

Kapitel 7c) Systementwurf – „Übergeordnete Belange“

Stand: 01.12.2019

Richtung der <<manifest>-Pfeile
korrigiert (Folie 22 bis 24)

Überblick

Bisher

1. Entwurfsziele

Definition
Abwägungen

2. System Dekomposition

Ebenen / Partitionen
Kohärenz / Kopplung

3. Hardware / Software Zuordnung

Kaufen vs. Selbermachen
Netzwerktopologie
Allokation

4. Persistente Datenverwaltung

Dateien
Datenbanken
Datenstrukturen

5. Nebenläufigkeit

Identifizierung von
Thread

Nun: Klärung übergeordneter Belange

8. Grenz- fälle

Initialisierung
Fehler
Ende

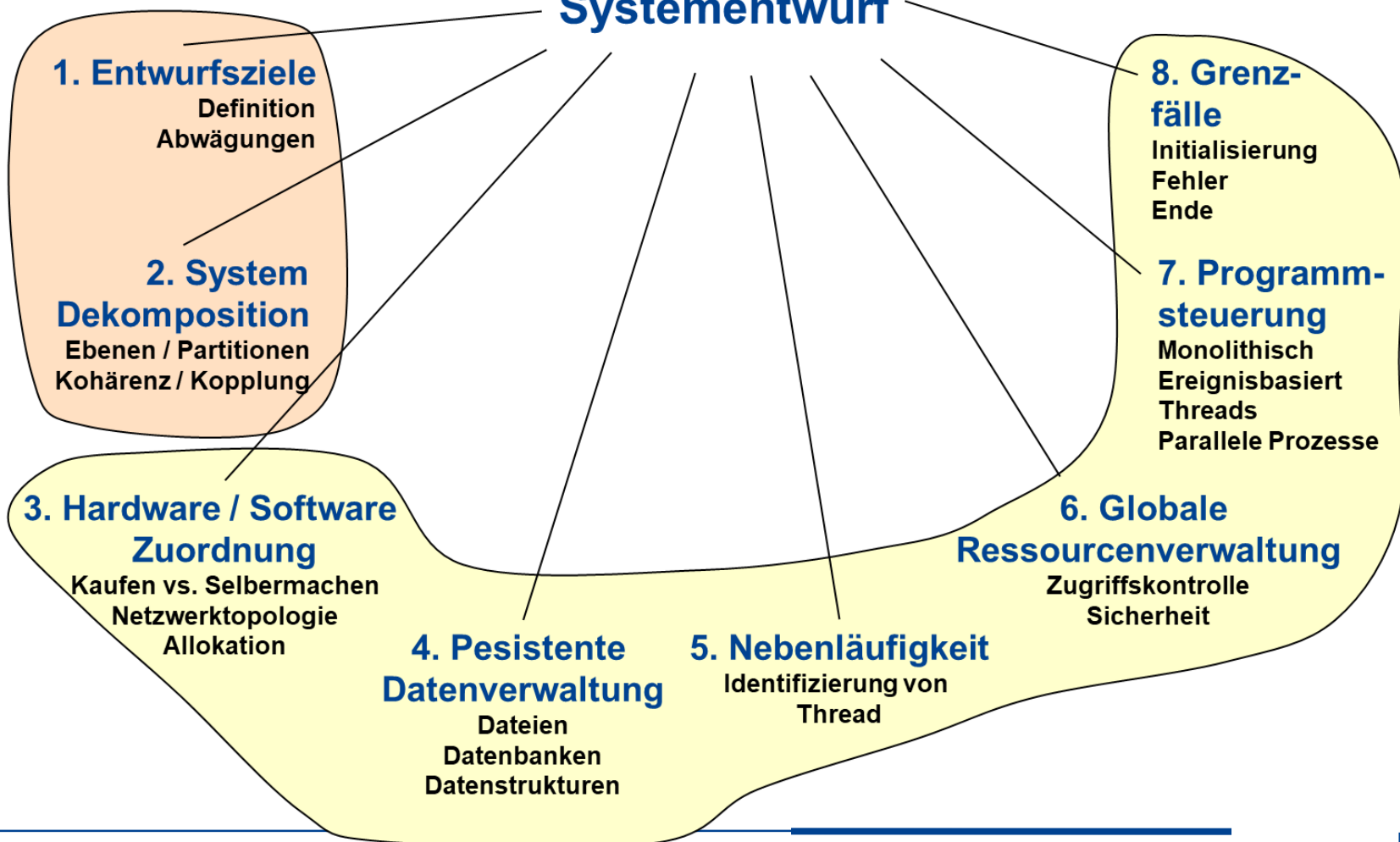
7. Programm- steuerung

Monolithisch
Ereignisbasiert
Threads
Parallele Prozesse

6. Globale Ressourcenverwaltung

Zugriffskontrolle
Sicherheit

Systementwurf



Nebenläufigkeit

3. Nebenläufigkeit

- Identifizieren nebenläufiger Ausführungsstränge und Behandeln von Fragen zur Nebenläufigkeit.
- Entwurfsziel: Reaktionszeit, Performance.
- Threads („Fäden“, „Stränge“)
 - ◆ Ein Thread ist ein Pfad durch eine Menge von Zustandsdiagramme, wobei stets genau ein Objekt zur selben Zeit aktiv ist.
 - ◆ Ein Thread bleibt in einem Zustandsdiagramm bis ein Objekt einen Event an ein anderes Objekt sendet und auf einen anderen Event wartet
 - ◆ Thread (ab-)spaltung: Ein Objekt sendet ein asynchrones Event

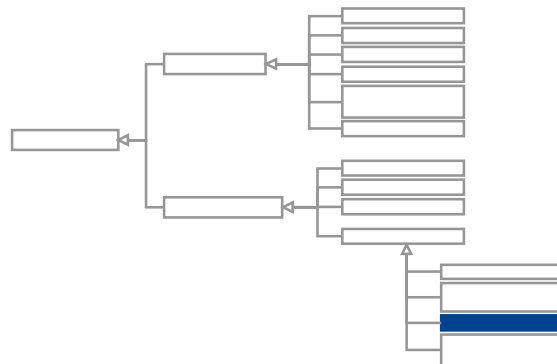
Nebenläufigkeit (Fortsetzung)

- Zwei Objekte heißen inhärent nebenläufig, wenn sie zur gleichen Zeit Events empfangen können ohne zu interagieren
- Inhärent nebenläufige Objekte sollten verschiedenen Threads zugeordnet werden
- Objekte mit sich wechselseitig ausschließenden Aktivitäten sollten demselben Thread zugeordnet werden (Warum?)

Fragen zur Nebenläufigkeit

- Welche Objekte des Objektmodells sind unabhängig?
- Welche Threads sind identifizierbar?
- Bietet das System vielen Nutzern Zugriff?
- Kann eine einzelne Anfrage an das System in mehrere Teilanfragen zerlegt werden?
- Können diese Teilanfragen parallel abgearbeitet werden?

