

Komposition

Echtzeitsysteme 2 - Vorlesung/Übung

Fabian Scheler
Michael Stilkerich
Peter Ulbrich
Wolfgang Schröder-Preikschat

Lehrstuhl für Informatik IV
Verteilte Systeme und Betriebssysteme
Friedrich-Alexander Universität Erlangen-Nürnberg

<http://www4.cs.fau.de/~{scheler,mike,wosch}>
{scheler,mike,ulbrich,wosch}@cs.fau.de



Übersicht

- Grundlagen
- Komposition: traditionell
- Komposition: Forschung



Grundlagen

■ Teil 1: **Abbildung**

Ereignisbehandlungen → BS-Dienste

- Verknüpfung von Ereignissen mit Ereignisbehandlungen
- Abbildung von Abhängigkeiten verschiedener Ereignisbehandlungen
- Synchronisation kritischer Abschnitte
- Modellierung von Datenabhängigkeiten
- ...

■ Teil 2: **Ablaufplanung**

- Berechnung statischer Ablaufpläne
- Auswahl eines geeigneten Ablaufplaners (RMA, DMA, EDF, ...)
- Planbarkeitsanalyse: Antwortzeitanalyse, CPU-Auslastung, ...
- ...



Komposition: Eingaben

■ Ereignisbehandlungen

- welche Ereignisse: periodisch, aperiodisch, sporadisch
- welche Deadlines: hart, fest, weich
- Abhängigkeiten (Datenfluß, kritische Abschnitte, ...)

■ Komponenten

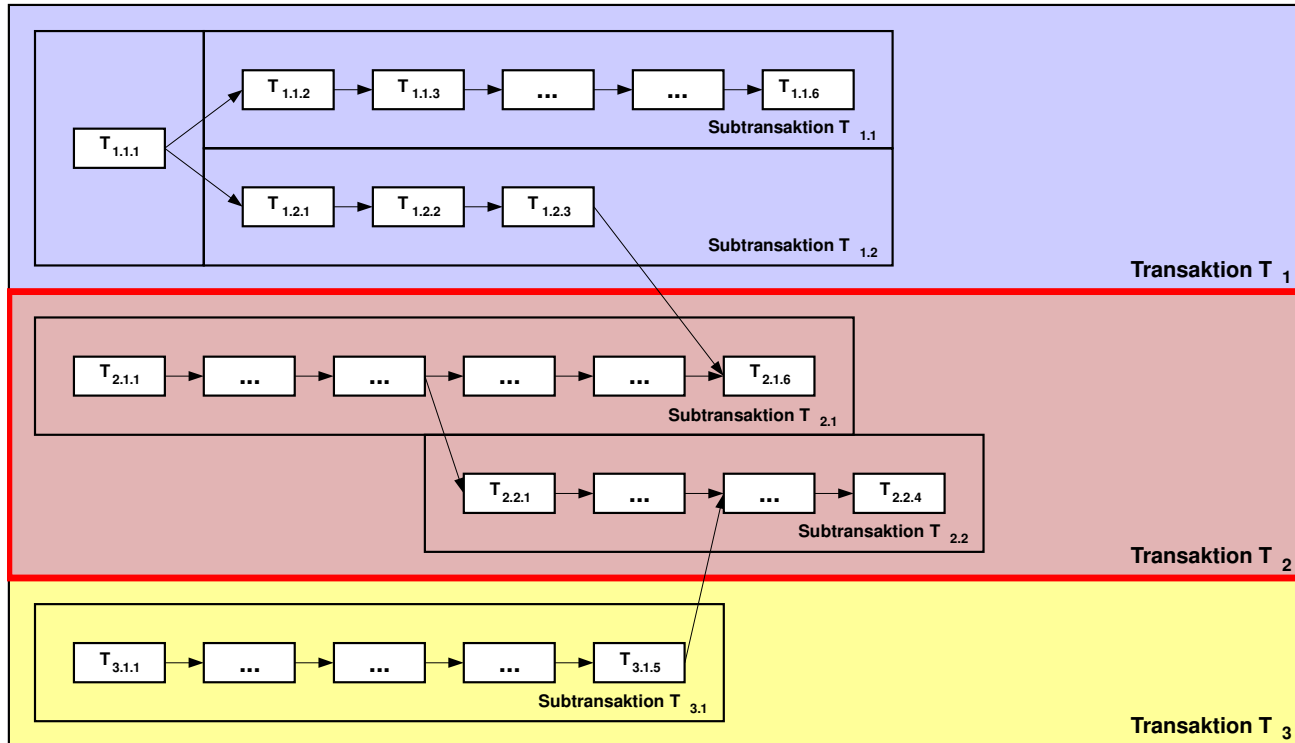
- funktionale und temporale Spezifikation

■ Laufzeitsystem

- funktionale und temporale Spezifikation



Ereignisbehandlungen

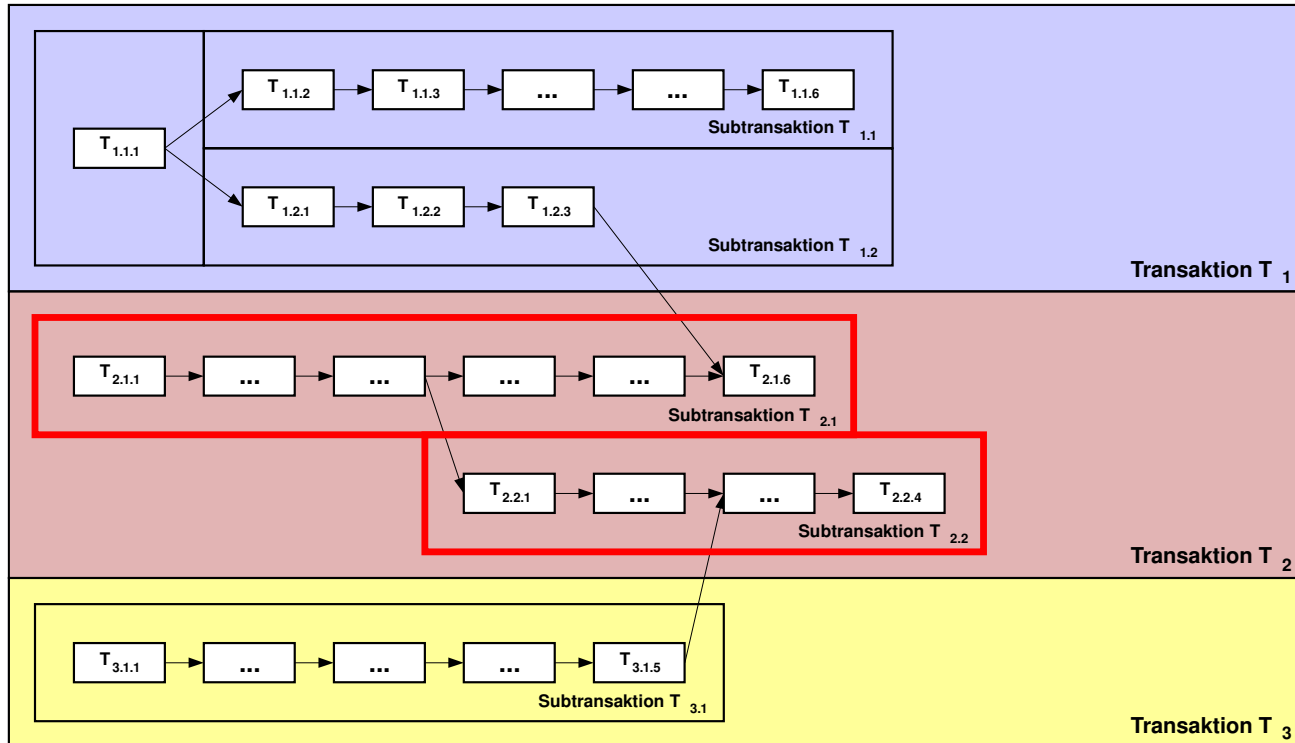


■ Transaktionen

Reaktionen auf Ereignisse aller Art sind **Ereignisbehandlungen** bzw. **Transaktionen**. Transaktionen und ihre **auslösenden Ereignisse (Auslöser)** sind durch eine **Periode** (periodische Ereignisse) oder **minimale Zwischenankunftszeit** charakterisiert (nicht-periodische Ereignisse).



Ereignisbehandlungen

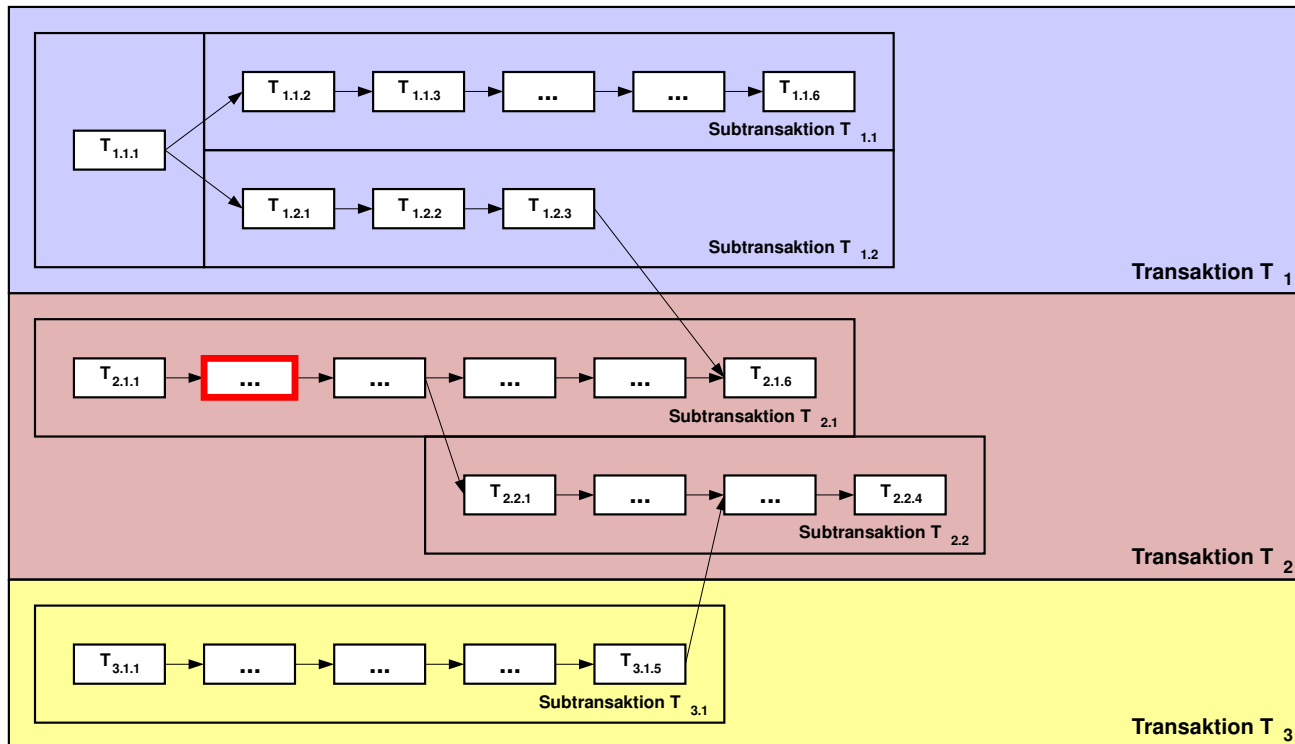


■ Subtransaktionen

Transaktionen zerfallen in **Subtransaktionen**, d.h. Kontrollflüsse, die ein Ereignis auf verschiedene Weisen behandeln. Subtransaktionen können mit einer **harten/festen/weichen Deadline** versehen sein.



