten

Queue Management

- Mehrere physikalische **Queues** können zu einer logischen Queue zusammengefasst werden.
- Queues können **vernetzt** werden.
- Intelligente **Routing-Regeln** können Nachrichten verteilen

→ Lastverteilung und Ausfallsicherheit

Service Level (QoS - Quality of Service)

Es können definierte Leistungsmerkmale einer Queue bestimmt werden, z.B.:

- **Priorität** einer Nachricht im Vergleich zu anderen Nachrichten
- **Transaktionsfähigkeit**
- Anzahl der erlaubten **Empfänger**
- **Gültigkeitsdauer** einer Nachricht
- Anzahl der wiederholten **Auslieferungsversuche**

Application Server

Beispiele:

- jBoss
- IBM Websphere
- Microsoft IIS

Grundfunktionen

- „Hosting“ und Ausführung von Komponenten
- Datenbankverbindungen verwalten
- Unterstützung verschiedener Typen von Benutzerschnittstellen wie Web-Schnittstellen oder Fat Clients

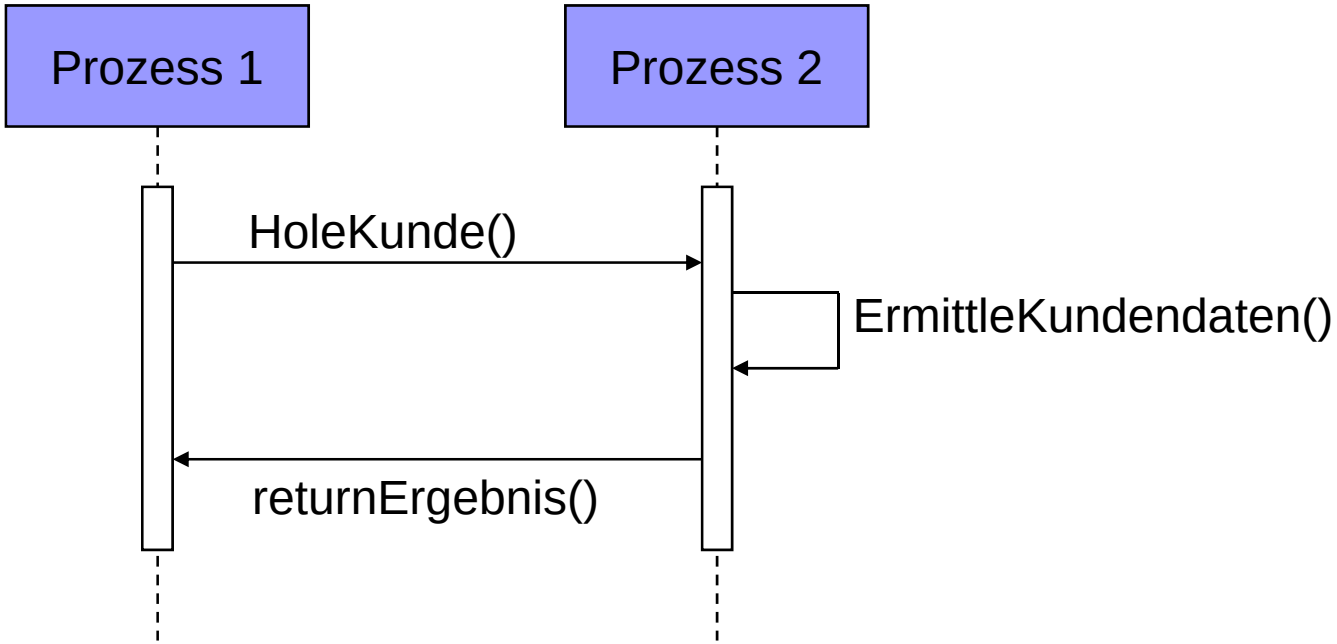
Erweiterte Eigenschaften

- Load-Balancing
- Caching
- Transaktionsmanagement
- Sicherheitsfunktionen

Schnittstellensemantik versus Inhaltssemantik (1)

