Struktur von Kapitel III - Middleware

I Einleitung

II Grundlegende Kommunikationsdienste

III Middleware

- A Einführung / Überblick
- B Web Services & REST
- C Message-Oriented Middleware (MOM)
- D Objekt-orientierte Middleware (CORBA)

IV Architekturen & Algorithmen

- A Synchronisierung
- B Konsistenz und Replication
- C Fehlertoleranz

V Beispiele bzw. Dienste

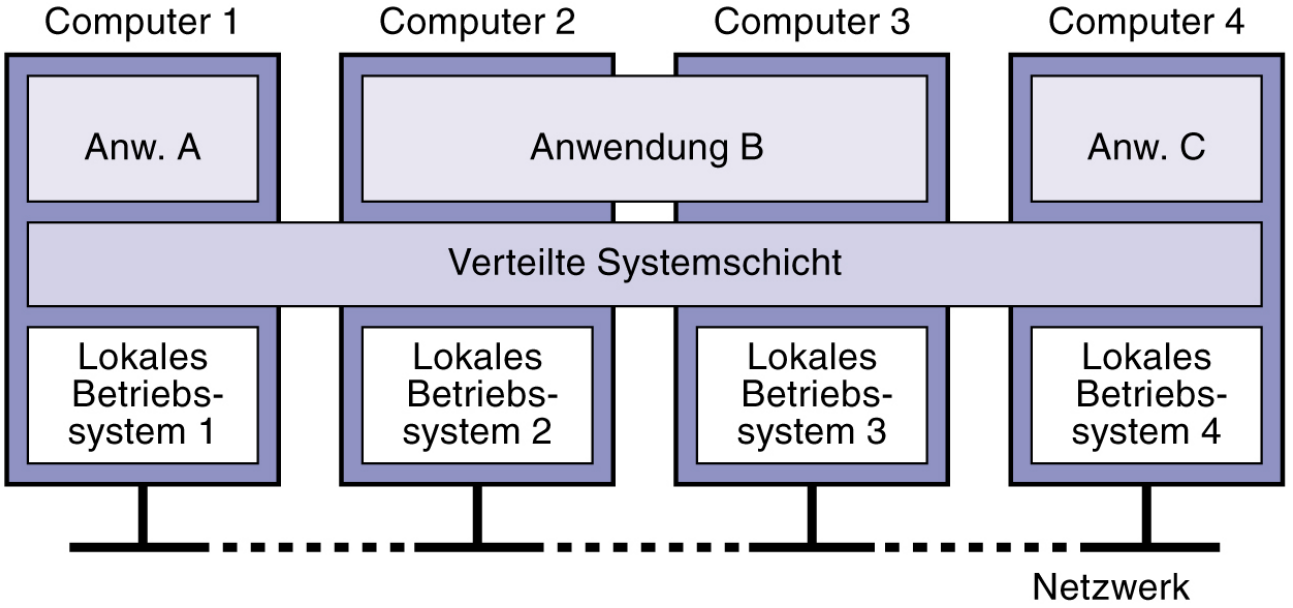
- A Verteilte Dateisysteme
- B Namensdienste

VI Sicherheits & Sicherheitsdienste

VII Zusammenfassung

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Was ist Middleware? (1)



Frage: Was unterscheidet Kommunikationsdienste (Kapitel II) von Middleware (Kapitel III)?

Antwort 0: **Nichts.** Kommunikationsdienste sind (eine Art von) Middleware – Laut Kapitel I (Einleitung).

Antwort 1: Unter „Middleware“ versteht (erwartet) man *mehr* als nur Kommunikationsdienste...

Was ist Middleware? (2)

Anfoderungen: Unter „Middleware“ versteht (erwartet) man :

- **Ein Infrastruktur:** Standard-Lösungen für Standard-Aufgaben (zum Beispiel Kommunikation)
- **Verteilungstransparenz:** Die Middleware sollte also Abstraktionen (eine bessere Umgebung, vgl. Kapitel I) anbieten :

Transparenz	Beschreibung
Zugriff	Verbirgt Unterschiede in der Datendarstellung und die Art und Weise, wie auf eine Ressource zugegriffen wird
Ort[1]	Verbirgt, wo sich eine Ressource befindet
Migration	Verbirgt, dass eine Ressource an einen anderen Ort verschoben werden kann
Relokation	Verbirgt, dass eine Ressource an einen anderen Ort verschoben werden kann, während sie genutzt wird
Replikation	Verbirgt, dass eine Ressource repliziert ist
Nebenläufigkeit	Verbirgt, dass eine Ressource von mehreren konkurrierenden Benutzern gleichzeitig genutzt werden kann
Fehler	Verbirgt den Ausfall und die Wiederherstellung einer Ressource

Von ISO/IEC IS10746, 1995 (vgl. TvS, S. 22).

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Drei Arten Middleware

- **Web-Services:** Wie XML-RPC, nur viel erhgeiziger...
- **Message-Oriented Middleware (MOM):** Wie verbindungsloser Sockets, nur viel ehrgeiziger...
- **Object-Oriented Middleware (an Hand von CORBA):** Wie Java RMI, ohne die JVM, mit Sprache-Wahlfreiheit

Warnung: Subjektiv

Ziele \ Middleware:	Web Services	MOM	OO-Middleware (CORBA)
Darstellungs-Transparenz	++	+	++
Mobile Code	0	0	++
Benennung	+	++	++
Synchronisierung	+	++	+
Replikation / Konsistenz	++	+	++

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Struktur von Kapitel III - Middleware

I Einleitung

II Grundlegende Kommunikationsdienste

III Middleware

- A Einführung / Überblick
- B Web Services & REST
- C Message-Oriented Middleware (MOM)
- D Objekt-orientierte Middleware (CORBA)

IV Architekturen & Algorithmen

- A Synchronisierung
- B Konsistenz und Replication
- C Fehlertoleranz

V Beispiele bzw. Dienste

- A Verteilte Dateisysteme
- B Namensdienste

VI Sicherheits & Sicherheitsdienste

VII Zusammenfassung

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Von XML-RPC zu Web-Services

Warnung: Subjektiv

Ziele \ Middleware:	XML-RPC	Web Services
Daten-Darstellung	Ganz einfache, vordefinierte XML	Beliebige XML Strukturen
Benennung	DNS-basiert, mit Load-Balancing	
Synchronisierung	HTTP für RPC (Anfrage->Antwort)	Meistens HTTP, andere Protokolle auch erlaubt
Replikation / Konsistenz	Transparente Load-Balancing	Load-Balancing + Service-Veröffentlichung
Automatische Software	Nicht vorgesehen	Verfügubar

Anders gesagt, Web Services sind XML-RPC *plus*...

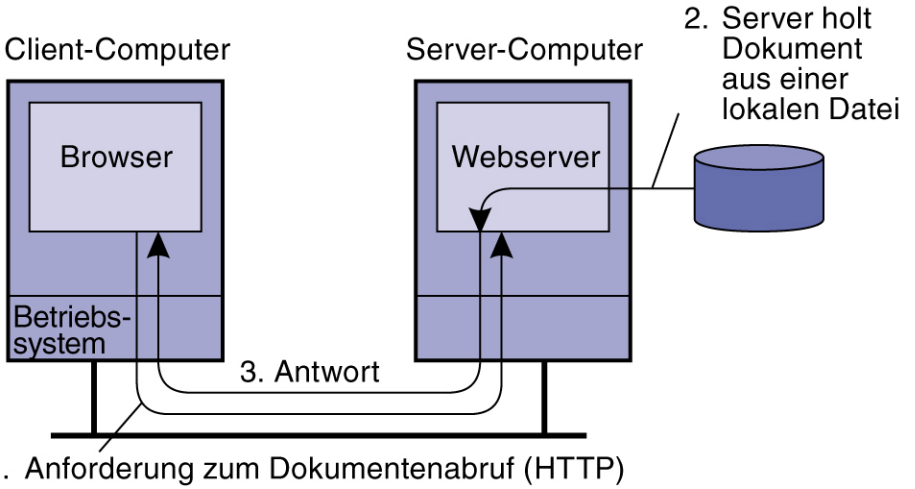
- mehr Freiheiten bei der Definition von Datenstrukturen und Datenaustausch (Protokolle),
- Unterstützung für Erzeugung von Software an Hand der Definitionen,
- Möglichkeiten für Veröffentlichung der Definitionen

Warnung!
Web-Services sind **nicht** einfach „Services, die im Web (verfügbar) sind“ oder „Services, die das Web verwenden“

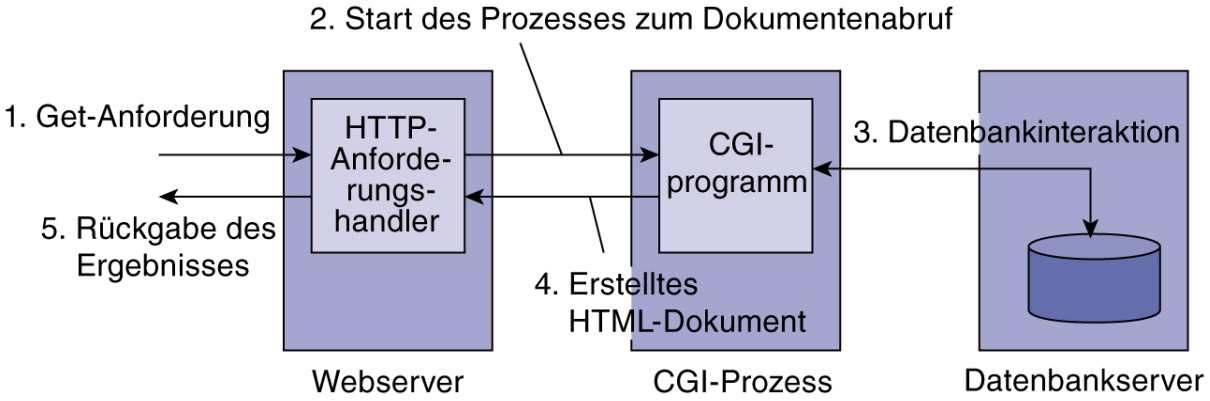
h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Basis: Web-Server & Web-Seiten

- Am Anfang gab es Web-Server, HTML und HTTP...



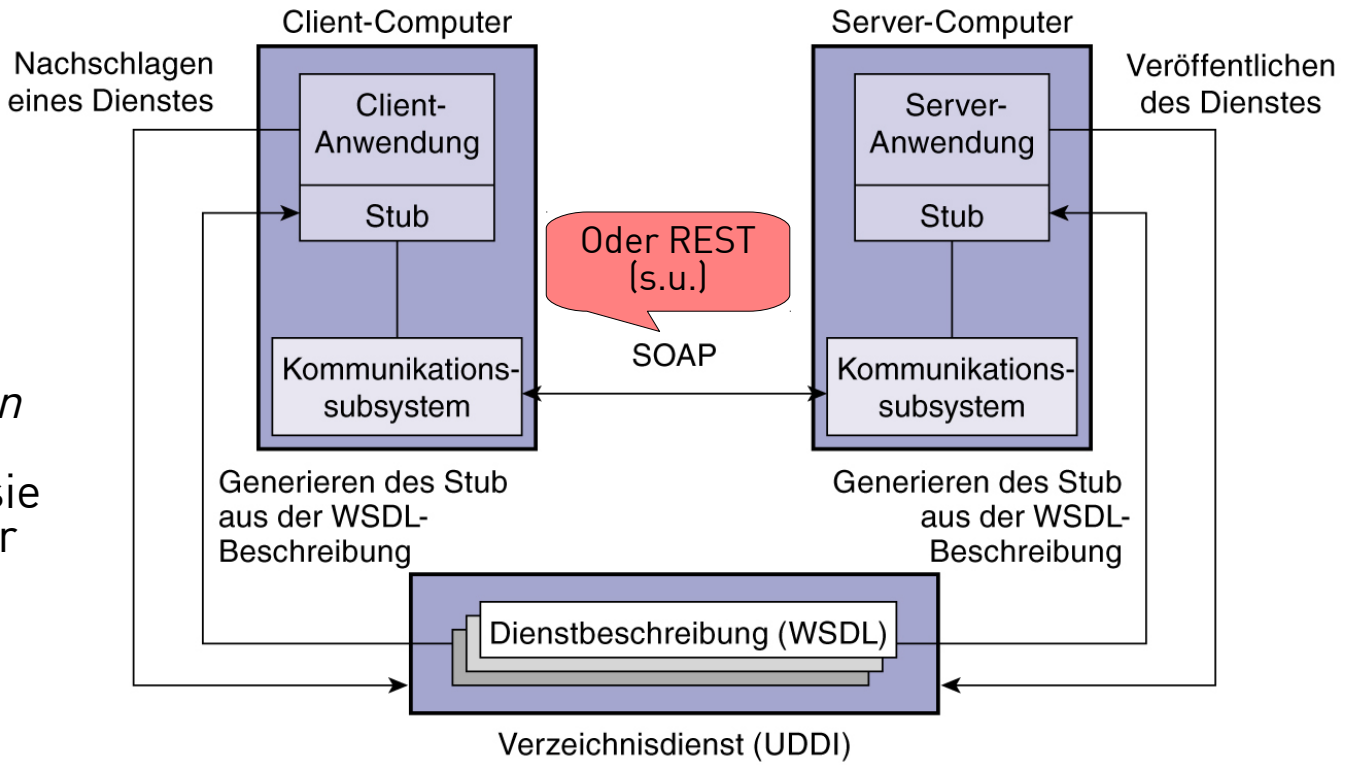
- Danach gab es Multi-Tier Web-Anwendung bzw. Anwendungs-Server...



Einsatzfeld: Mensch-Maschine Kommunikation

Idee: Web-Services

- Können Server dieselbe Technologie verwenden, um miteinander zu kommunizieren?
- Vision einer automatisierte Kommunikation unter Diensten:
  - Anwendungen *suchen* sich automatisch Dienste im Web, die sie zur Bearbeitung einer Anfrage benötigen.
  - Die Kommunikation bleibt transparent.



Einsatzfeld: Maschine-Maschine Kommunikation

Web Services – Zuerst, RESTful Services...

- Populär: REST (REpresentational State Transfer)
  - Roy Fielding: "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. thesis, 2000
  - Felding war an Spezifikation bzw. Entwicklung von HTTP, HTML und URI's beteiligt. Mitgrunder vom Apache-HTTP Server.
- Daten (Ressourcen) stehen im Vordergrund
  - Basiert meist auf HTTP (GET, POST, PUT, DELETE) Operationen zum Zugriff auf die Ressourcen ... *wie in HTTP definiert!*
  - Client/Server wie bei "normalen" Webanwendungen.
  - Praktisch für existierende Web-Architekturen: Infrastruktur existiert schon (wie bei Web Services, s.u.)
  - D.h. GET ist „read-only" PUT, POST & DELETE nicht ...
  - Server stellt Ressourcen als URLs bereit
  - Über Links werden weitere Ressourcen identifiziert

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Web Services – RESTful Services, fortgesetzt

- HTTP Anfragen evtl. mit Payload (PUT & POST/Modifizieren)
  - XML (z.B. beschrieben mit WSDL)
  - JSON
  - ...

JSON == Javascript Object Notation, vergleichbar mit XML, nur wesentlich leichter, einfacher... s.u.

- „Nachteil“ (?!?): Wenig Standardisierung (vgl. Web Services, s.u.)

Vielleicht weil ProgrammierInnen, die schon einen lauf-fähigen Web-Server betreiben, wenig Interesse auf Spezifierung haben?

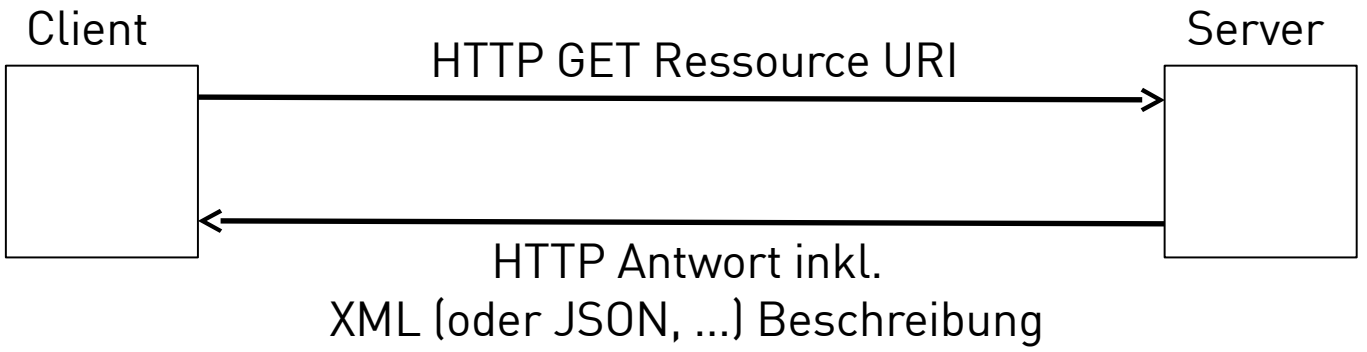
Warnung: reinste Spekulation...

