Verteilte Systeme

R. Kaiser, R. Kröger, O. Hahm

(HTTP: http://www.cs.hs-rm.de/~kaiser
E-Mail: eobert.kaiser@hs-rm.de)
Kai Beckmann
Sebastian Flothow
Alexander Schönborn

Sommersemester 2021

4. Anwendungsarchitektur

Hochschule RheinMain

http://www.mappenvorbereitungskurs.de/berufsbilder/architektur.html

Inhalt

Hochschule RheinMain

Haupttreiber kommerzieller IT-Produkte

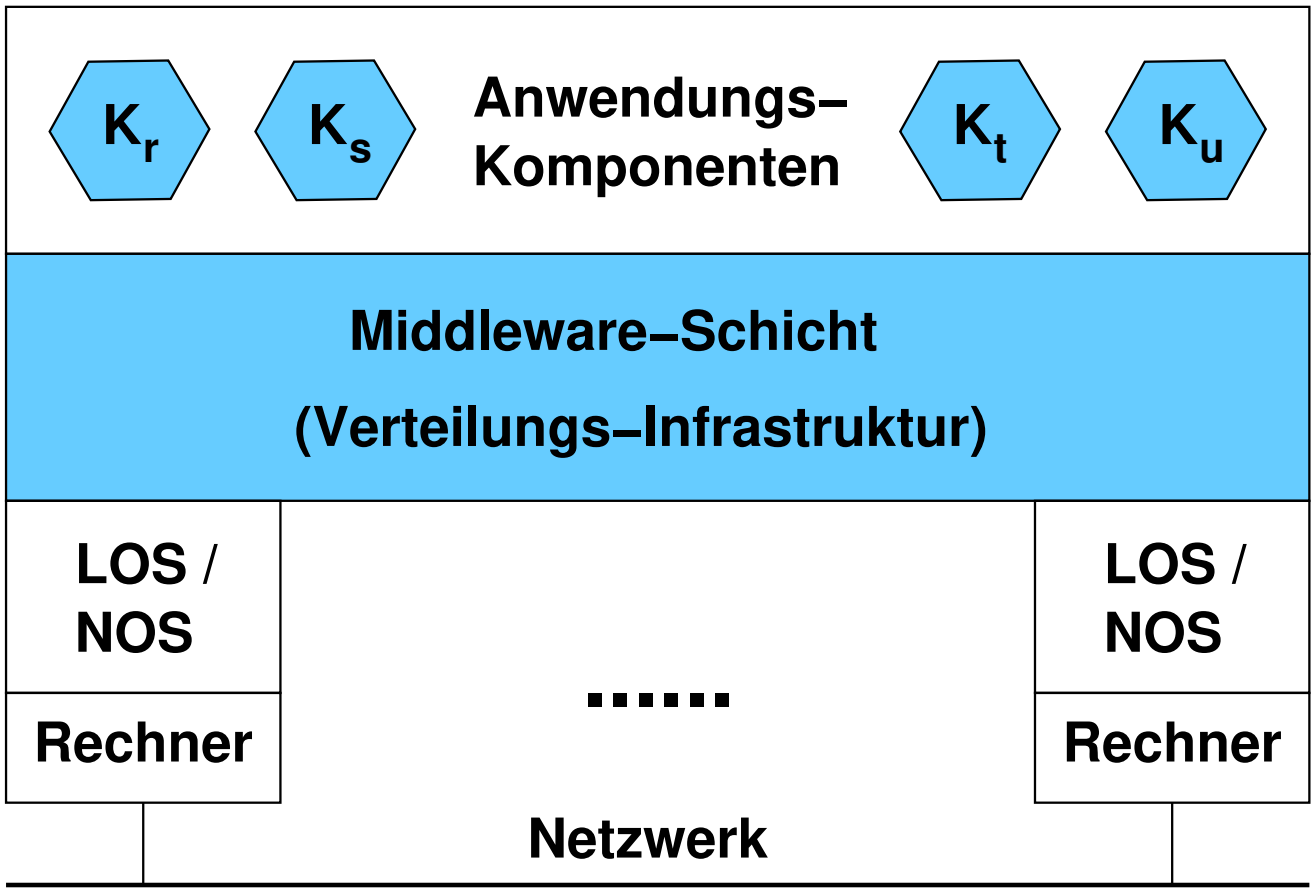
- Hohe Anpassungsfähigkeit
  - Flexibles Abbilden heutiger und künftiger Geschäftsprozesse
  - Verringerung der Entwicklungszeit (time-to-market)
  - Integration existierender (Teil)-Lösungen
  - Interoperabilität mit Fremdsystemen
  - Berücksichtigung der aktuellen technologischen Trends:
    - Internet of Things
    - Cloud Computing
    - Big Data
- Niedrige Kosten
  - Verringerung der Entstehungs-/Entwicklungskosten
  - Verringerung der Betriebs- und Management-Kosten (total cost of ownership)

Lösungsansätze

- Offene Systeme (Hersteller-Unabhängigkeit)
- Standardlösungen statt Individual-Entwicklung
- Client/Server, Distributed Computing
- Middleware
- Web-Anwendungen
- Application Server
- Wiederverwendung von Software (software reuse, componentware)
- Wiederverwendung von Diensten/ Service-Orientierte Architekturen

Middleware-basierte Anw.-Architekturen

- Aufgabe der Middleware:
  Standard-Software-Schicht als Verteilungsplattform zur Integration von Programm-Komponenten



Middleware-Architekturen

Jede Middleware ist charakterisiert durch ein Architektur-Paradigma mit Strukturmodell und Ablaufmodell

- Strukturmodell definiert
  - die verteilbaren Einheiten (Programmkomponenten)
  - ihre Benennung und Adressierung
  - eventuelle Hilfskomponenten
- Aktivitätsmodell (Dynamik) definiert
  - Akteure
  - Interaktionsmuster
  - kommunizierte Einheiten
  - Synchronisation
- Implementierung der Middleware erfordert Rückgriff auf Komponenten der untergelagerten Schichten (speziell NOS/LOS)

Middleware-Architekturen (2)

- Grad der Spezialisierung: kann sehr unterschiedlich sein, z.B.
  - Unterstützung eines allgemeingültigen Kooperationsansatzes (hier im Vordergrund)
  - Datenbank-zentriert (SQL-Middleware, Transaktionsmonitore)
  - Dokumenten/Workflow-orientiert
- Abhängigkeiten von Programmiersprachen
  - manchmal sehr hoch (z.B. nur mit Java nutzbar)
- Abhängigkeiten von unterlagerten Betriebssystemen
  - oft weniger stark
- Abhängigkeiten von unterlagerter Hardware
  - i.d.R. sehr gering

Historische Entwicklung

- Nachrichtenorientierung
- Dienstorientierung
- Objektorientierung
- Komponenten-Orientierung
- Service-Orientierung (Dokumenten-Orientierung)

→ werden im folgenden überblicksweise besprochen
ausgewählte Ansätze in folgenden Kapiteln detaillierter

Paradigma: Nachrichtenorientierung

- Grundmodell kommunizierender Prozesse klassischer Betriebssysteme übertragen auf verteilte Systemumgebung
  - Prozesse als verteilbare Einheiten
  - Nachrichten als kommunizierte Einheiten
- Programmierung paralleler Anwendungen
  (hier nicht behandelt)
  - Programmierung von Parallelrechner-Anwendungen
  - Basis bilden spezielle nachrichtenorientierte Library-Schnittstellen
  - Message Passing Interface (MPI) (Quasi-Standard)

Paradigma: Nachrichtenorientierung (2)

- Beispiel: Socket-Programmierung (vgl. Kap.2)
  - Berkeley Sockets (UNIX)
  - Winsock (MS Windows sockets API)
    - ★ Library mit i.w. Übernahme der UNIX/BSD-Funktionen
  - Transport Layer Interface (TLI)
    - ★ API für Netzwerkprogrammierung auf Transport-Ebene: `t_xxx`
    - ★ In UNIX SVR4 auf STREAMS implementiert.
    - ★ Kaum noch relevant und nicht mehr empfohlen.
  - Sockets heute noch de-facto-Standard, z.T. über Libraries oder Klassen verkleidet
  - Java Sockets (java.net)
    entspricht weitgehend Modell der Berkeley Sockets.