Ereignisbehandlungen

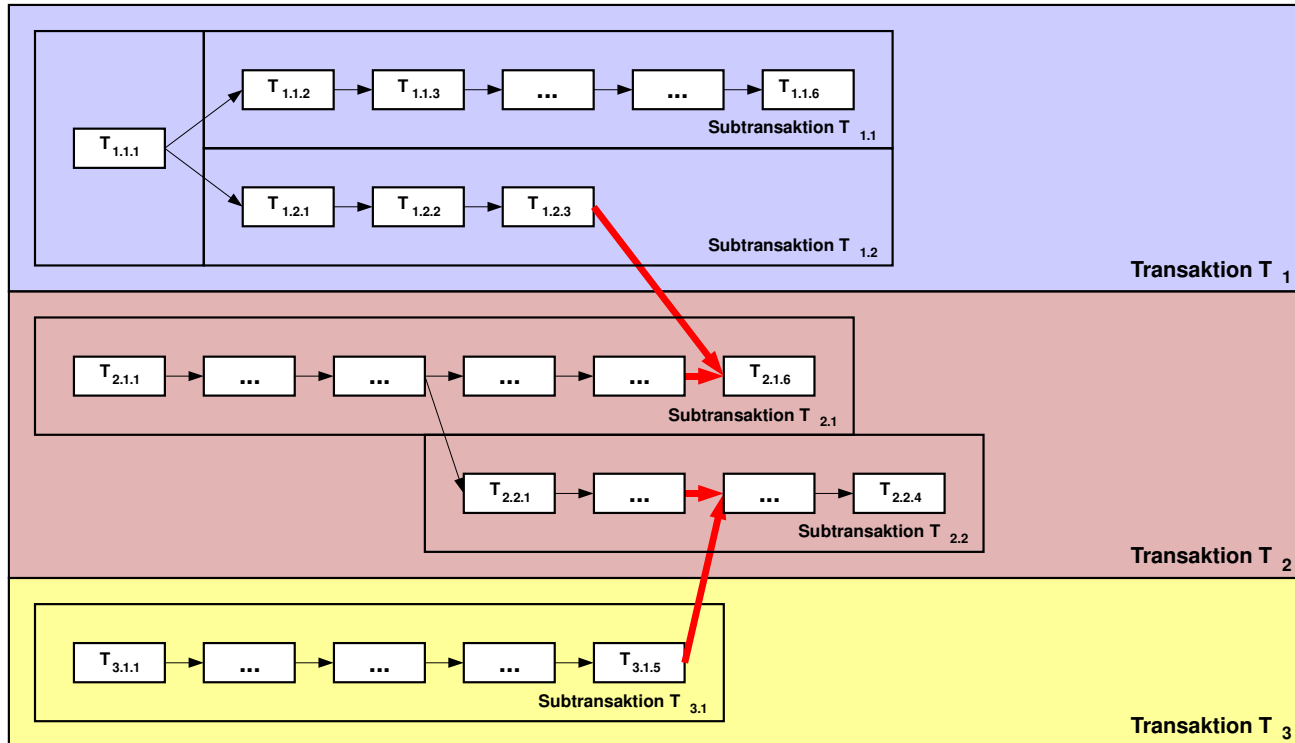


■ Sektionen

Subtransaktionen bestehen aus einer oder mehreren **Sektionen**. Innerhalb einer Sektion verändert sich die Verdrängbarkeit (also z.B. die Priorität) nicht. Endpunkte von Sektionen bilden z.B. die Freigabe eines Mutex, die Aktivierung eines Threads oder das Warten auf ein Event. Jeder Sektion ist eine **maximale Ausführungszeit (WCET)** zugeordnet, zusätzlich können auch sie mit **harten/festen/weichen Deadlines** versehen sein.



Abhängigkeiten



■ Oder-Abhängigkeiten

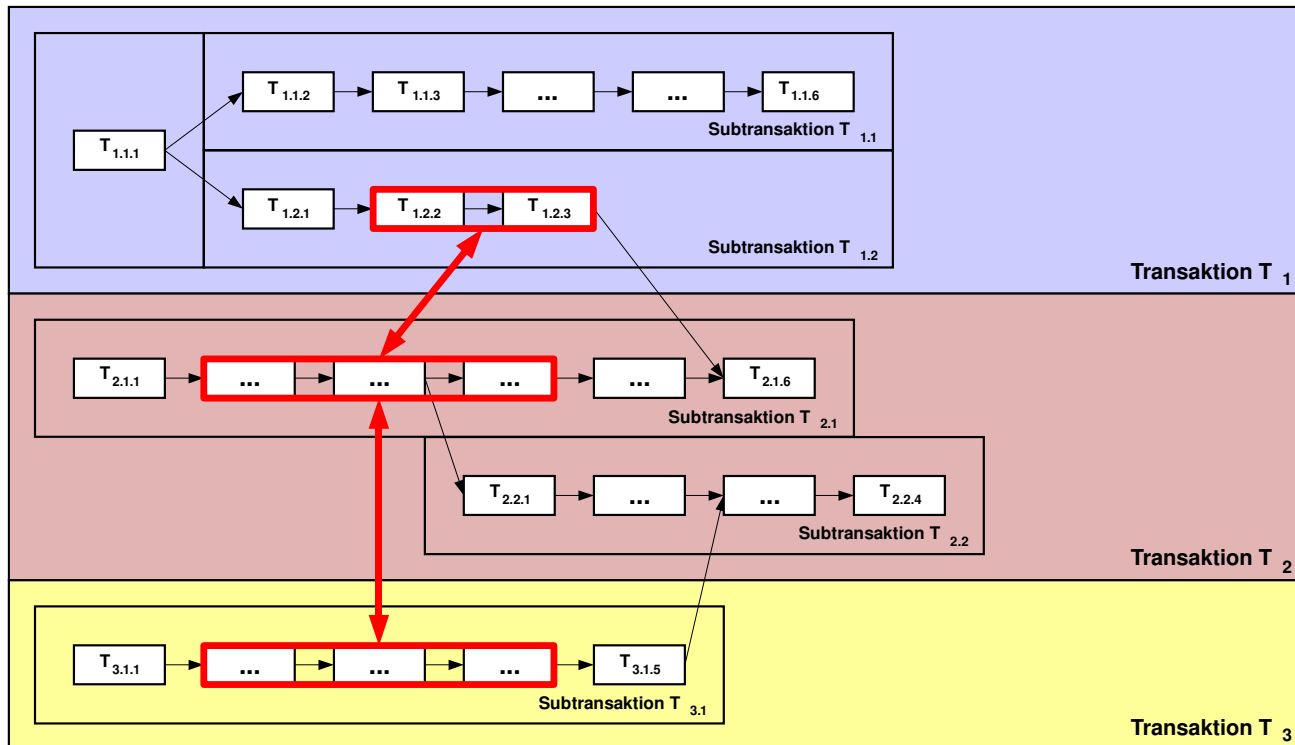
Die Ausführung des Nachfolgers ist möglich sobald die Ausführung **eines** Vorgänger abgeschlossen ist.

■ Und-Abhängigkeiten

Die Ausführung des Nachfolgers ist möglich sobald die Ausführung **aller** Vorgänger abgeschlossen ist.



Gegenseitiger Ausschluss



■ Gegenseitiger Ausschluss

Werden Datenstrukturen nebenläufig manipuliert, kann dies zu Inkonsistenzen und somit zu undefiniertem Verhalten führen. Man bezeichnet daher die Abschnitte, in denen diese Manipulationen durchgeführt werden, als **kritische Abschnitte**. Zwischen kritischen Abschnitten muss i.d.R. der **gegenseitige Ausschluss** gewährleistet werden, falls die Datenstrukturen nicht anderweitig synchronisiert werden können.



Abbildung

- das Laufzeitsystem muss die Implementierung folgender Konstrukte innerhalb von Transaktionen unterstützen:
 - Oder-Abhängigkeiten
 - Und-Abhängigkeiten
 - gegenseitiger Ausschluss
 - Aufteilen von Transaktionen in Subtransaktionen
 - Konkatenation von Sektionen zu Subtransaktionen



Statische Ablaufplanung

- **Constraint Programming** (*Schild & Würz*)

Das Scheduling-Problem wird als Problem der mathematischen, linearen Ganzzahlprogrammierung aufgefasst, das es unter Einhaltung diverser Nebenbedingungen zu lösen gilt.

- **Cyclic Executive Model** (Shaw & Baker)

Der Ablaufplan besteht aus Major- und Minor-Cycles, Konstruktion des Ablaufplans durch Festlegung/Berechnung gewisser Eigenschaften von Major- und Minor-Cycles.

- **Suchen** (*Fohler*)

Das Absuchen des kompletten Suchraums aller möglichen Ablaufpläne ist wegen der NP-Komplexität des Scheduling-Problems nicht möglich, daher wird auf heuristische Algorithmen wie A* und IDA* zurückgegriffen.

- **genetische Algorithmen** (*Hou*)

Verfahren für Scheduling in Multiprozessorsystemen. Die Suchknoten basieren auf der Reihenfolge der Tasks, die auf einem Knoten abgearbeitet werden, der genetische Operator auf den Abhängigkeiten zwischen den verschiedenen Tasks.