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

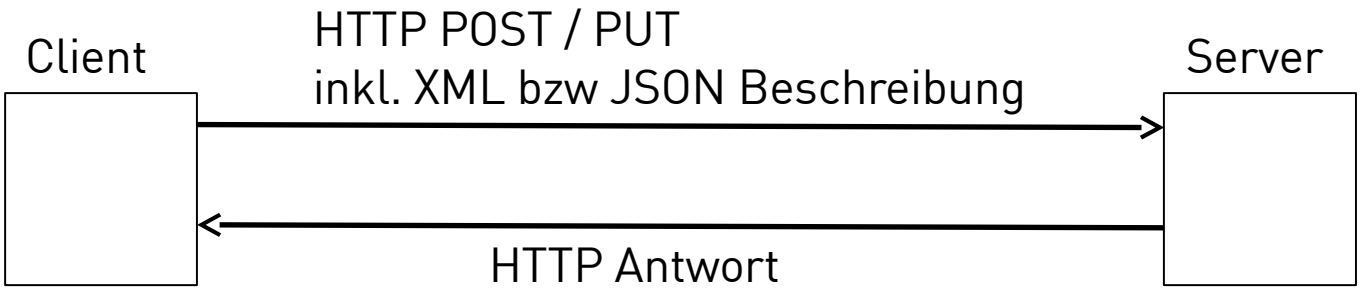
REST und HTTP

Client kann Ressourcen

- anfordern: HTTP GET
- modifizieren: HTTP POST oder HTTP PUT
- löschen: HTTP DELETE



- XML oder JSON zum Modifizieren von Zstand



REST Beispiel (0)

- Webshop für Ersatzteile (*parts*)
- Basis URL: http://www.parts-depot.com/parts
- Zugriff auf diese URL (HTTP GET) liefert (könnte auch JSON oder ein anderes Format sein):

Links mit weiteren Informationen

```
<?xml version="1.0"?>
<p:Parts xmlns:p="http://www.parts-depot.com"
    xmlns:xlink="http://www.w3.org/1999/xlink">
    <Part id="00345"
        xlink:href="http://www.parts-depot.com/parts/00345"/>
    <Part id="00346"
        xlink:href="http://www.parts-depot.com/parts/00346"/>
    <Part id="00347"
        xlink:href="http://www.parts-depot.com/parts/00347"/>
    <Part id="00348"
        xlink:href="http://www.parts-depot.com/parts/00348"/>
</p:Parts>
```

Quelle: http://www.xfront.com/REST-Web-Services.html

REST Beispiel (1)

- Informationen über ein Ersatzteil bekommt man durch Verfolgen der entsprechenden URL (HTTP GET):

  http://www.parts-depot.com/parts/00345

```
<?xml version="1.0"?>
<p:Part
    xmlns:p="http://www.parts-depot.com"
    xmlns:xlink="http://www.w3.org/1999/xlink">
    <Part-ID>00345</Part-ID>
    <Name>Widget-A</Name>
    <Description>This part is used within the frap assembly
    </Description>
    <Specification
     xlink:href="http://www.parts-depot.com/parts/00345/specification"/>
    <UnitCost currency="USD">0.10</UnitCost>
    <Quantity>10</Quantity>
</p:Part>
```

Link mit noch mehr Informationen

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

REST Beispiel (2)

- Ändern der Beschreibung eines Ersatzteils (mit HTTP PUT für neue Ersatzteile bzw. POST für „Updates“)

```
<?xml version="1.0"?>
<p:Part
    xmlns:p="http://www.parts-depot.com"
    xmlns:xlink="http://www.w3.org/1999/xlink">
    <Part-ID>00345</Part-ID>
    <Name>Widget-A</Name>
    <Description>This part is NOT used anywhere </Description>
    <Specification
     xlink:href="http://www.parts-depot.com/parts/00345/specification"/>
    <UnitCost currency="USD">0.10</UnitCost>
    <Quantity>10</Quantity>
</p:Part>
```

Neue Beschreibung

REST Beispiele:

- Google, z.B.
  https://developers.google.com/custom-search/json-api/v1/using_rest
  - Beispiel Aufruf:
    https://www.googleapis.com/customsearch/v1?q=SUCHBEGRIFF&...

- Doodle: http://doodle.com/xsd1/RESTfulDoodle.pdf

Hinweis: Dienst, nicht Ressource

- Twitter https://dev.twitter.com/docs/api
  - Beispiel Aufruf:

```
GET https://api.twitter.com/1.1/statuses/
mentions_timeline.json?count=2&since_id=14927799
```

  - Antwort


```
[
      { "coordinates": null,
        "favorited": false,
        "truncated": false,
        "created_at": "Mon Sep 03 13:24:14 +0000 2012",
        "id_str": "242613977966850048",
        "entities": {
            "urls": [
...]
```

Offene RESTful Fragen:

Frage: Sind RESTful Services besser als RPCs?

Was ist der Unterschied?

Was sind die Vorteile?

Was sind die Nachteile

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Basiskomponenten von Web Services

Ziel: SOA (Service Oriented Architecture)

- **XML (eXtended Markup Language):**

  Erweiterbares Textformat für den Austausch von strukturierten Daten. (Vgl. HTML).

- **XML-Schema –** XML-Beschreibung von XML-Strukturen.

- **SOAP (Simple Object Access Protocol):**

  Alternativ: REST (s.u.)

  Realisiert den entfernten Prozeduraufruf (RPC).
  Transportiert XML Nachrichten beispielsweise über HTTP.

- **WSDL (Web Service Description Language):**

  Für unsere Zwecke, WSDL == Web Service (!)

  XML Notation zur Beschreibung von Web Services.

- **UDDI (Universal Description, Discovery and Integration):**

  Verzeichnisdienst zur Veröffentlichung von Webservices.

Muß nicht sein. Nice to have.

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

XML - Beispiel

```
<?xml version="1.0" encoding="UTF-8" ?>
<bank>
    <kunde id=1>
        <name>Meier</name>
        <vorname>Emil</vorname>
        <telefon>1234</telefon>
    </kunde>
    <kunde id=2>
        .....
    </kunde>
</bank>
```

XML Kopf, mit XML Version & Zeichensatz

Tags für Datenstrukturierung

Tags mit Attributen

Elemente treten paarweise, geschachtelt auf

Problem: Die Tags sind noch nicht definiert bzw. beschrieben.

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

XML Schema

Es gibt zwei Möglichkeiten zur Definition der Semantik für XML Dokumente:

- **Alte:** DTD (Data Type Definition): Spezifikation für den Austausch von Dokumenten, komplexe Syntax, nicht XML basiert.
- **Neue:** XML Schema: Nachfolger der DTD, XML basiert.

XML Schema:

- Die XML Schema Spezifikation bietet:
  - eine Reihe an vordefinierten Basisdatentypen und
  - Ausdrucksmöglichkeiten zur Definition komplexer Datentypen.
- Die Datentypdefinitionen werden in einer Datei mit der Endung: .xsd beschrieben.
- Das .xml File erhält einen Link auf sein(e) XMLSchema Definition und kann danach interpretiert werden.

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

XML Schema Beispiel

```
<? xml version="1.0" encoding="UTF-8" ?>
<bank xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance"
      xsi:noNamespaceSchemaLocation="file:bank.xsd" >
   <kunde id=1>
      <name>Meier</name>
      <vorname>Emil</vorname>
      <telefon>1234</telefon>
   </kunde>
   <kunde id=2>
      .....
   </kunde>
</bank>
```

Hinweis auf xsd in xml

Schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="kunde" type="Kunde" />
   <xs:complexType name="Kunde" >
      <xs:sequence>
         <xs:element name="name"    type=xs:string/>
         <xs:element name="vorname" type=xs:string/>
         <xs:element name="telefon" type=xs:integer/>
      </xs:sequence>
   </xs:complexType>
   ...
</xs:schema>
```

Namespaces

SOAP

- SOAP ist ein weiteres Protokoll für Client-Server-Anwendungen. Definiert:
  - Envelope
  - Header
  - Body
- Kann auf verschiedenen Trägerprotokollen aufsetzen:
  - HTTP (bzw HTTPS)
  - SMTP
  - Messaging Service ... usw...
- In den meisten Fällen wird SOAP in Kombination mit HTTP verwendet.
- Wird wie XML vom W3C verwaltet.

Da HTTP zustandslos ist, ist auch eine SOAP Kommunikation zustandslos. SOAP (an sich) kann keinen Zustand zwischen zwei SOAP Aufrufen halten.

SOAP Beispiel

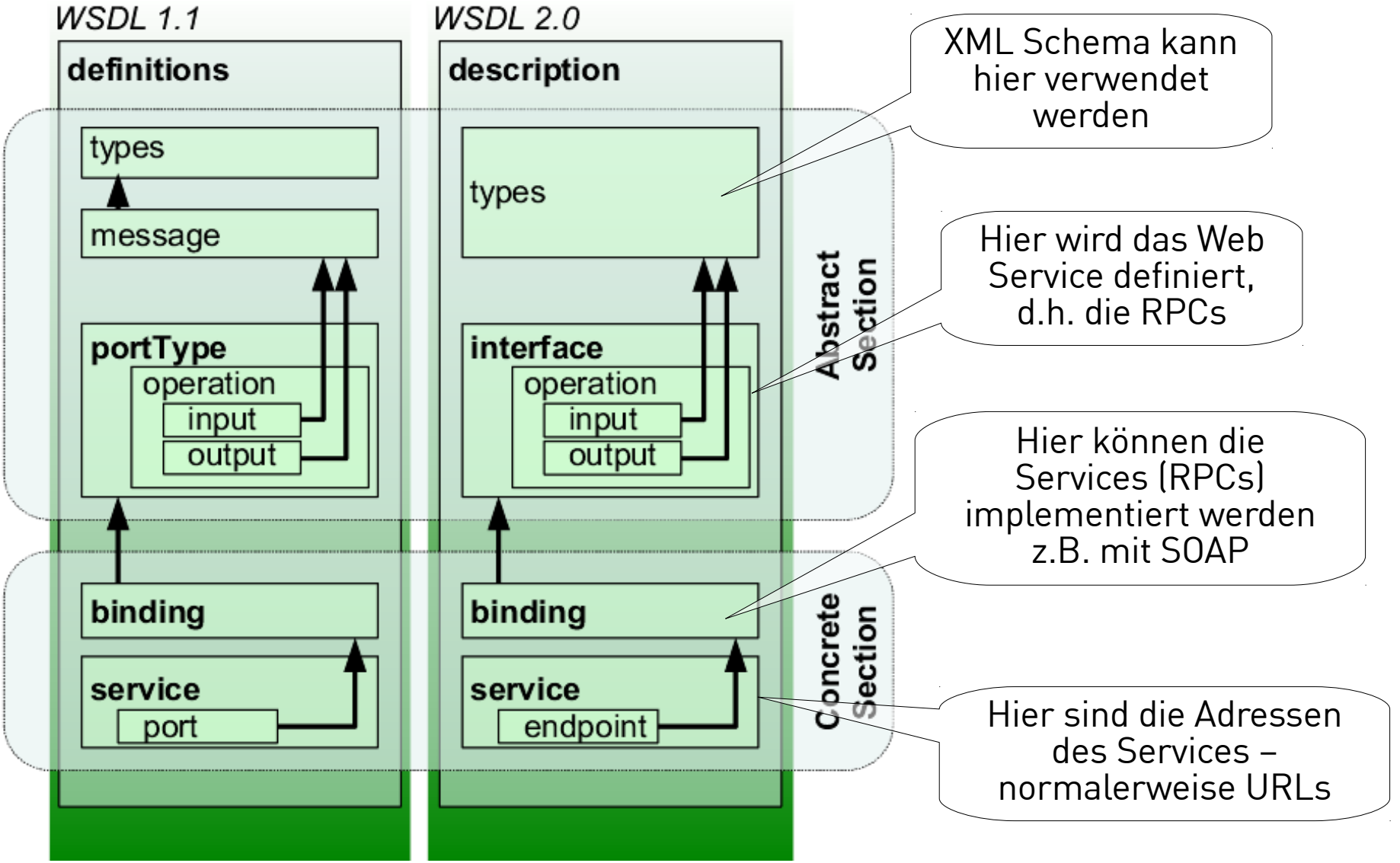
```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
     <env:Header>
        <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
            <n:priority>1</n:priority>
            <n:expires>2001-06-22T14:00:00-05:00</n:expires>
        </n:alertcontrol>
     </env:Header>
     <env:Body>
        <m:alert xmlns:m="http://example.org/alert">
           <m:msg>Mary um 14 Uhr von der Schule abholen</m:msg>
        </m:alert>
     </env:Body>
</env:Envelope>
```

Nutzlast

WSDL

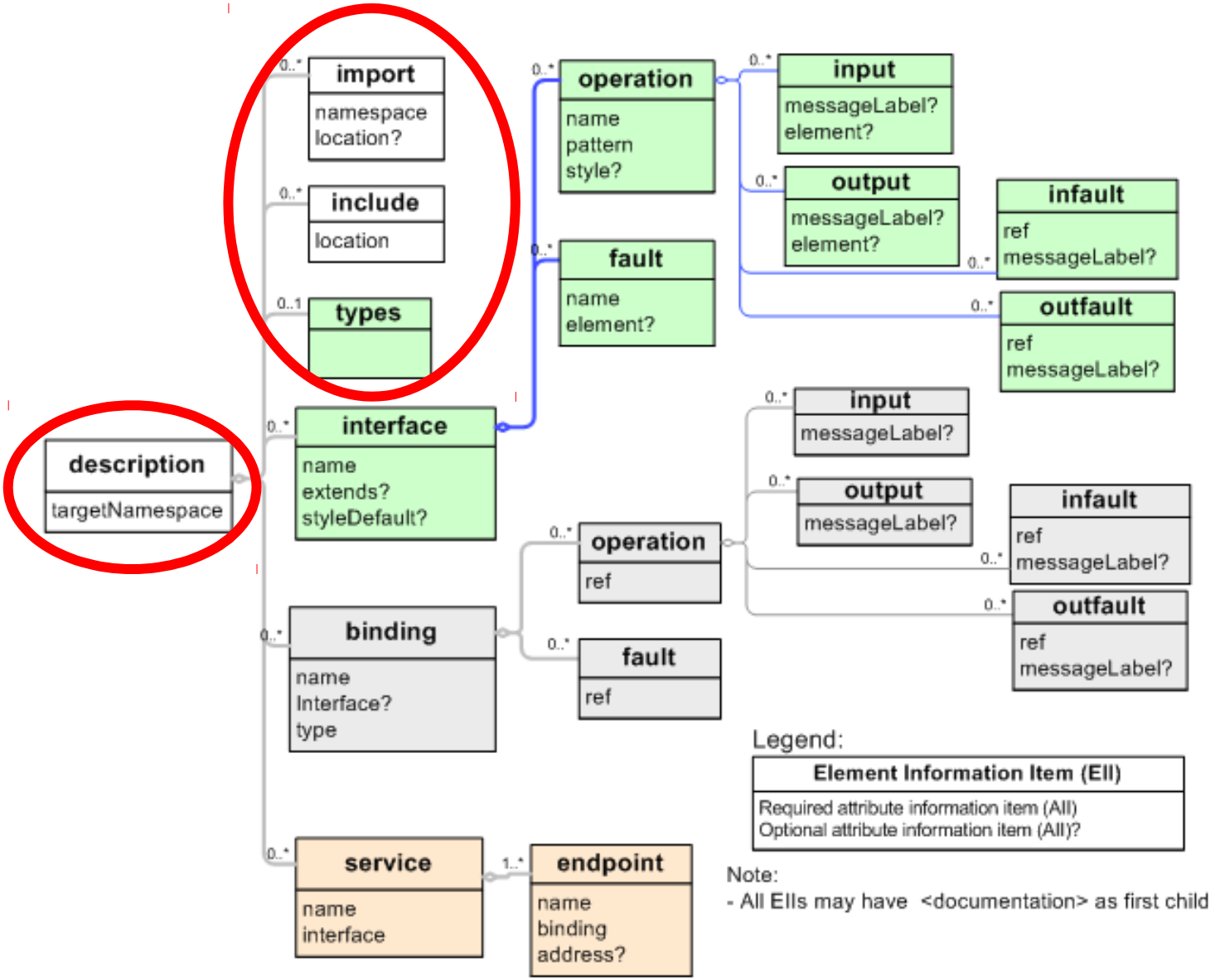
- Schnittstellensprache zur Veröffentlichung von Diensten als Web Services.
- Wird vollständig in XML formuliert. Der Standard definiert ein WSDL-spezifisches XML-Schema.
- **Ziel:**
  - Schaffen eines von Anwendungen lesbaren Zugangs zu angebotenen Diensten
  - Weitgehend automatisierte Integration von Diensten zu neuen Anwendungen
  - Integration zur Laufzeit
- WSDL Beschreibungen werden üblicherweise nicht manuell erstellt, sondern mithilfe von Werkzeugen generiert.
- Es gibt auch Werkzeuge, die aus einer WSDL-Datei die Client / Server Software erzeugen – sogar zur Laufzeit!

WSDL Struktur



Quelle: Web Services Description Language. . In **Wikipedia, The Free Encyclopedia**. Retrieved 06:50, April 21, 2010, from http://en.wikipedia.org/w/index.php?title=Web_Services_Description_Language&oldid=348616639

WSDL Struktur



Quelle: http://www.w3.org/TR/wsdl20-primer/

WSDL – Beispiel (1)

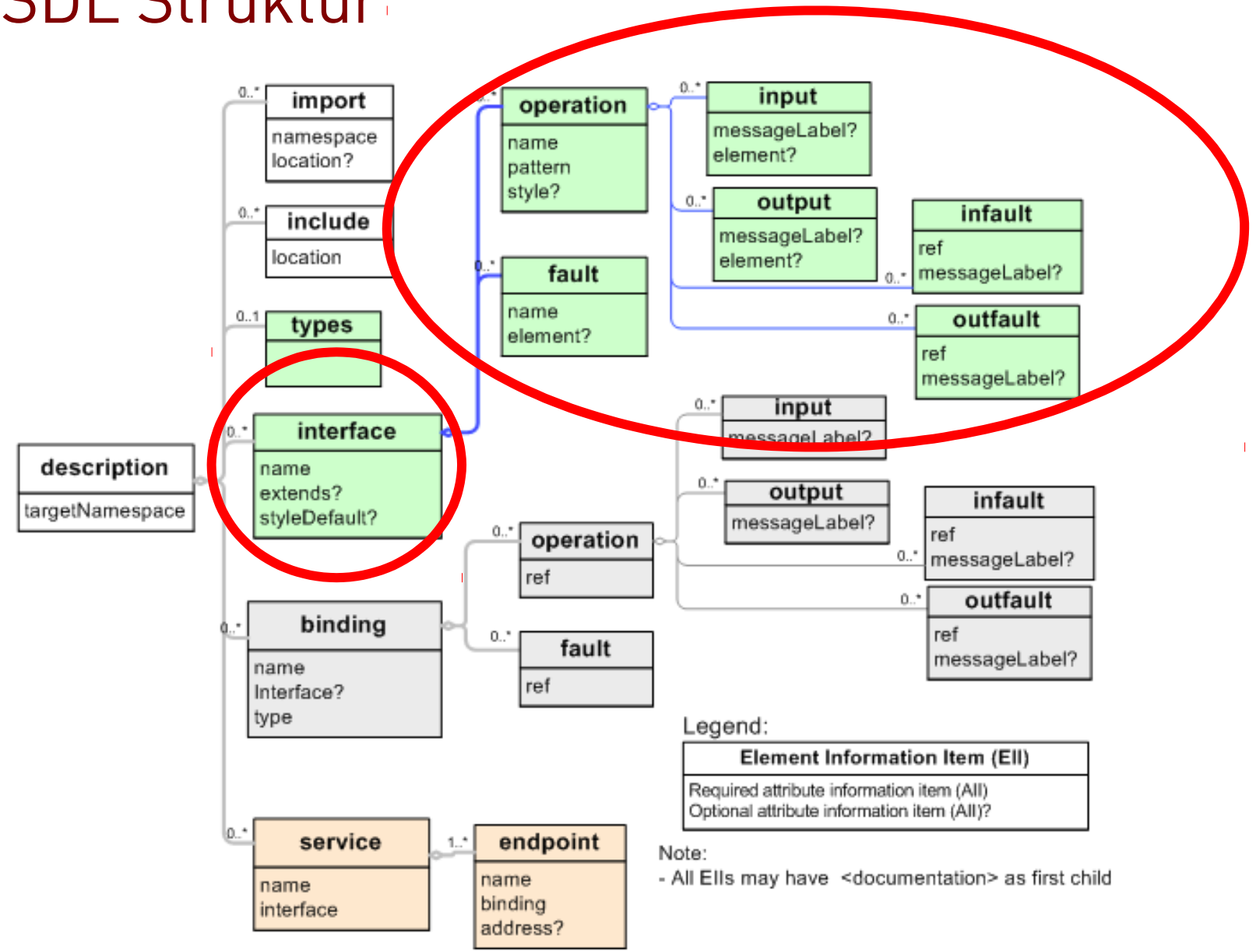
```
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/ns/wsdl"
    targetNamespace=
                "http://greath.example.com/2004/wsdl/resSvc"
    xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc"
    xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc"
    xmlns:wsoap= "http://www.w3.org/ns/wsdl/soap"
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsdlx= "http://www.w3.org/ns/wsdl-extensions">

  <documentation>
    This document describes the GreatH Web service. ...
  </documentation>

  <types>
    <xs:schema
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        ...