Paradigma: Nachrichtenorientierung (3)

- Message-oriented Middleware (MOM)
  - häufig Unterstützung für Persistenz, Transaktionen
  - Beispiele:
    - IBM Websphere MQ
    - Java Messaging Service (JMS) (Teil von J2EE)
    - RabbitMQ

Paradigma: Dienstorientierung

Basis: Remote Procedure Call (RPC)

- Dienste als verteilbare Einheiten
- Dienst = Service: Menge von offerierten Operationen/Funktionen
- Nutzung entfernter Dienste durch Prozeduraufrufe
- i.d.R. synchrone Verarbeitung
- kommunizierte Einheiten sind Requests und Responses, enthalten typisierte Parameter usw. in einer Netzdatendarstellung
- Basis für Client/Server-Anwendungen
- Bindung von Client und Server relativ statisch

Details im Kap. 3

Verbreitete RPC-Plattformen

SunRPC

- public domain, auf vielen Systemen lauffähig
- Bedeutung weniger durch Nutzung allgemeiner Anwendungen
- aber Netzwerkdateisystem NFS basiert auf SunRPC

OSF DCE RPC, Microsoft RPC

- DCE Distributed Computing Environment:
  erste reiche Diensteumgebung
- DCE RPC: ursprünglich aus Apollo NCS RPC entstanden
- zu komplex in der Nutzung
- Microsoft RPC weitgehend kompatibel zu DCE RPC
- heute kaum noch genutzt

Apache Thrift

- sehr flexibles RPC-System
- Unterstützung aller relevanten Programmiersprachen
- Weit verbreitet

Paradigma: Objektorientierung

Hochschule RheinMain

- Objekte (im Sinne der OO Programmierung) als verteilbare Einheiten
- Anwendung = verteiltes Objekt-„Geflecht“
- Interaktion durch Methodenaufrufe (mit Ortstransparenz, Zugriffstransparenz) auf Basis eines RPC-Mechanismus
- Wiederverwendung von Klassen auf Quellcodeebene
- Wesentliche Plattformen
  - OMG CORBA
  - Microsoft DCOM
  - Java RMI

Beispiel: RMI

- Java Remote Method Invocation (RMI) (Sun/Oracle)
  - Jüngste Plattform
  - Einfachste Nutzung
  - unterstützt ausschließlich homogene „Welt“ von verteilten Java-Objekten

Beispiel: Microsoft DCOM

- Microsoft DCOM
  - Erweiterung von COM/OLE unter Nutzung von Microsoft RPC
  - weitgehend proprietäre Plattform
  - 1999 an Open Group übergeben
  - Microsoft's Folge-Basis war .NET
  - Bedeutung gesunken, aber Nutzung noch in Automatisierungstechnik

Beispiel: OMG CORBA

- Object Management Group (OMG)
  - internationale Non-Profit Organisation von Herstellern, Software Häusern und Anwendern
  - gegründet 1989
    - ★ 3Com, American Airlines, Canon, Data General, HP, Philips, Sun, Unisys, ...
  - 1.000+ Mitglieder (Firmen, Organisationen, Hochschulen, ...)
  - In der Vergangenheit hohes Tempo für Standardisierungsgremium
  - offener, formaler Standardisierungsprozess
    basierend auf Request for Proposals (RFPs)
  - Ziel: Definition von Schnittstellen, nicht Produktentwicklung
  - `http://www.omg.org`: frei verfügbare Dokumente
  - heute auch relevant bzgl.
    - ★ UML-Standardisierung
    - ★ Model Driven Architecture (MDA)

CORBA

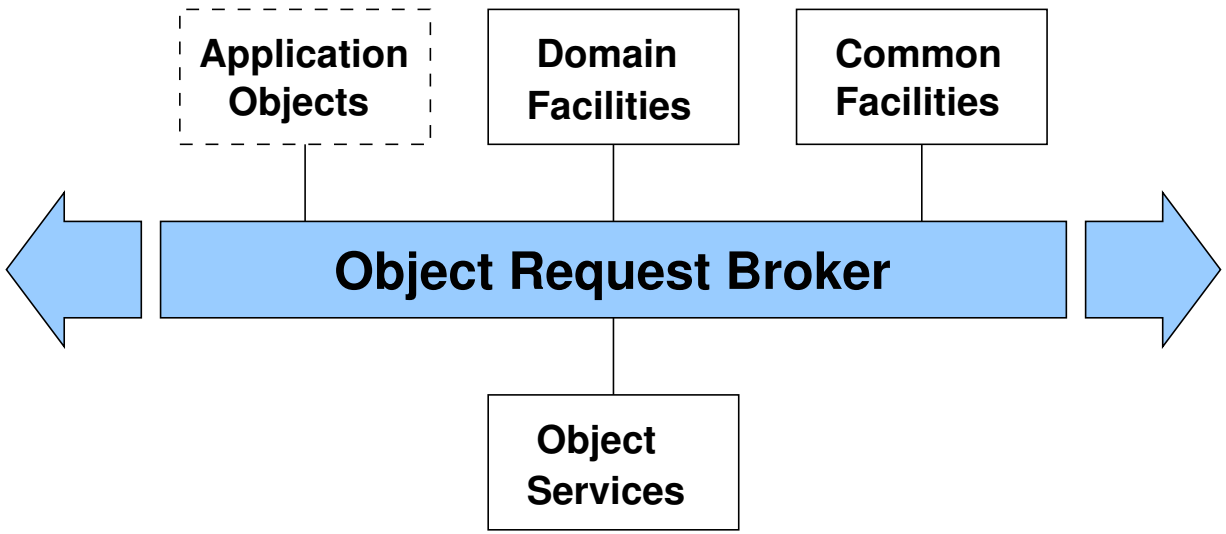
Hochschule RheinMain

- CORBA (Common Object Request Broker Architecture)
  - Architektur-, Betriebssystem- und Programmiersprachen-unabhängig
  - CORBA IDL an C++ angelehnte Schnittstellenbeschreibungssprache
  - IOR als systemweite Objektreferenz, GIOP/IIOP als Aufrufprotokoll
  - zahlreiche objektorientierte Dienste und Implemtierungen verfügbar
- Kaum noch Bedeutung für neue Geschäftsanwendungen, aber noch Pflege

Object Management Architecture

OMA = Object Management Architecture

- Referenzmodell für verteilte, objektorientierte Anwendungen in heterogenen Umgebungen



ORB = Object Request Boker

- „Objekt-Bus“ = Kern der OMA
- vermittelt Aufrufe zwischen Objekten (stellenübergreifend, plattformübergreifend, Programmiersprachen-unabhängig)
- Interoperabilität zwischen verschiedenen ORBs

OMA-Objektmodell

Objekt

- „gedachte“, gekapselte Einheit auf einem System.
- wird „real“, wenn Implementierung in einer Programmiersprache dazu existiert.
- muss nicht einem Objekt auf Programmiersprachenebene entsprechen.
- besitzt unveränderbare Identität (identity).
- hat Zustand.
- kann durch ORB lokalisiert werden.
- hat Attribute (von außen zugreifbar)
- offeriert Operationen (Methoden), die durch Anfragen von Klienten (client requests) in Anspruch genommen werden.

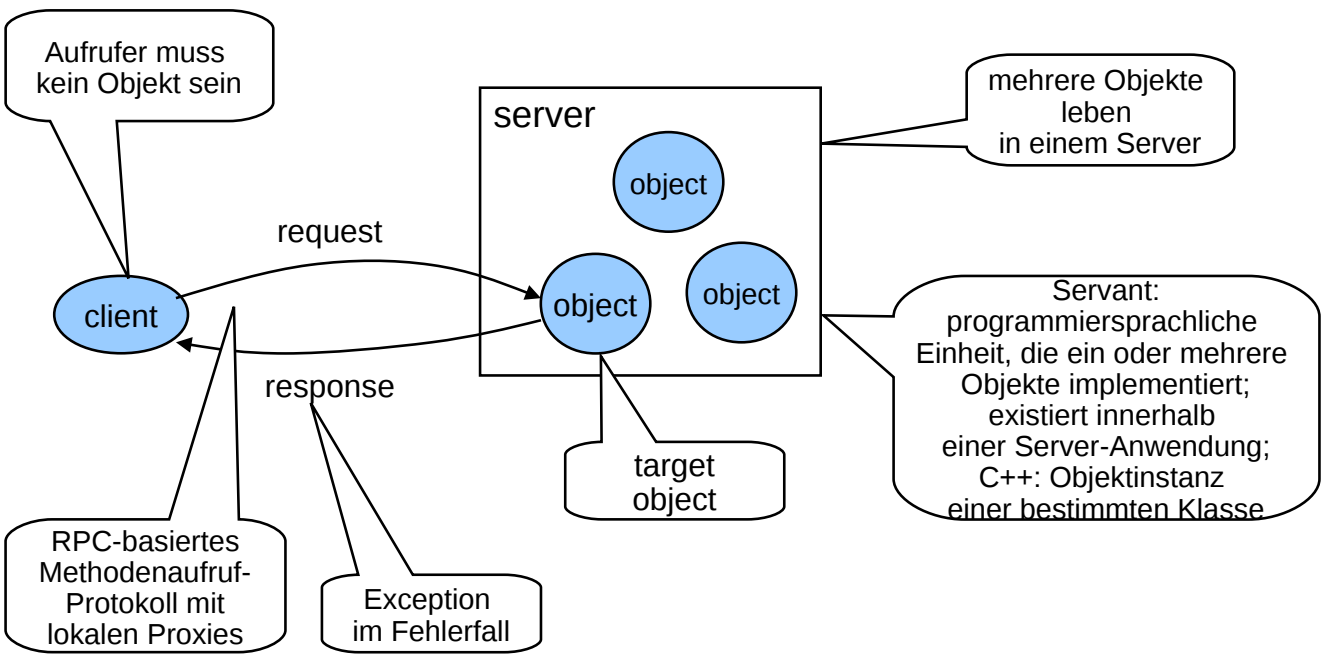
OMA-Objektmodell (2)

Hochschule RheinMain

Objektreferenz

- „handle“, um Objekt zu identifizieren, zu adressieren und zu lokalisieren
- interne Struktur für Klienten verborgen (opaque)
- bezieht sich auf ein bestimmtes Objekt

Zusammenhang



Arten von Requests

- synchron (*synchronous*)
  - Client blockiert, bis Response ankommt.
- verzögert synchron (*deferred synchronous*)
  - Client arbeitet nach Abschicken des Requests weiter, fragt später nach der Antwort (derzeit nur über DII möglich).
- Einwegaufruf (*oneway request*)
  - Best-effort-Zustellung ohne Response, muss nicht beim target object ankommen.
- asynchrone Aufrufe (*asynchronous requests*)
  - definiert im Rahmen von CORBA Messaging, Teil der Version CORBA 2.5 (2001).