Schnittstellensemantik

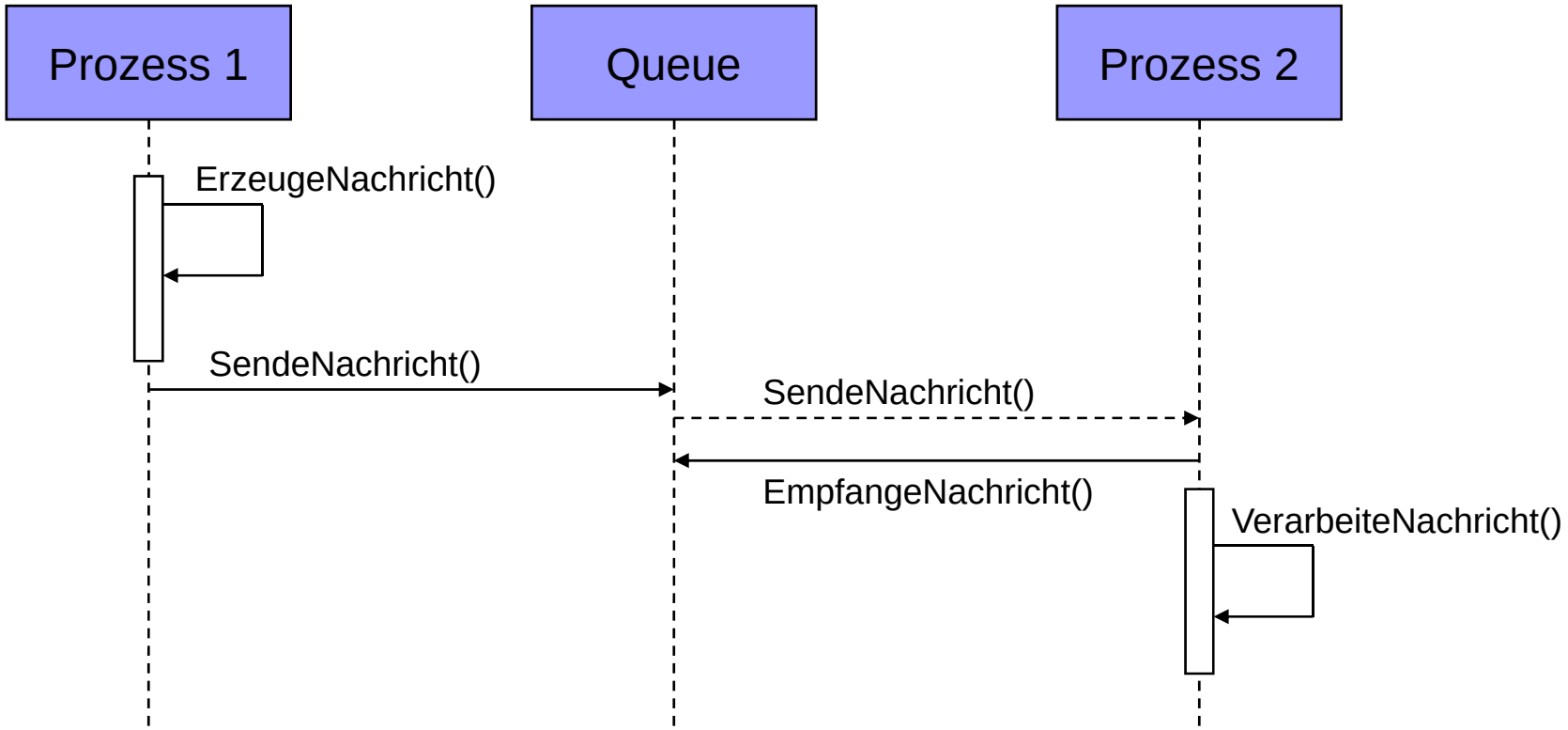
→ Es werden Funktionen verwendet, deren Namen selbsterklärend sind und deren Verbindung zur server-seitigen Funktion festgelegt ist (RPC-Style).



Schnittstellensemantik versus Inhaltssemantik (2)

Inhaltssemantik

→ Der Funktionsaufruf erfolgt über eine neutrale Funktion.
Die Entscheidung, welche Funktion auf dem Server auszuführen ist, ist **abhängig vom Inhalt der Nachricht**.



Schnittstellensemantik versus Inhaltssemantik

- Schnittstellensemantik ist für den Programmierer **intuitiv verständlich**.
- Bei einer Änderung an einer derartigen Schnittstelle müssen alle Systeme **geändert** werden, die diese Schnittstelle verwenden.

  → Enge Kopplung
- Bei Verwendung von **Inhaltssemantik** ist es möglich, neue Nachrichten zu definieren, ohne die bestehenden zu beeinträchtigen.
- Die Inhaltssemantik erlaubt nur **schwache Typprüfung**.

  → Lose Kopplung

Schnittstellensemantik lässt sich auch mit MOM abbilden.
Inhaltssemantik lässt sich auch mit RPCs abbilden.

Dokumenten-orientierte Nachrichten

- Mit Hilfe von selbstbeschreibenden Datenstrukturen wie XML können in einer Nachricht der Name der aufgerufenen Funktion und die zugehörigen Daten strukturiert übergeben werden.
- Die Datenstrukturen können flexibel erweitert werden, ohne dass sich die Schnittstelle selbst ändert.
- Sowohl der Aufruf, als auch die Antwort werden in Form des strukturierten Dokuments übermittelt.

Beispielimplementierung: SOAP

Beispiel: Flugbuchung

```
<SOAP-ENV:Envelope xmlns:SOAP-ENC="http://schemas.xmlsoap.org/
soap/encoding/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/
soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <SOAP-ENV:Body>
      <ns1:bookFlight xmlns:ns1="http://www.openuri.org/">
         <ns1:inbound>
            <ns1:flightNumber>LH400</ns1:flightNumber>
            <ns1:flightDate>2003-11-08</ns1:flightDate>
            <ns1:isConfirmed>false</ns1:isConfirmed>
         </ns1:inbound>
         <ns1:outbound>
            <ns1:flightNumber>LH401</ns1:flightNumber>
            <ns1:flightDate>2003-11-17</ns1:flightDate>
            <ns1:isConfirmed>false</ns1:isConfirmed>
         </ns1:outbound>
         <ns1:passenger>
             <ns1:Passenger>
                <ns1:firstName>Karl</ns1:firstName>
                <ns1:lastName>Banke</ns1:lastName>
                <ns1:birthday>1970-08-05</ns1:birthday>
             </ns1:Passenger>
          </ns1:passenger>
       </ns1:bookFlight>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Beispiel aus Krafzig, Enterprise SOA