    </xs:schema>
  </types>
  ...
```

Verwendete Namespaces

Für Menschen...

XML Schema für Datentypen

WSDL Struktur



Quelle: http://www.w3.org/TR/wsdl20-primer/

WSDL – Beispiel (2)

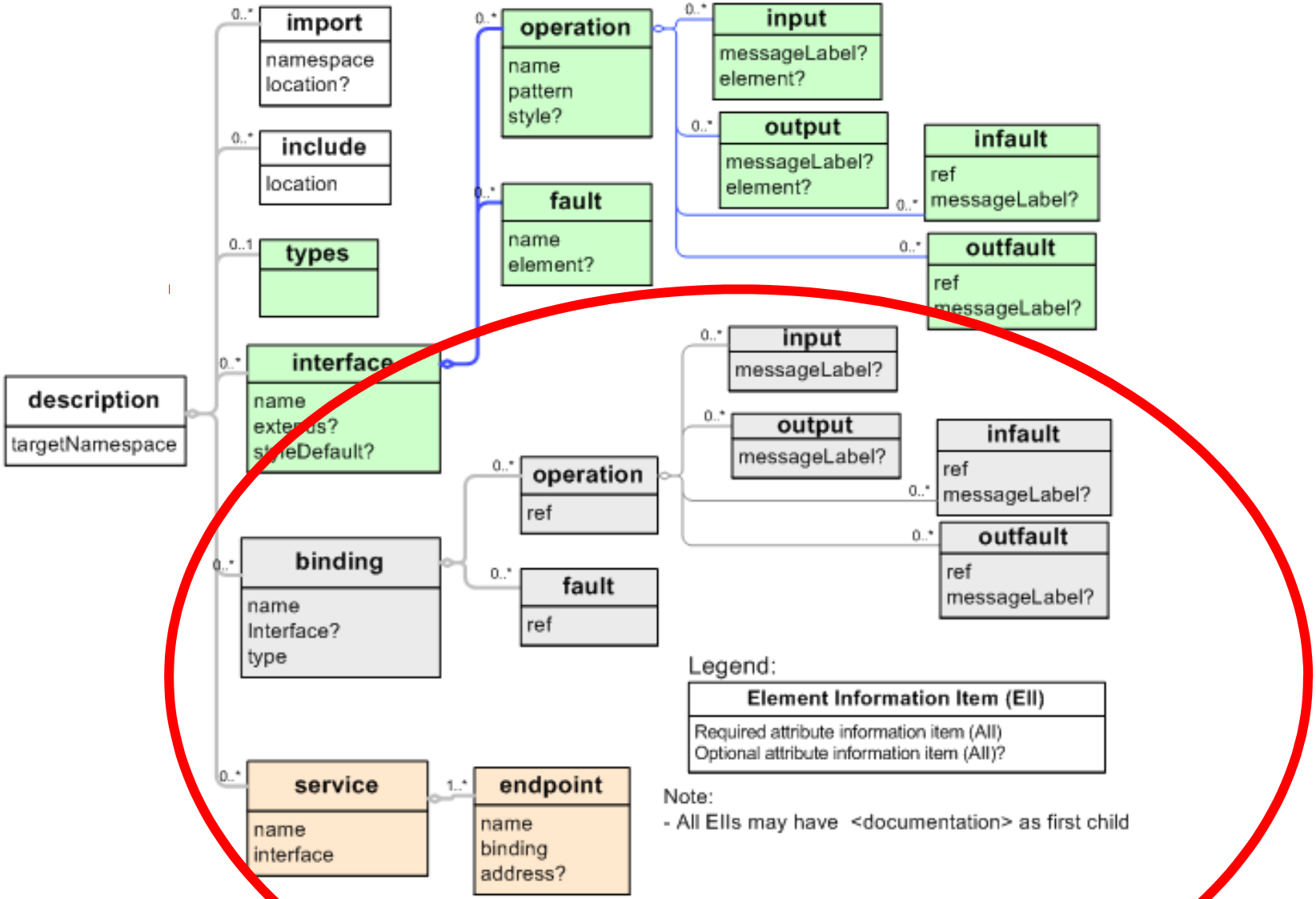
```
<interface  name = "reservationInterface" >

  <fault name = "invalidDataFault"
         element = "ghns:invalidDataError"/>

  <operation name="opCheckAvailability"
         pattern="http://www.w3.org/ns/wsdl/in-out"
         style="http://www.w3.org/ns/wsdl/style/iri"
         wsdlx:safe = "true">
      <input messageLabel="In"
           element="ghns:checkAvailability" />
      <output messageLabel="Out"
           element="ghns:checkAvailabilityResponse" />
      <outfault ref="tns:invalidDataFault"
                messageLabel="Out"/>
  </operation>

</interface>

...
```

Funktionen (& Exceptions)

Daraus kann eine API erzeugt werden!

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

WSDL Struktur



Quelle: http://www.w3.org/TR/wsdl20-primer/

WSDL – Beispiel (3)

```
...
<binding name="reservationSOAPBinding" ...>

  <fault ref="tns:invalidDataFault"
    wsoap:code="soap:Sender"/>

  <operation ref="tns:opCheckAvailability" .../>

</binding>

<service name="reservationService"
     interface="tns:reservationInterface">

   <endpoint name="reservationEndpoint"
             binding="tns:reservationSOAPBinding"
       address ="http://greath.example.com/2004/reservation"/>

</service>

</description>
```

Definiert Nachrichtenformate und Transportprotokoll wie sie zur Übertragung der Aufrufe verwendet werden. Häufig wird das SOAP Binding eingesetzt.

Definiert alle für den Zugriff auf den Dienst notwendigen Informationen wie Netzwerkadresse und Portnummer.

WSDL: Tools

- Beispiel: Apache Axis2 (Web Services engine)
  - Java und C Implementierung
  - Erzeugen von WSDL aus z.B. Java Code
  - Erzeugen von z.B. Java Code (Schnittstellen) aus WSDL Beschreibungen
  - Anscheinend aufgegeben ("apparently abandoned") seit 2009...
- Andere Sprachen (wie PHP) kommen mit relativ einfache Bibliotheke aus, um aus WSDL Code (Objekte) zu erzeugen...

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

WSDL – Verwendung (in PHP)

```
  <?php
$apiKey = 'XXXXXXXX';
$client = new SoapClient('http://api.google.com/GoogleSearch.wsdl');
$result = $client->doGoogleSearch(
        $apiKey,   // API key
        "php5",    // Suchbegriff
        0,         // Erster Eintrag der Ergebnisse
        10,        // Anzehl der Ergebnisse die geliefert werden
        false,     // Ähnliche Ergebnisse ausschließen
        '',        // Auf Themen einschränken
        true,      // Keine Inhalte, die nur für Erwachsene sind
        '',        // Auf Sprache einschränken
        '', ''     // werden ignoriert
       );

printf("Insgesamt ungefähr %d Ergebnisse gefunden\n",
       $result->estimatedTotalResultsCount);
$i = 0;
foreach ($result->resultElements as $ergebnis) {
    printf("%d. %s\n", ++$i, utf8_decode($ergebnis->title));
}
?>
```

Muss mit echtem Key ersetzt werden...

Erst jetzt weiß die Anwendung, wie der Google Web Service verwendet wird

Aber der Programmierer hat es anscheinend trotzdem schon gewußt...

Quelle: „PHP 5 Kochbuch“, §15.4,
Sklar & Trachtenberg, 2. Auflage(2005), O'reilly Verlag

SOAP *ohne* WSDL – (in PHP)

```
<?php
$client = new SoapClient(NULL,
                         array(
              "location" => "http://api.google.com/search/beta2",
              "uri"      => "urn:GoogleSearch",
              "style"    => SOAP_RPC,
              "use"      => SOAP_ENCODED,
            "exceptions" => 0                    ) );

$params = array( new SoapParam('XXXXXXXXXXXXXX', 'key'),
                 new SoapParam('php5', 'q'),
                 ...
                 new SoapParam('', 'oe')
                );

$options = array(   'uri' => 'urn:GoogleSearch',
                  'soapaction' => 'urn:GoogleSearch#doGoogleSearch'  );

$result = $client->__call('doGoogleSearch', $params, $options);