Anwendungsentwicklung

CORBA Interface Definition Language (IDL)

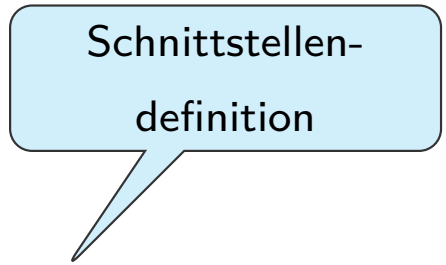
- deskriptive Sprache zur Definition von Objektschnittstellen (keine Kontrollkonstrukte)
- streng typisiert
- any-Typ erlaubt Flexibilität
- ISO 14750
- Beschreibung ist unabhängig von bestimmter Implementierungssprache
- syntaktisch an C++ angelehnt

Ein einfaches CORBA IDL-Beispiel


```
interface balance {
  exception out_of_tolerance {};
  readonly attribute long mode;
  long getweight_in_grams() raises (out_of_tolerance);
  void set_ref_weight_mode(in long ref_weight);
  unsigned short get_weight_in_percent();
  void reset_ref_weight_mode();
};


interface ext_balance : balance {
  exception out_of_tolerance {long difference};
  readonly attribute long mode;
  long getweight_in_carat() raises (out_of_tolerance);
  void set_tare_weight_mode(in long tare_weight);
  void reset_tare_weight_mode();
  void set_tolerance_weight_mode(in long min, in long max);
  void reset_tolerance_weight_mode();
};
```

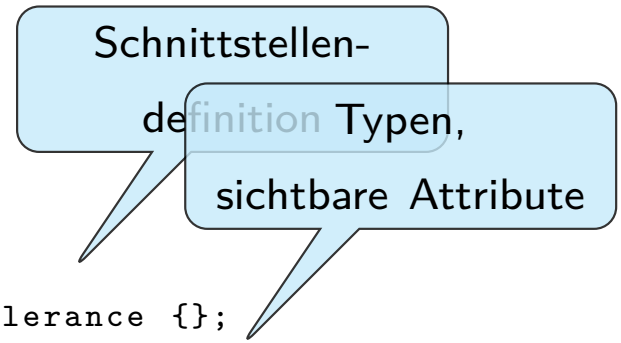
Ein einfaches CORBA IDL-Beispiel




```
interface balance {
  exception out_of_tolerance {};
  readonly attribute long mode;
  long getweight_in_grams() raises (out_of_tolerance);
  void set_ref_weight_mode(in long ref_weight);
  unsigned short get_weight_in_percent();
  void reset_ref_weight_mode();
};


interface ext_balance : balance {
  exception out_of_tolerance {long difference};
  readonly attribute long mode;
  long getweight_in_carat() raises (out_of_tolerance);
  void set_tare_weight_mode(in long tare_weight);
  void reset_tare_weight_mode();
  void set_tolerance_weight_mode(in long min, in long max);
  void reset_tolerance_weight_mode();
};
```

Ein einfaches CORBA IDL-Beispiel

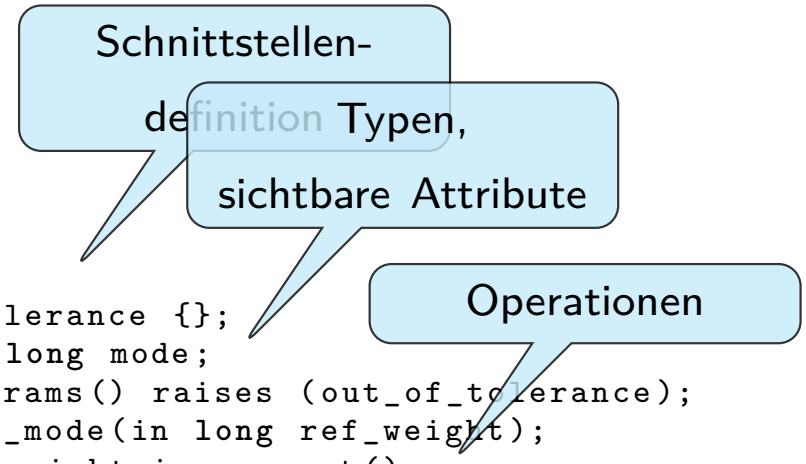



```
interface balance {
  exception out_of_tolerance {};
  readonly attribute long mode;
  long getweight_in_grams() raises (out_of_tolerance);
  void set_ref_weight_mode(in long ref_weight);
  unsigned short get_weight_in_percent();
  void reset_ref_weight_mode();
};


interface ext_balance : balance {
  exception out_of_tolerance {long difference};
  readonly attribute long mode;
  long getweight_in_carat() raises (out_of_tolerance);
  void set_tare_weight_mode(in long tare_weight);
  void reset_tare_weight_mode();
  void set_tolerance_weight_mode(in long min, in long max);
  void reset_tolerance_weight_mode();
};
```

Ein einfaches CORBA IDL-Beispiel

Hochschule RheinMain

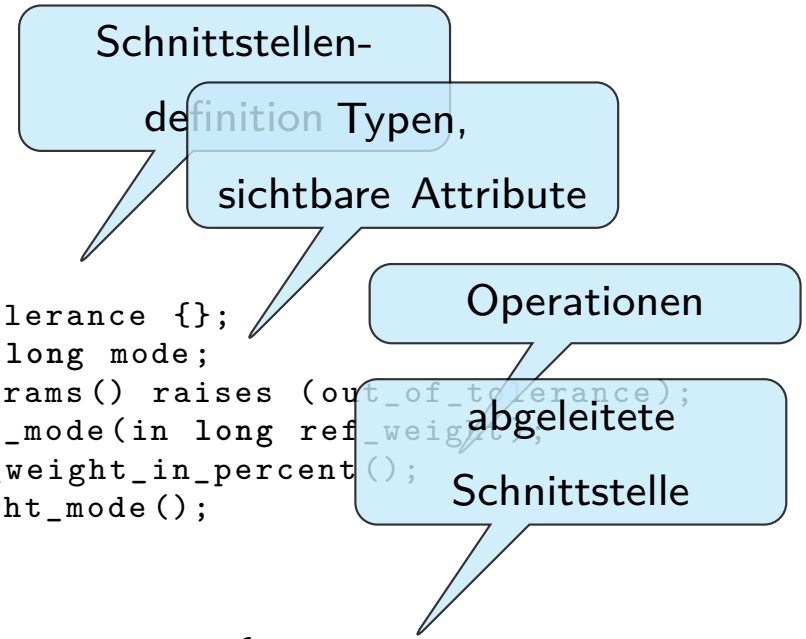



```
interface balance {
  exception out_of_tolerance {};
  readonly attribute long mode;
  long getweight_in_grams() raises (out_of_tolerance);
  void set_ref_weight_mode(in long ref_weight);
  unsigned short get_weight_in_percent();
  void reset_ref_weight_mode();
};


interface ext_balance : balance {
  exception out_of_tolerance {long difference};
  readonly attribute long mode;
  long getweight_in_carat() raises (out_of_tolerance);
  void set_tare_weight_mode(in long tare_weight);
  void reset_tare_weight_mode();
  void set_tolerance_weight_mode(in long min, in long max);
  void reset_tolerance_weight_mode();
};
```

Ein einfaches CORBA IDL-Beispiel

Hochschule RheinMain




```
interface balance {
  exception out_of_tolerance {};
  readonly attribute long mode;
  long getweight_in_grams() raises (out_of_tolerance);
  void set_ref_weight_mode(in long ref_weight);
  unsigned short get_weight_in_percent();
  void reset_ref_weight_mode();
};


interface ext_balance : balance {
  exception out_of_tolerance {long difference};
  readonly attribute long mode;
  long getweight_in_carat() raises (out_of_tolerance);
  void set_tare_weight_mode(in long tare_weight);
  void reset_tare_weight_mode();
  void set_tolerance_weight_mode(in long min, in long max);
  void reset_tolerance_weight_mode();
};
```

Language Mappings

- Spezifikation, wie IDL in verschiedene Programmiersprachen abgebildet wird
  - z.B. IDL Module auf C++ Namensraum oder Java package,
  - IDL Interface auf C++-Klasse,
  - IDL-Operationen auf deren Member-Funktionen.
- Standardisierte Language Mappings für
  - C, C++, Java, Smalltalk, COBOL, Ada, Lisp, PL/1, Python, IDLscript
- Andere definierte Language Mappings für:
  - Tcl, Perl, Eiffel, ...
- Konsequenz:
  - Unterschiedliche Teile einer verteilten Anwendung können mit verschiedenen Sprachen entwickelt sein
  - z.B. Server-Applikation in C++, Clients in Java.