- **von Hand**



Planbarkeitsanalyse

■ Bestimmung der **maximalen CPU-Auslastung**

- Periode p_i , Deadline d_i und WCET e_i aller n Tasks t_i sind bekannt
- CPU Auslastung:
$$U = \sum_{i=1}^n \frac{e_i}{\min(p_i, d_i)}$$
- EDF:
$$U \leq 1$$
- RMA & DMA:
$$U(n) = n(2^{1/n} - 1); d_i = p_i \forall 1 \leq i \leq n$$

■ **Antwortzeitanalyse**

- Periode p_i , Deadline d_i und WCET e_i aller n Tasks t_i sind bekannt
- $\text{Priorität}(t_i) > \text{Priorität}(t_{i+1})$
- Zeitbedarf:
$$w_i(t) = e_i + \sum_{k=1}^{i-1} \text{ceiling}\left(\frac{t}{p_k}\right) e_k$$
- prüfe:
$$w_i(t) \leq t; t = jp_k; k = 1, 2, \dots, i; j = 1, 2, \dots, \text{floor}(\min(d_i, p_i)/p_k)$$



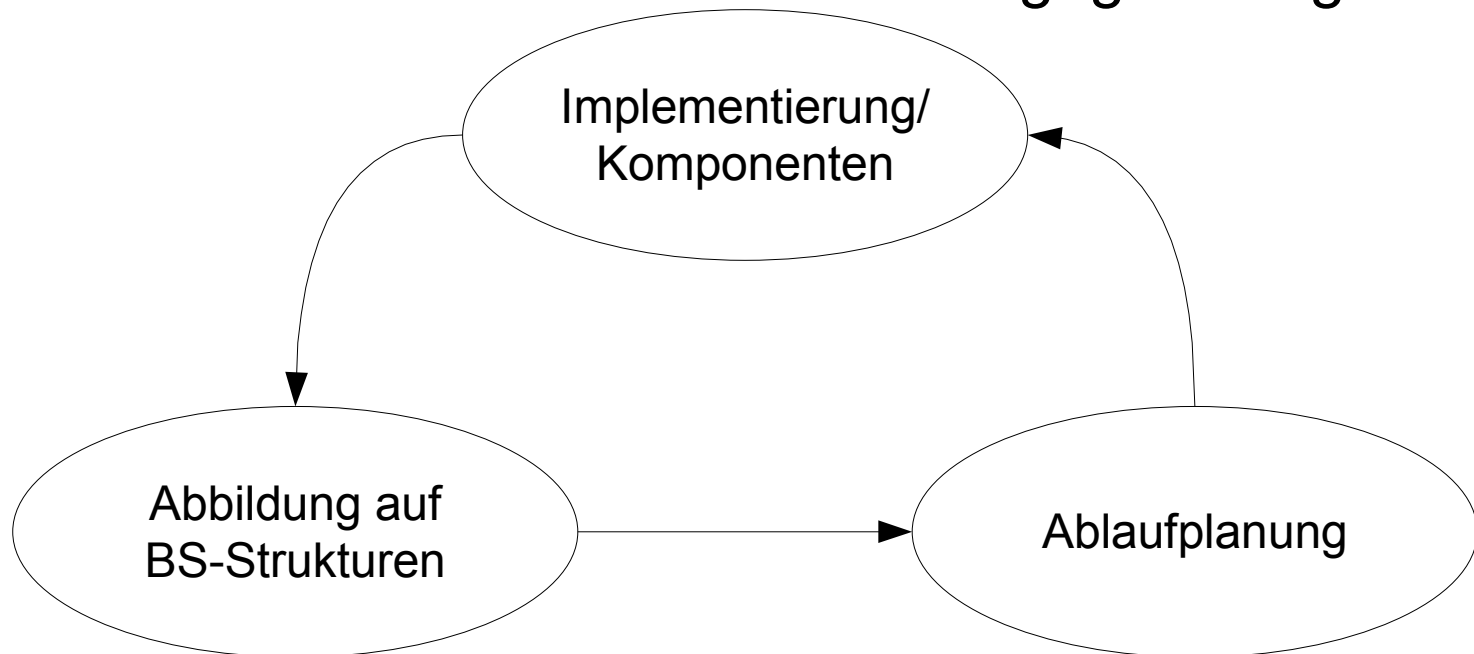
Komposition: traditionell

- oft erfolgen einige Schritte simultan:
 - Entwicklung der Komponenten
 - Abbildung auf BS-Strukturen
 - Berechnung statischer Ablaufpläne / Planbarkeitsanalyse
- das ist **problematisch**