Realzeitanforderungen und UML Timing-Diagramme

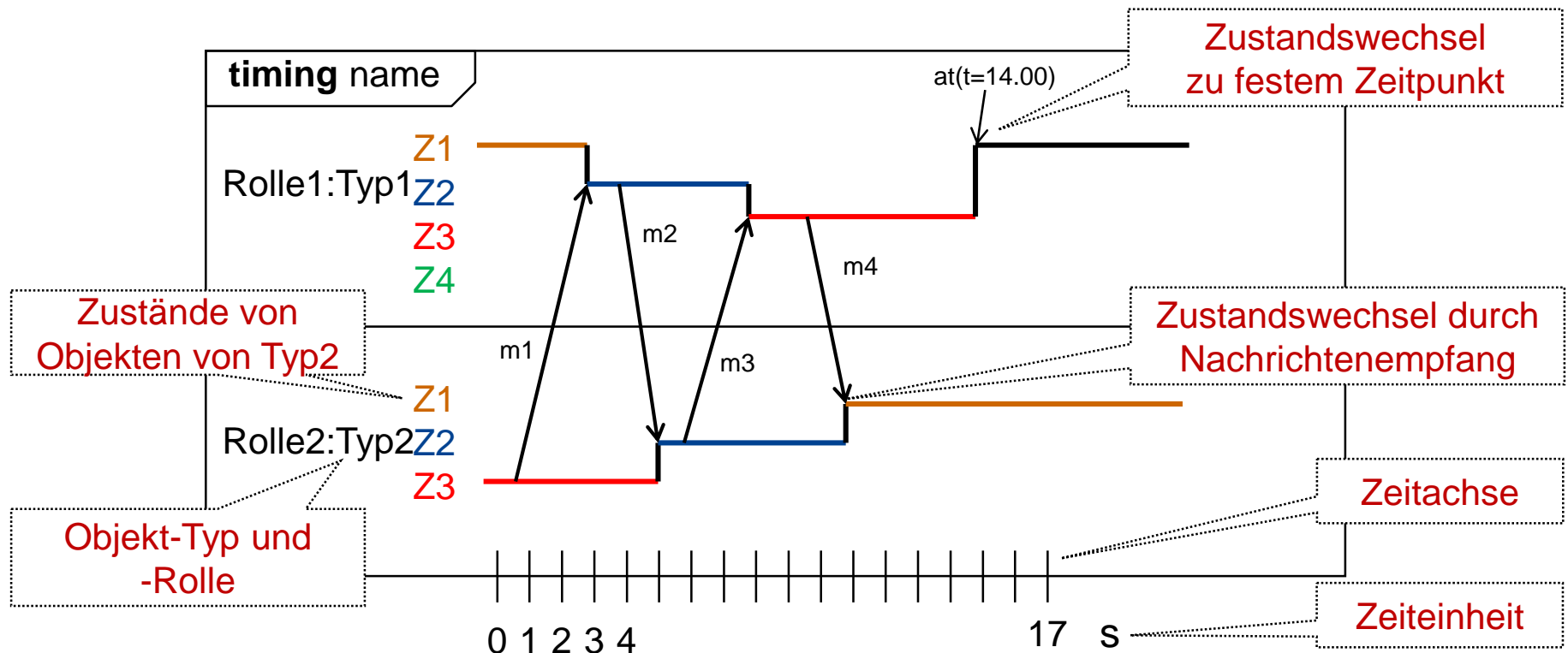


3a. Realzeitanforderungen

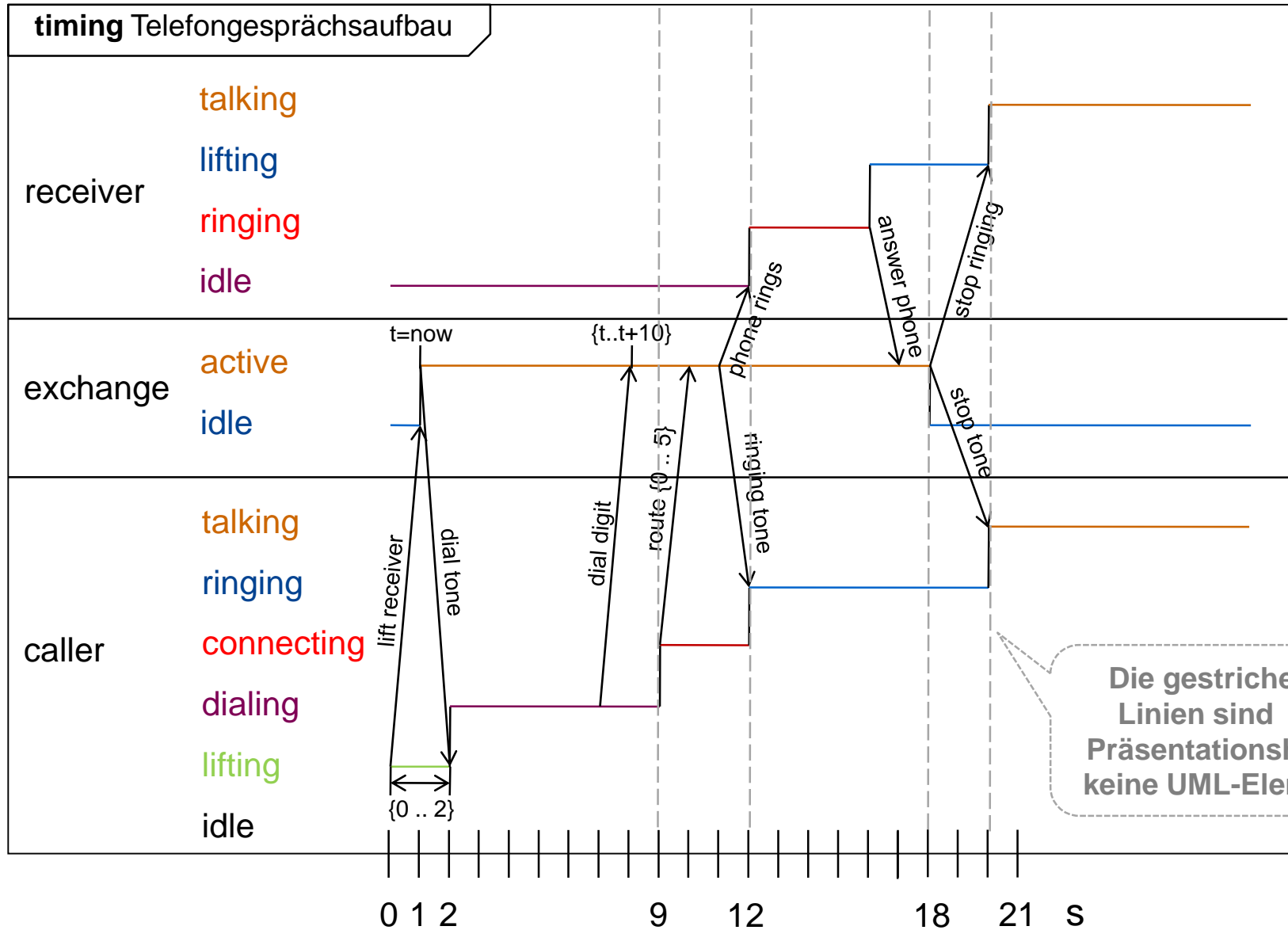
- Realzeitanforderungen bezeichnen Anforderungen an Software in einer **bestimmten Zeitspanne** zu reagieren oder etwas zu einem **bestimmten Zeitpunkt** zu tun
- Meistens geht es um sehr kurze Zeitspannen (Sekunden und alles darunter)
- Typischerweise bei "Eingebetteten Systemen" (Mischung aus Hard- und Software)
 - ◆ Im Fahrzeug- und Maschinenbau, Telekommunikation, Anlagen, ...
- Dilemma
 - ◆ Software ist flexibler (leichter austauschbar und wartbar) und kostengünstiger
 - ◆ Aussagen über die absolute Zeit, die ein bestimmter Aufruf dauert sind aber sehr schwer zu treffen
 - ⇒ Problem "Dynamisches Binden": Was wird denn nun aufgerufen?
 - ⇒ Problem "Garbage Collection": Wann unterbricht sie evtl. einen Aufruf?
 - ⇒ ...

Zeitverlaufdiagramm (Timing Diagram)

- Modelliert zeitabhängige Zustandsübergänge der Interaktionspartner
 - ◆ Auslöser ist fester Zeitpunkt oder Nachrichtenaustausch
- Nützlich bei Interaktionen mit zeitkritischem Verhalten (Realzeitanforderungen)
- Stellt exemplarischen Ablauf da (keine Kontrollstrukturen o.ä.)