Produkte

Hochschule RheinMain

Wichtige kommerzielle ORBs:

- BEA M3 (Teil von BEA Tuxedo) (BEA gekauft von Oracle, 2008)
- IONA Orbix (IONA gekauft von Progress, 2008)
- ORBexpress RT, Orbriver RT, PrismTech OpenFusion (für Echtzeitanwendungen)

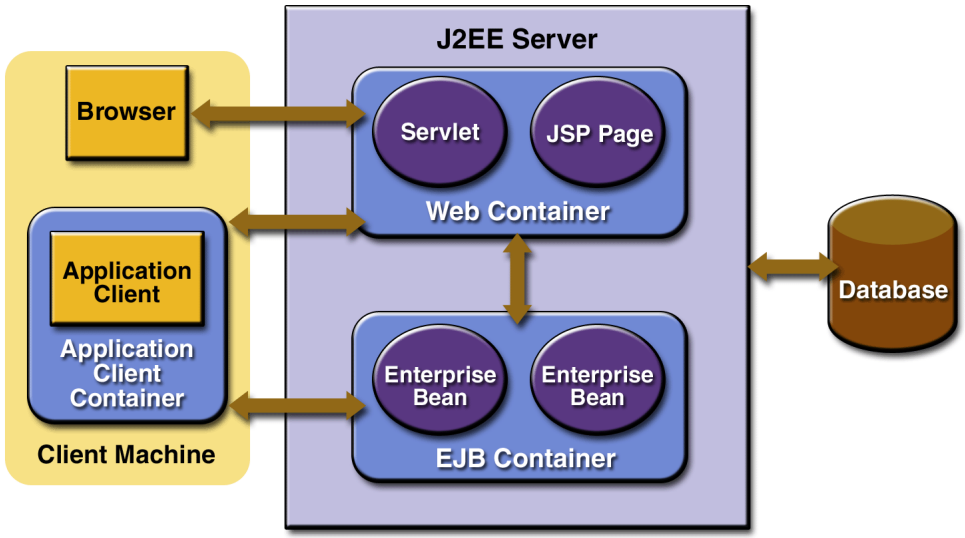
Wichtige freie ORBs:

- OOC ORBacus (noch weitgehend frei verfügbar, 2001 aufgekauft von IONA)
- MICO (Open Source Projekt, ursprünglich Uni Frankfurt)
- JacORB (FU Berlin, jetzt PrismTech OpenFusion)
- TAO (WUSTL) (Echtzeitverarbeitung)
- ORBit (Middleware für GNOME)

Paradigma: Komponentenorientierung

- Komponenten als verteilbare Einheiten
- Starke Unabhängigkeit und Austauschbarkeit der Komponenten
- Interaktion durch Methodenaufrufe (basierend auf RPC)
- Enterprise Java Beans (EJB) als am weitesten verbreitetes Komponentenmodell, daneben Microsoft .NET
  - Teil der Spezifikation von Java-Schnittstellen für Server-seitige Komponenten (J2EE, jetzt JEE)
  - enger Bezug zu CORBA
  - Ziel: Vereinfachung der Anwendungsentwicklung
  - Application Server als integrierte Infrastruktur für transaktionsorientierte Geschäftsanwendungen
  - Schnittstellen zu standardisierten Diensten (Persistenz, Transaction Management, Directory-Dienste, Messaging) zum Deployment-Zeitpunkt gebunden
  - Hohe Skalierbarkeit für Server-seitige Webanwendungen

Enterprise Java Beans



http://www.rizzimichele.it/enterprise-java-beans-and-all-j2ee/

- Komponenten
  - Stateless und Stateful Session Beans (Ausführung einer Task für Client ohne bzw. mit Gedächtnis für denselben Client)
  - Entity Beans (Repräsentierung von Geschäftsobjekten im persistenten Speicher, Unterstützung für Transaktionen)
  - Message-driven Beans (asynch. Verarbeitung von Nachrichten, JMS-API)

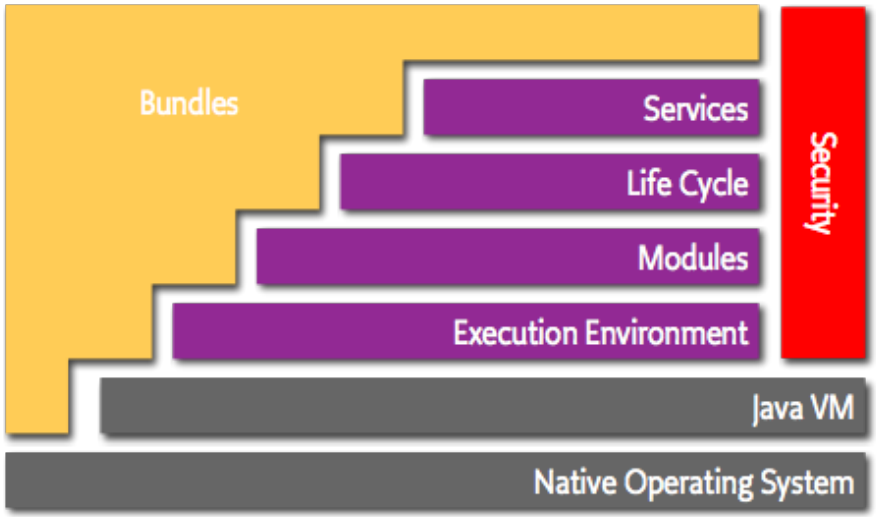
Verbreitete Produkte

- Freie:
  - JBoss Application Server (steil aufgestiegen, jetzt Red Hat)
  - Geronimo (Apache)
  - JOnAS (Object Web, Bull)
- IBM Websphere
- Oracle/BEA Weblogic
- SAP NetWeaver
- Sun GlassFish

OSGi - Komponentenmodell

- Aktuelles verbreitetes Komponentenmodell mit Java-Bezug für große verteilte Systeme bis zum Embedded-Bereich
- Von Entwicklern erzeugte Komponenten heißen „Bundles“
- Dynamisches Management von Komponenten (Lifecycle, incl. Updates, Remote Management)
- Unterstützung für Versionierung
- Einsatz: Technologie auch z.B. enthalten
  - als Equinox Plattform in Eclipse für dyn. Plugin-Management
  - zur internen Modularisierung in vielen Applikationsservern
  - Ursprung Home Automation, auch dort noch sehr aktiv (Smart Home, Residential Gateways, z.B. Telekom Qivicon)
  - Automotive / Telematik u.a.

OSGi - Architektur



https://www.osgi.org/developer/architecture/

- Services verbinden Bundles dynamisch
- Life-Cycle - API für install, start, stop, update, uninstall
- Modules - Layer, der Import/Export von Code definiert
- Execution Environment - definiert Klassen und Methoden der Plattform