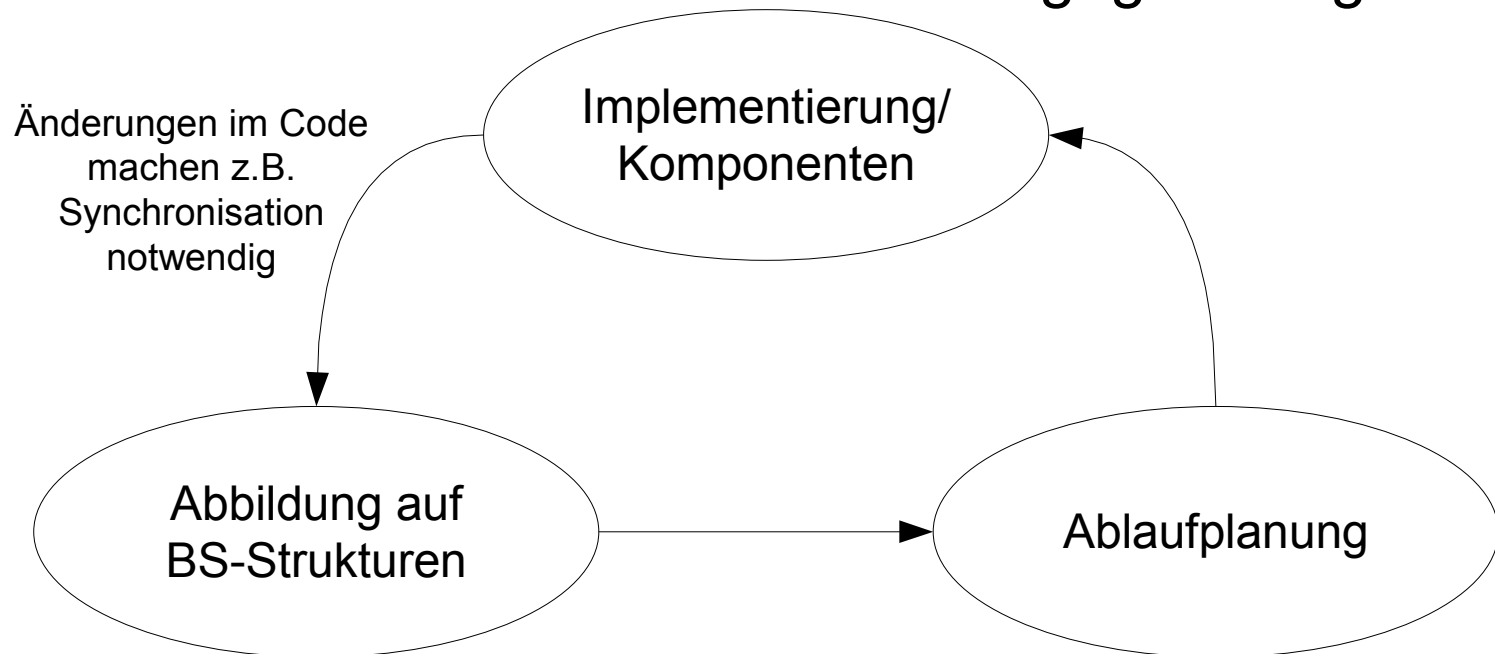
Probleme & Folgen

- Problem: Schritte beeinflussen sich gegenseitig



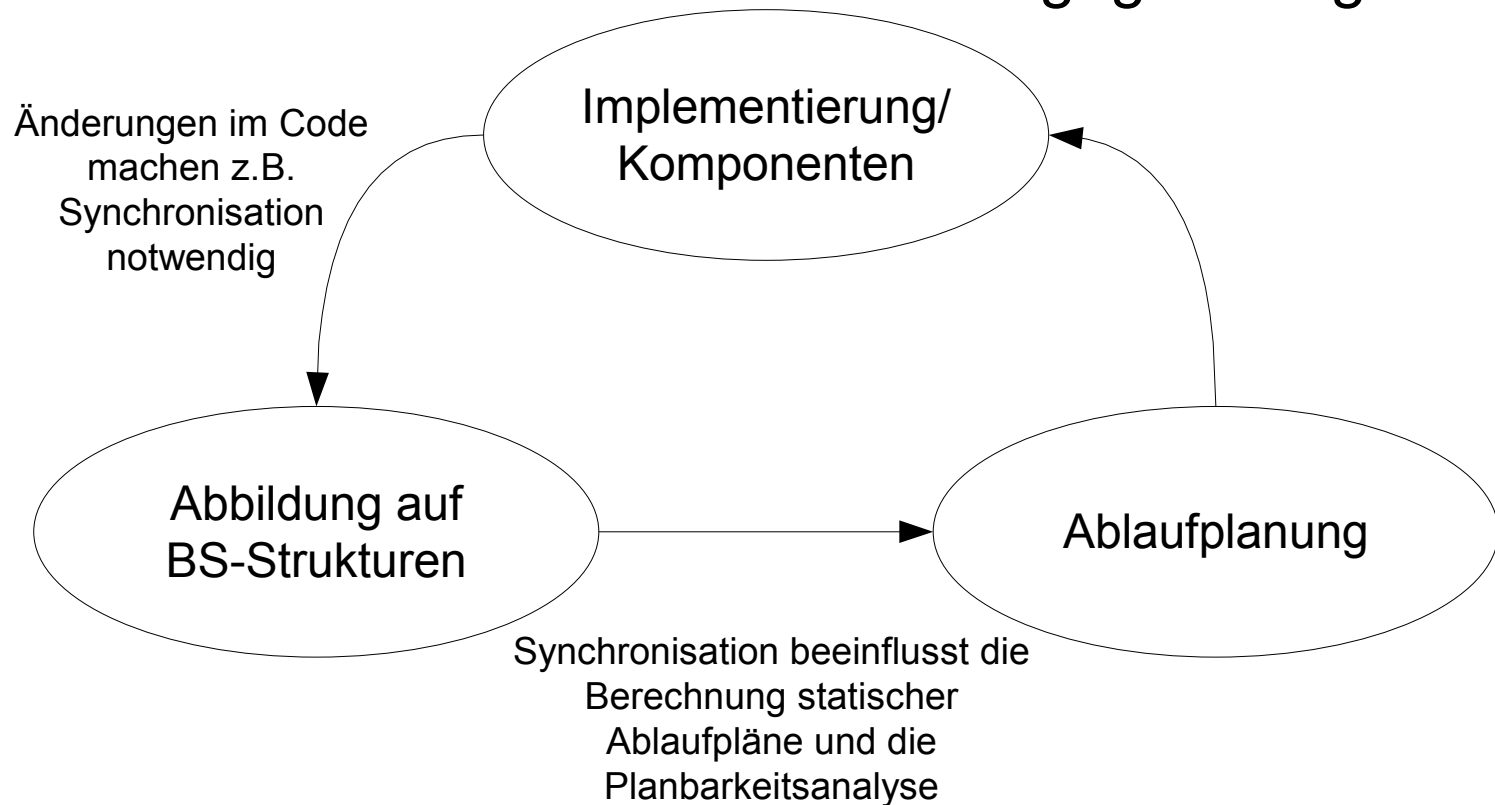
Probleme & Folgen

- Problem: Schritte beeinflussen sich gegenseitig



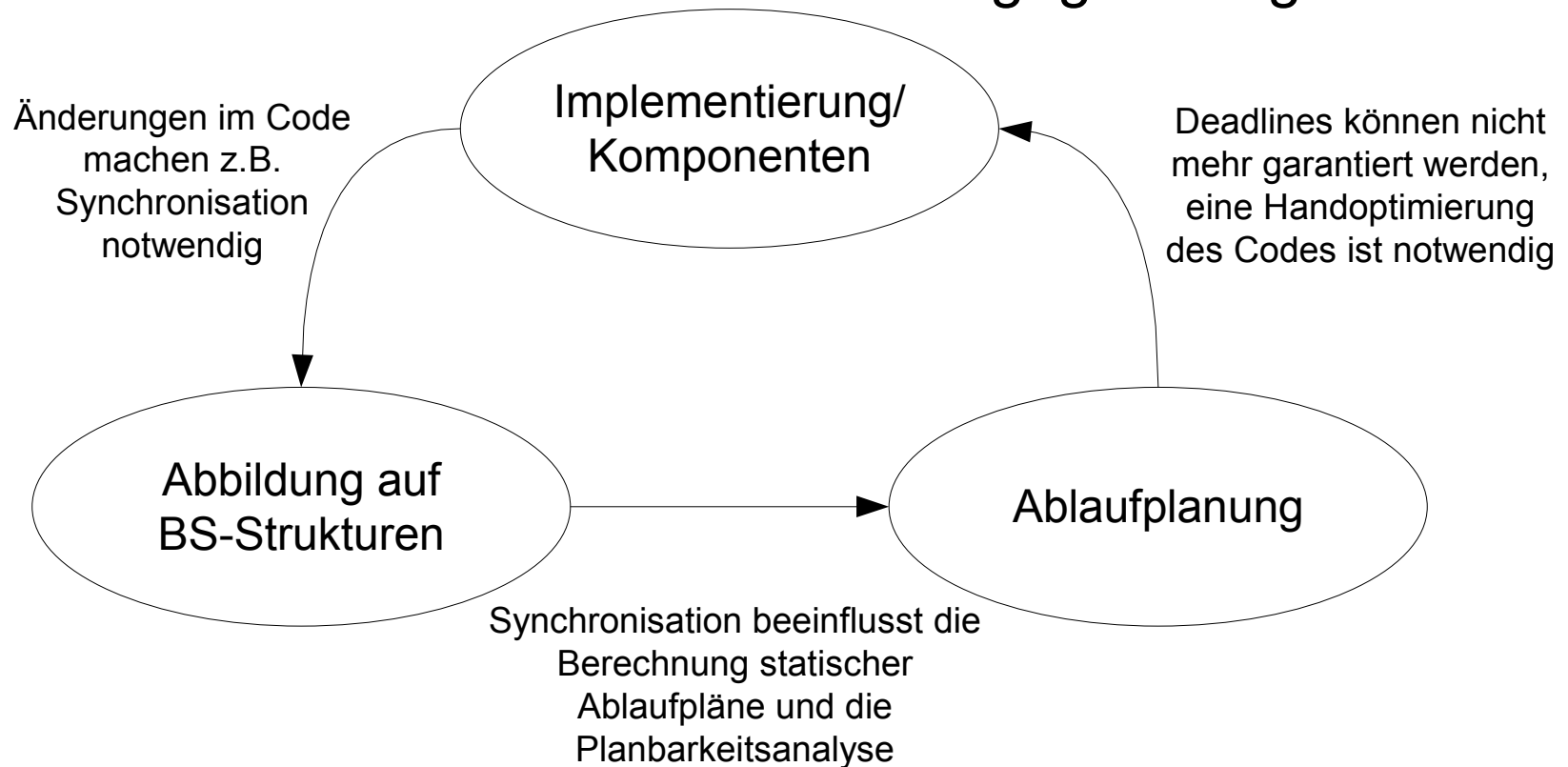
Probleme & Folgen

- Problem: Schritte beeinflussen sich gegenseitig



Probleme & Folgen

- Problem: Schritte beeinflussen sich gegenseitig



- Folge
 - langwieriger Entwicklungsprozess
 - schlecht wartbarer Code



Probleme & Folgen

- **Problem:** Abbildung auf BS-Dienste erfolgt zu früh
Modularisierung findet auf der Ebene von BS-Strukturen statt, z.B. LED-Task, Lichtschranken-ISR ...

- **Folgen:**
 - System ist **nicht wiederverwendbar**
 - bestimmte BS-Dienste existieren nicht unbedingt in allen BS
 - Portierungsaufwand
 - Architekturunterschiede: TT vs. ET
 - **Overhead**
 - Einsatz von BS-Diensten ist **immer teurer!**
 - Komponenten / Module auf der Ebene von BS-Diensten ziehen ausgiebigen Gebrauch von BS-Diensten nach sich!



What you need ...

*„for operating-system technology, the cat is out of the bag. Fundamentally, **every RTOS is the same as every RTOS. What you need is a way to divide your problem into tasks** and have sufficient computing power. Then you want to have a priority-based pre-emptive kernel. And they're all the same, whether you get your OS out of a book or with free source code included, or you get something else free. ...“ [1]*

Michael Barr
(former editor-in-chief Embedded Systems Design)



Komposition: Forschung

■ Ziele:

- Entkopplung der einzelnen Schritte
- Automatisierung der einzelnen Schritte
- Einhaltung gewisser Randbedingungen:
 - Rechtzeitigkeit
 - Security / Safety
 - Zuverlässigkeit
 - ...



Forschung: Aufteilung

■ Grundlagenforschung

- stellt Mechanismen zur Verfügung,
 - um Ereignisbehandlungen zu implementieren
 - die es erlauben, Aussagen über gewisse Eigenschaften zu treffen
- Forschungsgebiete
 - Scheduling-Theorie
 - WCET-Analyse
 - Echtzeitbetriebssysteme
 - ...

■ Werkzeugunterstützung

- basiert auf den Ergebnissen der Grundlagenforschung
- (halb)automatische Anwendung der entwickelten Mechanismen



Forschung: Beispiele

- Grundlagenforschung:
 - Time-Triggered Architecture (TTA)
- Werkzeugunterstützung
 - Cluster Compiler
 - SysWeaver



TTA & Cluster Compiler

- Entwicklungsziel:
sicherheitskritische, verteilte Echtzeitsysteme
- TTA: Grundlage/Architektur
- Cluster Compiler: Entwicklungswerkzeug
- Entwickler: TU Wien – Kopetz et. al
- Einsatzgebiete
 - Luftfahrt
 - Automobilbau
 - Industrieanlagen
 - Antriebssysteme
 - Beispiel: Kabinendrucksteuerung des A380



TTA: Architekturkonzepte

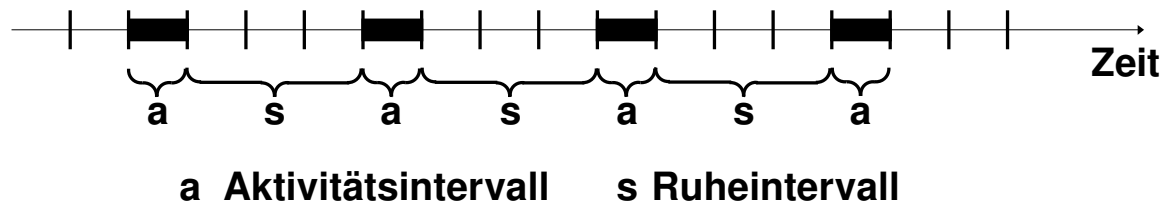
- Modell der Zeit
- Zeit & Zustand
- Echtzeitentitäten und Echtzeitabbilder
- Zustandsinformation & Ereignisinformation
- Struktur
 - einzelner Knoten
 - TTA-Cluster
- Topologien



TTA: Architekturkonzepte

■ Zeit

- basiert auf physikalischer Zeit
 - Zeitpunkte auf der Zeitachse sind Ereignisse
→ Beobachtung eines Zustands ist ein Ereignis
 - Ereignisse tragen Zeitstempel
 - global synchronisierte Zeitbasis
 - perfekte Synchronisation nicht möglich
→ oft keine Aussagen über die Folge von Ereignissen möglich
 - Lösung: ***sparse time base***
 - Ruhe- und Aktivitätsintervalle
 - Ruheintervalle werden durch TTA überwacht
- Ereignisse lassen sich zeitlich nach ihren Zeitstempeln ordnen



TTA: Architekturkonzepte

■ Zeit & Zustand

- der Zustand entkoppelt Zukunft und Vergangenheit
- Zukunft hängt nur von Zustand und zukünftigen Eingaben ab
- ***sparse time base*** erlaubt globale Trennung zwischen
 - Zukunft und
 - Vergangenheit

■ Echtzeitentität und Echtzeitabbild

- EZ-Entität: relevante Variablen, die den Zustand kapseln
- EZ-Abbild: zeitlich akkurates Abbild einer Echtzeitentität

■ Zustands- und Ereignisinformation

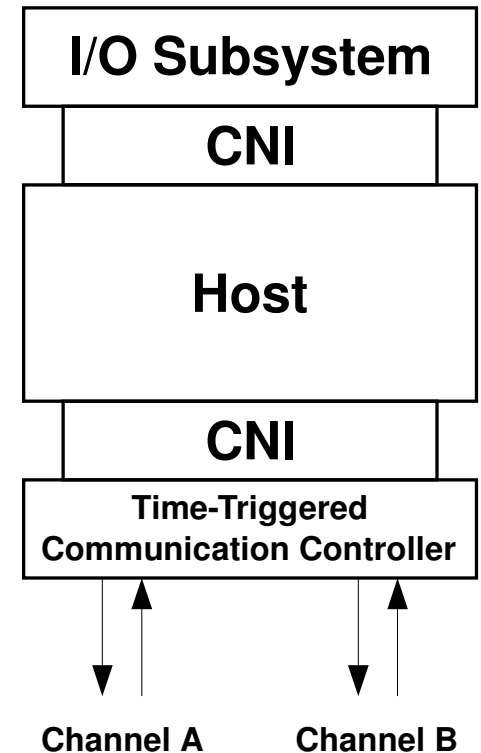
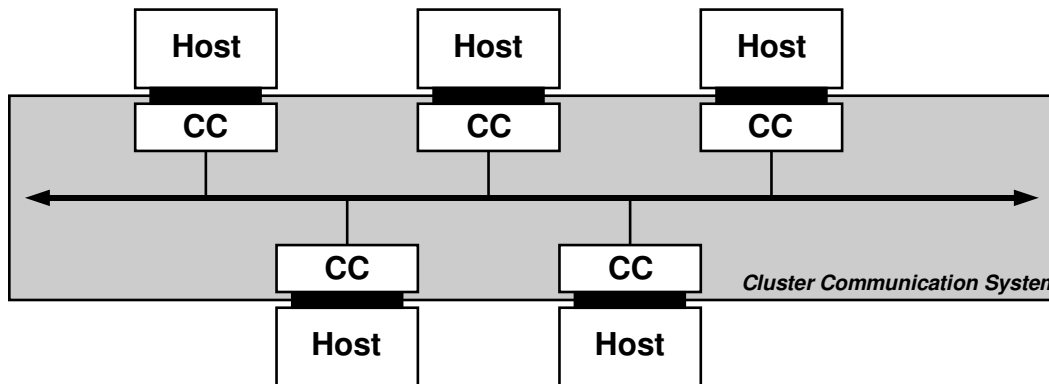
- Zustandsinformation: EZ-Abbild mit Zeitstempel
- Ereignisinformation: Änderung des Zustands
- TTA verwendet ausschließlich Zustandsinformation