Zeitdiagramm ▶ „Telefongespräch“



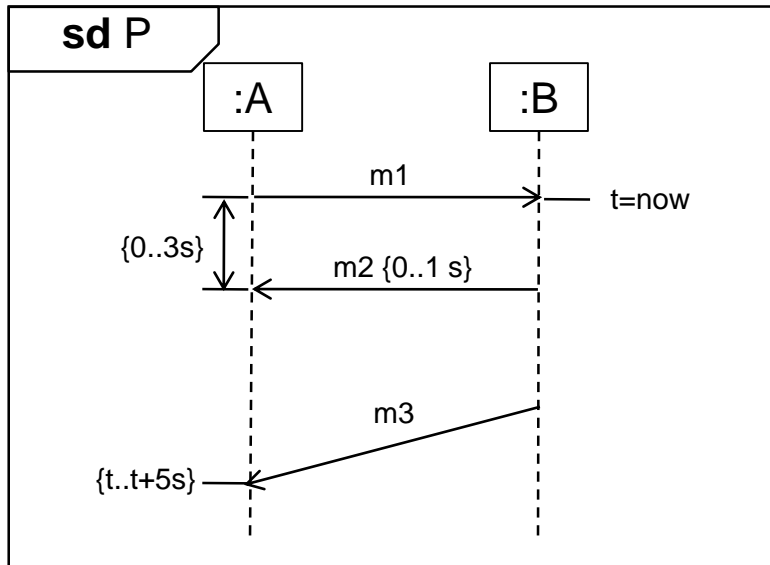
Die gestrichelten Linien sind nur Präsentationshilfen, keine UML-Elemente

Sequenz- versus Zeitverlaufdiagramm

Gleiche Zeitinformationen darstellbar (Zeitpunkt, relativer und absoluter Zeitraum)

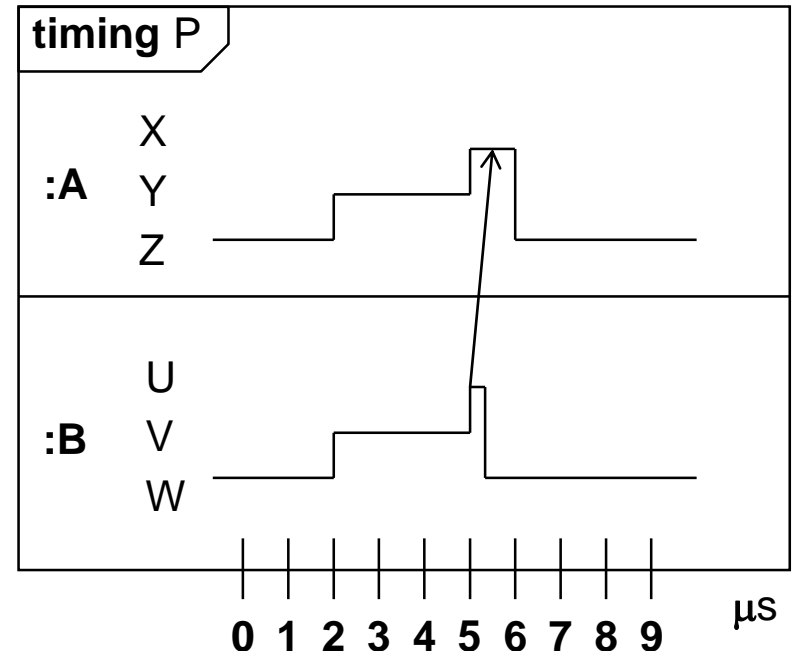
Sequenzdiagramm mit Zeit

- Kontrollstrukturen („Fragmente“ für Bedingung, Wiederholung, Abbruch, ...)



Timing Diagramm

- Zusammenhang von Nachrichten und Zustandsübergängen



Hardware- / Software-Zuordnung

4. Hardware/Software Zuordnung

- Diese Aktivität befasst sich mit drei Fragen:
 - ◆ Wie sollen wir jedes einzelne Subsystem realisieren
 - ⇒ In Hardware oder in Software?
 - ◆ Welche Hard- / Software ist schon verfügbar?
 - ⇒ Komponenten von Drittanbietern die man nutzen kann
 - ⇒ Altsysteme die man integrieren muss
 - ◆ Wie wird das Objektmodell auf die gewählte Hard- und Software abgebildet?
 - ⇒ Komponenten in die Realität abbilden → auf Rechner / Prozessor, Speicher, I/O
 - ⇒ Assoziationen in die Realität abbilden → auf Bus-/Netzwerktopologie
- Viele Schwierigkeiten beim Entwurf eines Systems sind die Folge von Kunden-Vorgaben bzgl. Hard- und Software.
 - ◆ „Wir haben gerade erst X Millionen für System Y ausgegeben...“
 - ◆ „Aufgabe X muss von Hard- /Software Y gelöst werden.“

Zuordnung von Objekten / Klassen / Komponenten

- Auf Rechner / Prozessor
 - ◆ Ist die geforderte Berechnungsgeschwindigkeit zu hoch für einen einzelnen Prozessor?
 - ◆ Bringt die Verteilung der Aufgaben auf verschiedene Prozessoren einen Geschwindigkeitsgewinn?
 - ◆ Wie viele Prozessoren sind für den dauerhaft stabilen Betrieb unter Dauerlast notwendig?
- Auf Speicher
 - ◆ Ist genug Speicher vorhanden, um Belastungsspitzen abzufangen?
- Auf I/O
 - ◆ Reicht die Kommunikationsbandbreite zwischen den Hardware-Einheiten, auf denen Subsysteme eingesetzt werden, um die gewünschte Reaktionszeit zu garantieren?

20.000.000 Kunden
1% Änderungen / Tag
200.000 Anfragen / Tag
60% zwischen 18 -20 Uhr
120.000 / 2*3.600 TPS
= 1200 / 72 TPS
= 100 / 6 TPS
→ Mindestens 17
Transaktionen pro Sek.

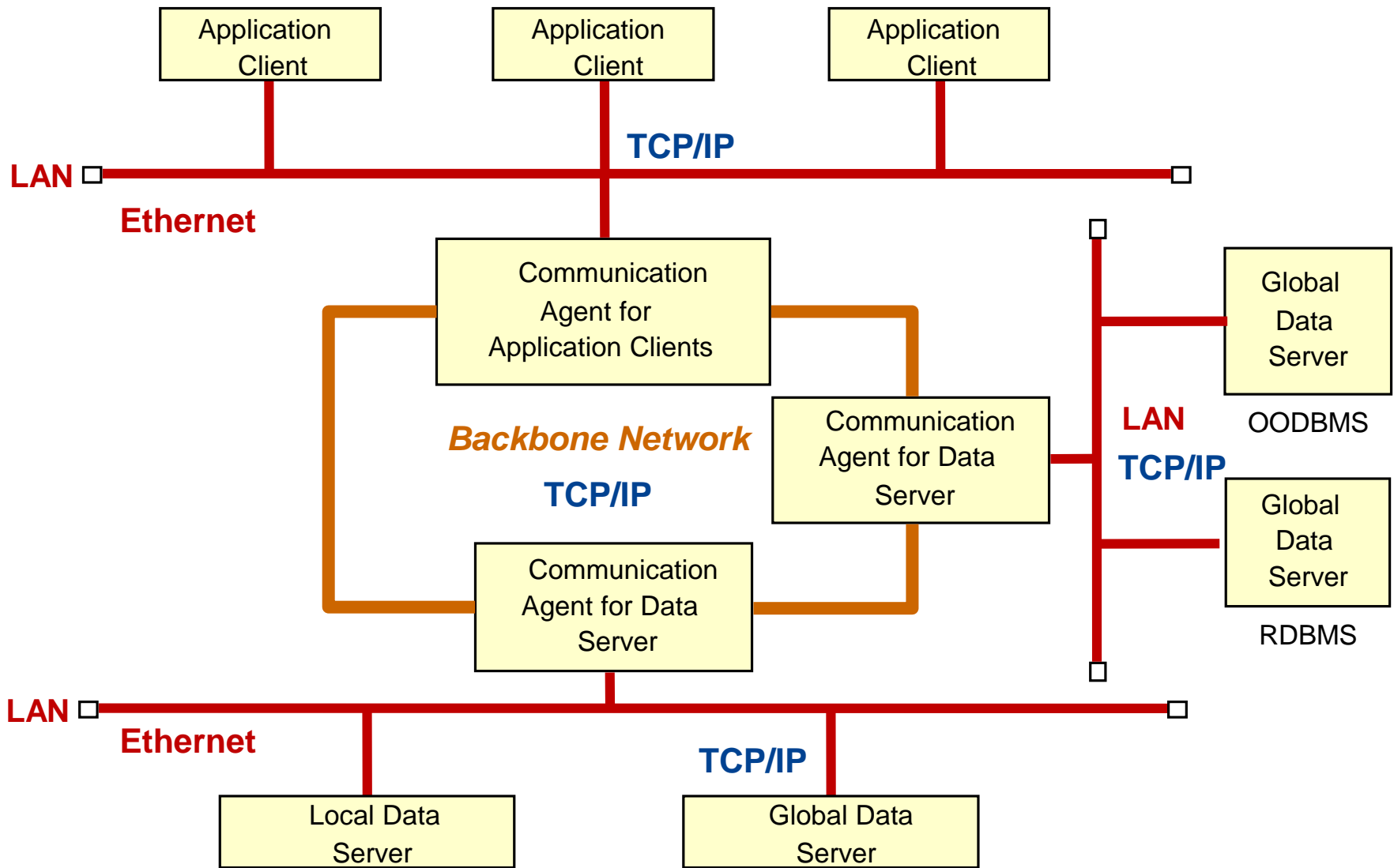
Zuordnung der Assoziationen: Kommunikationstopologie

- Beschreibe die physikalische Topologie der Hardware
 - ◆ Entspricht oft der physikalischen Schicht in ISO's OSI Referenzmodell
 - ⇒ Welche Assoziationen werden auf physikalische Verbindungen abgebildet?
 - ⇒ Welche <<benutzt>>-Beziehungen aus dem Analyse-/Entwurfsmodell korrespondieren mit physikalischen Verbindungen?
- Beschreibe die logische Topologie (Assoziationen zwischen den Subsystemen)
 - ◆ Identifiziere Assoziationen, die nicht direkt auf physikalische Verbindungen abzubilden sind:
 - ⇒ Wie sollen diese Assoziationen implementiert werden?