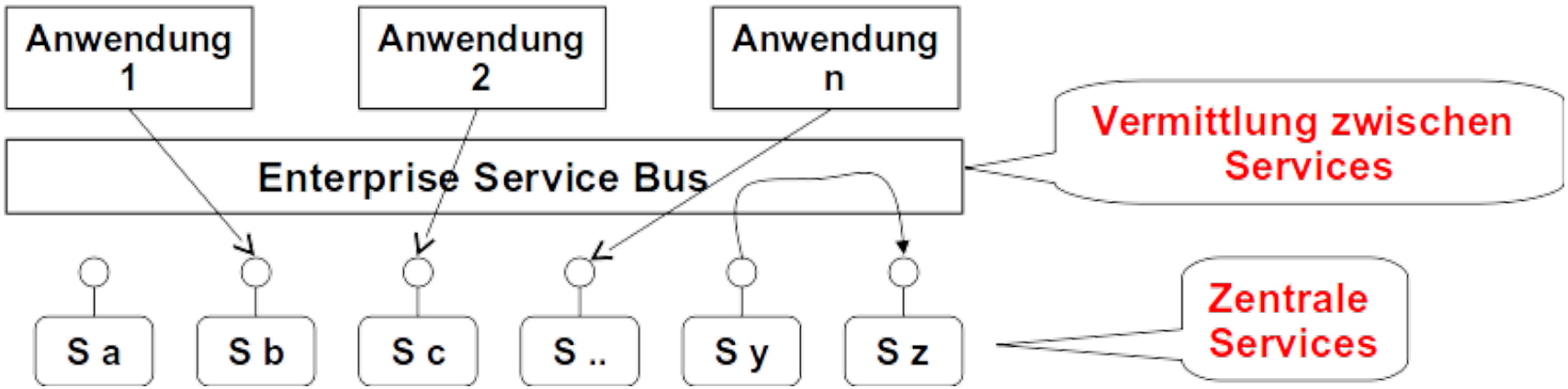
Paradigma: Service-Orientierung

- Service-Orientierte Architekturen (SOA)
- Architekturansatz für Geschäftsanwendungen zur Strukturierung und Nutzung von verteilten Diensten, die möglicherweise unter Kontrolle verschiedener Eigentümer stehen, mit dem Ziel, eine fachliche Strukturierung von Anwendungsmengen zu erreichen.
- Erwartete Vorteile:
  - Definition von Diensten anhand der Geschäftsprozesse
  - Gleichzeitige mehrfache Nutzung von Diensten in verschiedenen Anwendungen
  - Dadurch Reduktion ansonsten mehrfach zu pflegender Funktionalität
  - Zentrale Integration verschiedener Anwendungen statt paarweiser Schnittstellen

SOA: Dienste – Anwendungen

Hochschule RheinMain

- Idealvorstellung:



- Zunehmende Kritik wegen Problemen:
  - Komplette Dekomposition bestehender Anwendungen ist schwierig, aufwendig und für Nutzer nicht sichtbar
  - Veränderung an zentralen Dienste betreffen viele Anwendungen
  - Formalisierung der Geschäftsprozesse aus Diensten für Fachabteilungen schwierig

SOA: Technische Sichtweise

- Technische Sichtweise:
  - Dienste autonom mit formalen Schnittstellen (Service Contracts) beschrieben in XML Schema Dokumenten
  - Dienste halten möglichst keinen Zustand
  - XML-Dokumente als kommunizierte Einheiten (Messages)
  - Dienstbeschreibungen (Metadaten) in Verzeichnis (Service Registry)
  - Dienste können über ihre Beschreibung dynamisch erkannt und angesprochen werden (kein Linking notwendig)
  - Implementierungssprache / -technologie irrelevant
  - Web Services als aktueller Hype zur Implementierung von SOA-Diensten
  - Enterprise Service Bus (ESB) zur losen Kopplung der Dienste

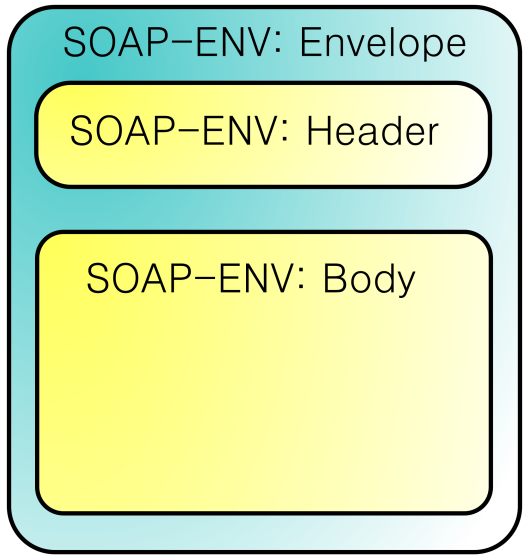
WSDL

- WSDL (Web Service Description Language)
  - Schnittstellen/Contract-Beschreibungssprache (abstrakt): Types, Messages, Interfaces, Services
  - W3C-Standard
  - XML-basiert

SOAP

- SOAP (ehemals Simple Object Access Protocol)
  - W3C-Standard
  - Objekte im Sinne der Objektorientierung existieren aber nicht
  - XML-Dokumenten-basiertes Interaktions-Framework für Web Services
    - ★ SOAP Messages (Envelopes aus opt. Header und Body)
    - ★ asynchrone Verarbeitung prinzipiell möglich
    - ★ SOAP Request/Response-Nachrichten für RPC-Stil
  - Protocol Binding Framework sieht verschiedene unterlagerte Transport-Dienste vor, neben HTTP/HTTPS auch z.B. SMTP, JMS
  - Java API for XML Web Services (JAX-WS) Teil von Java SE

SOAP: Beispiel



https://commons.wikimedia.org/wiki/File:SOAP.svg

```
<?xml version="1.0"?>
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope">
    <s:Body>
        <m:TitleInDatabase xmlns:m="http://www.lecture-db.de/soap">
            DOM, SAX und SOAP
        </m:TitleInDatabase>
    </s:Body>
</s:Envelope>
```

Geschäftsprozesse

- Modellierung von Geschäftsprozessen
  - Geschäftsprozess = komplexe Interaktion zwischen Diensten
  - auch Web Services Orchestrierung genannt
  - Programmieren im Großen
    - ★ Web Services als elementare Einheiten
  - WS-BPEL (Business Process Execution Language)
    - ★ OASIS Standard
    - ★ Programm ist selbst XML-Dokument
    - ★ Mittlerweile untergeordnete Bedeutung
  - BPMN (Business Process Model and Notation)
    - ★ Bisher genannt: Business Process Modelling Notation
    - ★ OMG Standard, verwandt zu UML-Aktivitätsdiagr., aktuell 2.0.2 (2013)
    - ★ = ISO/IEC 19510
    - ★ Soll Verständnis zwischen Technikern und Managern fördern

Grundlegende Architekturmodelle

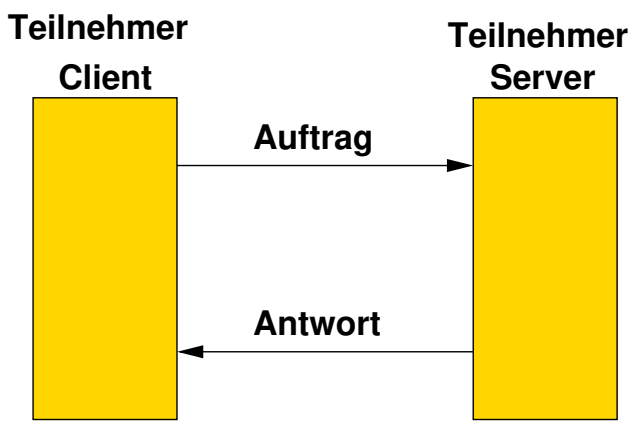
Grundlegende Strukturmodelle für komplexe verteilte Anwendungen

1. Client/Server-Modell
2. Peer-to-Peer-Modell
3. Multi-Tier-Modell
4. SOA Modell

Client/Server-Modell

Hochschule RheinMain

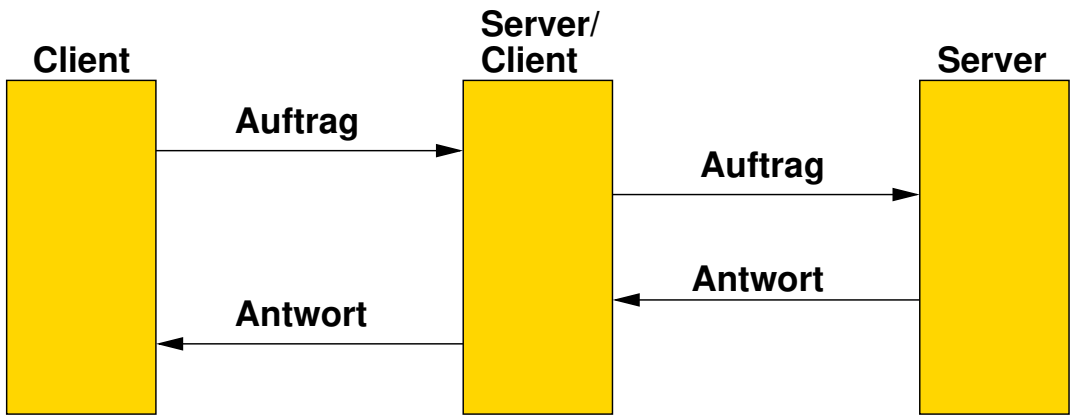
- Das Client/Server-Modell ist ein Software-Architekturmodell für verteilte Anwendungen
- Es unterscheidet Rollen:
  - Server: Erbringer eines Dienstes (Service), z.B. Web-Server liefert Seiten
  - Client: Dienstnutzer, Kunde, Klient z.B. Browser fordert Seiten an
- Client und Server i.d.R. auf verschiedenen Rechnern



Client/Server-Modell (2)

Hochschule RheinMain

- Kommunikationsvorgänge basieren auf Auftrag/Antwort-Interaktionsmuster und werden vom Client initiiert.
- Ein Client kann im zeitlichen Verlauf mit mehreren Servern arbeiten
- Ein Server kann Aufträge für verschiedene Klienten ausführen.
- Ein Server kann als Client gegenüber weiteren Servern auftreten (Wechsel der Rolle):