TTA: Architekturkonzepte

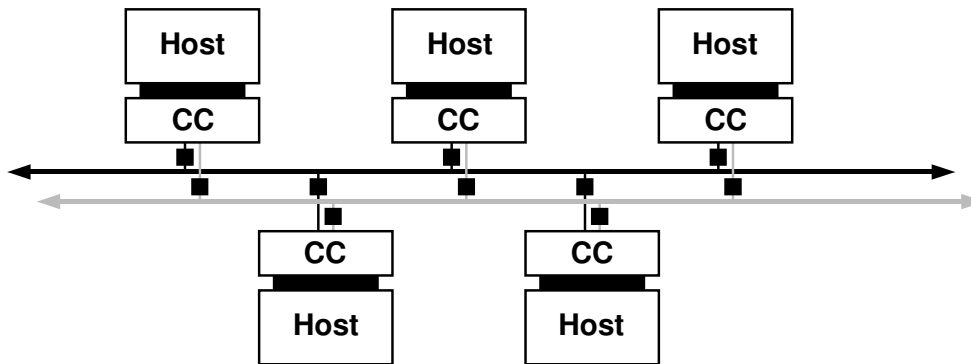
■ **Struktur:** ein TTA-System besteht aus

- Knoten
 - Prozessor/Speicher
 - Communication Network Interface
 - zeitgesteuerter Kommunikationscontroller
 - I/O Subsystem
 - BS + Anwendung
- die zu einem TTA-Cluster gekoppelt sind
 - Cluster Kommunikationssystem – TTP
 - replizierter Broadcast-Kanal (zuverl. Broadcast)

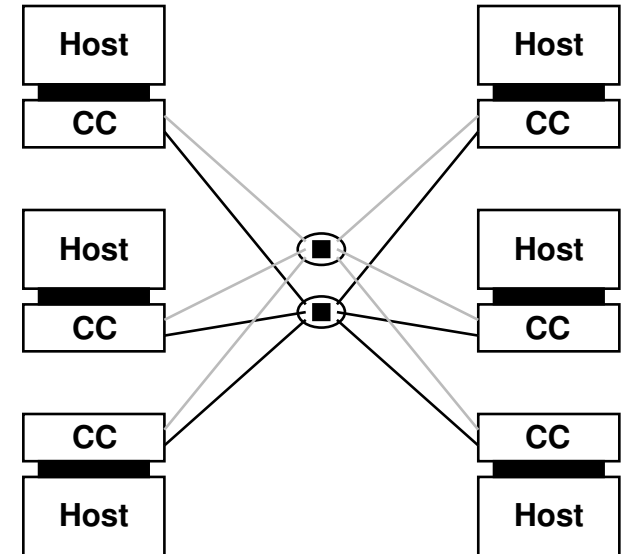


TTA: Architekturkonzepte

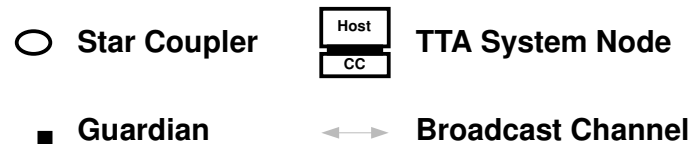
■ Topologie



TTA-Bus



TTA-Stern



TTA: Designkonzepte

- zuverlässige, verteilte Rechnerplattform
- Temporal Firewalls
- Zusammensetzbarkeit
- Skalierbarkeit
- transparente Fehlertoleranz



TTA: Designkonzepte

■ zuverlässige, verteilte Rechnerplattform

- kurze Fehlererkennungslatenz durch zeitgesteuerte Kommunikation
 - Fehlererkennung auf Protokollebene
 - Membership-Service
 - korrektes Verhalten oder Stille (***fail-silent nodes***)
- Mitglieder verhalten sich korrekt

■ Temporal Firewalls

- CNI ist die wichtigste Schnittstelle innerhalb eines TTA-Systems
 - zwei unidirektionale Kanäle (Senden & Empfangen)
 - elementare Schnittstelle: unidirektionaler Kontrollfluß (Sender übergibt Daten an das CNI, Empfänger holt Daten vom CNI ab)
 - Kontrollfehlerausbreitung aufgrund des Designs unmöglich
- Schnittstelle mit diesen Eigenschaften: ***Temporal Firewall***



TTA: Designkonzepte

■ Zusammensetzbarkeit

- unabhängige Entwicklung der einzelnen Knoten
Dienste, die ein Knoten zur Verfügung stellt müssen funktional und zeitlich auf Architekturebene spezifiziert werden können.
- Stabilität existierender Dienste
Die Integrität bestehender Dienste wird nicht durch die Integration neuer Knoten kompromittiert.
- konstruktive Integration neuer Knoten
Die Integrität existierender Knoten, darf nicht durch die Integration neuer Knoten beeinträchtigt werden.
- Replikdeterminismus
Replizierte Knoten müssen zur selben Zeit denselben von außen sichtbaren Zustand einnehmen und dieselben Nachrichten versenden.



TTA: Designkonzepte

■ Skalierbarkeit

- Komplexität einzelner Funktionen (Knoten) darf nicht mit der Komplexität des Gesamtsystems wachsen
- horizontale & vertikale Schichten (Abstraktion & Partitionierung)
- CNIs bieten exakt diese Abstraktionen

■ transparente Fehlertoleranz

- in EZ-Systemen basierend auf aktiver Redundanz & Mehrheitsentscheid
 - Replizierung
 - Mehrheitsentscheid
 - (Re)integration ausgefallener Knoten
- in der Anwendung: hohe Komplexität nur durch Fehlertoleranz
- separate Fehlertoleranzschicht in TTA

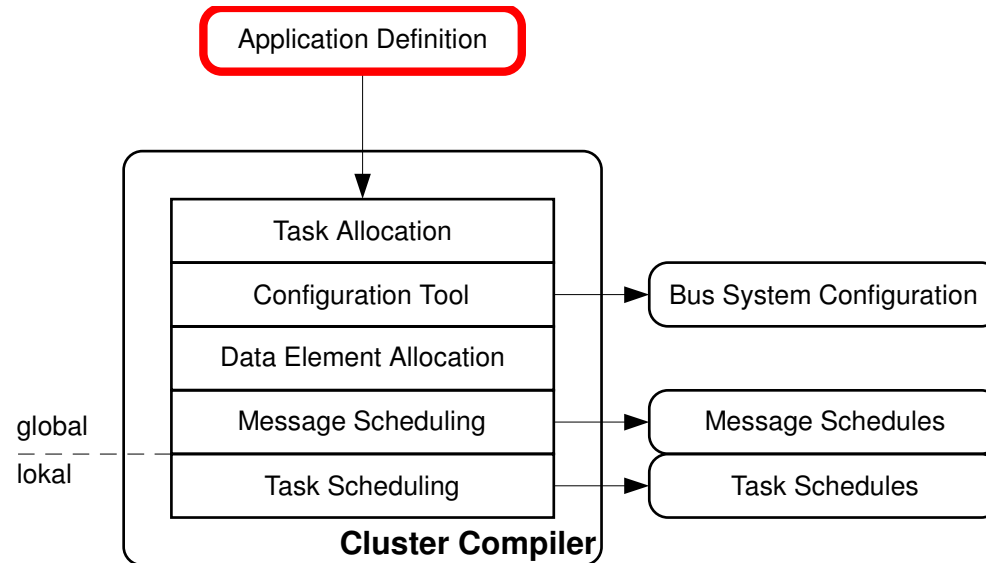


Cluster Compiler

- Entwicklungswerkzeug für zeitgesteuerte, verteilte Echtzeitsysteme
- Offline-Scheduling für
 - Tasks und
 - Nachrichten
- verwendet Heuristiken
 - A*-, IDA*-Search
 - Tabu-Search
 - genetische Algorithmen
 - Simulated Annealing
- Grundprinzip: strikte Trennung von
 - globalen Aspekten und
 - lokalen Aspekten eines verteilten Echtzeitsystems



Cluster Compiler: Struktur

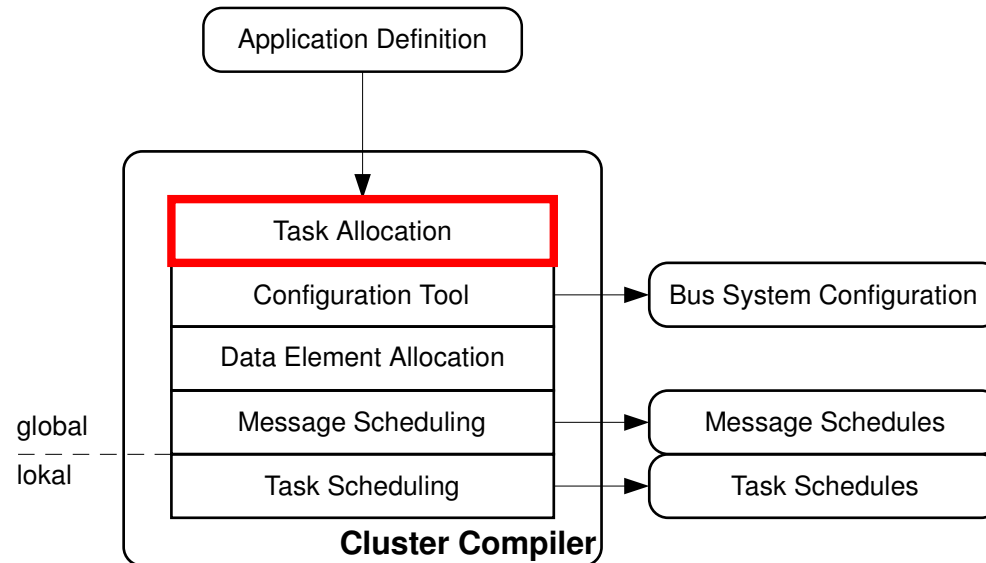


■ Eingabe

- Periode / WCET für alle Tasks
- Daten Elemente
 - Gültigkeitsintervalle, sender/emfänger Task
- Abhängigkeitsgraph
 - gegenseitiger Ausschluss, Datenabhängigkeiten, ...