if (is_soap_fault($result)) { ... }

 // result verwenden wie oben ...
```

Vgl. letzte Seite („mit WSDL“).
Alle Informationen, die hier neu sind, waren in der WSDL Datei!

Quelle: „PHP 5 Kochbuch“, §15.5,
Sklar & Trachtenberg, 2. Auflage(2005), O'reilly Verlag

UDDI

- Verzeichnisdienst für Web Services
- Veröffentlicht werden
  - WSDL Definitionen von Web Services
  - Zusatzinformationen zu Inhalten, verantwortliche Organisation, Grobe thematische Einordnung, ...
- Definiert zwei APIs:
  - Publishing API: Anbieter können über dieses API ihren Web Service veröffentlichen.
  - Inquiry API: Anwender können über dieses API nach einem Web Service suchen.
- UDDI wurde ursprünglich im Rahmen eines unabhängigen Projekts entwickelt: www.uddi.org (Beginn 2000).
- Wurde 2002 an OASIS übergeben, in der Web Services Interoperability (WS-I) Standard integriert.

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Web Services – Wofür soll das gut sein?

- Offene Frage: Was bringen WSDL & UDDI & Co.?
  - Im Vergleich mit XML-RPC?
  - In einer großen Firma?
  - Für KMUs (Kleine & Mittlere Unternehmen)?
- Noch eine Frage: Welche Transparenzeigenschaften kann man erreichen?
  - Zugriff?
  - Ort?
  - Fehler?
  - Migration?
  - Nebenläufigkeit?
  - Replikation?

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Struktur von Kapitel III - Middleware

I Einleitung

II Grundlegende Kommunikationsdienste

III Middleware

- A Einführung / Überblick
- B Web Services & REST
- C Message-Oriented Middleware (MOM)
- D Objekt-orientierte Middleware (CORBA)

IV Architekturen & Algorithmen

- A Synchronisierung
- B Konsistenz und Replication
- C Fehlertoleranz

V Beispiele bzw. Dienste

- A Verteilte Dateisysteme
- B Namensdienste

VI Sicherheits & Sicherheitsdienste

VII Zusammenfassung

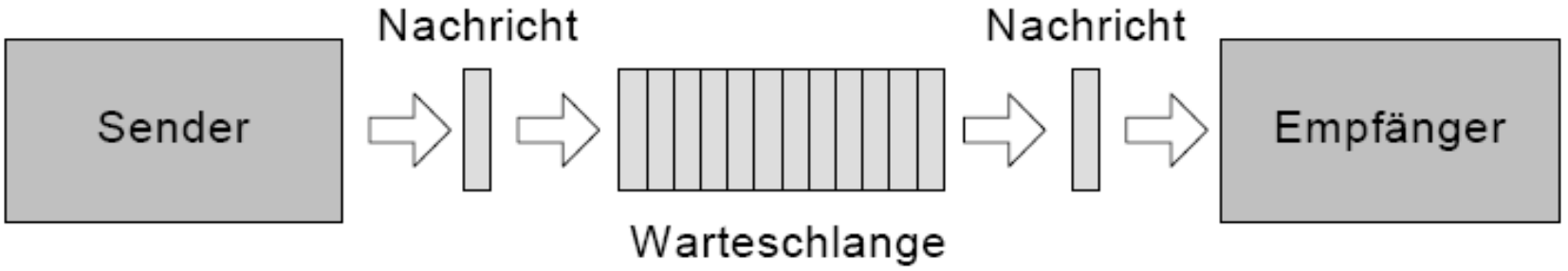
Von asynchrone Sockets zu MOM

Ziele \ Middleware:	Sockets	MOM
Daten-Darstellung	Selbst-definiert	Middleware-Unterstützung
Benennung	DNS-basiert	Middleware-basiert
Synchronisierung	Asynchron – Anwendung-synchronisiert	
Replikation / Konsistenz	Nicht (unbedingt) vorgesehen	
Fehler Toleranz	Nicht vorgesehen	Middleware-basiert

Anders gesagt, MOM ist wie Sockets, *plus*...

- Eine Infrastruktur für Kommunikation,
- Warteschlangen für Zwischenspeicherung, Fehlertoleranz,
- Infrastruktur mit „Boni“ wie z.B. Benennung, Routing, Verteilungstransparenz... S.u.

Messaging, Message Passing, Message Queuing



- Message Passing ist ein Programmiermodell zum **asynchronen** (vgl. Sockets) Transport von Daten zwischen Prozessen (vgl. RPC, RMI = Call/Respond).
- Die Daten werden innerhalb von Nachrichten (Messages) übertragen.
- Message Passing arbeitet auf der Basis von **Warteschlangen**:
  - Nachrichten werden vom Sender in eine Warteschlange gelegt...
  - ...und werden an den Empfänger weitergeleitet.

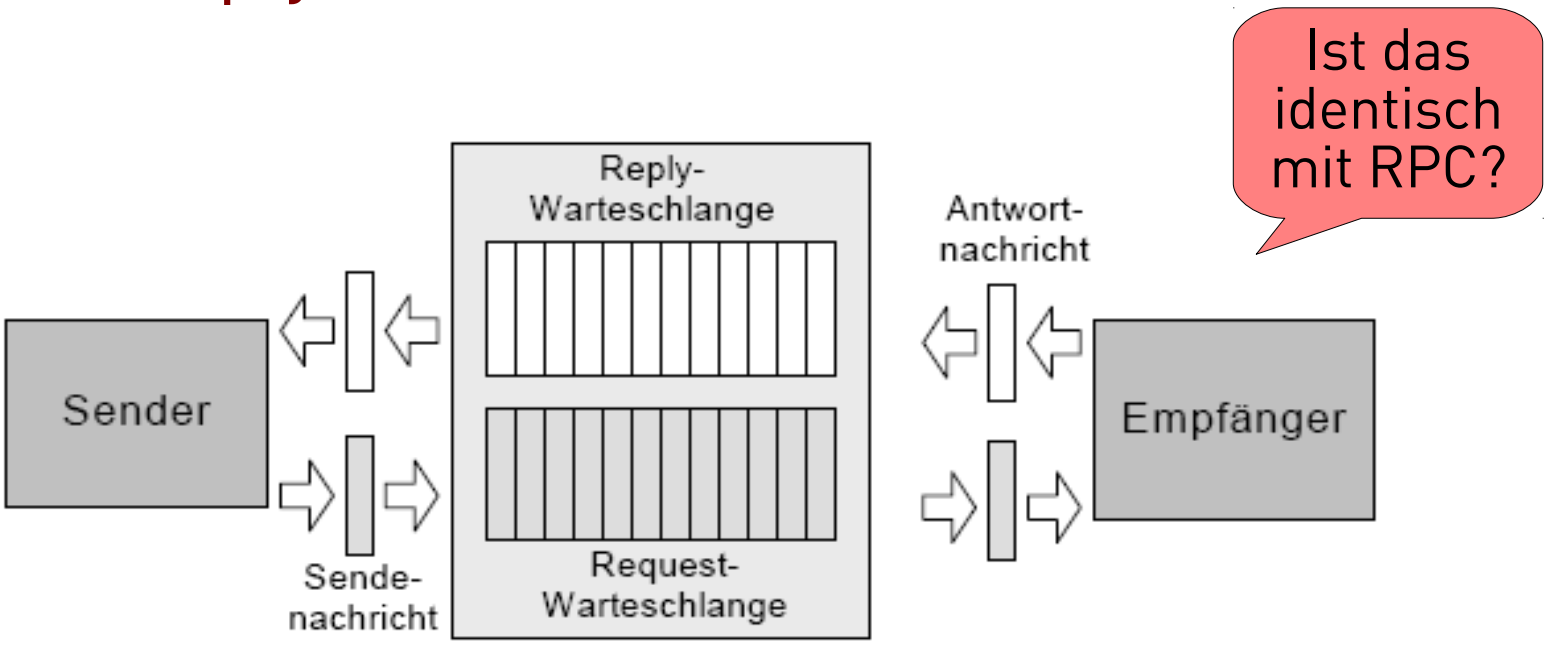
Message-Passing-Varianten

- **Point-to-point:**
  Die Nachrichtenübertragung findet zwischen zwei festgelegten Partnern statt.
  - Genauer gesagt, jeder Nachricht wird in eine Warteschlange (Queue) geschrieben, und genau einmal dort entnommen.
  - Eine spezielle Art des Point-to-point Message Passing ist das **Request-Reply Modell**.
  - … allerdings muss es nicht immer eine Antwort (Reply) geben! Auch möglich: "Fire and forget" = **Ereignis-gesteuerte Architektur**

- **Multi- bzw. Broadcasting:**
  Eine Nachricht wird an alle erreichbaren Empfänger versendet.
  - Eine Umsetzung des Broadcastings ist das **Publish-Subscribe Modell** *(s. u.)*.
  - In diesem Modell heißen die Warteschlangen oft **Themen** (**Topics**)

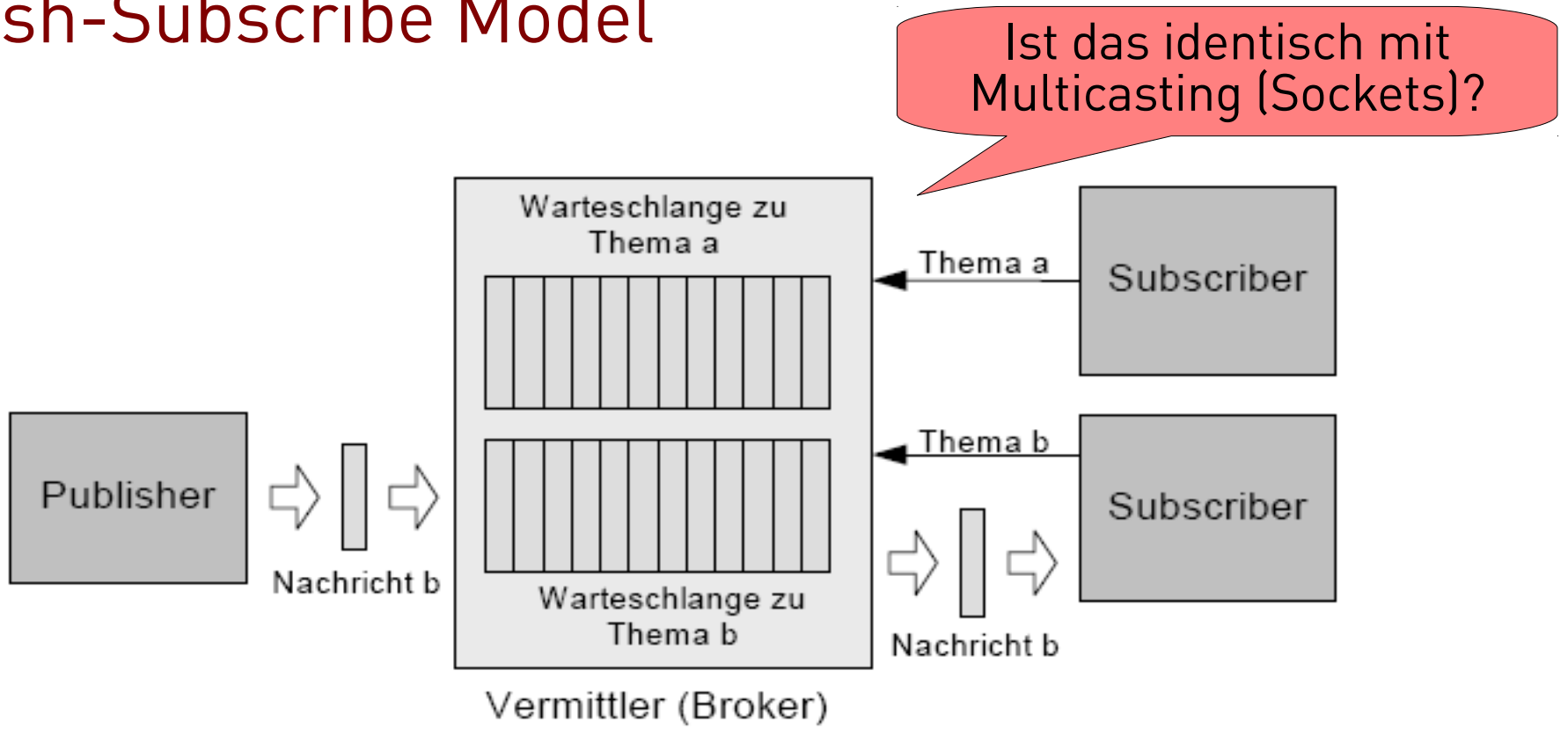
Use-Case: Ein(e) Mitarbeiter(in) verlässt eine Firma. Das hat verschiedenste Konsequenzen! Verschiedene System-Komponenten müssen das erfahren und darauf reagieren!

Request-Reply Modell



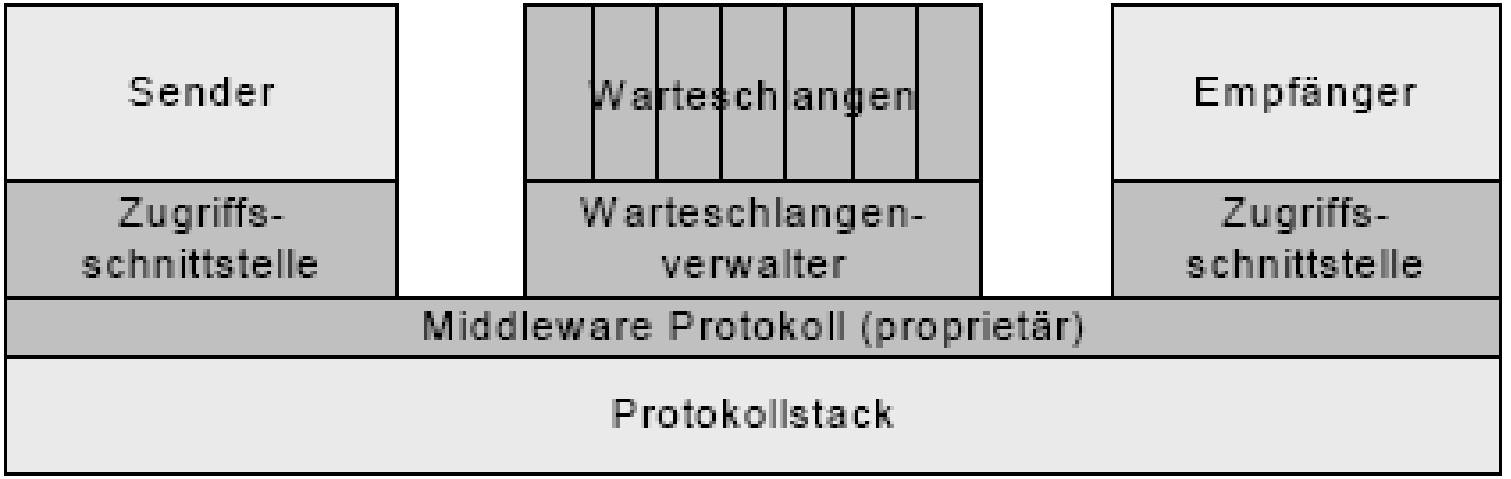
- Das Verschicken einer Nachricht (Request) und das Empfangen einerAntwortnachricht (Reply) erfolgen als eine Einheit (nur wenn beide erfolgreich ablaufen, war auch die Kommunikation erfolgreich).
- Ermöglicht eine quasi-synchrone Kommunikation über eine asynchrone Middleware.
- Eine mögliche Umsetzung erfolgt über unterschiedliche Warteschlangen für Request und Reply Nachrichten.

Publish-Subscribe Model



- Prozesse erhalten Rollen als Publisher bzw. Subscriber.
- Subscriber abonnieren Nachrichten zu einem Thema.
- Publisher veröffentlichen Nachrichten zu einem Thema.
- Broker übernehmen die Vermittlung.
- Eine mögliche Realisierung erfolgt durch themenspezifische Warteschlangen.

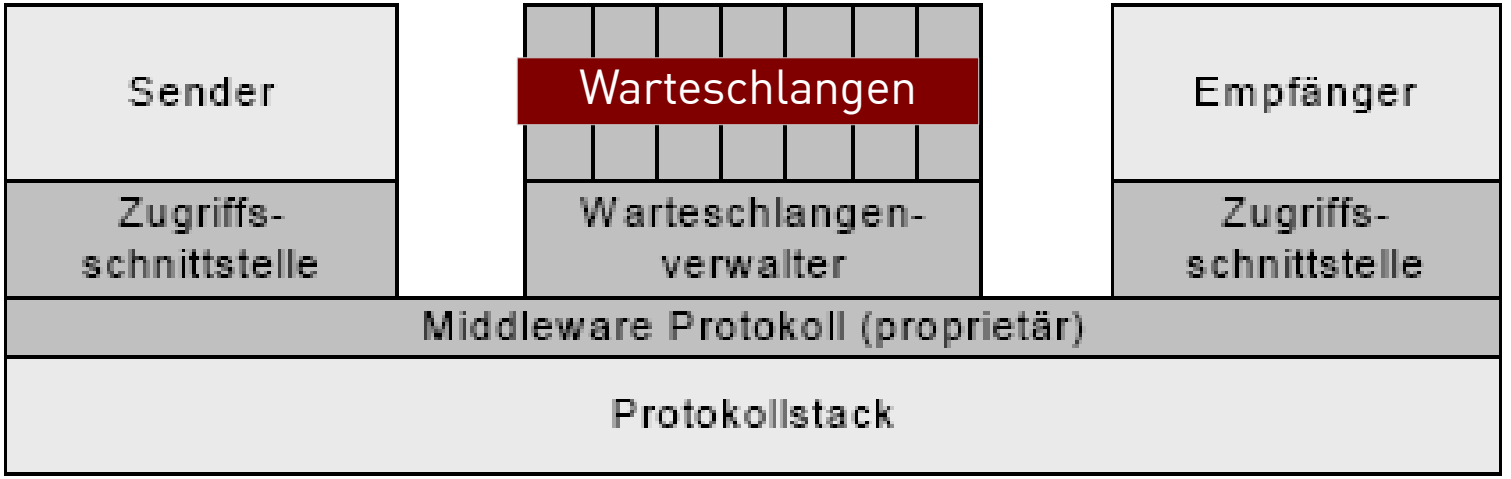
Message Oriented Middleware (MOM) Architektur



- Auch *nachrichtenorientierte Middleware* genannt.
- MOM ist die Middleware-Technologie zum Message-Passing-Programmiermodell.

- MOM bietet neben der rein asynchronen Kommunikation weitere Mechanismen und Dienste:
  - Unterstützung der verschiedenen Messaging Modelle
  - Warteschlangenverwaltung
  - Verbindungsmanagement
  - Quality-of-Service Zusicherungen (QoS)

Persistente Warteschlangen

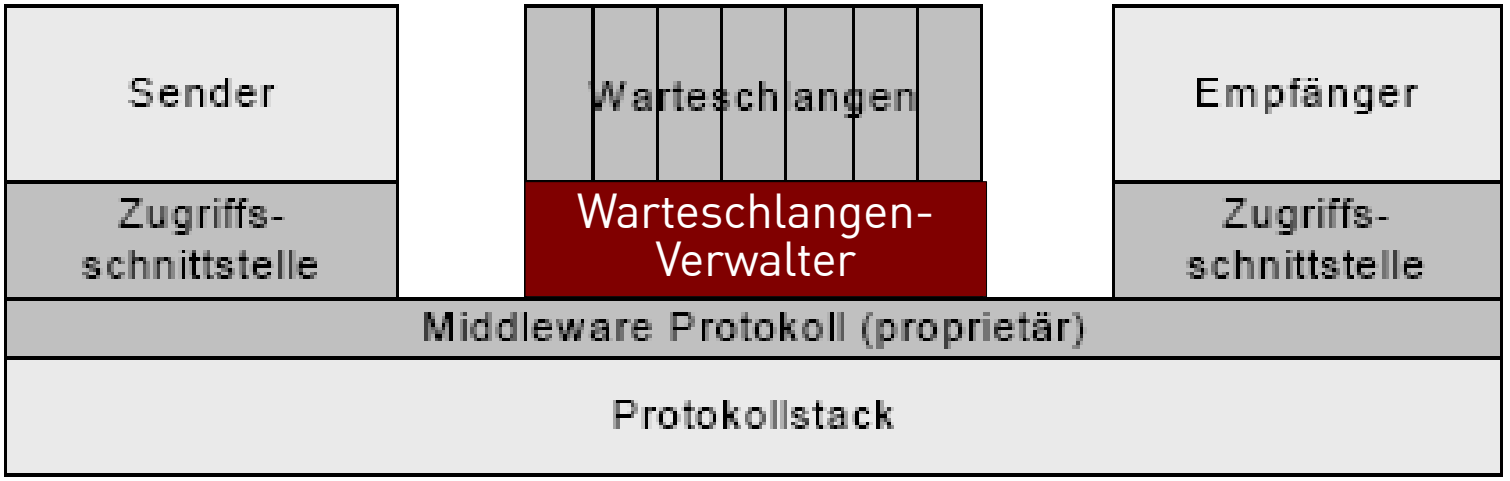


- Eine wesentliche Quality of Service Eigenschaft ist die **garantierte Auslieferung** einer Nachricht.
- Erreicht durch das persistente Zwischenspeichern von Nachrichten.
  - z.B. Datenbank, Dateisystem, ...