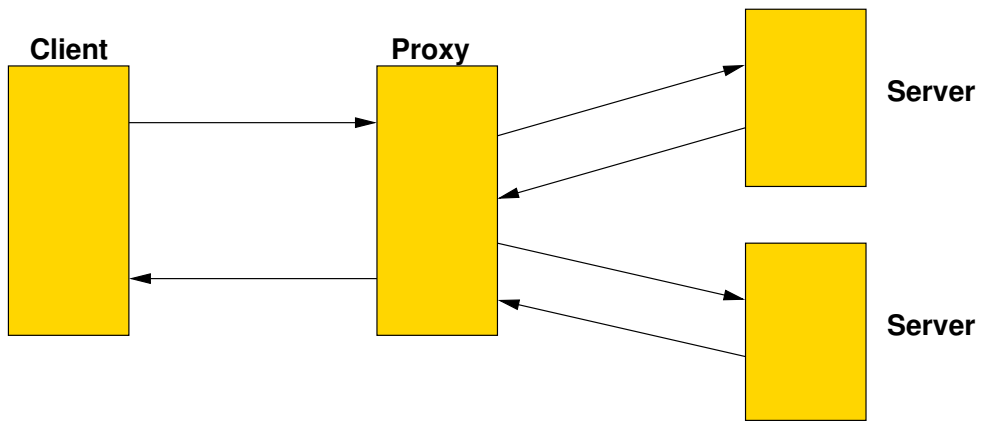


Client/Server-Modell (3)

Hochschule RheinMain

Proxy

- zwischengeschaltete Instanz
- Server-Rolle gegenüber Client,
- Client-Rolle gegenüber eigentlichen Servern
- Aufgaben z.B. Caching, Modifikation der Anfragen, ...
- Beispiel: Proxy-Server für Web-Seiten



Peer-to-Peer-Modell (P2P)

Hochschule RheinMain

- dezentrale Kommunikation zwischen Gleichrangigen (= Peer)
- keine zusätzliche Infrastruktur (z.B. Server)
- Basis für Ad-hoc-Kommunikation
- Netzwerk- oder Anwendungsebene
- beliebige nachrichtenorientierte Interaktion
- Beispiele
  - File-Sharing, z.B. BitTorrent, Gnutella, eMule
  - P2P-Entwicklungsplattformen JXTA, MSP2P

Multi-Tier-Modell

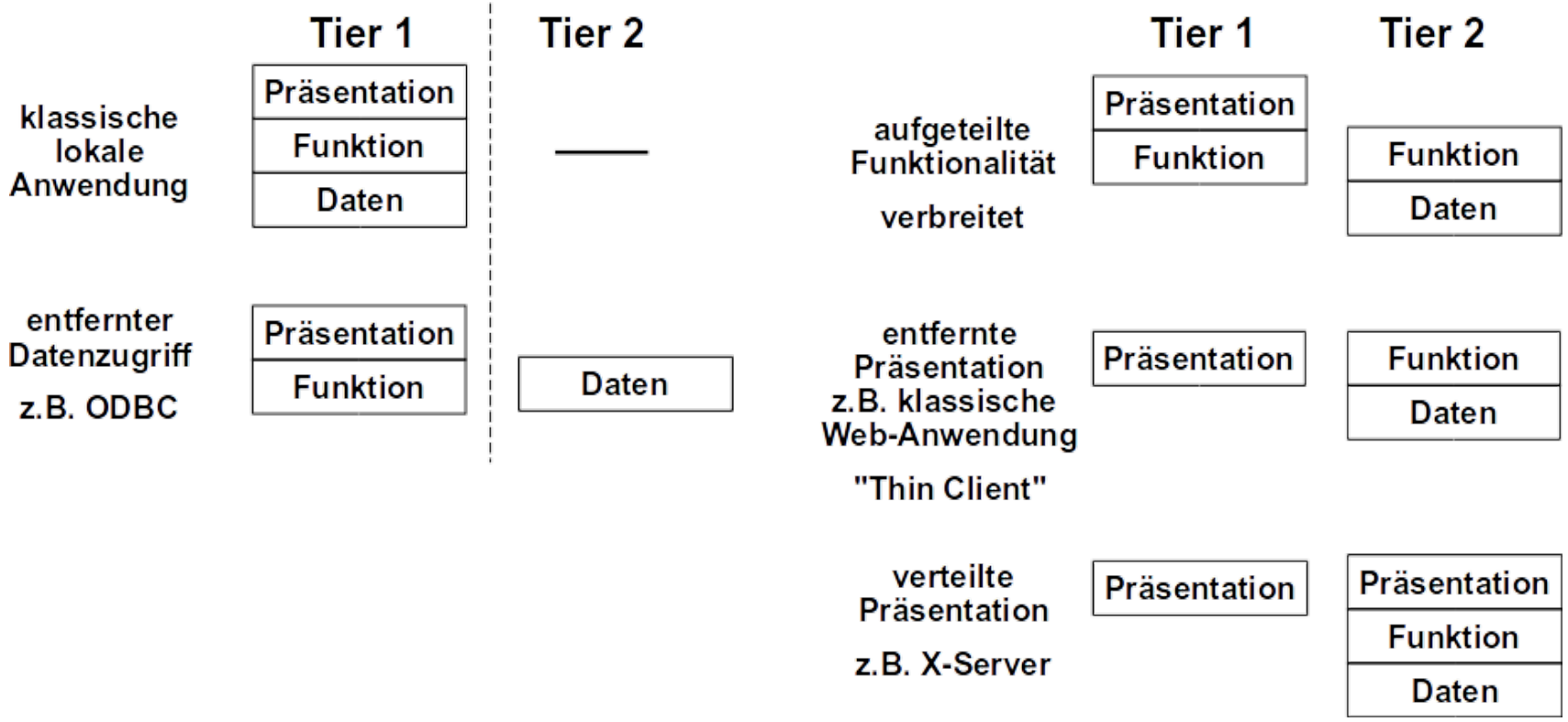
Hochschule RheinMain

- *Tier* = Reihe, Strang
- Stränge eher orthogonal zu (Abstraktions)-Schichten, i.d.R. orientiert an
  - Benutzerschnittstelle / Präsentation
  - Anwendungslogik / Funktion
  - Datenhaltung
- enthält keine Festlegung über verwendete Middleware
- heute sehr verbreitet
- üblich
  - Two-Tier-Architektur
  - 3-Tier-Architektur
  - N-Tier-Architektur

Two-Tier-Architektur

Hochschule RheinMain

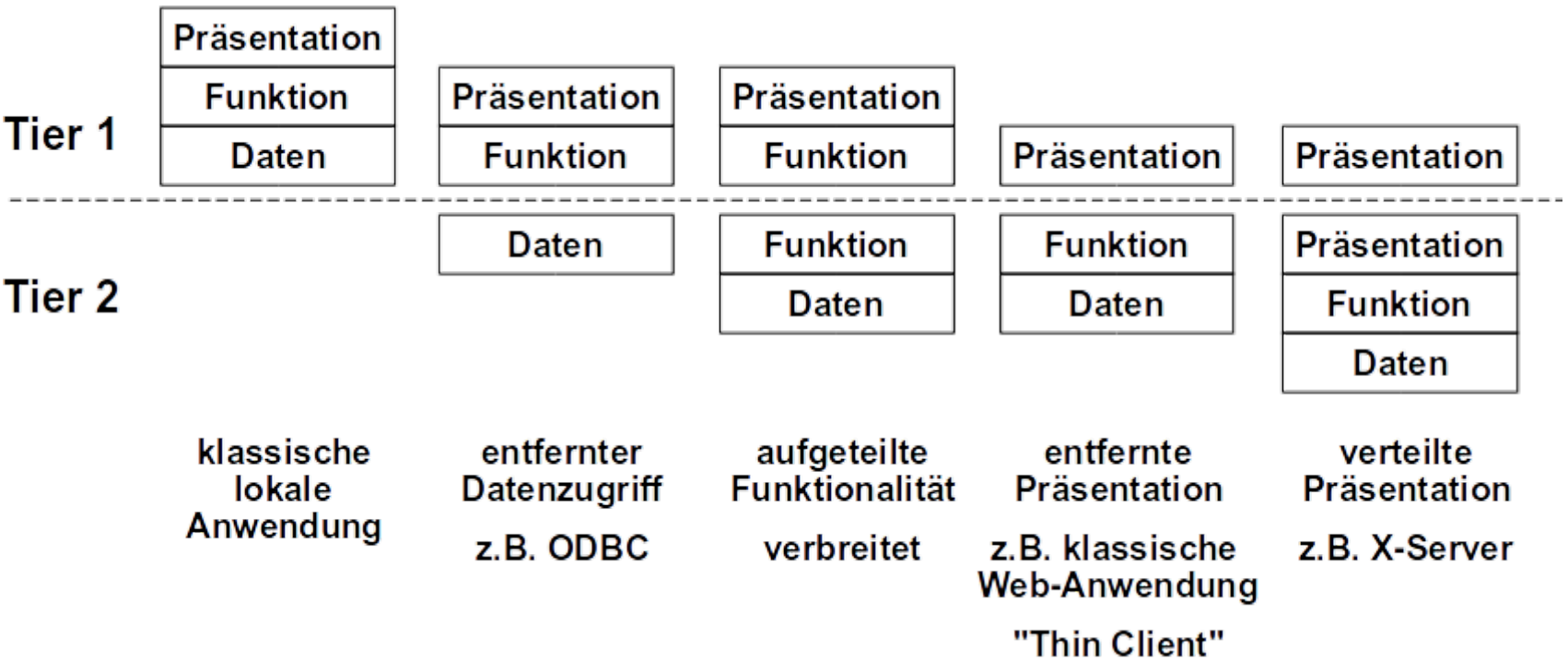
- Enthält Client-Strang (Tier 1) und Server-Strang (Tier 2)
- Einfache mögliche Aufteilungen



Two-Tier-Architektur (2)

Hochschule RheinMain

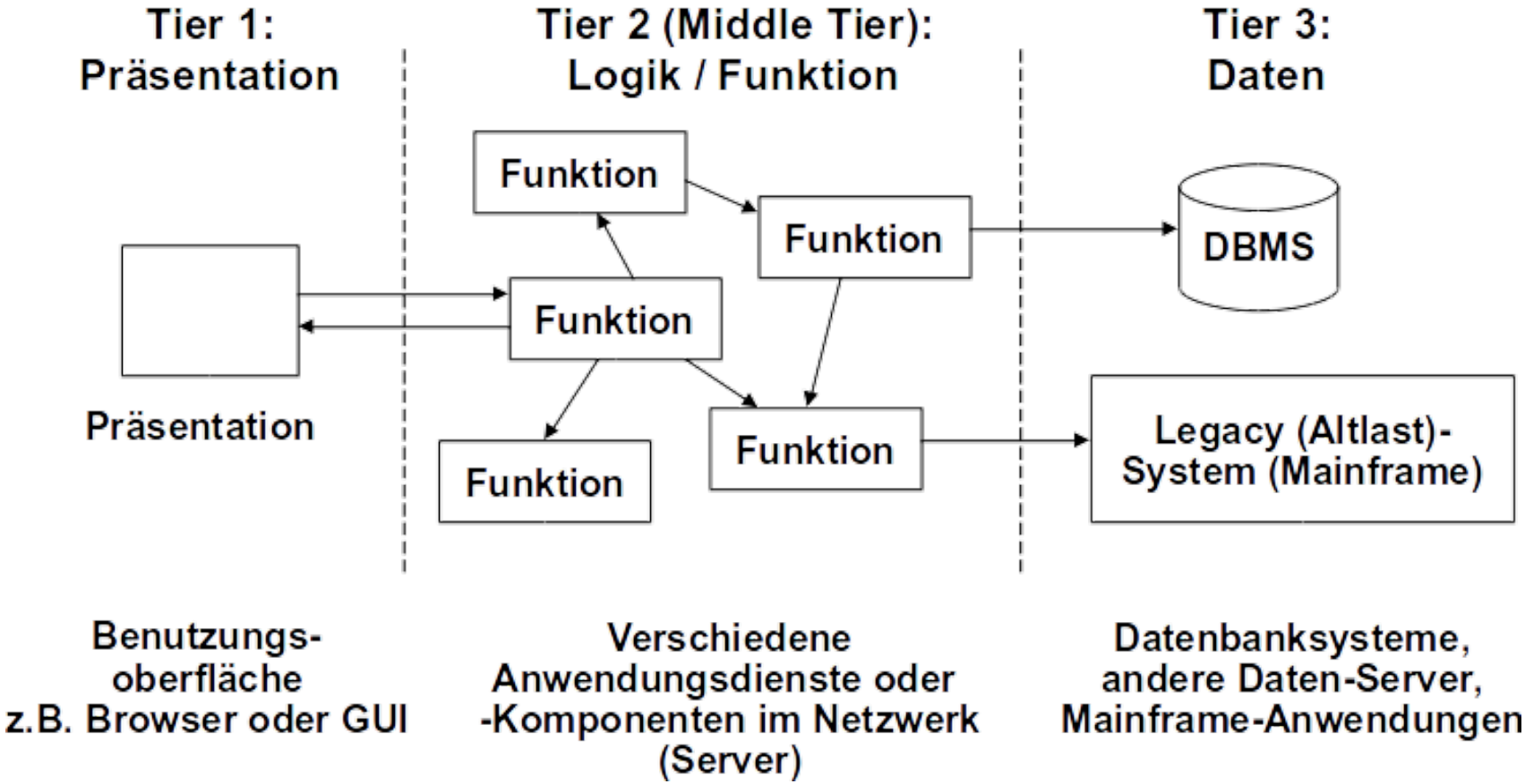
- Übersichtlicher, aber Tier-Anordnung falsch.



3-Tier-Architektur

Hochschule RheinMain

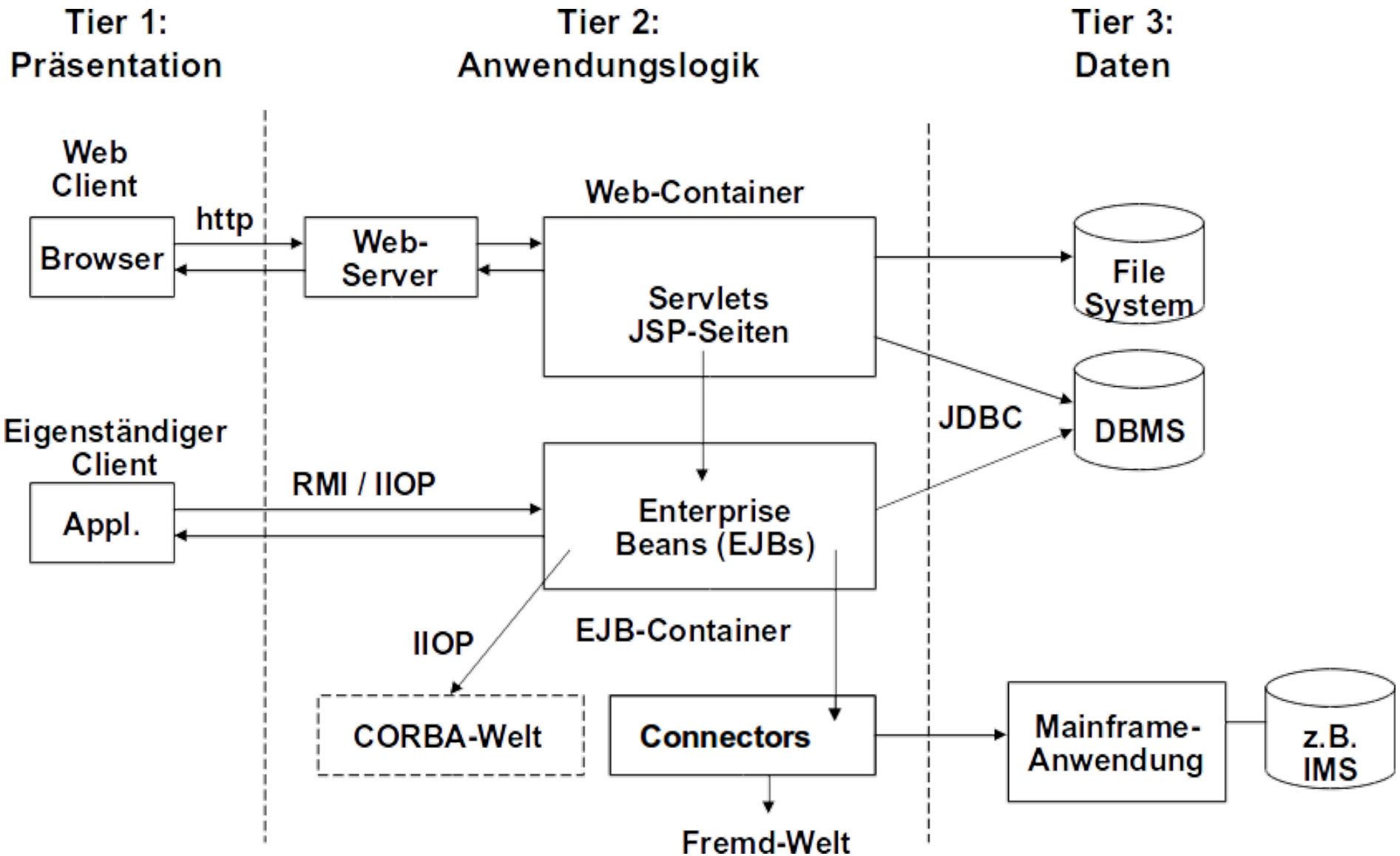
- aktuelles Strukturmodell für komplexe Anwendungen



- Erweiterung auf N-Tier-Architektur
  Spalten primär des Middle Tiers

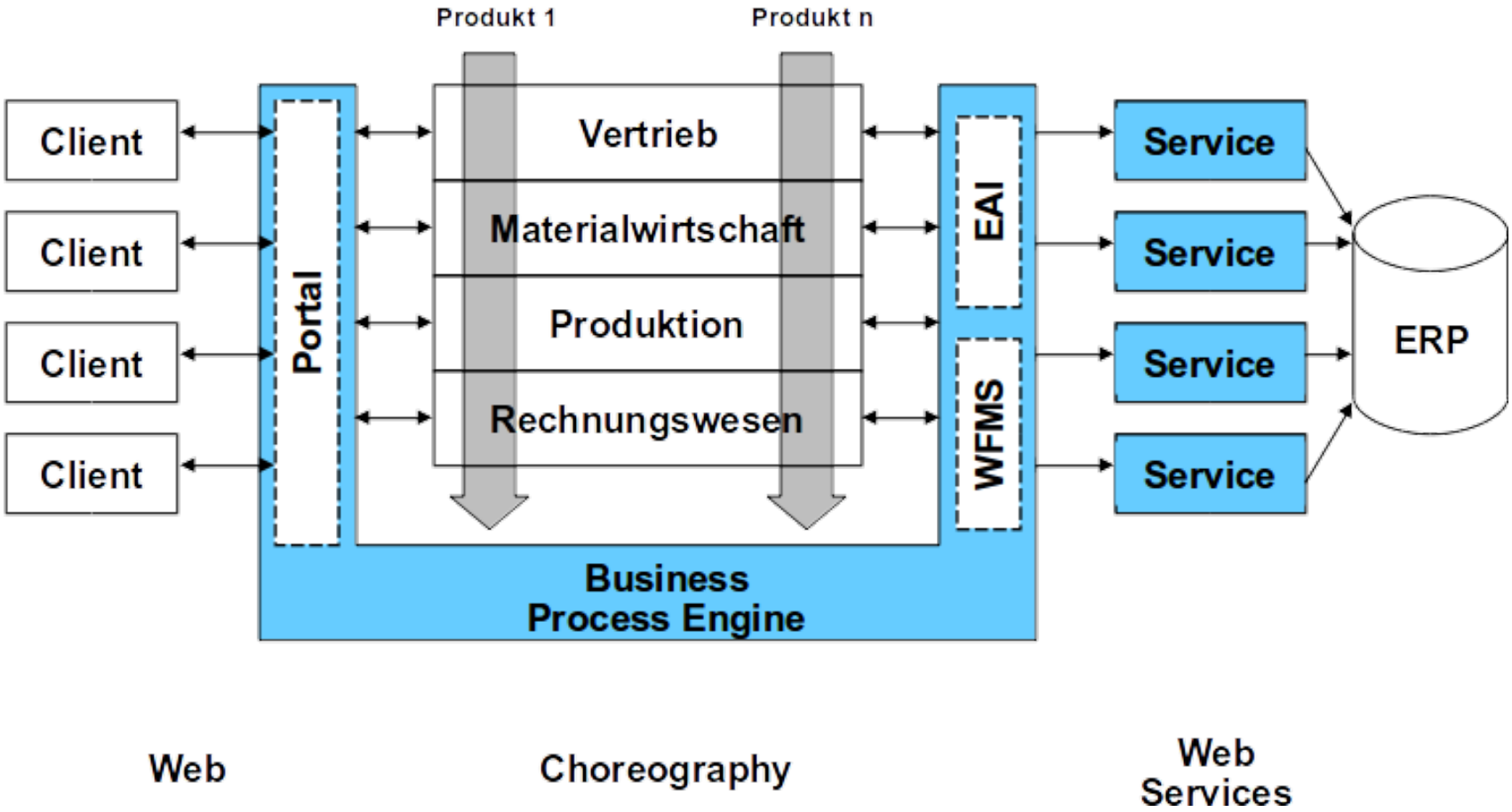
Beispiel: J2EE-Anwendung (vereinfacht)

Hochschule RheinMain



SOA-Modell

Hochschule RheinMain



Nach Scheer

SOA aus technischer Sicht

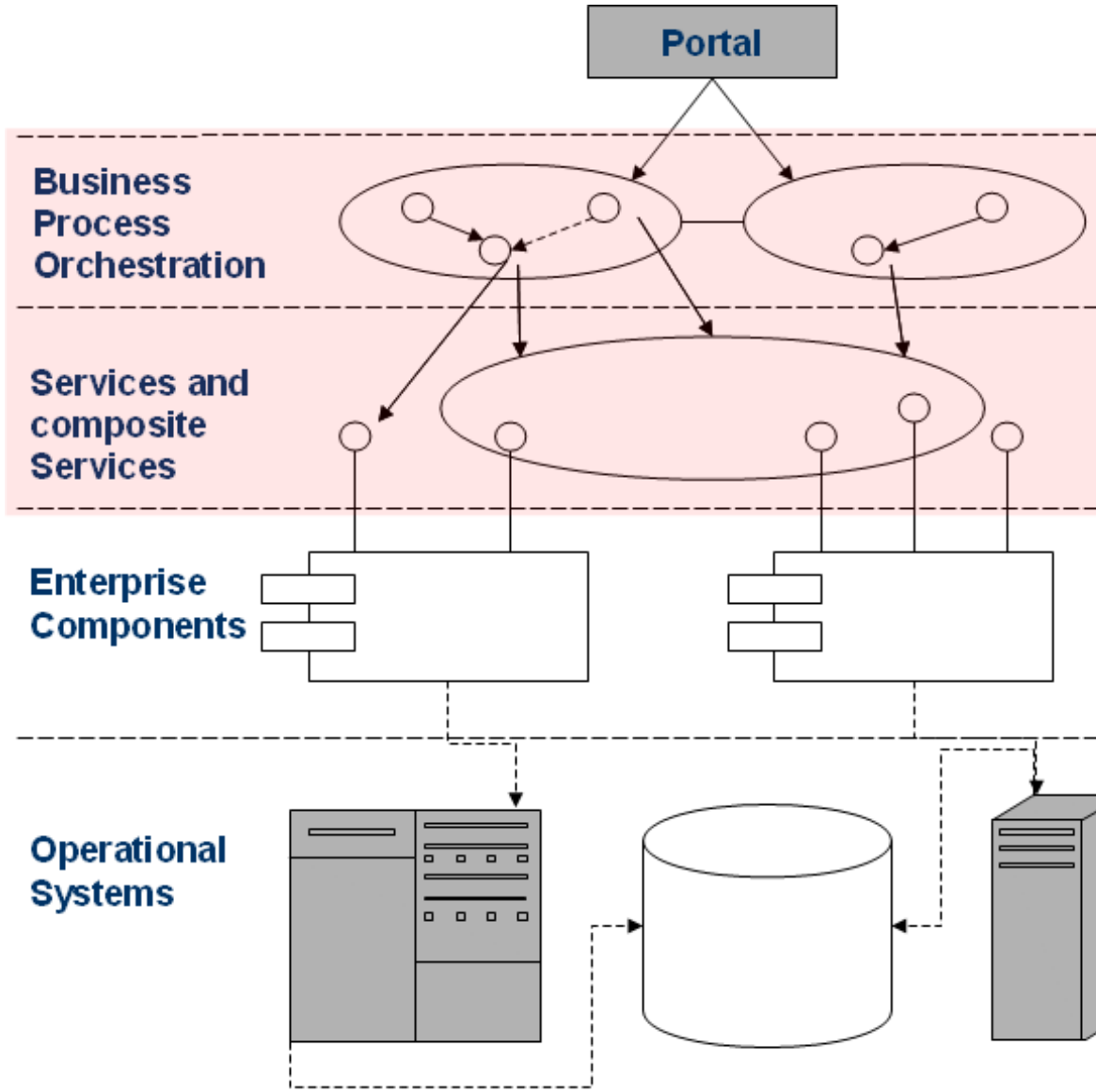
Hochschule RheinMain

Geschichtetes System

- IT-Geschäftskomponenten nutzen Ressourcen
- Komponenten stellen Teilfunktionalität als Dienst bereit
- Komplexe Dienste können aus einzelnen Basisdiensten kombiniert werden
- Geschäftsprozesse verknüpfen Dienste zu Anwendungen (Choreography / Orchestration)

Optional

- Enterprise Service Bus zur Kommunikation über Protokollgrenzen hinweg



Zusammenfassung

Hochschule RheinMain

- Eine Middleware stellt eine Schicht zwischen Betriebssystem und Anwendung bereit, um verteilte Anwendungen von den darunter liegenden Schichten abstrahieren zu können.
- Middleware-Architekturen beschreiben die verteilbaren Einheiten und Interaktionsmodelle.
- Bei dem Entwurf eines verteilten Systems können je nach Anforderung unterschiedliche Architekturmodelle zum Einsatz kommen.