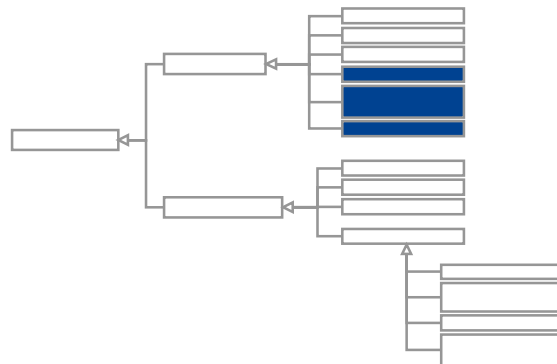
Netzwerktopologie bei verteilten Systemen

- Wenn die Architektur verteilt ist, müssen wir auch die Netzwerkarchitektur beschreiben (Kommunikationssystem).
- Zu stellende Fragen
 - ◆ Was ist das Übertragungsmedium? (Ethernet, Wireless)
 - ◆ Welche “Quality of Service” (QOS) ist vorhanden/erforderlich? Welche Art von Kommunikationsprotokollen können genutzt werden?
 - ◆ Sollen Interaktion asynchron, synchron oder sperrend sein?
 - ◆ Welche Anforderungen gibt es an die Bandbreite zwischen den Subsystemen?
 - ⇒ Stock Price Change -> Broker
 - ⇒ Icy Road Detector -> ABS System

Typisches Beispiel für physikalische Topologie



Verteilungsdiagramme (Deployment Diagrams)



Verteilungsdiagramm (Einsatzdiagramm / Deployment Diagram)

- Zeigt
 - ◆ **Ausführungsknoten** (Rechner und Laufzeitumgebungen),
 - ◆ **Kommunikationsbeziehungen** zwischen Ausführungsknoten,
 - ◆ **Manifestation** (=Realisierung) von Komponenten durch Artefakte,
 - ◆ **Einsatz** von Artefakten auf Ausführungsknoten,
 - ◆ **Konfiguration** des Einsatzes,
 - ◆ sonstige Beziehungen (Abhängigkeits-Pfeile zeigen von der abhängigen Komponente weg)

- Nutzen: Spezifikation der
 - ◆ Hardware/Software Zuordnung
 - ◆ Subsystemdekomposition
 - ◆ Verteilung im Netzwerk
 - ◆ Einsatz zur Laufzeit

Verteilungsdiagramm ▶ Knoten

- Komponente

- ◆ Logische Einheit mit expliziten Abhängigkeiten
- ◆ Wie im Komponentendiagramm definiert



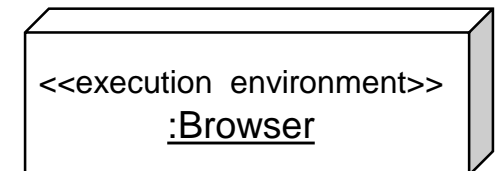
- Artefakt

- ◆ Physische Einheit, z.B. Modell, Hilfetext, Quellcode, ausführbarer Code (class-file, jar-Archiv, o-file).
- ◆ Realisierung einer oder mehrerer Komponenten



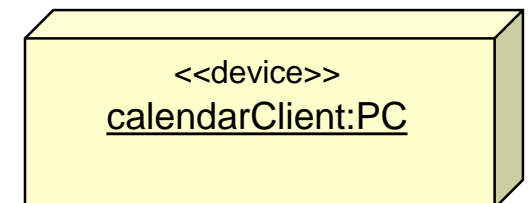
- Laufzeitumgebung („execution environment“)

- ◆ Softwaresystem in dem Artefakte zum Einsatz kommen
- ◆ Z.B. Java Virtual Machine, Applikationsserver, ...



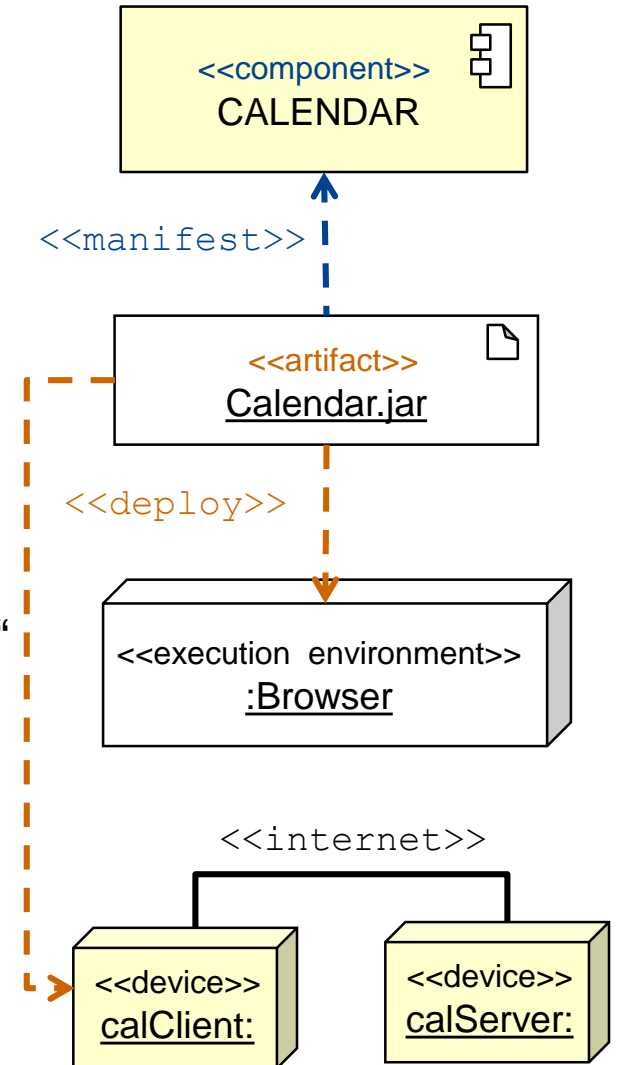
- Gerät („device“)

- ◆ Physikalisches Gerät auf dem Artefakte zum Einsatz kommen (Rechner)