- Garantierte Auslieferung
  - stellt sicher, dass eine Nachricht an den Empfänger ausgeliefert wird.
  - sagt jedoch nichts über den Zeitpunkt der Auslieferung aus.

    (Dies kann über andere QoS Eigenschaften sichergestellt werden).

Warteschlangenverwalter (Queue Manager)

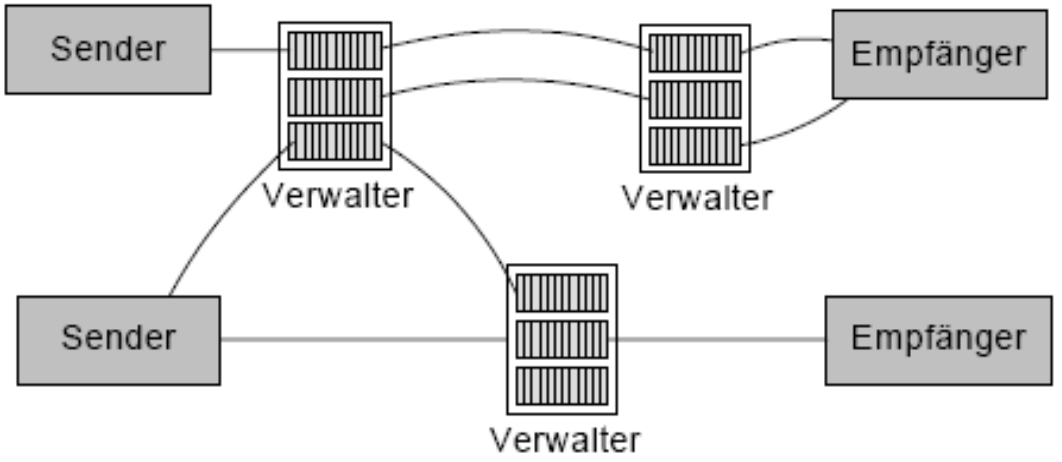


- Häufig existieren in einem MOM-Server mehrere parallele Warteschlangen unterschiedlicher Anwendungen.
- Verantwortlich für die Verwaltung der Warteschlangen ist ein Warteschlangenverwalter.

- Dieser hat folgende Aufgaben:
  - Zuordnung von ankommenden Nachrichten zu Warteschlangen
  - Überwachung der Nachrichten (Größe, Format)
  - Überwachung und Sicherstellung der QoS-Eigenschaften (Timeout, Priorität, ...)
  - Überwachung des Lebenszyklus einer Warteschlange
  - Füllungsgrad der Warteschlangen

Warteschlangeninfrastruktur

- Der Zugriff auf eine Warteschlange (lesend, schreibend) kann immer nur lokal erfolgen (im gleichen Prozess, im gleichen Subnetz).
- Für alle anderen Szenarien ist eine komplexe **Warteschlangen-infrastruktur** notwendig.
- Nachrichten werden zwischen Warteschlangen unterschiedlicher Verwalter weitergereicht. Verwalter dienen als Router für Nachrichten.



Beispiel: HPC, PVM und MPI (1)

- Nachrichten-orientierte Programmierung kommt häufig in **HPC – High Performance Computing –** hoch-performante, rechner-intensive Anwendungen vor.
- Zwei viel verwendete Bibliotheke sind:
  - **PVM** – Parallel Virtual Machine
  - **MPI** – Message Passing Interface
- Beide bestehen aus (i.d.R.):
  - eine **Schnittstelle-Bibliothek**
  - einen **Dämon** (Warteschlagne-Verwalter bzw. -Vermittler).

- **PVM:**
  - akademisch,
  - open-source,
  - einfache Menge von Funktionen:
    - → Marshalling,
    - → Demarschalling,
    - → Senden und
    - → Empfangen
  - **Schwerpunkt:** *Interoperabilität* (Offenheit gegenüber Heterogenität).

Beispiel: HPC, PVM und MPI (2)

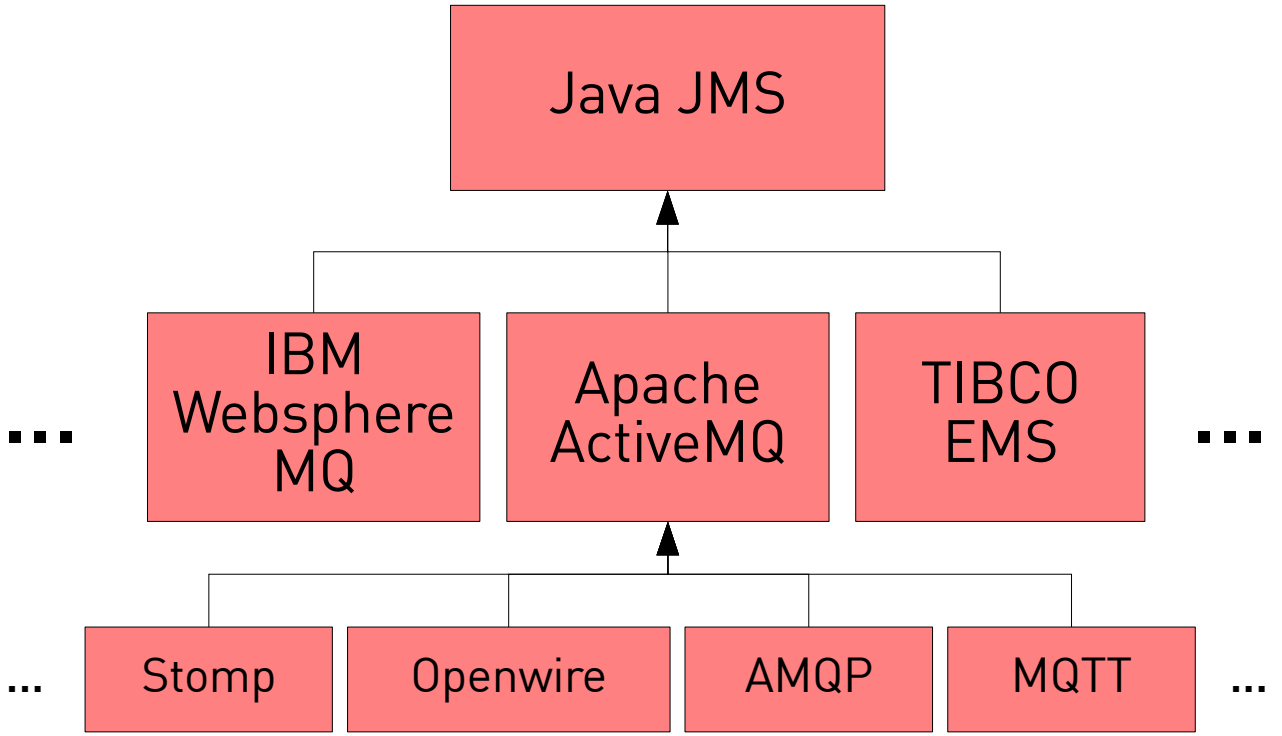
- **MPI:**
  - ein Standard mit verschiedene industrielle bzw. open-source Realisierungen,
  - einfache Menge von Funktionen (wie PVM, s.o.)...
  - ... + extra Fuktionalität für Fortgeschrittene (z.B. Multicasting).
  - **Schwerpunkt:** *Performanz*, besonders für *Parallelrechner*

Primitive	Bedeutung
MPI_bsend	Die ausgehende Nachricht an einen lokalen Sendepuffer anhängen
MPI_send	Senden einer Nachricht und Warten, bis sie in den lokalen oder entfernten Puffer kopiert wurde
MPI_ssend	Nachricht senden und warten, bis der Empfang startet
MPI_sendrecv	Nachricht senden und auf Antwort warten
MPI_isend	Referenz an die ausgehende Nachricht übergeben und weitermachen
MPI_issend	Referenz an die ausgehende Nachricht übergeben und warten, bis der Empfang startet
MPI_recv	Nachricht empfangen; blockieren, wenn es keine gibt
MPI_irecv	Überprüfen, ob es eine eingehende Nachricht gibt, aber nicht blockieren

Einige der intuitivsten Primitive zur Nachrichtenübergabe bei MPI

Beispiel: Java Messaging Service (JMS)

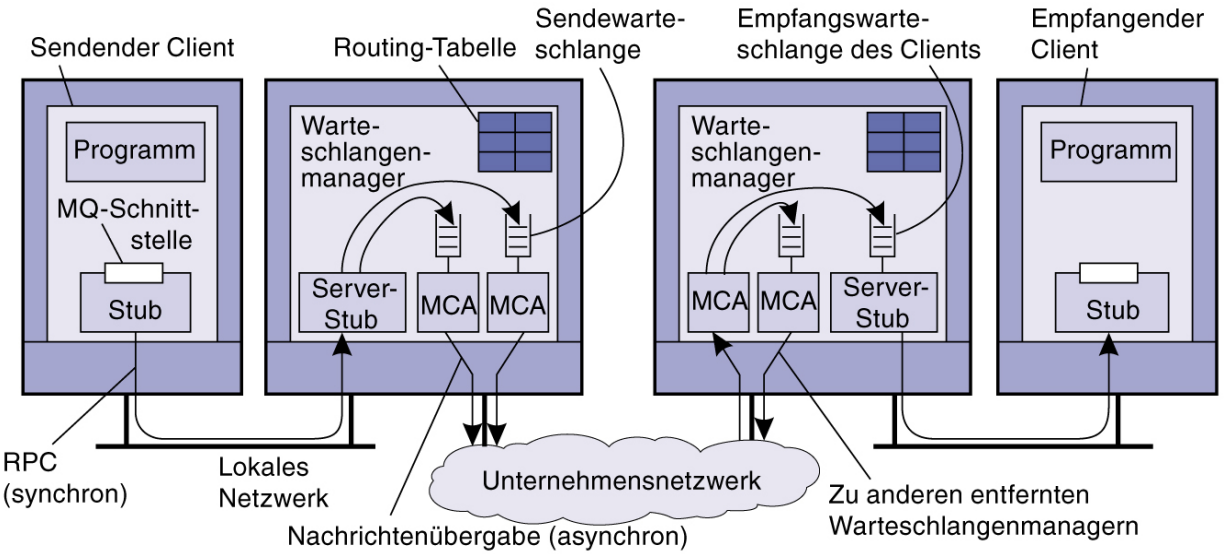
- **JMS = Java Messaging Service:**
  - Eine Bibliothek, die eine Schnittstelle für verschiedene "enterprise" MOM Frameworks an bietet
  - Manche Realisierungen unterstützen wiederum mehrere "Wire Protocols"
  - **Hauptfokus:** Arbeit mit Heterogenität (Diversität)



Beispiel: IBM WebSphere MQ (1)

- Bekanntester MOM Server ist **WebSphere MQ** (früher **MQSeries** genannt) von IBM.
- Kann als Quasi-Standard betrachtet werden.
- Alle Warteschlangen werden von *Warteschlangen-Manager* verwalten.
- *Warteschlangen-Manager* kommunizieren paar-weise über *Message Channel Agents* (MCAs).

  Es gibt sendende und empfangende MCAs



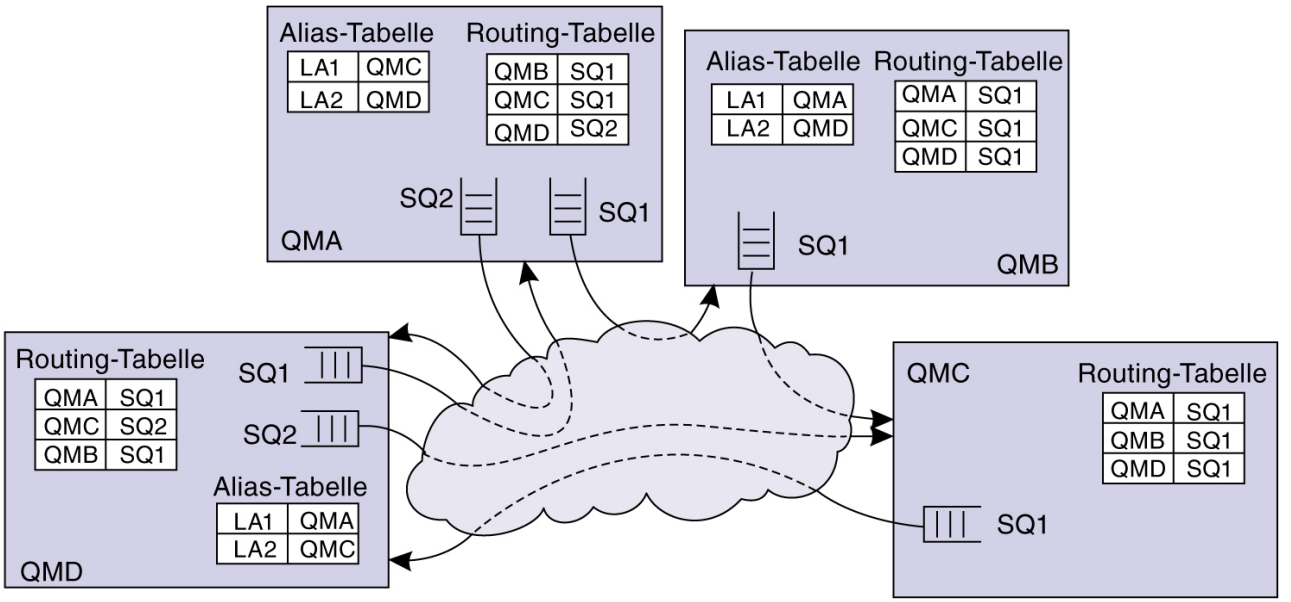
Allgemeiner Aufbau des Warteschlangensystems von IBM

- Ein *Warteschlangen-Manager* kann ein Teil vom Anwendungsprozess sein...
- ...oder ein eigener Prozess auf einen anderen Rechner (in dem Fall wird RPC verwendet – vgl. Abbildung).

In dem Fall: Wo ist der Unterschied zwischen MOM & RPC? Wie wird es *asynchron*?

Beispiel: IBM WebSphere MQ (2)

- Jeder *Warteschlangen-Manager* hat eine eindeutige Name...
- ...kann aber auch über ein *Alias* angesprochen werden.
- Warteschlangenmanager müssen also eine Routing-Tabelle und eine Alias-Tabelle verwalten.



Der allgemeine Aufbau eines MQ-Warteschlangennetzwerkes unter Verwendung von Routing-Tabellen und Aliasen

Beispiel:

- Eine Anwendung ist mit Manager QMA angebunden, möchte LA1 (Local Alias) eine Nachricht schicken.
- Alias-Tabelle: LA1 = QMC.
- Routing-Tabelle: QMC kann über SQ1 erreicht werden (indirekt).
- Die Nachricht geht via QMA's SQ1 an QMB
- QMB's Routing-Tabelle: QMC kann über SQ1 erreicht werden (direkt).
- Die Nachricht geht via QMB's SQ1 an QMC (fertig!)

Beispiel: IBM WebSphere MQ (3)

Vorteil:

- Einfache Schnittstelle

Primitive	Beschreibung
MQopen	Eine (möglicherweise entfernte) Warteschlange öffnen
MQclose	Eine Warteschlange schließen
MQput	Eine Nachricht in eine geöffnete Warteschlange stellen
MQget	Eine Nachricht aus einer (lokalen) Warteschlange abrufen

Nachteil:

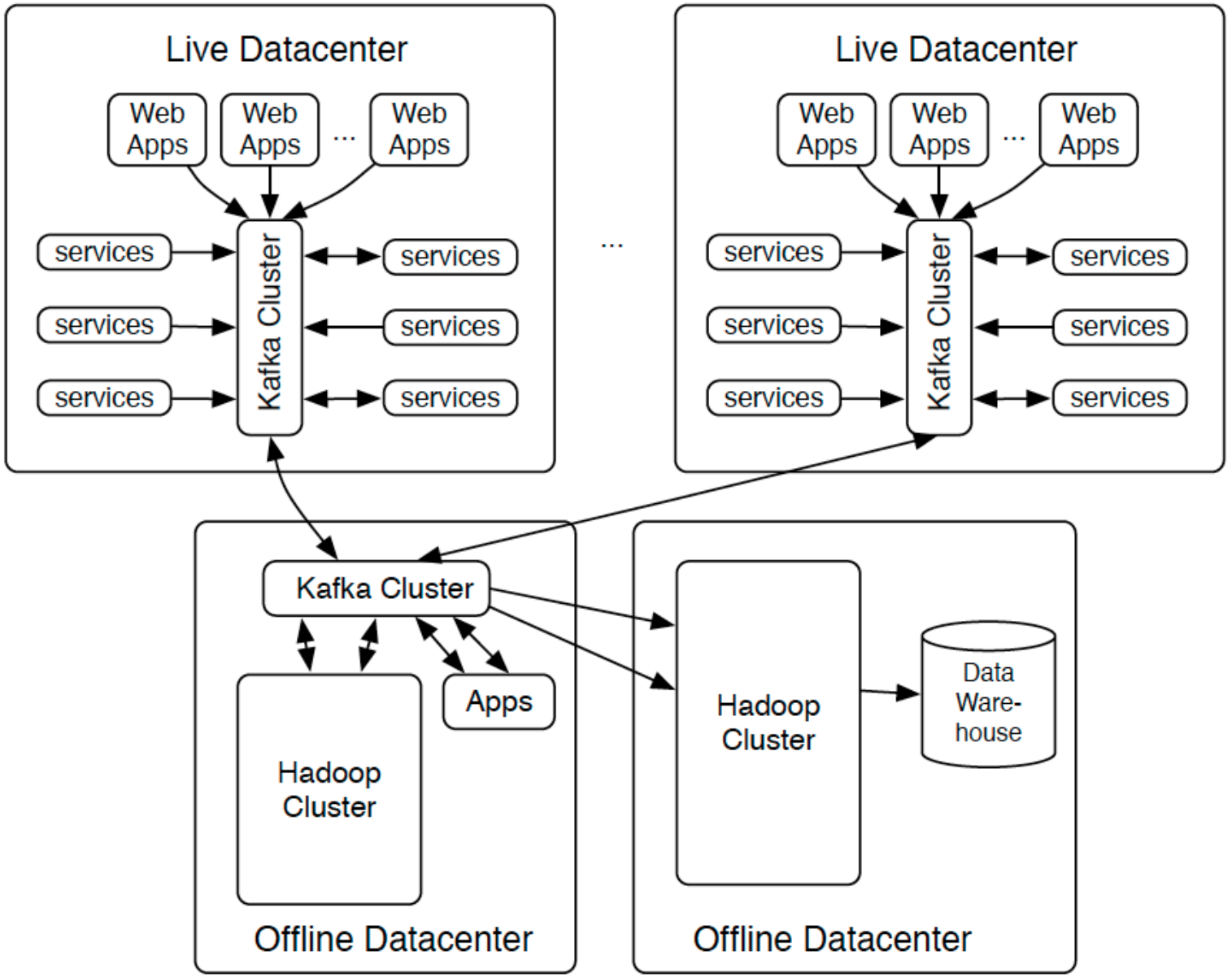
- Verwaltung der *Overlay-Networks* schwierig

Beispiel: Apache Kafka

- Sehe http://www.kafka.apache.org
- Ursprünglich entwickelt von LinkedIn
- Ziele
  - Echtzeit (!) Monitoring und Logging von Nutzeraktivitäten und Servern (z.B. als Früherkennungssystem für Probleme mit dem Dienst)