Cluster Compiler: Struktur



■ Zuordnung: Tasks - Knoten

■ Optimierung

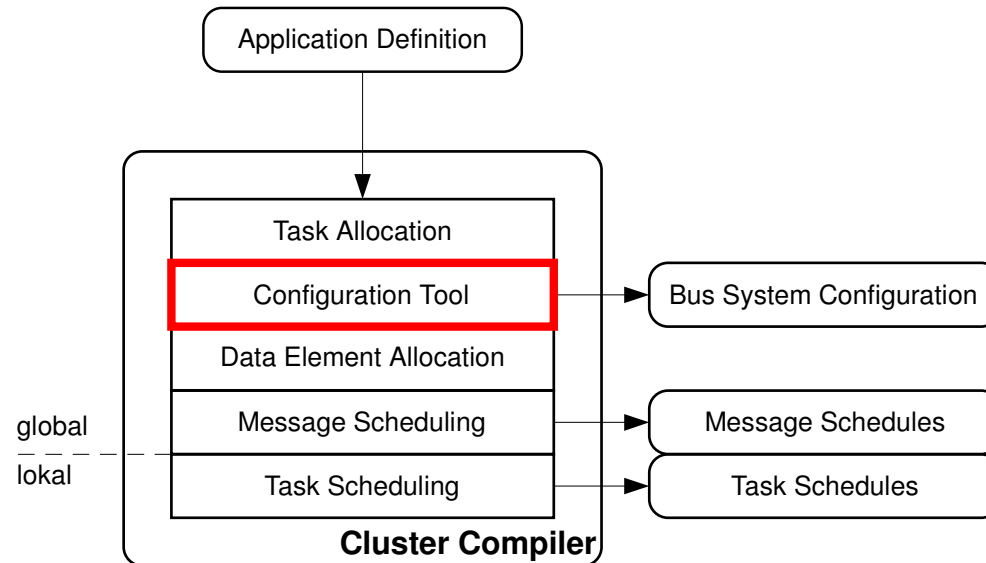
- Lastausgleich
- Minimierung der Kommunikation zwischen Knoten

■ Einschränkungen

- feste Zuordnung von Tasks zu bestimmten Knoten (z.B. wenn auf Hardware zugegriffen werden muss)



Cluster Compiler: Struktur



■ Konfiguration des Bussystems

■ Optimierung

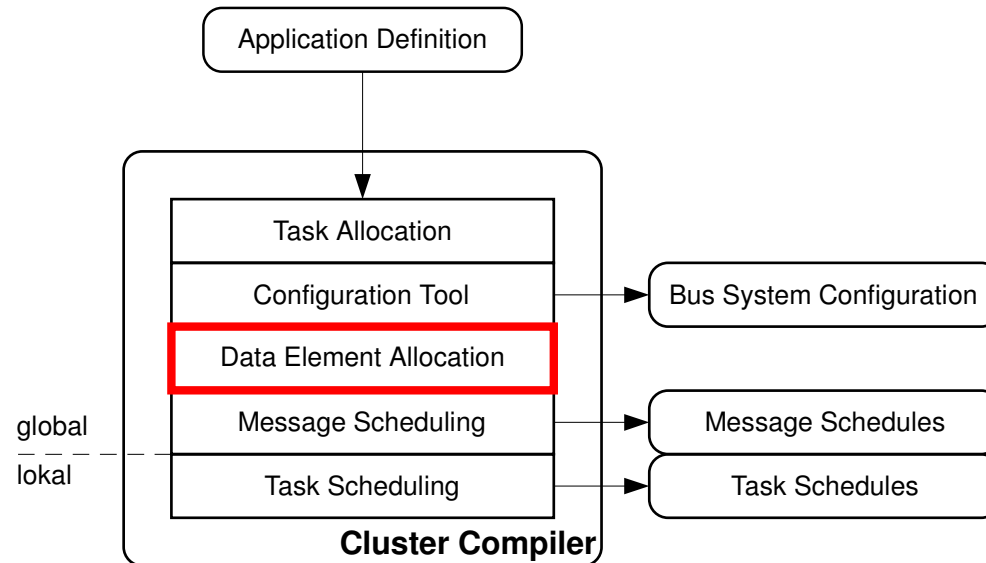
- Übertragungslatenz
- Anzahl Busse/Gateways

■ Einschränkungen

- *Locality Constraints*: vom Entwickler vorgegebene Zuordnung von Knoten zu bestimmten Bussen



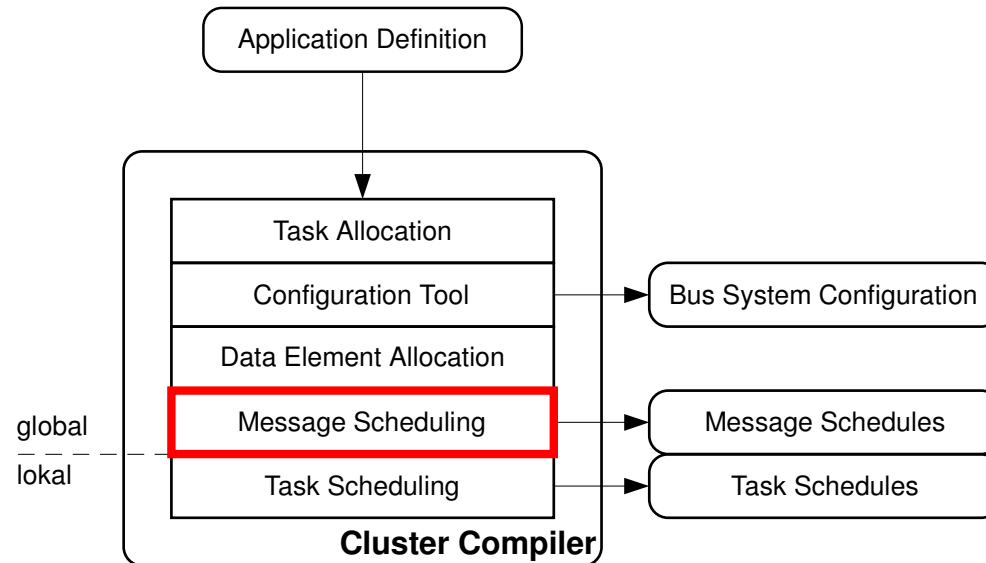
Cluster Compiler: Struktur



- Kombination von Datenelementen zu TTP-Nachrichten
 - Berechnung der Periode und der Länge der Nachrichten
 - Einschränkungen
 - gleiche Reihenfolge von Sendern/Empfänger pro TDMA-Runde
 - Bandbreite des Mediums
 - Gültigkeitsintervalle der Datenelemente



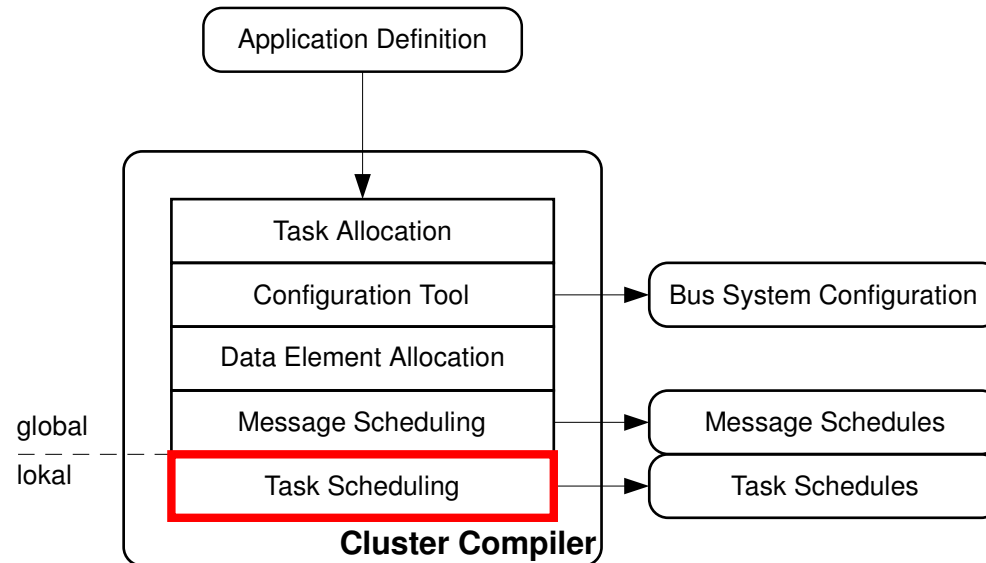
Cluster Compiler: Struktur



- Berechnung Sende- und Empfangstabellen
 - fasse Nachrichten zu Cluster-Zyklen zusammen
 - erzeuge Kontrollinformation zur Protokollausführung
 - Optimierung
 - Übertragungslatenz
 - benötigte Bandbreite / Auslastung der Bandbreite



Cluster Compiler: Struktur



- Berechnung der Ablauftabellen für lokale Knoten
 - Einschränkung
 - Sende- und Empfangstabellen der Nachrichten
 - Gültigkeitsintervalle der Datenelemente



SysWeaver – Motivation

- Software ist sehr flexibel, das ist ihre große
 - Stärke und
 - Verpflichtung
- herkömmliche Programmiersprachen
 - Korrektheit der Syntax überprüft der Compiler
 - Korrektheit des Verhalten schwer zu prüfen
 - durch Änderungen (Flexibilität) kann Software *beschädigt* werden
- kaum Möglichkeiten zur Formulierung von NFPs wie
 - Rechtzeitigkeit
 - Safety/Security
 - Fehlertoleranz



SysWeaver – Ziel

- SysWeaver
 - Werkzeug für modellbasierte Softwareentwicklung
 - schlägt die Brücke vom Modell zum ausführbaren Programm
- Entwurfsziele
 - Zusammensetzbarkeit
 - Wiederverwendbarkeit
 - Skalierbarkeit
 - Korrektheit durch Konstruktion
 - **mehrdimensionale** Modellierung
 - Verwendung von Mustern
 - Kommunikation, Berechnung, Nebenläufigkeit
 - Plattformunabhängigkeit

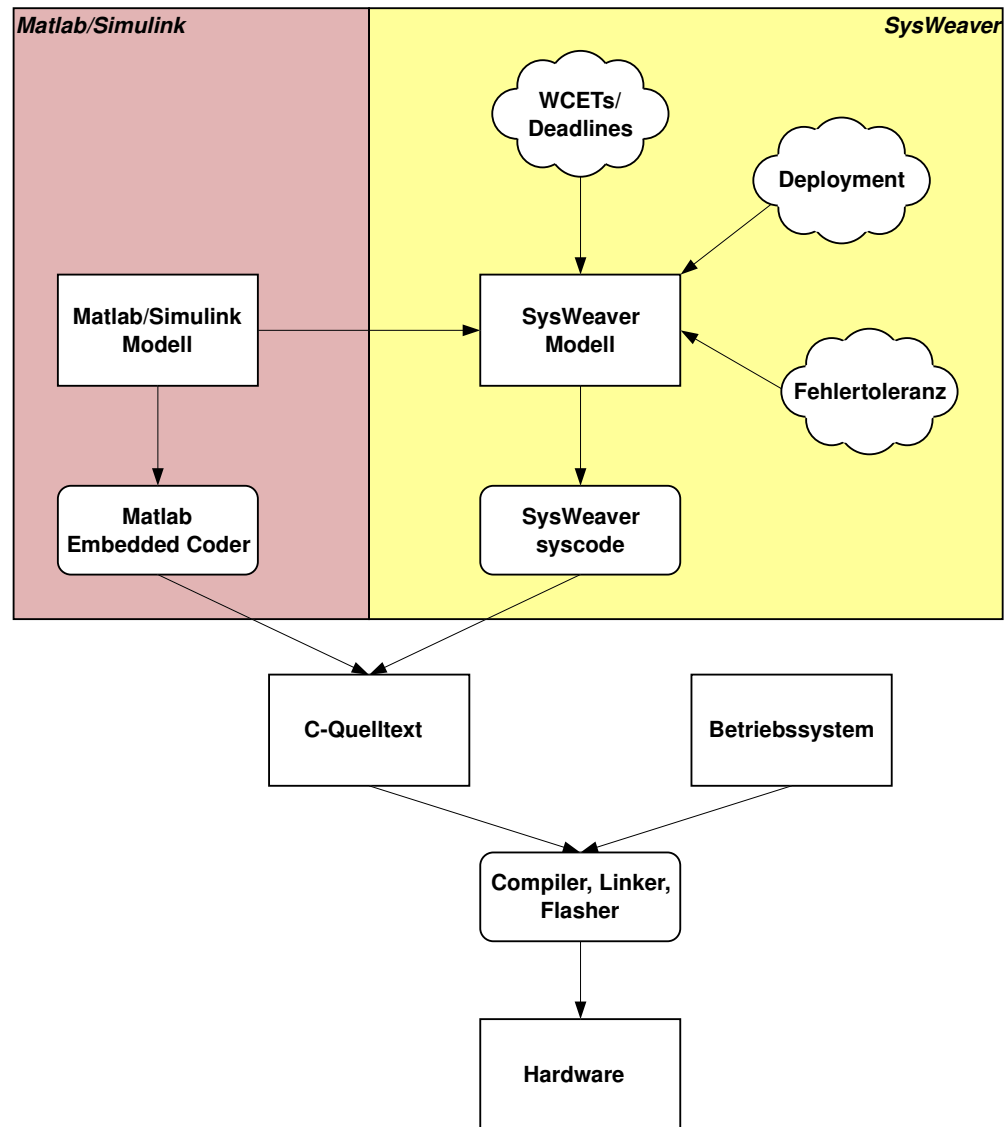


SysWeaver – Grundkonzepte

- modellbasierte Entwicklung
 - vom Modell zum ausführbaren Programm
 - unabhängig von Programmiersprache und Betriebssystem
- mehrdimensionale Modellierung
 - Spezialisten für Echtzeit, Fehlertoleranz, Netzwerke, ...
 - verschiedene Modellansichten
 - Ansichten fungieren als *Filter*
- Modellierung nicht-funktionaler Eigenschaften
 - Modellierung funktionaler Eigenschaften: gängig (z.B. Simulink)
 - Modellierung nicht-funktionaler-Eigenschaften
 - Speziallösungen, modellieren genau eine nicht-funktionale Eigenschaft
 - Modellierung **verschiedener** nicht-funktionaler Eigenschaften



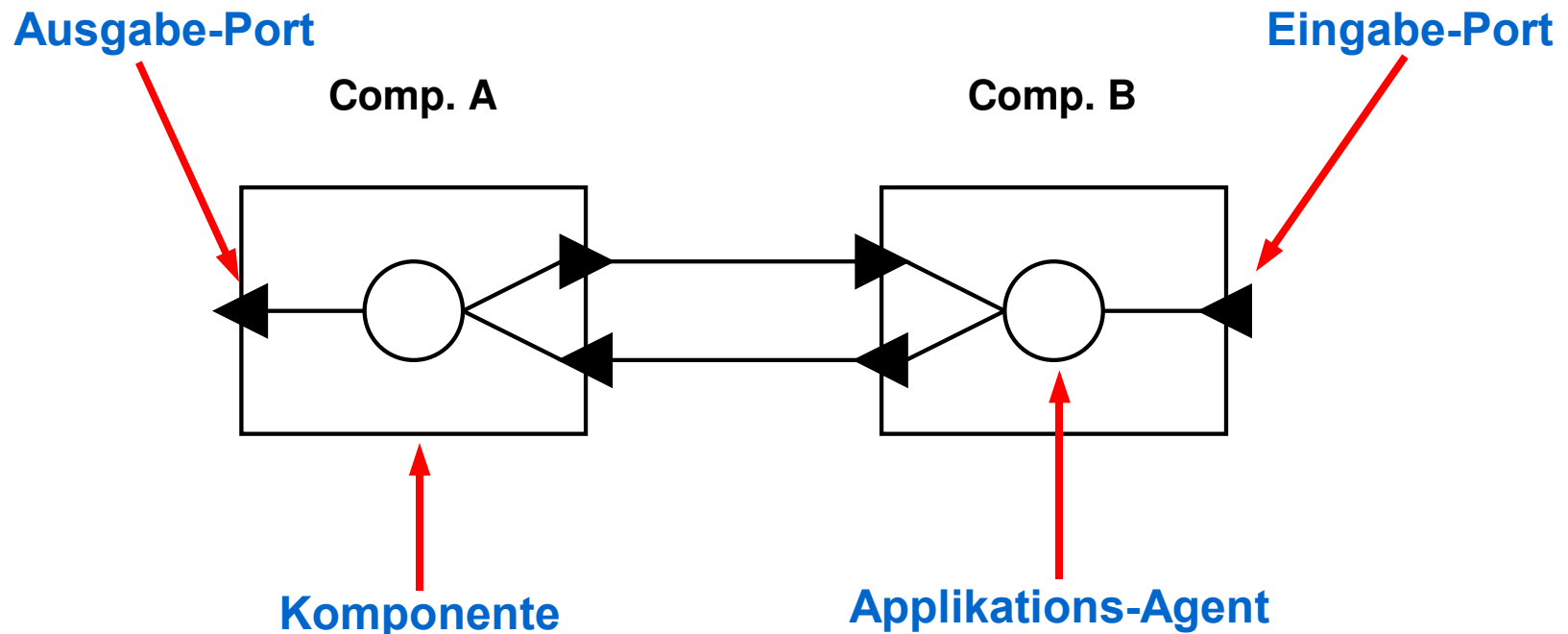
Workflow



SysWeaver – Modelle

■ Komponenten

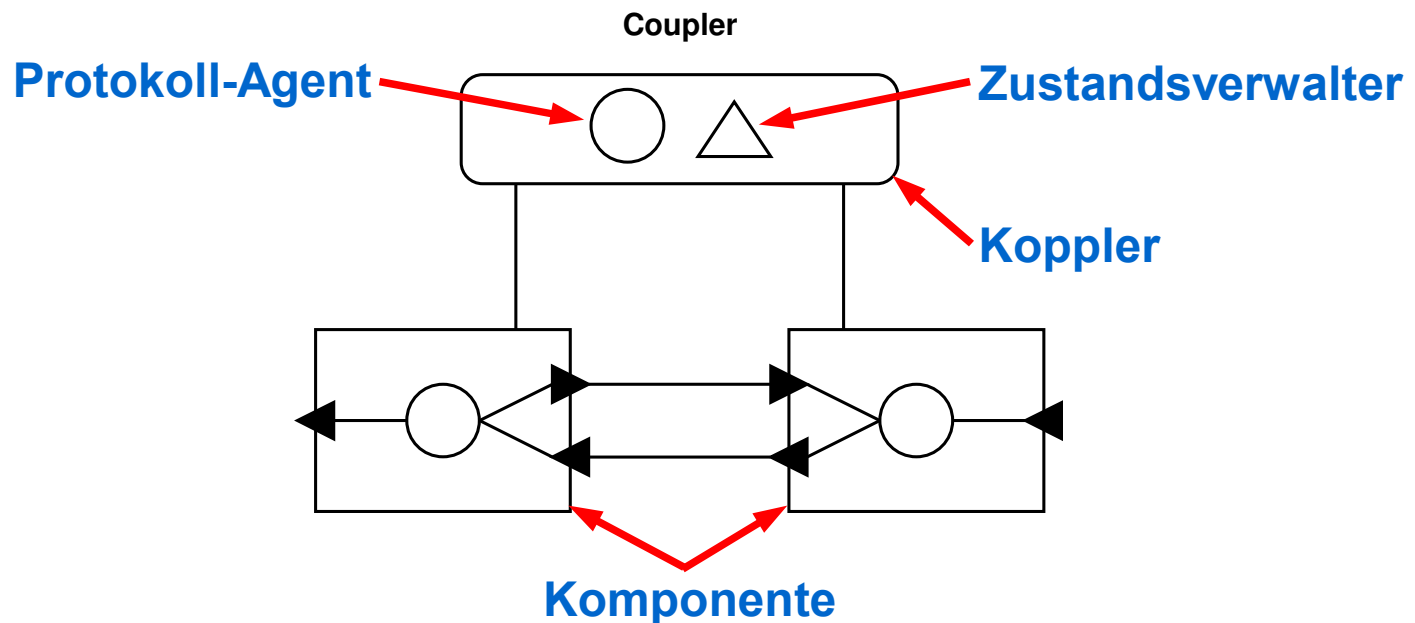
- enthalten **Applikations-Agenten**
- Applikations-Agenten konsumieren und erzeugen **Ereignisse**
- Ereignisse werden über **Ports** zu andern Komponenten propagiert



SysWeaver – Modelle

■ Koppler

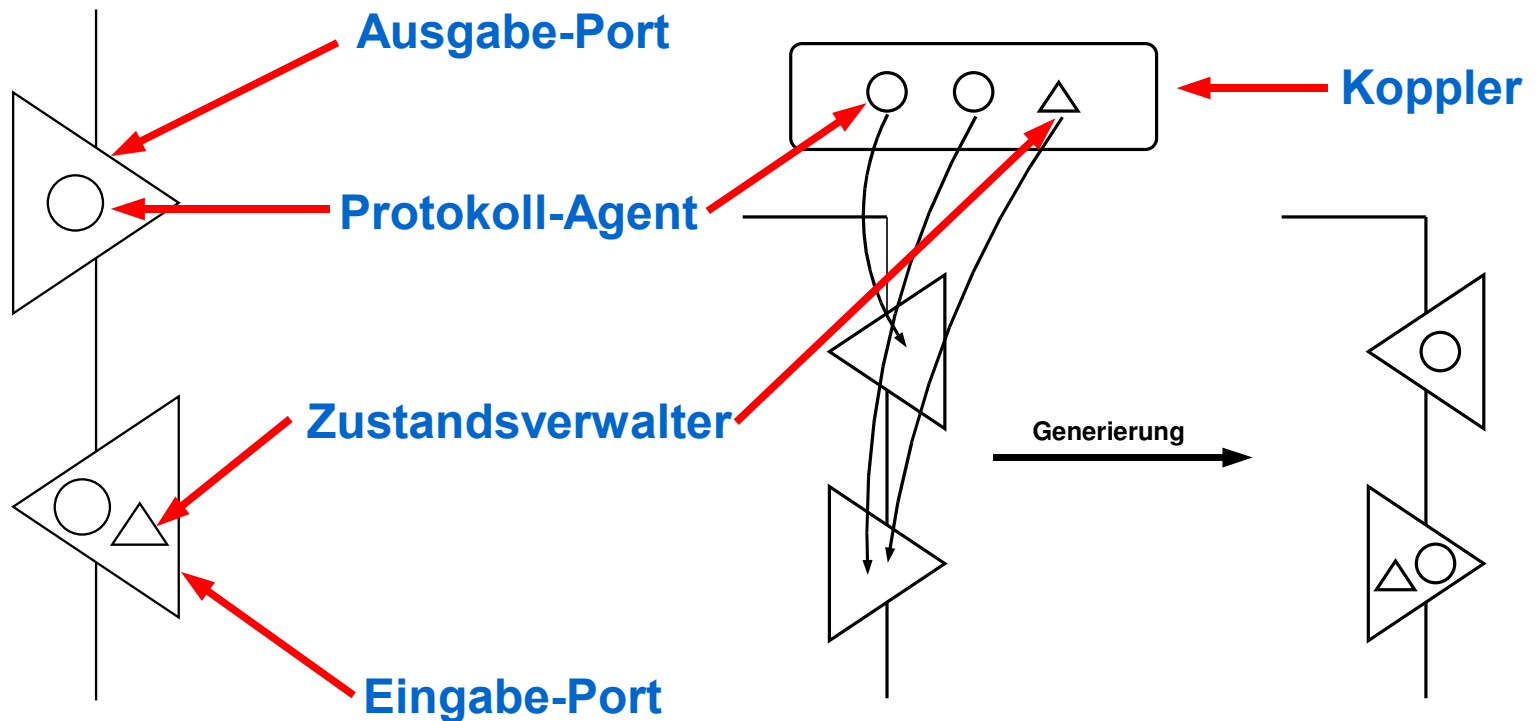
- verbinden Komponenten
- kapseln nicht-funktional Eigenschaften
- definieren **Protokoll-Agenten** und **Zustandsverwalter**
- Hierarchien möglich
- existieren nur im Modell



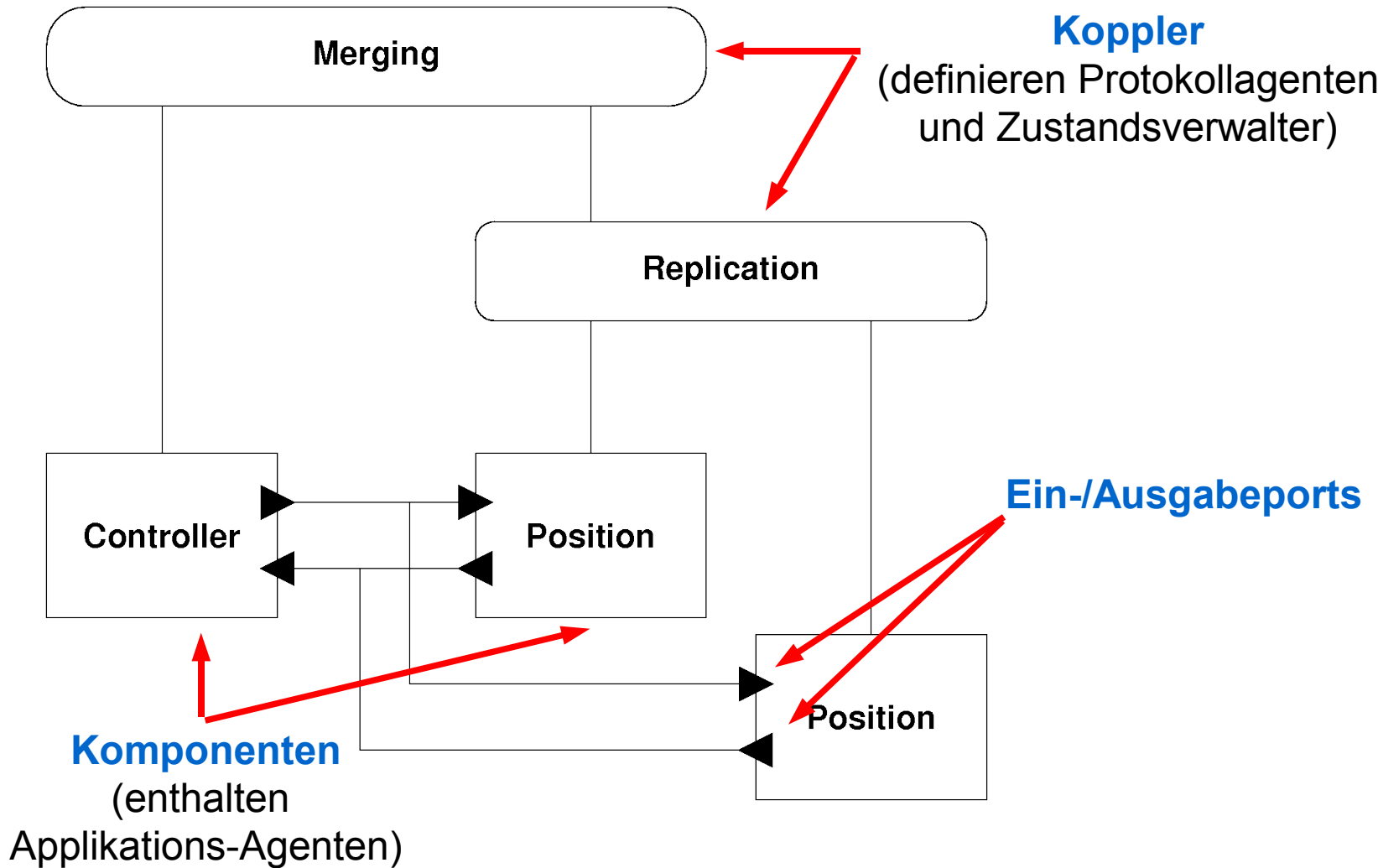
SysWeaver – Modelle

■ Ports

- enthalten Protokoll-Agenten und Zustandsverwalter
- Protokoll-Agenten und Zustandsverwalter werden während der Generierung *injiziert*

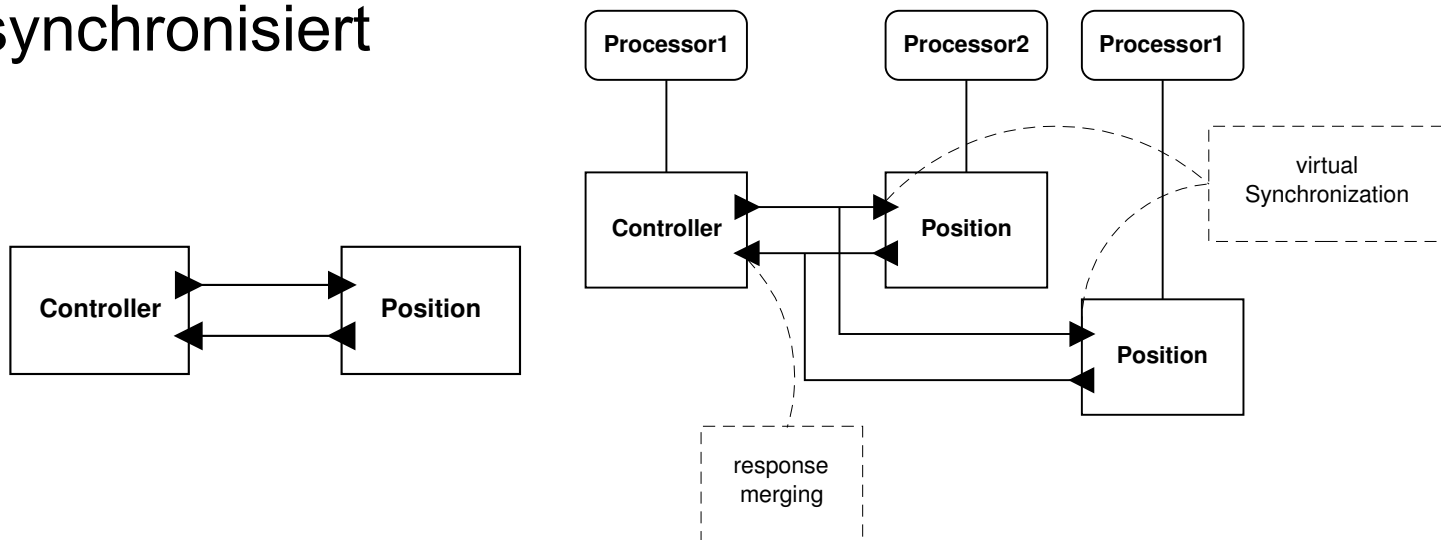


SysWeaver – Modelle: Beispiel



Mehrdimensionale Modellierung

- es existiert nur ein globales Modell
- verschiedene Ansichten sind Projektionen dieses Modells
 - hinsichtlich Fehlertoleranz
 - hinsichtlich Deployment
 - hinsichtlich ...
- Änderungen in den verschiedenen Sichten werden synchronisiert



rein funktionale Ansicht

funktionale Ansicht, Deployment

Zusammenfassung

- Grundlagen
 - Ereignisbehandlungen
 - Abbildung auf Betriebssystemdienste
 - Berechnungen statischer Ablaufpläne
 - Planbarkeitsanalyse
- traditionelle Komposition
 - Probleme
- Komposition in der Forschung
 - Entkopplung und Automation der Einzelschritte
 - Beispiele
 - TTA & Cluster Compiler
 - SysWeaver



Literatur

1. *Richard Nass.*
Annual study uncovers the embedded market.
Embedded Systems Design, 2007.