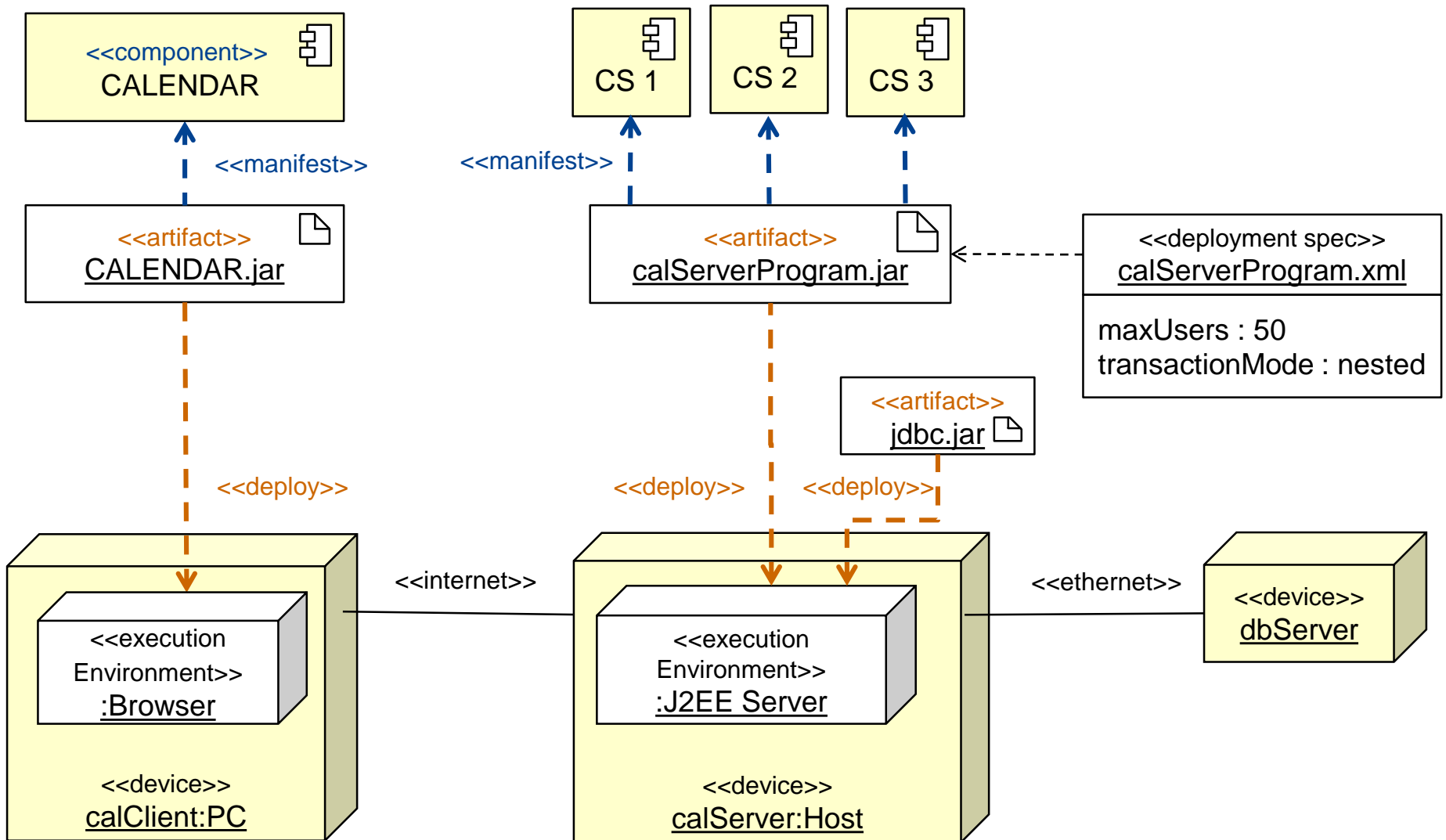


Verteilungsdiagramm ▶ Kanten

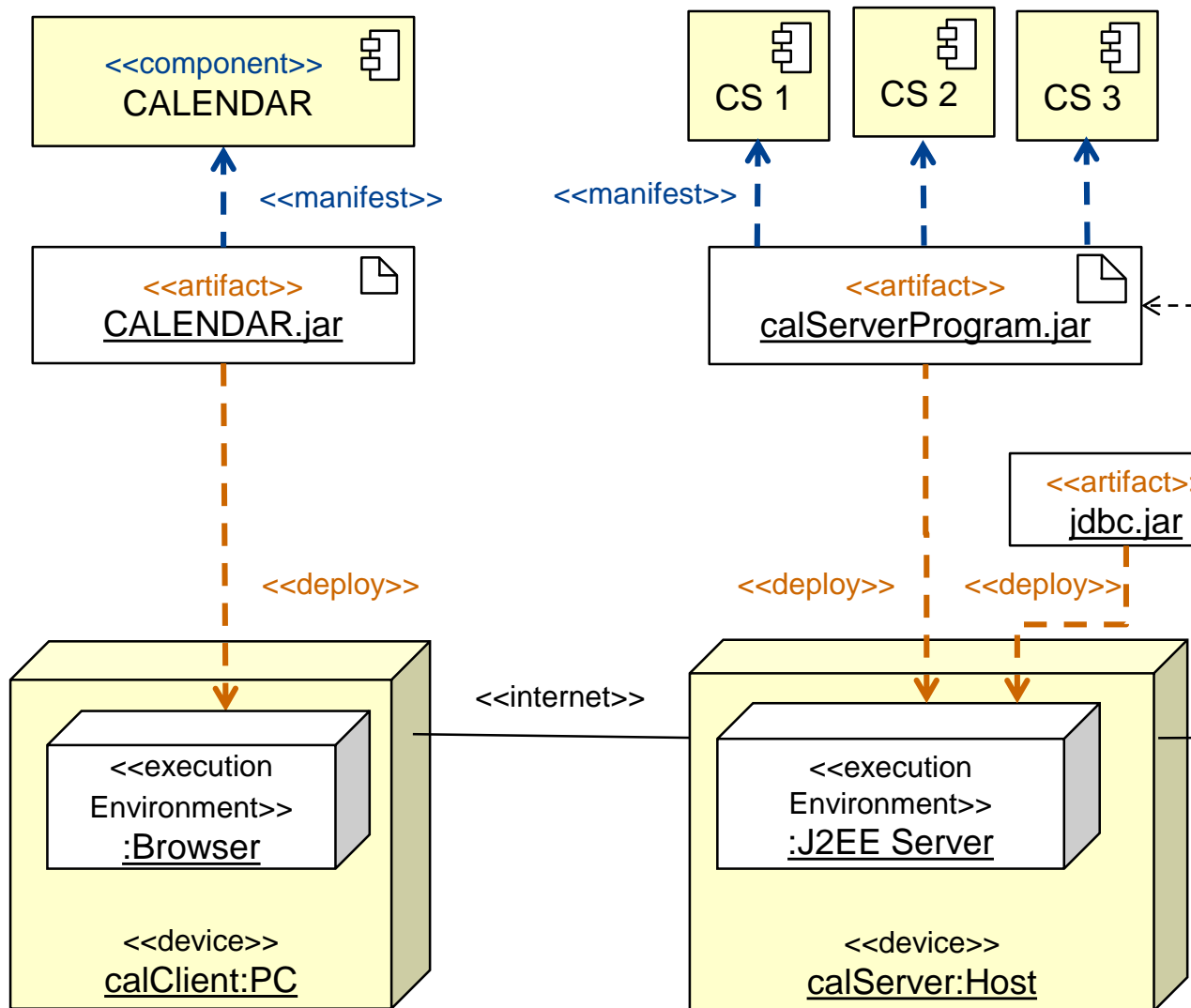
- **Manifestation** (<<manifest>>)
 - ◆ Komponente ist durch Artefakt realisiert
 - ◆ „Artefact is manifestation of component“
- **Einsatzbeziehung** (<<deploy>>)
 - ◆ Artefakt wird auf Ausführungsumgebung oder Gerät eingesetzt
 - ◆ „Artefact is deployed on device / in environment“
- **Kommunikationsbeziehung**
 - ◆ Physische Verbindung über die Ausführungsumgebungen kommunizieren
 - ◆ Art kann als Stereotyp angegeben werden, z.B. <<internet>>, <<ethernet>>, ...



Verteilungsdiagramm ▶ Beispiel



Verteilungsdiagramm ▶ Beispiel ▶ Erläuterung



“Die Kalender-Komponente kommt in Form der Datei “Calendar.jar” im Browser auf dem Kunden-PC zum Einsatz.”

“Die Komponenten CS1, ..., CS3 kommen in Form der Datei „calServerProgram.jar“ in dem J2EE-Applikationsserver auf dem „calServer“ Rechner zum Einsatz.

Sie unterstützt geschachtelte Transaktionen und max. 50 Benutzer.“

“Die Kunden-PCs greifen über das Internet auf den Server zu. Der kommuniziert lokal per Ethernet mit dem Datenbankserver“.



Datenmanagement

Datenmanagement

- Einige Objekte in den Modellen müssen persistent sein
 - ◆ Trenne sauber zwischen den Subsystemen, die Persistenz-Dienste anbieten und denen, die sie nutzen.
 - ◆ Definiere klare Schnittstellen.
- Ein nicht persistentes Objekt kann durch (interne) Datenstrukturen realisiert werden
- Ein persistentes Objekt kann folgendermaßen realisiert werden
 - ◆ Dateien
 - ⇒ Billig, einfach, permanente Speicherung
 - ⇒ Low level (Lese-/Schreiboperationen)
 - ⇒ Der Anwendung muss gegebenenfalls Code hinzugefügt werden, um eine angemessene Abstraktion zu realisieren
 - ◆ Datenbank
 - ⇒ Mächtig, leicht zu portieren
 - ⇒ Unterstützt mehrere Schreiber und Leser

Datei oder Datenbank?