  - Performance
    - bis zu 172.000 eingehende Nachrichten pro Sekunde
    - bis zu 55 Milliarden ausgehende Nachrichten pro Tag

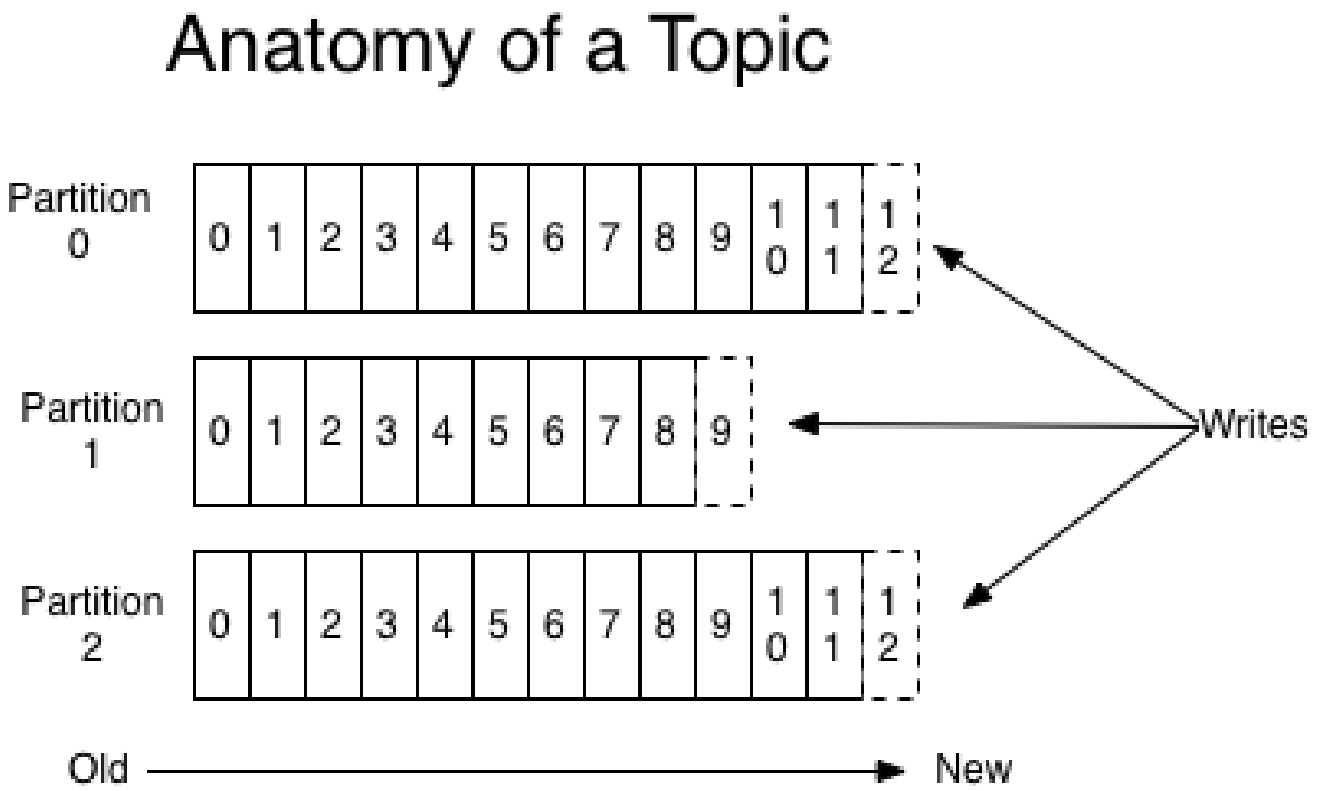
h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Beispiel: Apache Kafka



Quelle: Goodhope et.al.: Building LinkedIn's Real-time Activity Data Pipeline, IEEE Data Eng. Bull. 35(2):33--45

Beispiel: Apache Kafka

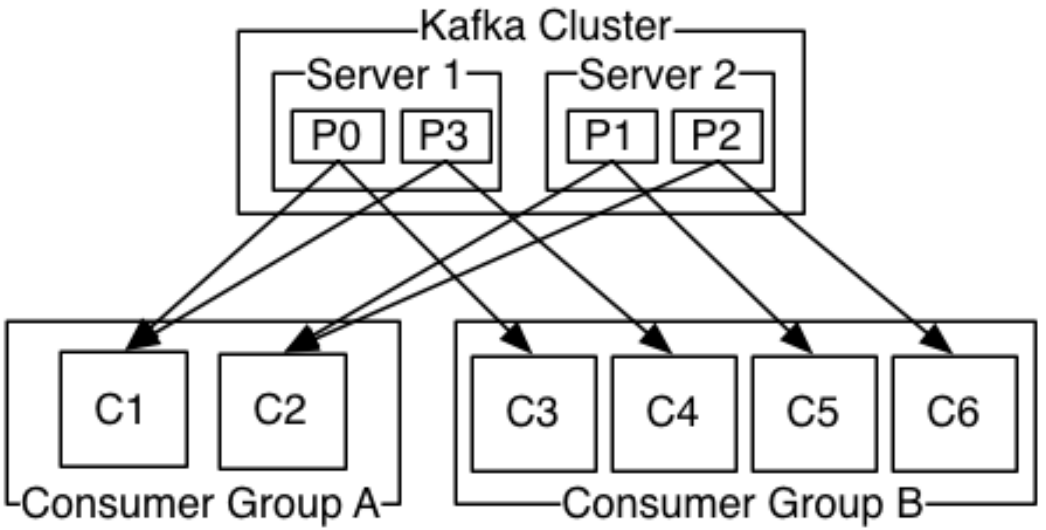


Quelle: http://www.kafka.apache.org

Pro Thema (*topic*) werden eingehende Nachrichten in Partitionen aufgeteilt (z.B. mittels eines Schlüssels, durch Hashing)

Beispiel: Apache Kafka

Kafka arbeitet als Cluster



Quelle: http://www.kafka.apache.org

- Ein Server ist für mehrere Partitionen zuständig
- Jede Partition wird auf mehrere Server repliziert (!) *Ein Master, mehrere Helpers*
- Ordnung der ausgelieferten Daten wird nur pro Partition garantiert!
  - Ein Subscriber bekommt Nachrichten zu einem Thema möglicherweise aus mehreren Partitionen
  - Keine globale Ordnung mehr
- Subscriber können in Consumer Groups aufgeteilt werden: Subscriber arbeiten auch im Cluster

Struktur von Kapitel III - Middleware

I Einleitung

II Grundlegende Kommunikationsdienste

III Middleware

- A Einführung / Überblick
- B Web Services & REST
- C Message-Oriented Middleware (MOM)
- D Objekt-orientierte Middleware (CORBA)

IV Architekturen & Algorithmen

- A Synchronisierung
- B Konsistenz und Replication
- C Fehlertoleranz

V Beispiele bzw. Dienste

- A Verteilte Dateisysteme
- B Namensdienste

VI Sicherheits & Sicherheitsdienste

VII Zusammenfassung

Objekt-Orientierte Middleware

- **Motive** – Objekt-Orietierung für:
  - Weniger Komplexität
  - Wiederverwendung
  - Leichtere Entwicklung
- Objekt-Orientierung heißt:
  - Objekt, Klasse, Methode
  - Vererbung, Polymorphismus
  - Kapseln von Daten und Operationen
  - Instanzen, nach Bedarf erzeugt

- **Motive** – Warum OO für *verteilte Systeme*?
  - Verteilte Anwendungen lassen sich mit OO leichter strukturieren und in Teilkomponenten zerlegen
  - Einheitliche Methodik des Operationsaufrufs durch Nachrichten (Request/Reply).
- **Beispiele:**
  - Java RMI (s.o.).
  - Enterprise Java Beans
  - Tannenbaum's Globe
  - **OMG's CORBA**

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Object Management Group (OMG)

- Initiator und Verwalter des CORBA Standards
- 1989 gegründet
  - Nicht-kommerzielles Standardisierungsgremium
  - Ziel: Standardarchitektur für verteilte objektorientierte Anwendungen, die Object Management Architecture (OMA).
  - Über 800 Mitglieder, vor allem aus Unternehmen (IBM, Bea, IONA, etc).
  - ...aus unterschiedlichen Bereichen: Telekommunikation, Internet, Medizin, Finanzen, Datenbanken, Echtzeit-Programmierung u.s.w.
- Heute arbeitet die OMG neben den verschiedenen OMA Standards an weiteren Standards:
  - UML (Unified Modeling Language)
  - XMI (XML Metadata Interchange)
  - MDA (Model Driven Architecture)

Common Object Request Broker Architecture (CORBA)

CORBA umfasst:

- objektorientierter RPC-Mechanismus (Remote Procedure Call)
- Object Request Broker (Vermittler) = ORB
- Object Services (z.B. Name Services)
- Language mappings für verschiedene Programmier-sprachen
- Interoperable Protokolle
- Richtlinien zur Programmierung

- CORBA ersetzt spezialisierte **Mechanismen** durch eine offene, skalierbare und portierbare **Plattform**

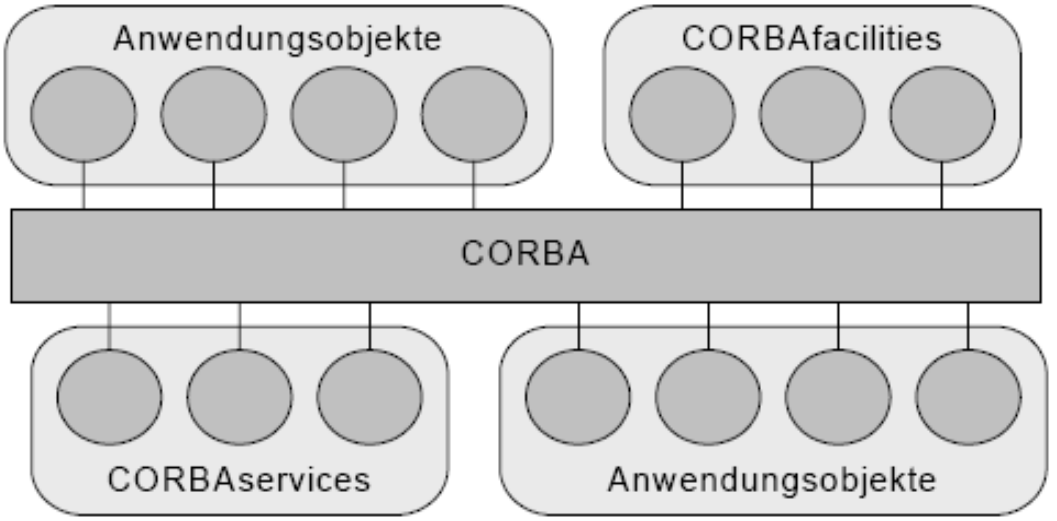
Historie:

- Erste stabile Version: CORBA 1.1 im Jahre 1991
  - Probleme: fehlende Interoperabilität zwischen verschiedenen ORB Herstellern
- CORBA 2.0 im Jahre 1997
  - Einführung eines gemeinsamen Protokolls, das IIOP (Internet Inter ORB Protokoll)
  - Überarbeitung bestehender Standards
- CORBA 3.0 im 2003
  - Einführung des CORBA Component Models
  - Spezifikation des Persistenzdienstes
  - Einführung einer Firewall Spezifikation

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

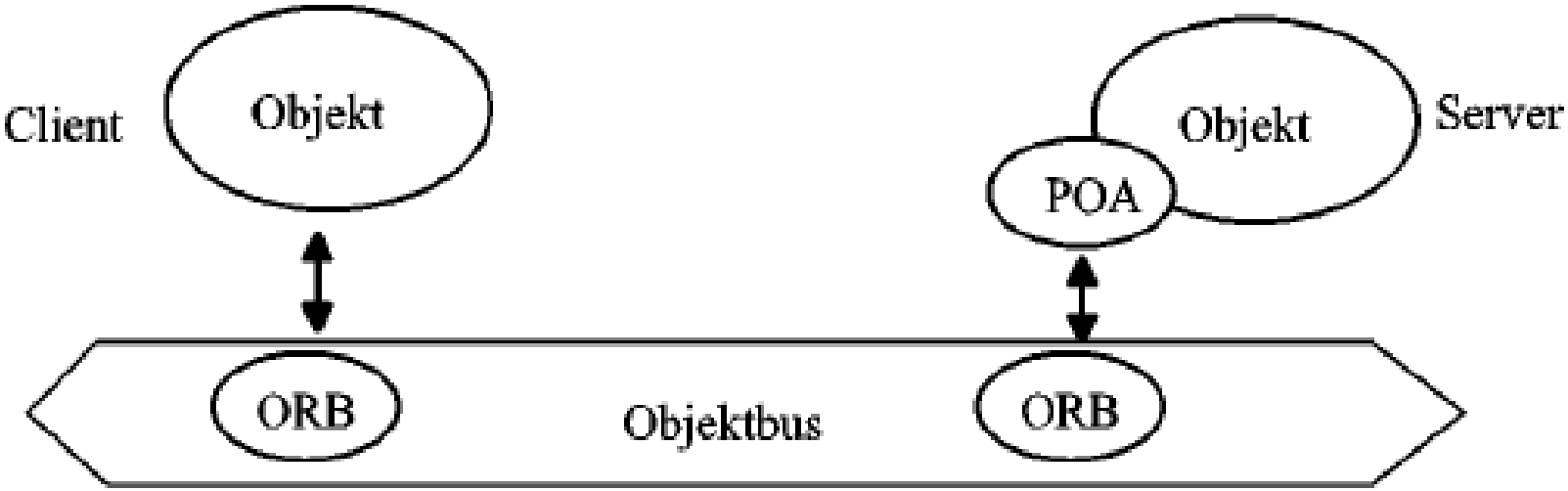
Object Management Architecture (OMA)

- Die OMA legt den Rahmen der verschiedenen Standards in diesem Bereich fest, sie ist nicht selbst standardisiert.
- Die Standards lassen sich in drei Gruppen einteilen:
  - **CORBA:** definiert die Basisinfrastruktur zur Kommunikation
  - **CORBAservices:** definieren verschiedene technische Dienste für Anwendungen
  - **CORBAfacilities:** definieren fachbezogene Dienste für Anwendungen (finden eher selten Verwendung in der Praxis).



- Zentrale Aufgabe von CORBA ist die **Abstraktion** von den folgenden Fragen:
  - Welche Programmiersprache (Implementierung) wird verwendet?
  - Wo ist das Server-Programm abgelegt?
  - Wie wird ein Server-Programm bei Bedarf aktiviert?
  - Welches Kommunikations-Protokoll, welches Message Passing wird verwendet?

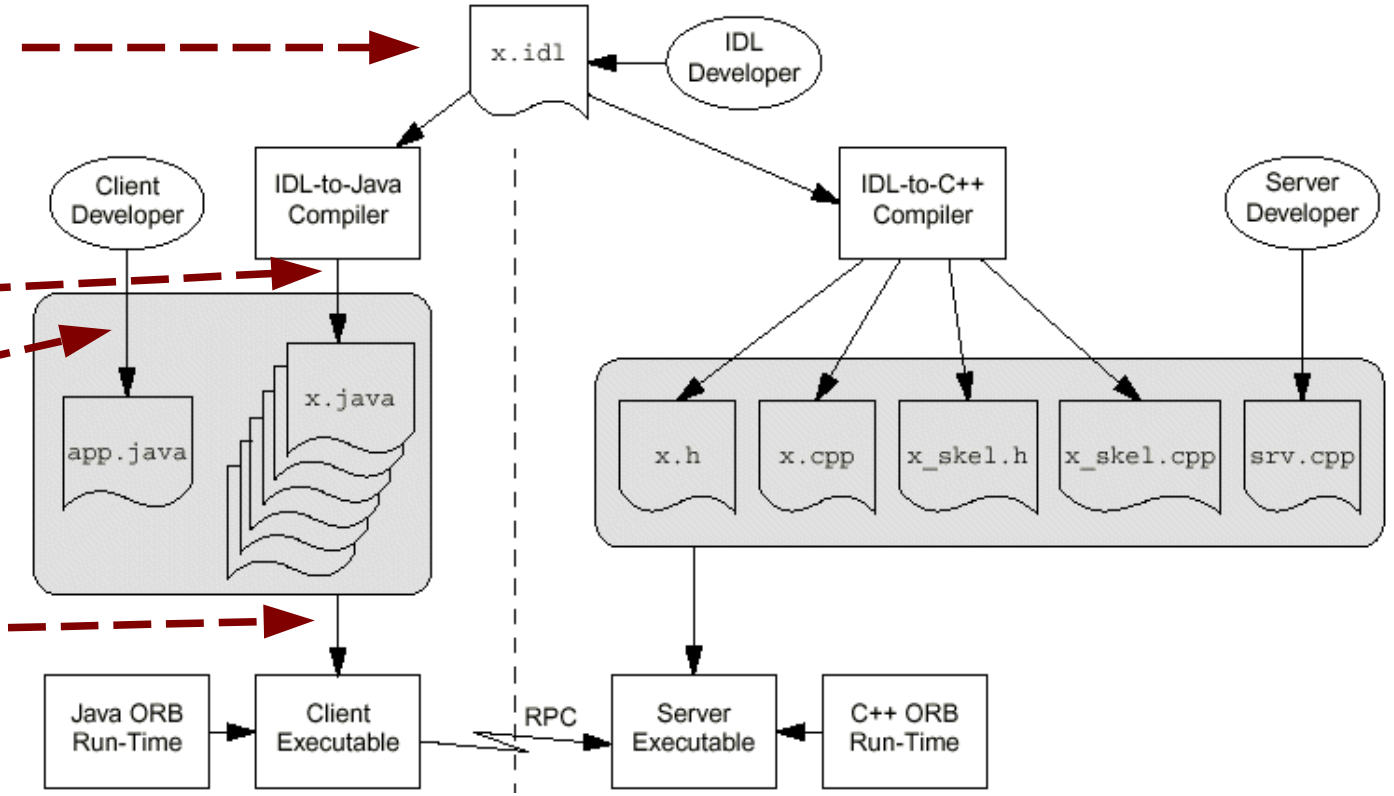
Objektbus



- ORB: Object Request Broker (Vermittlung der Nachrichten)
- POA: Portable Object Adapter (Life Cycle Management)

Entwicklungsprozess

1. Schnittstellen in IDL (Interface Description Language) schreiben.
2. Skeletons und Stubs erzeugen lassen.
3. Server und Client programmieren, ggf. Stubs anpassen.
4. Compilieren (für Java und C++ usw).
5. Interfaces und Implementations in Repositories speichern.
6. Laufen lassen!



OMG IDL (Interface Description Language)

- Beispiel:

```
module finance {

  interface account {

    readonly attribute string owner;

    attribute float balance;

    void deposit ( in float amount,
                   out float newBalance);

    void withdraw (in float amount,
                   out float newBalance);

    float balance( );

  };

};
```

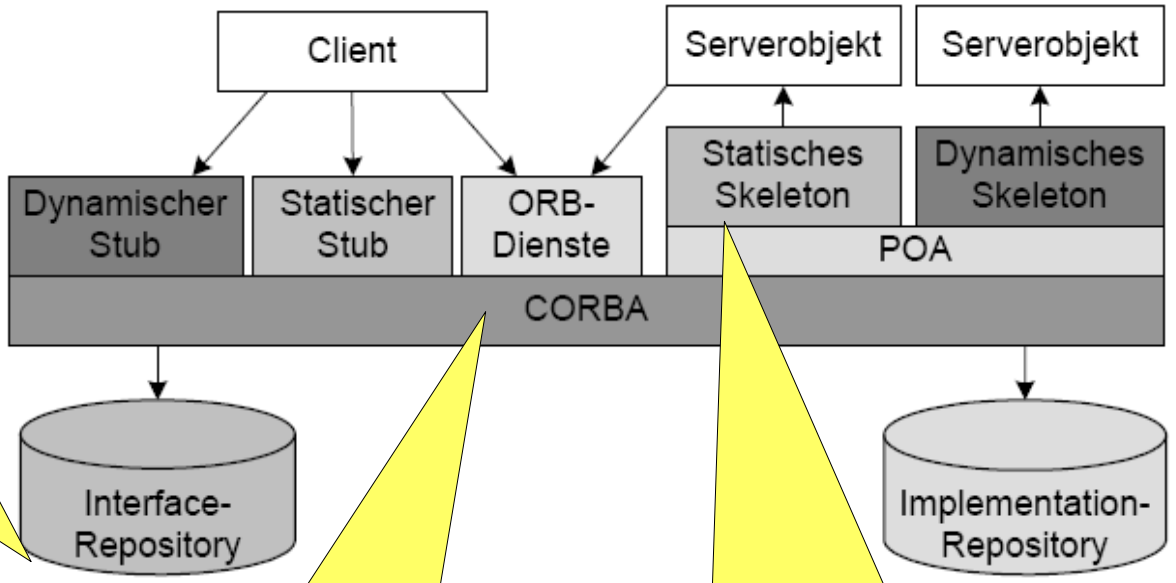
- Module ≈ Namespaces
- Interfaces ≈ Java Interface ≈ Abstract C++ Classes
- Attributes ≈ Felder eine Klasse – sind aber Methode!
- Operationen ≈ Methoden
- Argumente können in, out oder inout sein (Rückgabewerte sind implizit out).

Abbildung: IDL Java

IDL	Java
IDL Module *Foo*	package *Foo*
IDL Interface *Bar*	Java Interface Klasse: *Bar.java* Holder Klasse: *BarHolder.java* Helper Klasse: *BarHelper.java*
IDL Attribute	Zugriffsmethoden auf Attribute (Lese- und ggf. Schreibzugriff)
IDL Operationen	werden auf Java Klassenmethoden abgebildet. Für *inout* und *out* Parameter werden Holderklassen generiert.
Enum, Struct, Unions	Werden auf Java Klassen abgebildet, die die jeweilige Semantik nachbilden
Sequenzen, Arrays	Sequenzen und Arrays werden auf Java Arrays abgebildet. Für beide werden Holderklassen generiert.

Holderklassen sind Container für Datentypen, ermöglichen inout und out Parameter Übergabe. Sind bereits für alle Basistypen in den ORBBibliotheken da; werden für benutzerdefinierte Typen vom IDL Compiler generiert.

CORBA Architektur

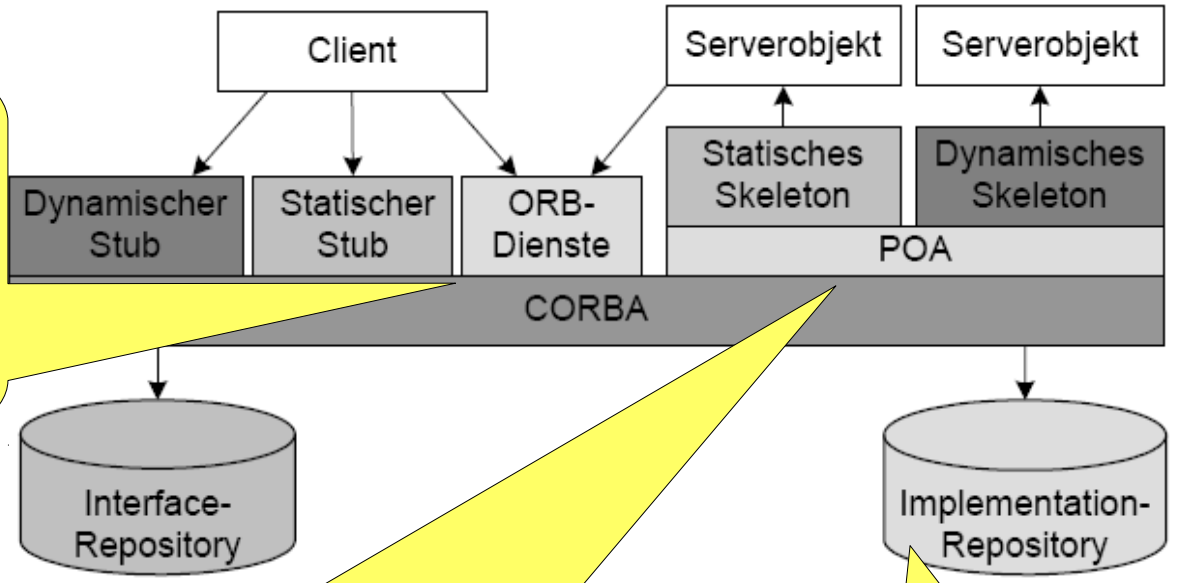


Interface Repository
– Unterstützt dynamische Stubs.
– Verwaltet Schnittstellen von Objekten, die zur Laufzeit gebunden werden können.

Object Request Broker
– Kern der Architektur
– Transparente Vermittlung von Aufrufen zwischen Objekten.
– Benötigt Objektreferenzen zur Adressierung der Objekte.
– Referenzen werden im Namensdienst veröffentlicht.

Stubs / Skeleton
– Schnittstelle zwischen Broker und Anwendungsobjekten.
– Kernaufgabe = Marshalling, Unmarshalling, Vermittlung
– Es gibt zwei Arten Aufrufe:
• *statische*: Bindung von Objekt an Stub zur Compilezeit
• *dynamische*: Bindung zur Laufzeit

CORBA Architektur *fortgesetzt*



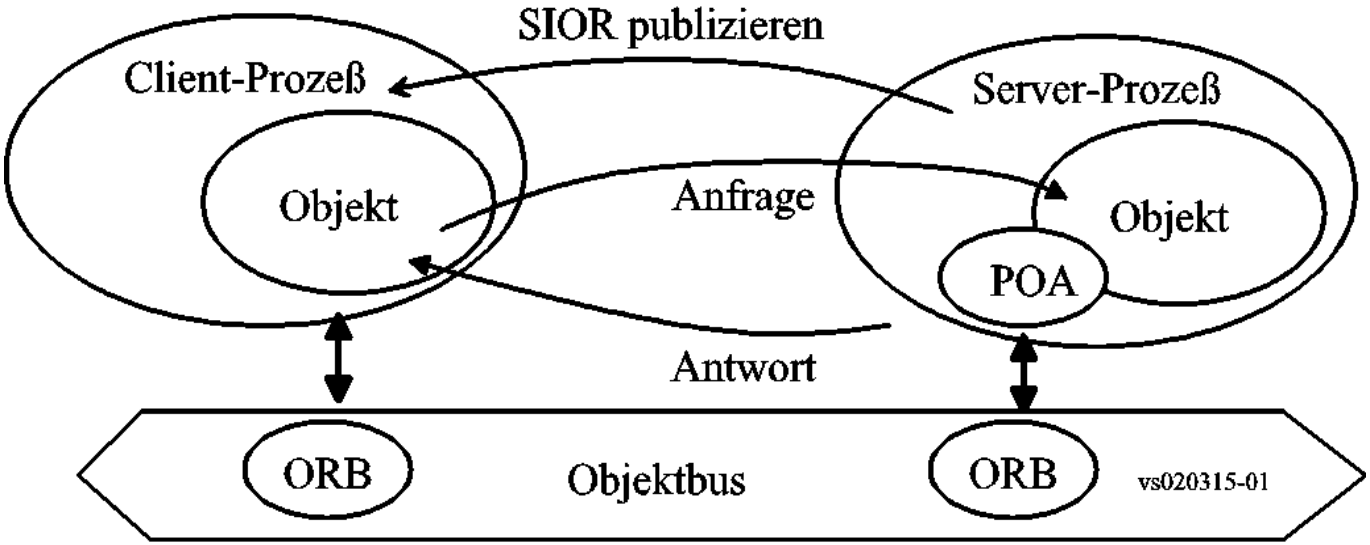
ORB Dienste
– Schnittstelle zu Diensten des ORB für Objekte
– Beispielsweise
• Zugriff auf Standard-dienste wie POA oder Namensdienst.
• Initialisierung des ORB

Object Adapter
– Verwaltet Serverobjekte in ihren verschiedenen Lebensphasen (initialisieren, aktivieren, passivieren, löschen)
– Generiert eindeutige Objektreferenzen zu Serverobjekten
– Nimmt Aufrufe vom ORB entgegen und sucht entsprechendes Skeleton
– Ursprünglich Basis Object Adapter (BOA), seit CORBA 2.2 Portable Object Adapter (POA):
• Vollständige Standardisierung der Adapter Schnittstelle
• Dadurch Portabilität der Anwendungen möglich

Implementation Repository
– Unterstützt Object Adapter bei der Verwaltung der Serverobjekte
– Enthält notwendige Informationen zur Initialisierung und Aktivierung

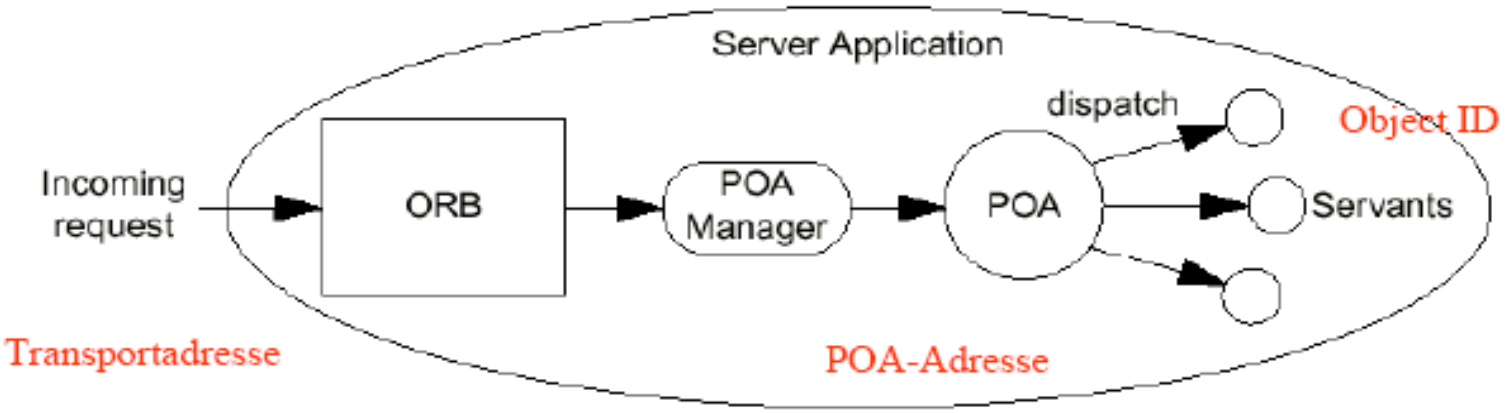
Client und Server finden zusammen

1. Client und Server werden als eigenständige Prozesse gestartet.
2. Client und Server finden sich über **interoperable Objektreferenz (IOR).**
3. Der Server erzeugt eine Objekt-Referenz und gibt diese in druckbarer Form aus: **SIOR (Stringified Interoperable Objectreference)**



4. Client bezieht jede Anfrage an den Server auf diese Objektreferenz.
5. Der ORB des Client kennt den Aufbau der IOR und adressiert den richtigen Ziel-ORB.

Verwalten und Verteilen der Client-Anfragen



- Eintreffende Anfragen werden direkt zu einem ORB geleitet.
- Falls der ORB die Anfrage akzeptiert, wird diese dem POA-Manager übergeben.
- Ist der POA-Manager aktiv, gibt er die Anfrage weiter zu einem POA, der diese wiederum an den richtigen Servant leitet.
- *Servants* sind die lokale Objekte, die die Arbeit leisten. Ein Servant kann 1 bis *n* sichtbare Objekte realisieren.

ORB und POA

Object Request Broker - ORB

- Verbindet die vorhandenen *Interfaces*. Ermöglicht die Kommunikation zwischen Clients und Objekten
  - Per CORBA-ORB können Software-Komponenten mit Hilfe verschiedener Dienste über Plattformgrenzen hinweg kommunizieren.
- Aktiviert transparent Objekte, wenn Anfragen an diese gerichtet werden.
- Ein ORB fängt den Aufruf eines Client ab (Proxy), ist verantwortlichfür:
  - Objekt-Vermittlung,
  - Parameter-Übergabe
  - Methode-Invokation
  - Rückgabe der Resultate.

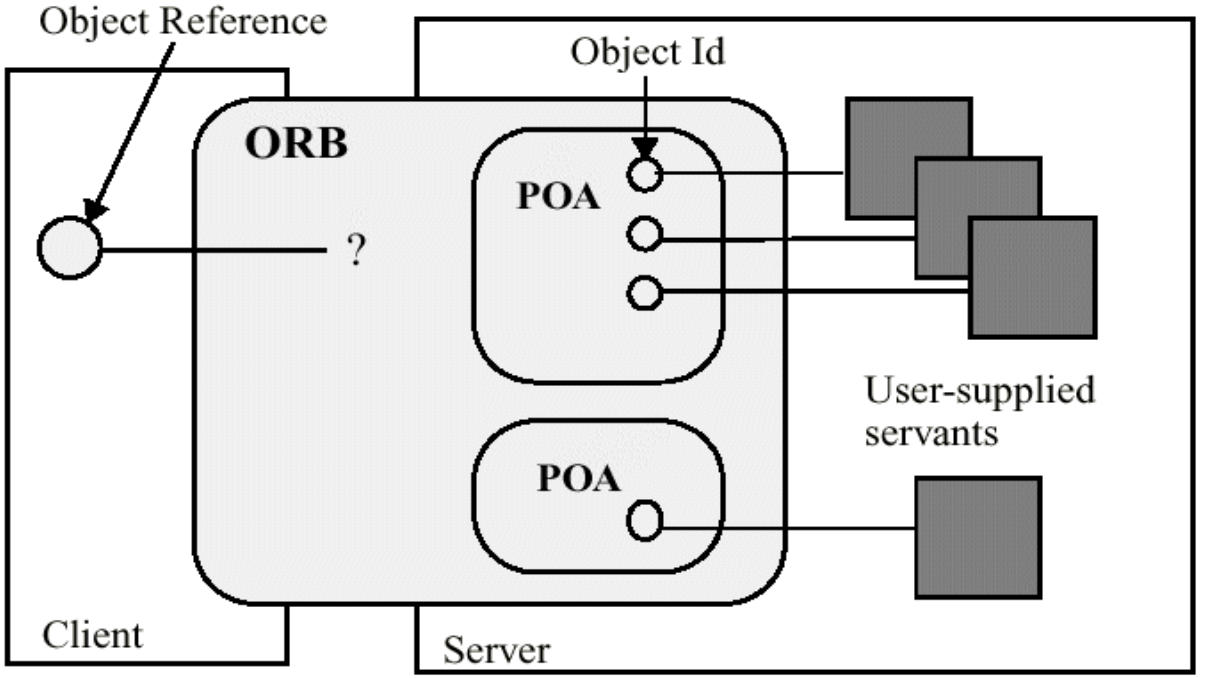
Portable Object Adapter - POA

- Realisierung von Objekt-Implementierungen, die portabel sind zwischen verschiedenen ORB-Produkten
- Entlastung des ORBs von der Verwaltung der Objekte (Identität, Zustand, Lebensdauer)
- Gleichzeitige Existenz mehrerer POA-Instanzen in einem Server
- Objekte mit persistenten Identitäten versehen
- Unterstützung transparenter Aktivierung von Objekten
- Realisierung statischer und dynamischer Schnittstellen

ORB und POA (2)

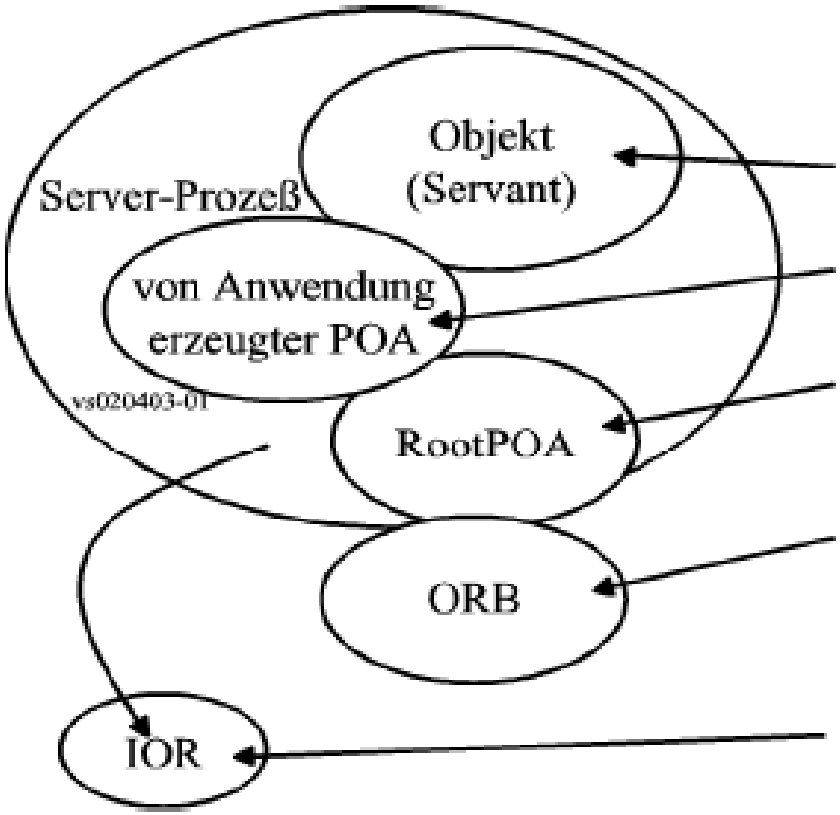
Portable Object Adapter

- Macht registrierte Servant-Objekte den Clients zugänglich.
  - bildet eintreffende Anfragen auf Konstrukte der Programmiersprache wie Objekte oder Prozeduren ab
  - kennt die existierenden Objekte
  - verwaltet Life Cycle (z.B. definiertes Löschen eines Servants)
- Überbrückt die Grenze zwischen IDL und konkreter Programmiersprache



- POA ist nur dem Server sichtbar (nicht dem Client).
- User-supplied Servants werden bei einem POA registriert.
- POA befaßt sich mit Object ID und aktivem Servant.

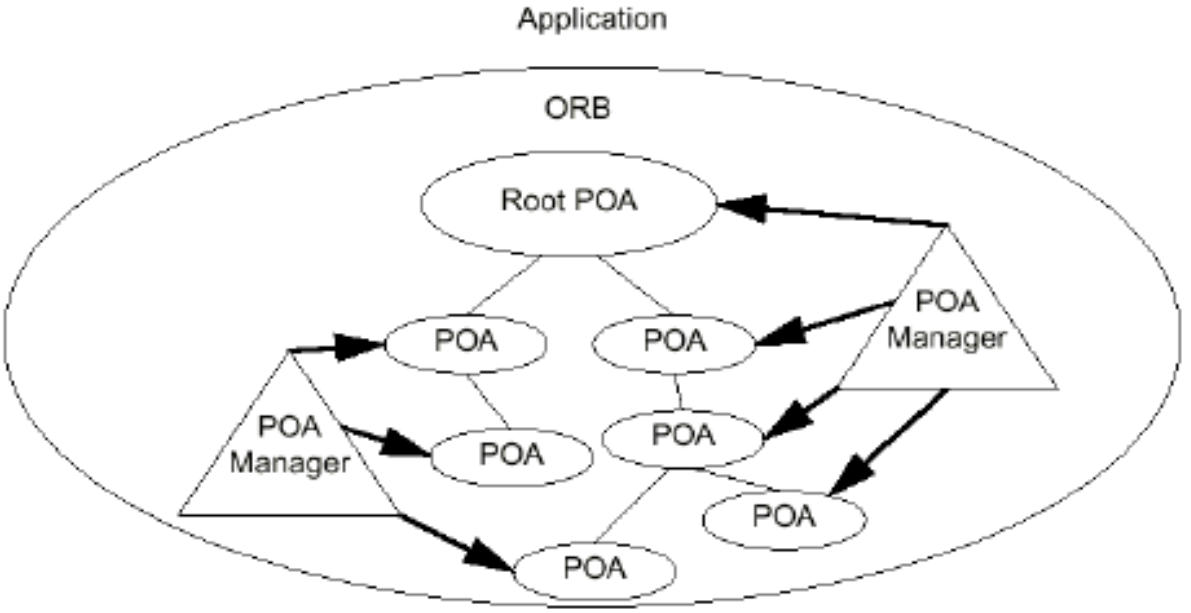
Persistente Objektreferenz



3. Objekt erhält einen vom SW-Entwickler festgelegten Namen
2. Beliebige Policies möglich, z.B. persistente IOR, vom Benutzer erzeugte Objekt-ID
1. Feste Policies, z.B. transiente Objektreferenz, automatische Vergabe der Objekt-ID
4. ORB erhält beim Start die Information, sich auf einen festen Port zu binden: -ORB_iiop_port 4711
5. IOR ist bei jedem Start des Servant identisch

POA-Manager und POA-Hierarchie

- Oft wird für jedes Interface des Servers ein eigener POA verwendet.
- Die POA-Hierarchie bestimmt die Reihenfolge, wie Objekte beim Herunterfahren des ORB gelöscht werden.
- Mehrere POA-Manager erlauben, die Verarbeitung von Anfragen getrennt für verschiedene Objektgruppen zu steuern.



Beispiel „bank“ (Serverobjekt)

```
module bank
{
  interface Account
  {
    void deposit (in float amount,
                  out float newBalance );
    void withdraw (in float amount,
                  out float newBalance );
     };
};
```

- Mit dem IDL Compiler werden aus der IDL Schnittstelle Stubs und Skeletons in der entsprechenden Sprache generiert.

Beispiel „bank“ (die Schnittstelle)

```
package bank_a;
public class Account extends AccountServerPOA
{
    float deposit (float amount) { … }
    float withdraw (float amount) { … }
}
```

- Am Server werden das Serverobjekt selbst, sowie eine Serverimplementierung benötigt.
- `Account` ist das Serviceobjekt, auf das Clients zugreifen um den `bank_a` Service zu nutzen.
- Die Einbettung des Serviceobjekts in das CORBA System erfolgt über die Superklasse `AccountServerPOA`.

Beispiel „bank“ (der Server 0)

```
public class AccountServer {
 public static void main( String[] args ) {
 try {
   org.omg.CORBA.ORB orb =
                org.omg.CORBA.ORB.init(args,null);
   ...
 } catch …
}
```

Beispiel „bank“ (der Server 1)

Schritt 1: Initialisierung des ORBs und des POAs:

Der ORB wird mit der Methode init() initialisiert.

```
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
```

Die Methode `resolve_initial_reference()` bietet einen Mechanismus um Zugriff auf wichtige Dienste zu erhalten (e.g. POA, Namingservice)

```
org.omg.CORBA.Object obj = orb.resolve_initial_references("RootPOA");
```

Mit Hilfe der POA Stubklasse POAHelper wird das allgemeine CORBA-Objekt auf ein konkretes Objekt vom Typ POA gecastet. Narrow prüft gleichzeitig die Korrektheit des Castings.

```
org.omg.PortableServer.POA poa
org.omg.PortableServer.POAHelper.narrow(obj);
```

Beispiel „bank“ (der Server 2-3)

Schritt 2: Aktivieren des POA Managers:

Der Manager verwaltet den POA, insbesondere den Fluss an Aufrufen. Er kann vier Zustände einnehmen:

i. active: Requests werden angenommen,
ii. inactive: Requests werden abgewiesen,
iii. holding: Requests werden zwischengespeichert bis das Objekt bereit ist,
iv. discarding: Bei großer Last werden Requests zurückgewiesen.

Nur wenn der Manager im Zustand aktiv ist, kann der POA Requests empfangen.