- Persistenz via **Dateien** benutzt man für
 - ◆ Große, unstrukturierte Daten (Bitmaps, Core Dumps, Event Traces)
 - ◆ Daten mit geringer Informationsdichte (Archivdateien, Logdateien)
 - ◆ Daten, die nur kurzzeitig zu speichern sind
- Persistenz via **Datenbanken** benutzt man für
 - ◆ Strukturierte Daten (→ Relationen),
die in verschiedenen Detailstufen (→ Sichten)
von vielen Nutzern (→ Transaktionsmanagement)
zugreifbar sein müssen
 - ◆ Daten, die von vielen Anwendungen (→ Transaktionsmanagement)
benutzt werden
 - ◆ Daten, die auf verschiedenen Plattformen (→ Datenabstraktion)
zur Verfügung stehen müssen

Was muss bei Benutzung einer Datenbank beachtet werden?

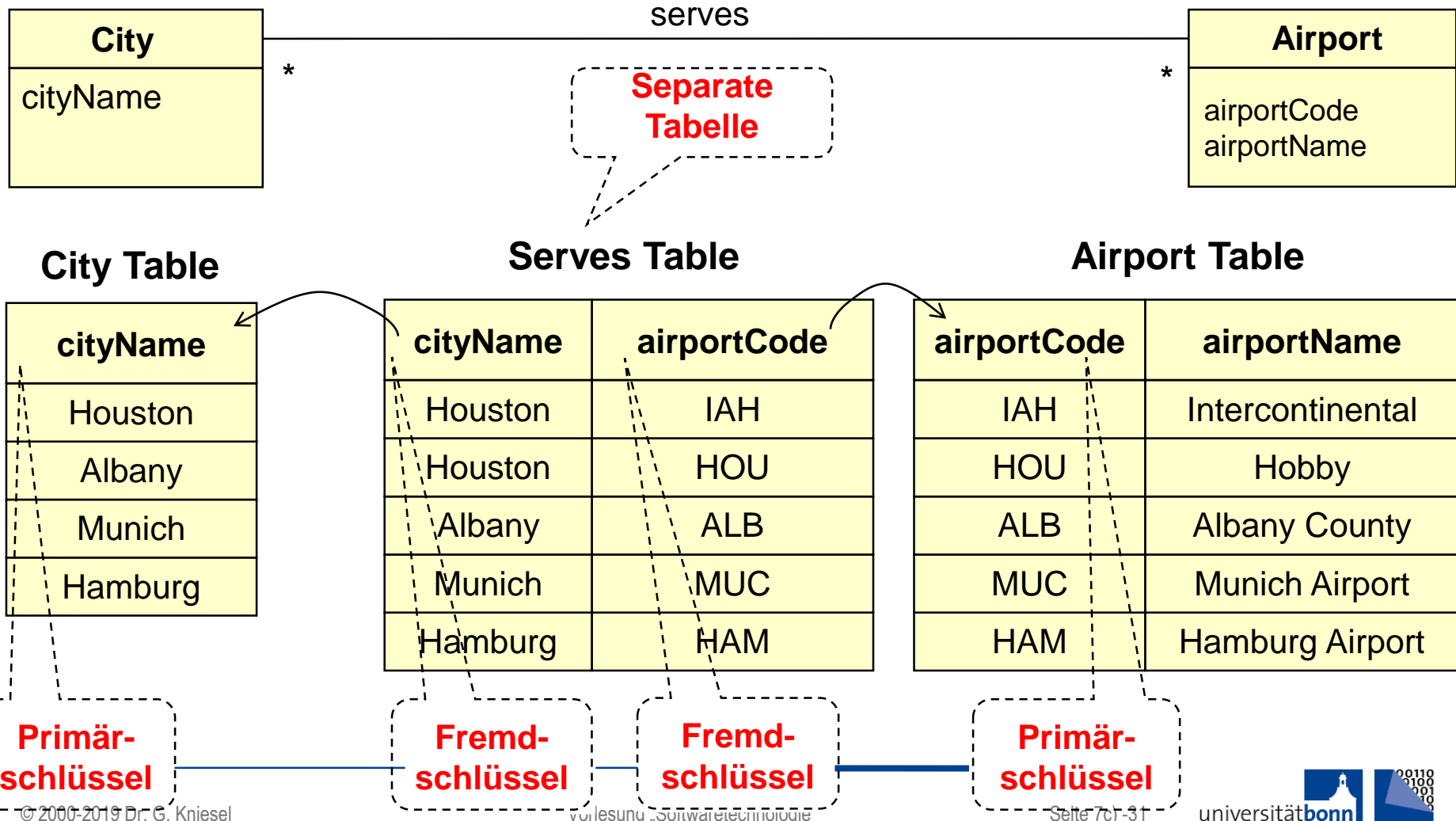
- Speicherplatz
 - ◆ Die Datenbank benötigt in etwa die dreifache Größe der Daten (→ Indices)
- Antwortzeit
 - ◆ Datenbanken sind I/O- oder kommunikationsabhängig (verteilte Datenbanken). Die Antwortzeit wird auch von der CPU Zeit, Locks und Verzögerungen durch häufige Bildschirmausgaben beeinflusst.
- Lock Arten
 - ◆ **Pessimistisches Locking**: Lock wird vor dem Zugriff auf ein Objekt gesetzt und erst wieder aufgehoben wenn der Zugriff beendet wurde.
 - ◆ **Optimistisches Locking**: Häufiger Lese- / Schreibzugriff (hohe Parallelität!) Wenn die Aktivität beendet wurde, prüft die Datenbank ob Konflikte bestehen; wenn ja werden alle Änderungen verworfen.
- Administration
 - ◆ Große Datenbanken benötigen ausgebildete Support Mitarbeiter, um Sicherheits-Mechanismen, Datenträgerplatz und Backups zu verwalten, die Leistung zu überwachen und Einstellungen anzupassen.

Abbildung eines Objektmodells auf eine relationale Datenbank

- UML Objektmodelle können auf relationale Datenbanken abgebildet werden:
 - ◆ Etwas Verlust entsteht, weil alle UML-Konstrukte auf ein einziges relationales Datenbankkonstrukt abgebildet werden – die Tabelle.
- UML Zuordnungen
 - ◆ Jede **Klasse** wird auf eine **Tabelle** abgebildet
 - ◆ Jedes **Attribut** einer Klasse wird auf eine **Spalte** abgebildet
 - ◆ Eine **Instanz** einer Klasse repräsentiert eine **Zeile** in der Tabelle
 - ◆ Eine **n-zu-m Beziehung** wird in eine **eigene Tabelle** abgebildet
 - ◆ Eine **1-zu-n Beziehung** wird als **Fremdschlüssel** implementiert
- Methoden werden nicht abgebildet ☹
 - ◆ „Stored procedures“ können nicht einzelnen Relationen zugeordnet werden (keine Kapselung, kein dynamisches Binden, ...)

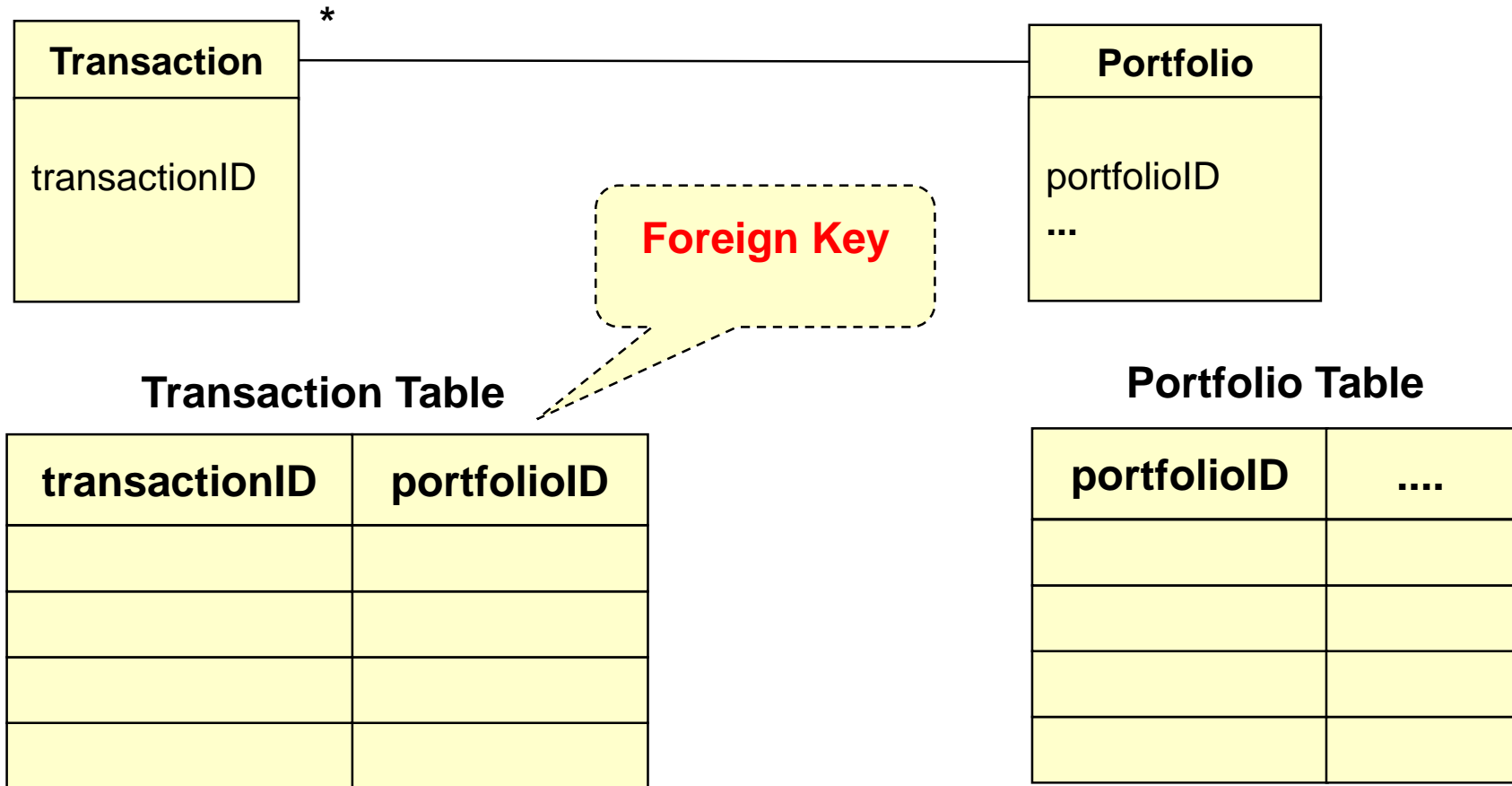
Von Objektmodellen zu Tabellen I

- N-zu-M Assoziation: Eigene Tabelle für die Assoziation



Von Objektmodellen zu Tabellen II

- 1-zu-n oder n-zu-1 Assoziationen: Verdeckte Fremdschlüssel



Von Objektmodellen zu Tabellen ▶

Werkzeuge

- Applikationsserver und ähnliche Werkzeuge erledigen die Abbildung eines Objektmodells auf ein relationales Schema („object relational mapping“) **automatisch**
- Gängige Systeme / Frameworks
 - ◆ Java Data Objects (JDO) – Teil der Java Enterprise APIs
 - ◆ Hibernate – Teil des Applikationsservers JBoss
 - ◆ ... Google nach „object relational mapping“ ...

Globale Ressourcenverwaltung

Globale Ressourcenverwaltung

- Ressourcenverwaltung beschreibt
 - ◆ die Zugriffsrechte für verschiedene **Akteure**
 - ◆ wie **Objekte** sich vor unberechtigtem Zugriff schützen
- Zugriffskontrollmatrix
 - ◆ **Zeilen = Akteure**
 - ◆ **Spalten = Objekte**
 - ◆ **Inhalt der Felder = zulässige Operationen**

	Objekttyp 1	Objekttyp 2	Objekttyp 3
Actor A	Op1.1, Op1.2	---	Op3.1
Actor B	Op1.2	Op2.2, Op2.3	Op3.2
Actor C	---	Op2.1	Op3.3

Bsp: Actor C darf Operation Op2.1 auf Objekten des Typs Objekttyp 2 ausführen.

Globale Ressourcenverwaltung: Realisierung der Zugriffskontrollmatrix

- Spaltenweise Aufteilung = Jedes Objekt weiss wer, was damit tun darf
 - ◆ Access Control Lists (Beispiel: „Unix“)
- Zeilenweise Aufteilung = Jeder Actor besitzt ein „Ticket“ das besagt, welche Operationen er ausführen darf
 - ◆ Capabilities (Beispiel: „Amoeba“)
 - ◆ Synonyme: „Ticket“ / „Ausweis“ / „Schlüssel“

	Objekttyp 1	Objekttyp 2	Objekttyp 3	
Actor A	Op1.1, Op1.2	---	Op3.1	
Actor B	Op1.2	Op2.2, Op2.3	Op3.2	
Actor C	---	Op2.1	Op3.3	

Bsp: Actor C darf Operation Op2.1 auf Objekten des Typs Objekttyp 2 ausführen.

Fragen zu globalen Ressourcen

- Benötigt das System eine Authentifizierung?
- Wenn ja, welches Authentifizierungsschema?
 - ◆ Nutzernamen und Passwort? → Zugriffskontrollliste (ACL)
 - ◆ Tickets? → Capability-based
- Welche Benutzerschnittstelle für die Authentifizierung?
- Wann und wie wird ein Dienst dem Rest des Systems bekannt gemacht?
 - ◆ Zur Laufzeit?
 - ◆ Beim Kompilieren?
 - ◆ Über einen TCP-IP-Port?
 - ◆ Durch einen Namen?
- Benötigt das System einen netzweiten „Name Server“?

Bestimmung des Kontrollparadigmas (Programmsteuerung)

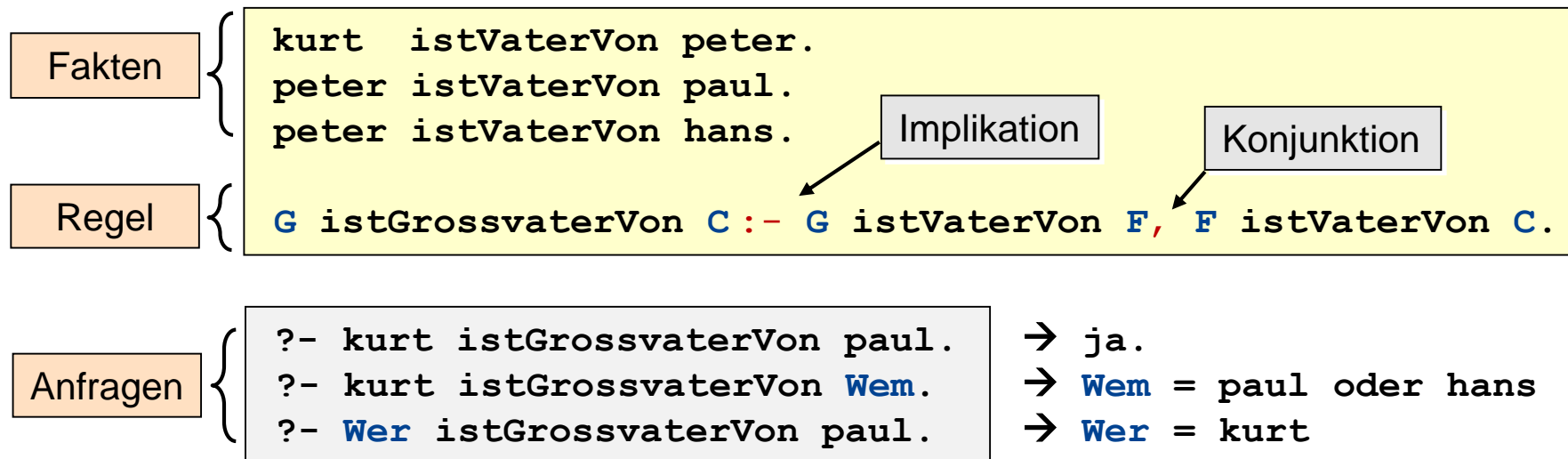
Bestimmung des Kontrollparadigmas (Programmsteuerung)

A) Implizite Kontrolle (deklarative Sprachen)

- Regelbasierte Systeme
- Logische Programmierung (Prolog)
- Datenbankabfragesprachen (SQL)

Prinzip: Sie programmieren Sachverhalte, nicht Algorithmen.

- Beispiel: Verwandtschaftsbeziehungen in „Prolog“



Bestimmung des Kontrollparadigmas (Programmsteuerung)

B. Explizite Kontrolle (prozedurale und objektorientierte Sprachen)

◆ Zentrale Kontrolle

- ⇒ Kontrolle befindet sich in einem Objekt / einer Komponente

◆ Dezentrale Kontrolle

- ⇒ Kontrolle befindet sich in verschiedenen unabhängigen Objekten
- ⇒ Geschwindigkeitsgewinn durch Parallelität versus mehr Kommunikation.
- ⇒ Evtl. Höhere Ausfallsicherheit
- ⇒ Beispiel: Nachrichten-basiertes System

◆ Prozedurgesteuerte Kontrolle

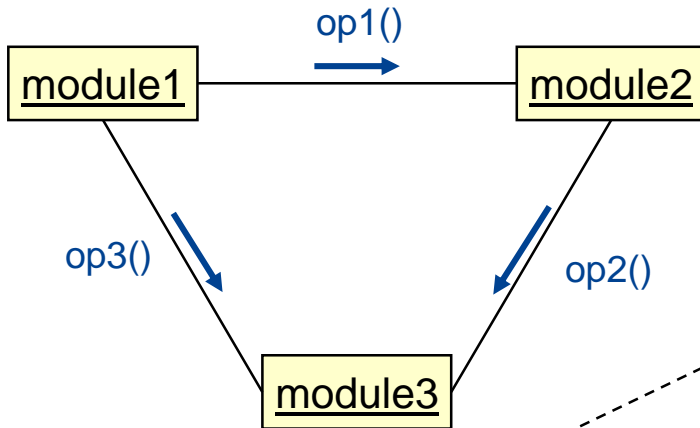
- ⇒ Kontrolle befindet sich im Programmcode.
- ⇒ Beispiel: Hauptprogramm ruft Prozeduren in Subsystemen auf.
- ⇒ Einfach, leicht zu bauen

◆ Eventgesteuerte Kontrolle

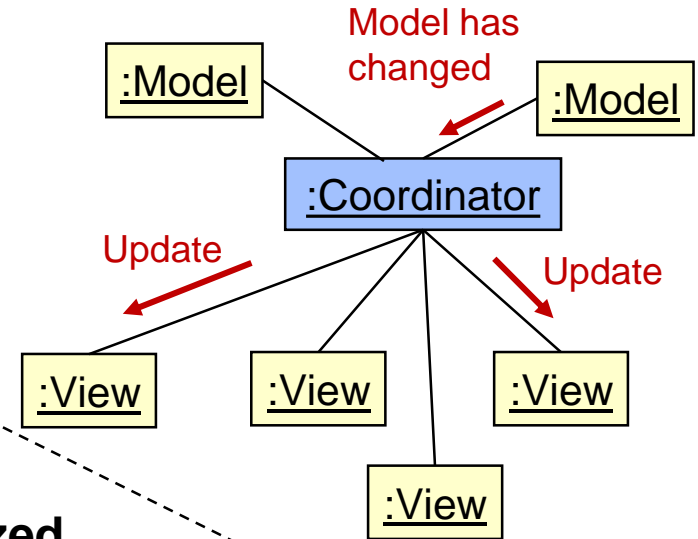
- ⇒ Kontrolle sitzt in einem Dispatcher, der Funktionen von Subsystemen durch Rückfragen aufruft.
- ⇒ Flexibel, gut für Benutzerschnittstellen

Prozedurgesteuerte vs. eventgesteuerte Kontrolle

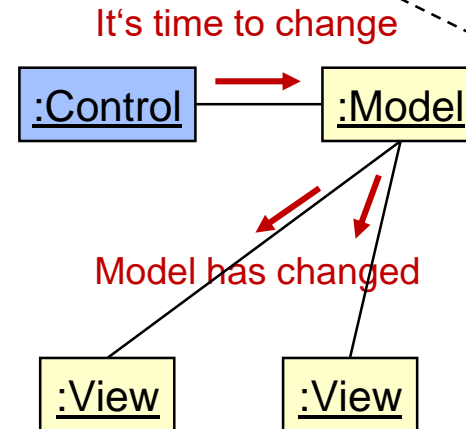
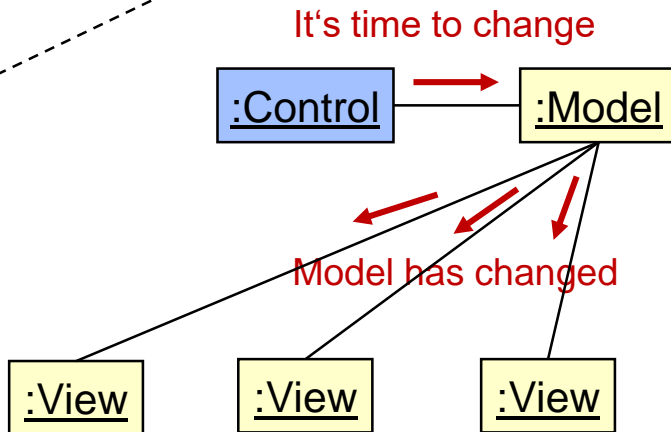
Procedure-Driven Control



Event-Based Control, centralized



Event based, decentralized



Zentrale vs. dezentrale Kontrolle

- Welches von beiden soll man benutzen?
- Zentrale Kontrolle
 - ◆ Ein Kontrollobjekt oder Subsystem kontrolliert alles (“Spinne im Netz”)
 - ◆ Änderungen in der Kontrollstruktur sehr einfach durchzuführen
 - ◆ Möglicher Flaschenhals für die Performance
 - ◆ Mögliche Vermischung verschiedener Kontrollaufgaben („tangling“)
- Dezentrale Kontrolle
 - ◆ Kontrolle ist verteilt
 - ◆ Streut die Verantwortung
 - ◆ Passt gut in die objektorientierte Entwicklung
 - ◆ Übersicht geht eventuell verloren
 - ◆ Koordination schwierig

Grenzfälle

- Die meiste Zeit beschäftigt man sich beim Systementwurf mit dem Verhalten im Betriebszustand.
- Abschliessend muss man sich aber auch mit Grenzfällen befassen.
 - ◆ Initialisierung
 - ⇒ Beschreibt, wie das System aus einem nicht initialisierten Zustand in einen Betriebszustand gebracht wird ("startup use cases").
 - ◆ Terminierung
 - ⇒ Beschreibt, welche Ressourcen vor der Beendigung aufgeräumt werden und welche Systeme benachrichtigt werden ("Terminierungs-Use Cases").
 - ◆ Fehler
 - ⇒ Viele mögliche Gründe: Programmierfehler, externe Probleme (Stromversorgung).
 - ⇒ Guter Systementwurf sieht fatale Fehler voraus ("Fehler-Use Cases").

Fragen zu den Grenzfällen

- Initialisierung

- ◆ Wie startet das System?

- ⇒ Auf welche Daten muss beim Hochfahren zugegriffen werden?

- ⇒ Welche Dienste müssen registriert werden?

- ◆ Was tut die Benutzerschnittstelle beim Startvorgang?

- ⇒ Wie präsentiert sie sich dem Nutzer?

- Terminierung

- ◆ Dürfen einzelne Subsystemen terminieren?

- ◆ Werden andere Subsysteme benachrichtigt, wenn ein einzelnes terminiert?

- ◆ Wie werden lokale Updates der Datenbank mitgeteilt?

- Fehler

- ◆ Wie verhält sich das System, wenn ein Knoten oder eine Kommunikationsverbindung ausfällt? Gibt es Backupverbindungen?

- ◆ Wie stellt sich das System nach einem Fehler wieder her?

- ◆ Unterscheidet dieser Vorgang sich von der Initialisierung?

Rückblick ▶ Zielgerichteter Systementwurf

Aktivitäten

- Identifikation von Nebenläufigkeit
- Hardware/Software Zuordnung
- Management persistenter Daten
- Globale Ressourcenverwaltung
- Wahl der Programmsteuerung
- Grenzfälle

Nutzen

- Jede Aktivität überprüft die gewählte Architektur in Hinsicht auf eine bestimmte Frage.
- Nach Beendigung dieser Aktivitäten können die Schnittstellen der Subsysteme **abschließend** definiert werden.
 - Start frei für den Objektentwurf der Subsysteme!

Rückblick ▶ Systementwurf (Gesamt)