```
poa.the_POAManager().activate();
```

Schritt 3: Erzeugen und Aktivieren des Serviceobjekts:

```
AccountServer account = new AccountServer();
org.omg.CORBA.Object servantobj = poa.servant_to_reference(account);
```

Beispiel „bank“ (der Server 4-5)

Schritt 4: Generieren und Veröffentlichen der Objektreferenz:

Die Veröffentlichung erfolgt hier in einem File mit Namen „Filename“ (Später schauen wir uns auch die Veröffentlichung im Namensdienst an).

```
String ref = orb.object_to_string(servantobj);
PrintWriter ps = new PrintWriter(new FileOutputStream(„Filename“));
ps.println(ref);
ps.close();
```

Schritt 5: Aktivieren des ORBs:

Der ORB beginnt an seinem IP Port auf Requests zu horchen.

```
orb.run();
```

Beispiel „bank“ (der Client 1-2)

Schritt 1: Initialisiere den ORB (keinen POA!):

```
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
```

Schritt 2: Hole Objektreferenz:

Hier wird die Objektreferenz aus einem File ausgelesen

```
BufferedReader in = new BufferedReader(„Filename“)

String ior = in.readLine();
```

Aus der stringifizierten Objektreferenz wird ein allgemeines CORBA-Objekt erstellt

```
org.omg.CORBA.Object o = orb.string_to_object(ior);
```

Das CORBA-Objekt wird auf den korrekten Typ (Account) gecastet. narrow prüft ob der Cast innerhalb der IDL Hierarchie korrekt ist.

```
AccountServer account = AccountServerHelper.narrow(o);
```

Beispiel „bank“ (der Client 3-4)

Schritt 3: Zugriff auf die Service Methode account.withdraw() (usw) über die Objektreferenz:

```
account.withdraw( … );
account.deposit( … );
```

Schritt 4: Geordnetes Herunterfahren des ORBs:

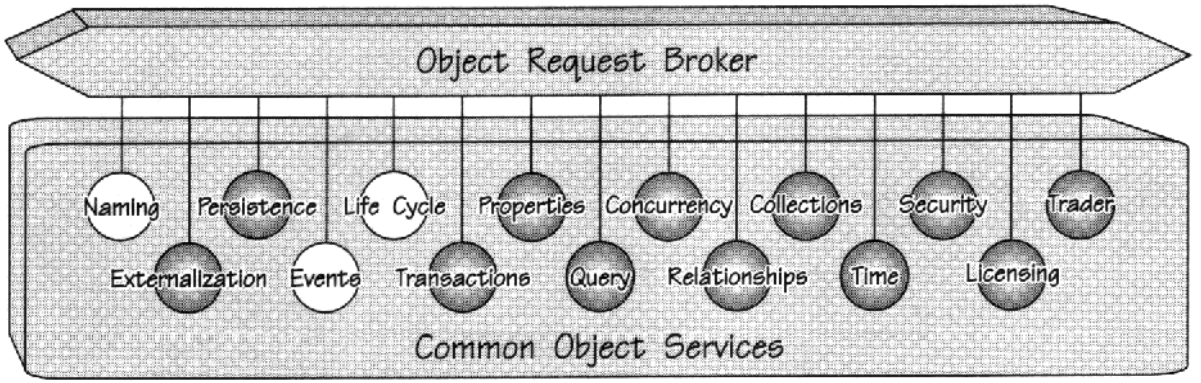
```
orb.shutdown(false);
```

Alle offenen Verbindungen können so korrekt geschlossen werden.

Der Parameter bestimmt, ob die Methode wartet bis der ORB vollständig heruntergefahren ist, oder nicht.

System-Dienste

- **Naming Service:**
  erlaubt Objekten, sich über ihre Namen zu finden
- **Event Service:**
  Objekte können einen Ereigniskanal abonnieren, um bestimmte Ereignisse regelmäßig zu erhalten.

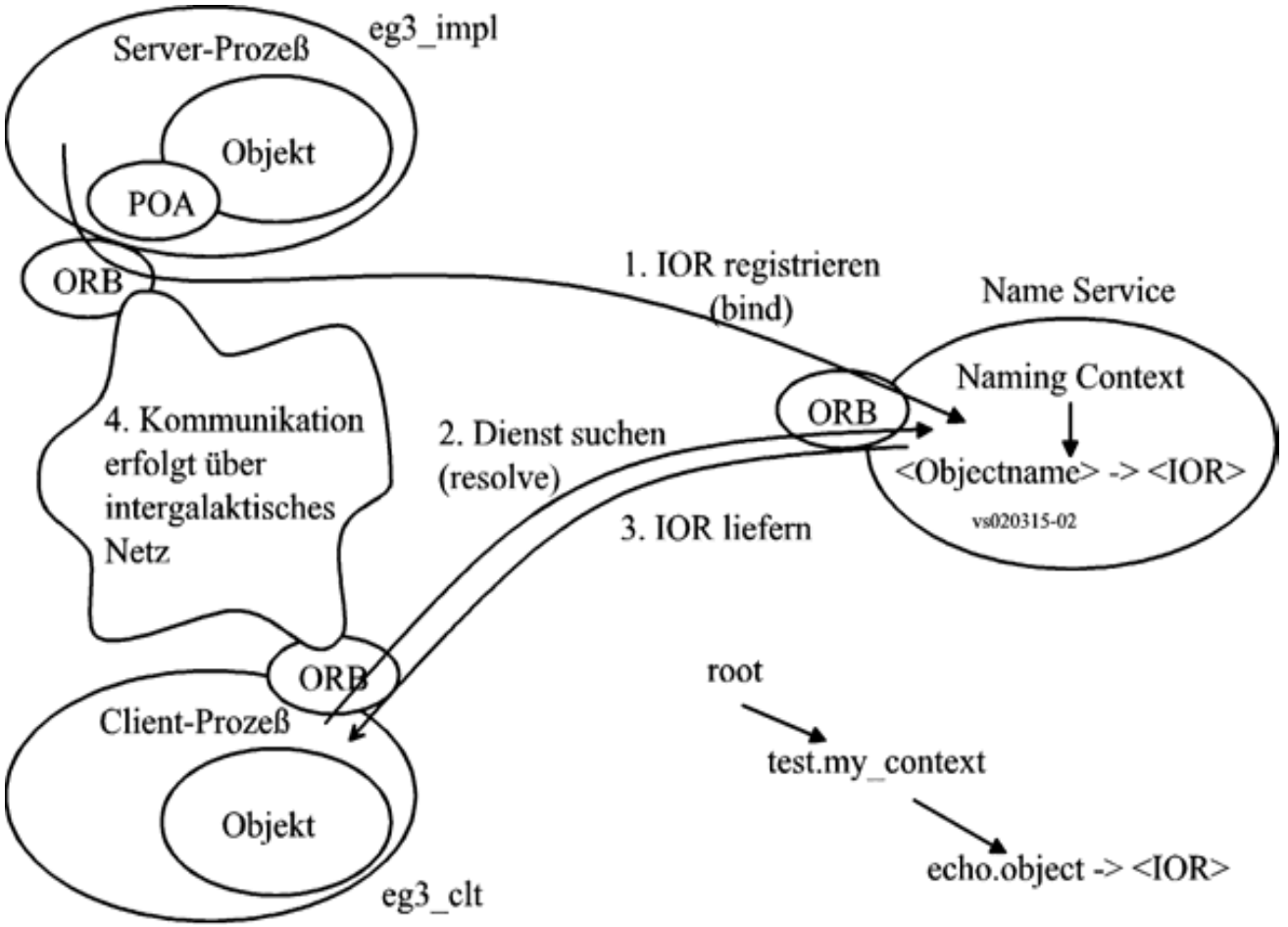


- **Transaction Service:**
  koordiniert Transaktionen auf Basis des Verfahrens "Two-Phase-Commit" (s. Kapitel IV).
- **Security-Service:**
  zuständig für Authentifizierung und Autorisierung von Nutzern und Verschlüsseln sensitiver Daten

Naming-Service (1)

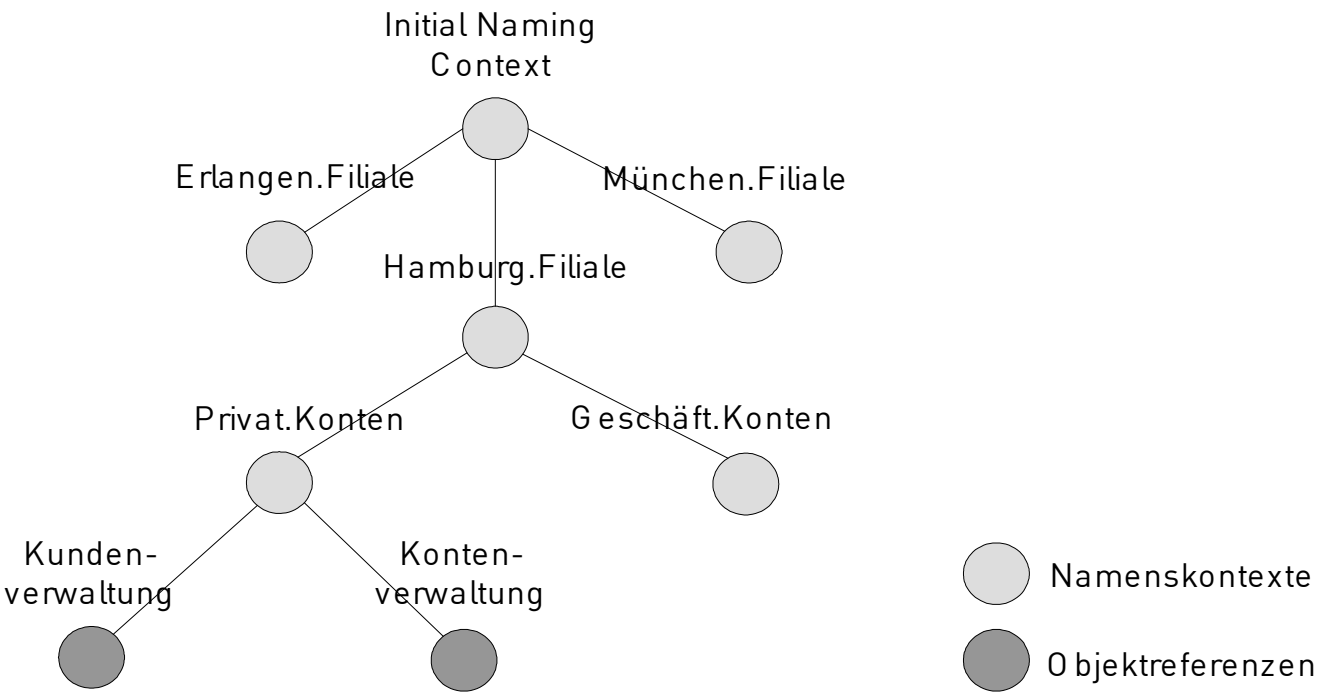
Naming-Service:

- Ist ein ‚Bootstrap' Service:
- Ersetzt Verteilen der Stringified Interoperable Objektreferenz (SIOR) durch Registrieren des Namens
- Ermöglicht Registrieren, Nachschlagen und Benutzung von Namen...
- ...und andere Services im Netz.



Naming-Service (2)

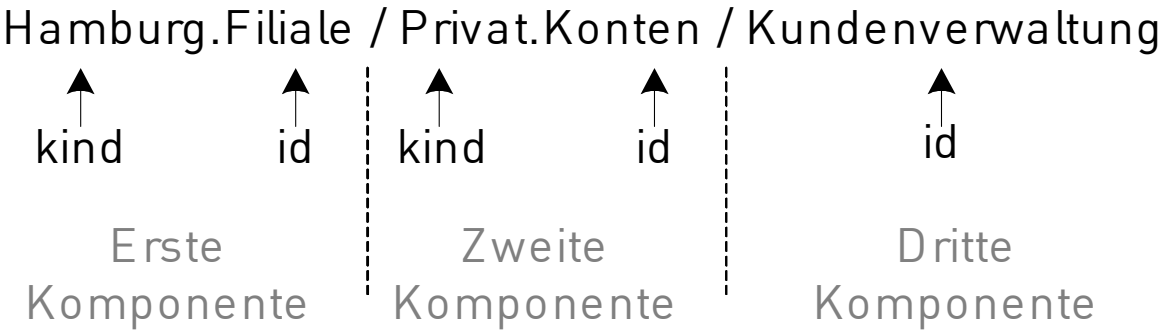
- Der Naming Service verwaltet zwei Typen von Assoziationen (Bindings):
  - **Context Binding:** Assoziationen von Namen zu Namenskontext (Naming Context)
  - **Object Binding:** Assoziation von Namen zu Objektreferenz
- Der Naming Service stellt lediglich die Verwaltungsstruktur bereit.



- Die Verwaltungsstruktur ähnelt einer Verzeichnisstruktur in einem Dateisystem:
  - Knoten → Namenskontexte → Verzeichnisse
  - Blätter → Objekt Referenzen → Dateien.

Naming-Service (3)

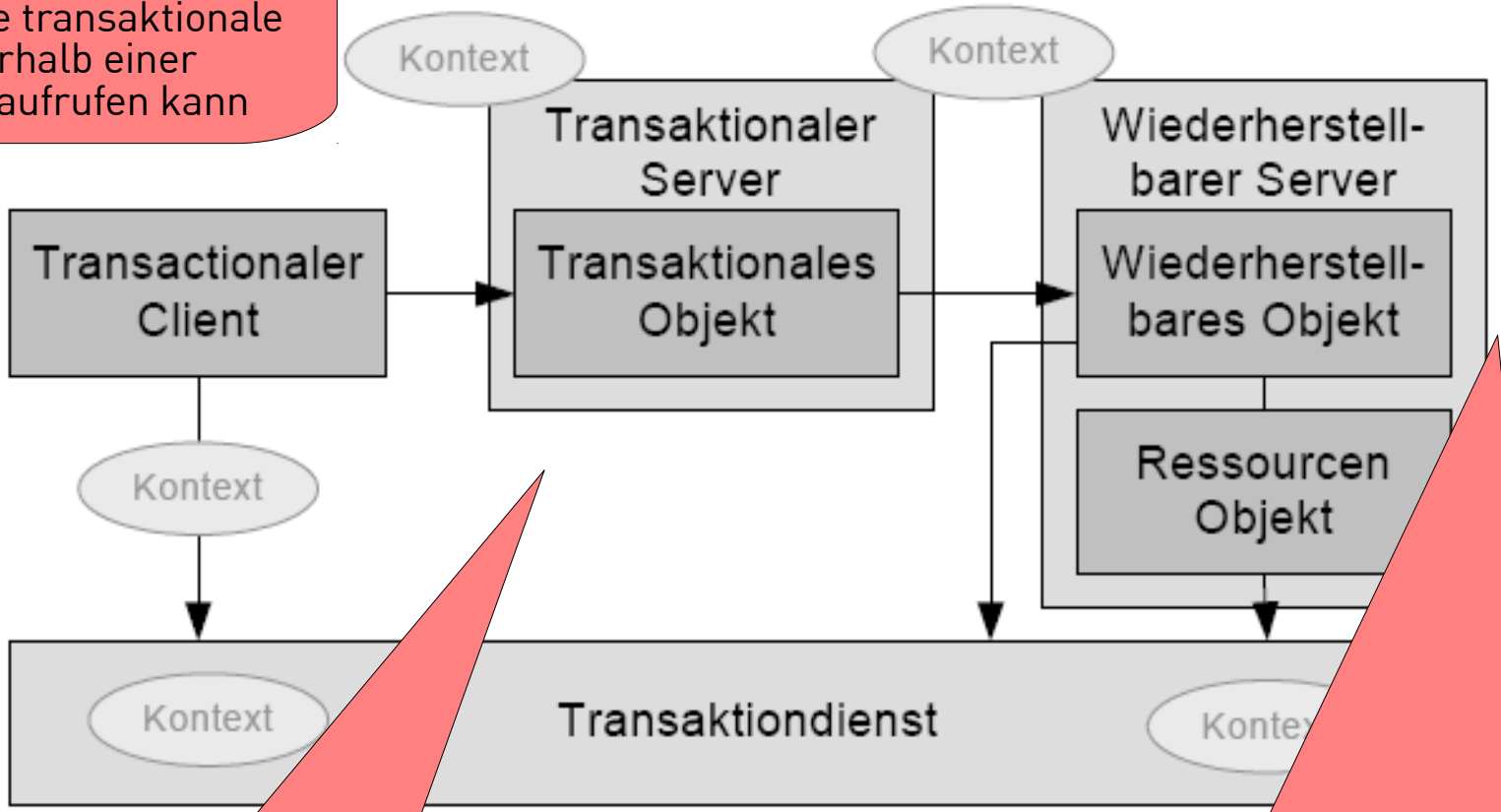
- Ein Name wird unterteilt in Komponenten. Komponenten werden getrennt durch „/“.
- Eine Komponente setzt sich zusammen aus den Feldern id und kind, getrennt durch „.“ (Punkt)
- Wird das „kind“ Feld weggelassen wird es implizit als leer angenommen.



Frage: Gibt es hier das Objekt (bzw. Name) „Hamburg/Geschäft.Konten/Kontenverwaltung“?

Andere Service: Transaktionen

Transaktionaler Client: beliebiges Programm, das verschiedene transaktionale Objekte innerhalb einer Transaktion aufrufen kann



Transaktionaler Objekt: beliebiges Objekt, das innerhalb einer Transaktion aufgerufen werden kann und dessen Verhalten durch die Transaktion bestimmt wird.

Wiederherstellbare (Recoverable) Objekt: transaktionale Objekte haben keinen eigenen persistenten Speicher. Hierzu stehen wiederherstellbare Objekte zur Verfügung. Interagieren direkt mit dem Transaktionsdienst. Bei einem Ausfall sind sie in der Lage den letzten gültigen Zustand zu rekonstruieren.

Zusammenfassung Kapitel III

- **Middleware:**
  Kommunikation + andere Dienste
- **MOM = Message Oriented Middleware:**
  - Nicht nur RPC, auch Publish-Subscribe.
  - HPC (MPI) oder „Enterprise“ (IBM MQ & Co.) oder „Big Data“ (Apache Kafka & Co.)
- **CORBA:**
  - RMI, ohne Festbindung an Java
  - Viele Dienste, viele Freiheiten…
  - Viel „Design by Committee“
- **Web Services**
  - Nicht alles, was auf HTTP basiert, sondern alles, wofür man eine WSDL-Datei schreiben könnte
  - XML-Schema *beschreibt* XML, Soap *verwendet* XML
  - WSDL *beschreibt* SOAP, *verwendet* XML-Schema, erlaubt automatische Software-Erzeugung
  - UDDI = Suchdienst für WSDL
  - Oder REST (mit oder ohne WSDL, mit XML oder JSON oder…)
  - Schwerpunkte: Klarer Schnittstellen, über Grenzen hinweg, automatische Software.

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK