Strukturelemente in Echtzeitsystemen

Florian Franzmann Martin Hoffmann Tobias Klaus
Peter Wägemann

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
http://www4.cs.fau.de

24. November 2014

Ausführungsmodell in tt-eCos

- *Ereignisorientiertes* Ausführungsmodell (≠ ereignisgesteuert!)
  - *keine* Endlosschleife in der Anwendung
  - ⇝ Betriebssystem startet Faden, der Jobs abarbeitet und sich beendet
- Einlastung erfolgt *verdrängend*
  - Neue Aufgabe unterbricht Ausführung laufender Aufgabe
  - Anschliessend Fortsetzung der unterbrochenen Aufgabe
  - ⇝ Terminüberprüfung möglich
- *Aber:* Faden blockiert sich nie selbst
  - sonst würde kein Fortschritt mehr stattfinden
  - ⇝ *run-to-completion-Semantik*

Vergleich mit eCos: *Prozessorientiertes* Ausführungsmodell

- Anwendungsthread implementiert Endlosschleife . . .
- . . . die sich blockiert und auf Ereignis wartet

Übersicht

Software Komponenten / Module:

- Identifikation
- Spezifikation
- Modellierung
- Implementierung
- Ergebnis

Identifikation von Modulen

Modularisierung:

- Was ist ein Modul?
- Wie teilt man ein System in Module auf?
- Welche Module gibt es in meinem System?
- Wie sind diese Module gekoppelt?
- Wie interagieren diese Module?
- Welche Module kann man zusammenfassen?

Implementierung:

- Softwarekomponente (-modul)
  - Klasse,
  - Prozedur,
  - Übersetzungseinheit, ...
- hat eine wohldefinierte Funktion
- hat eine wohldefinierte Schnittstelle

Wie gliedert man ein System in Module?

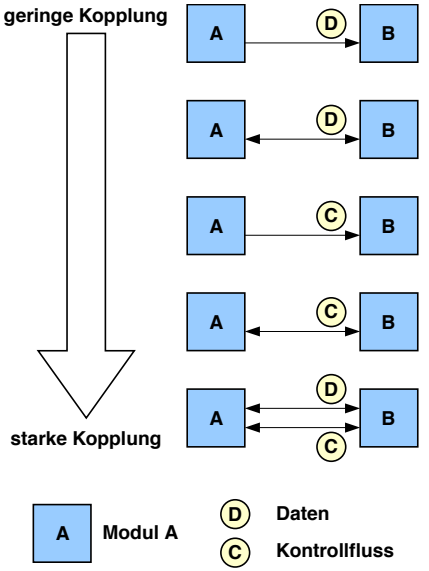
- **Algorithmus existiert nicht!**
- ist **anwendungsbezogen** – hängt vom System ab
- ist **subjektiv** – hängt vom Designer ab
- mögliche Kriterien
  - Partitionierung des Systems
  - Größe der Module
  - Komplexität der Module
  - externe Schnittstelle der Module
  - Beziehung und Kommunikation der Module untereinander
  - Bereich der Kontrolle und des Einflusses eines Moduls

⇝ **Unabhängigkeit**! Identifikation erfordert Erfahrung!

Kopplung von Modulen

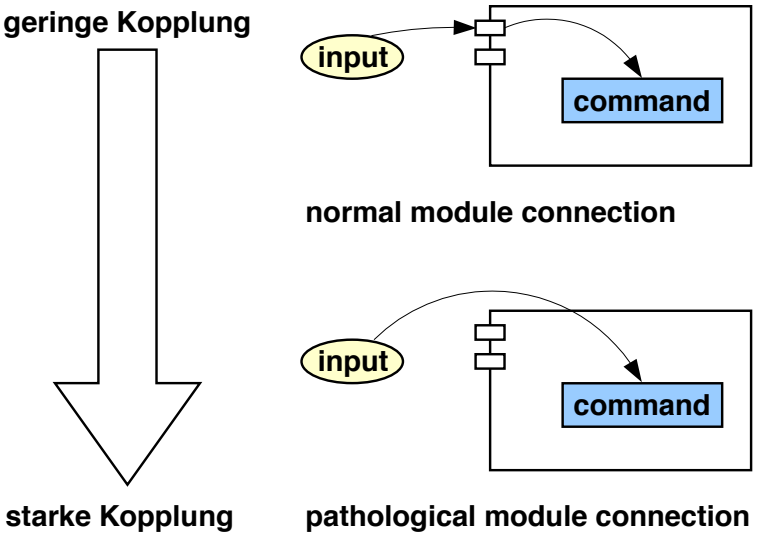
- Maß für die **Unabhängigkeit** von Modulen
- **niedrige** Kopplung
  - einfach zu verstehen
  - wartbar
  - verlässlich
  - stabil
- **hohe** Kopplung
  - hohes Maß an "Collateral Evolution"
- Hauptfaktoren
  - Komplexität von Information
  - **Informationstyp**
  - **Kommunikationsmethode**
  - Art der **Kopplung**

Kopplung – Informationstyp



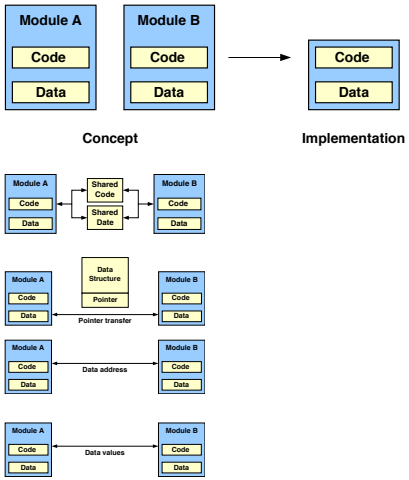
- unidirektionaler Datenfluss
- bidirektionaler Datenfluss
- unidirektionaler Kontrollfluss
- bidirektionaler Kontrollfluss
- bidirektionaler Daten- und Kontrollfluss

Kopplung – Kommunikationsmethode



Kopplung – Art

starker Kopplung

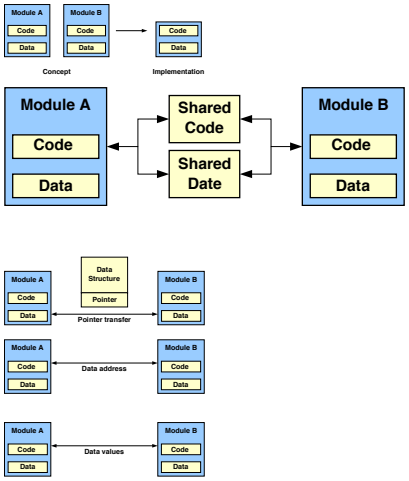


- **Content Coupling**
  - Module existieren im Konzept
  - keine Trennung in der Implementierung
- Common Coupling
- Stamp Coupling
- Data Coupling – by reference
- Data Coupling – by value

geringe Kopplung

Kopplung – Art

starker Kopplung

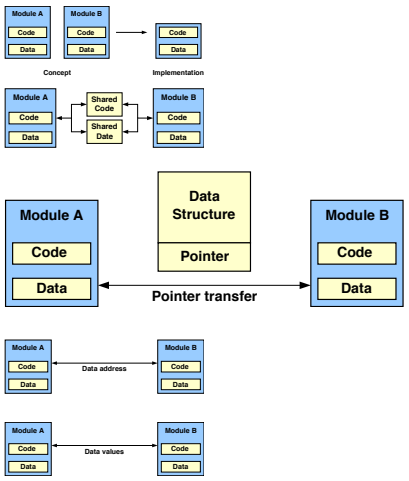


- Content Coupling
- **Common Coupling**
  - Unterscheidung: globale vs. private Daten- und Programmbereiche
  - globale Bereiche von allen Modulen zugreifbar
- Stamp Coupling
- Data Coupling – by reference
- Data Coupling – by value

geringe Kopplung

Kopplung – Art

starker Kopplung

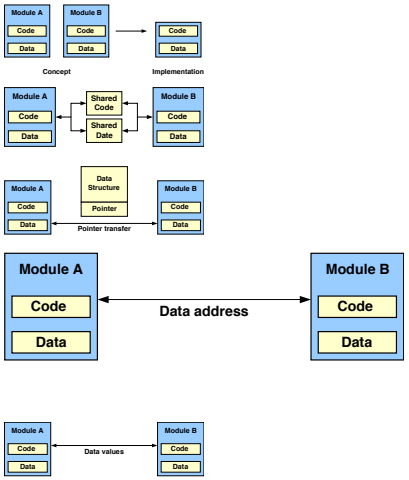


- Content Coupling
- Common Coupling
- **Stamp Coupling**
  - spezifische Datenstruktur statt globaler Bereiche
  - nur bestimmte Bereiche werden geteilt
  - teilende Module müssen konsistente Sicht auf die Datenstruktur haben
- Data Coupling – by reference
- Data Coupling – by value

geringe Kopplung

Kopplung – Art

starker Kopplung

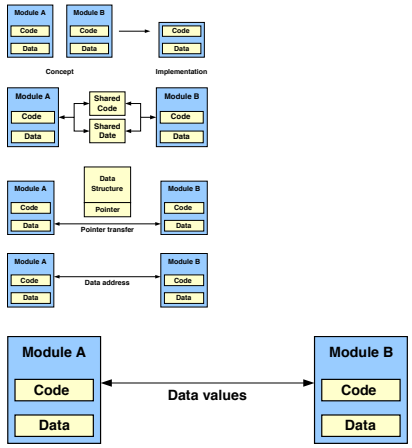


- Content Coupling
- Common Coupling
- Stamp Coupling
- **Data Coupling – by reference**
  - Zugriff auf Daten per Referenzen
  - Konsistenz der Daten durch Programmiersprache sicher gestellt
- Data Coupling – by value

geringe Kopplung

Kopplung – Art

starker Kopplung

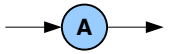


- Content Coupling
- Common Coupling
- Stamp Coupling
- Data Coupling – by reference
- **Data Coupling – by value**
  - nur Werte werden weiter gegeben
  - Module können Zustände anderer Module nicht ändern
  - sehr sicher
  - hoher Speicherbedarf

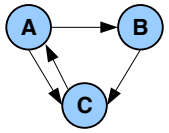
geringe Kopplung

Kohäsion

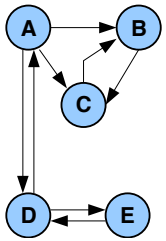
- **ideales Module**
  - sehr einfach
  - 1 Eingang, 1 Ausgang
- **einfache Systeme**
  - übersichtlich
- **komplexere Systeme**
  - Systemkomplexität steigt schnell an
  - unübersichtlich
- Kompromiss
  - einfache Module vs. einfaches System



ideales Modul



einfaches System

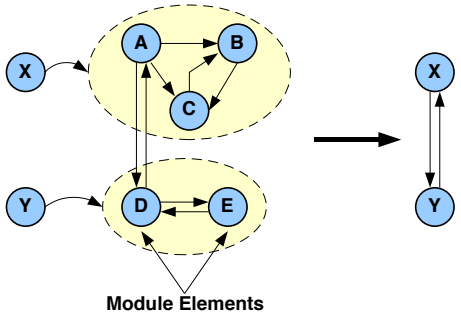


komplexeres System

Kohäsion

fasse Module zu **Super-Modulen** zusammen

- Module werden Elemente des Super-Moduls



Kohäsion: ein Maß für den **Zusammenhalt** der Elemente

- Anzahl der Verbindungen **untereinander**
- Anzahl der Verbindungen **nach außen**

Spezifikation

↝ **Beschreibung des Moduls**

- **Abhängigkeiten**
  - von welchen anderen Modulen hängt dieses Modul ab
  - welche anderen Module hängen von diesem Modul ab
- **Schnittstelle**
  - Syntax
  - Semantik
  - Vor- und Nachbedingungen

↝ Beschreibung des Moduls

- Zustandsdiagramme
- Sequenzdiagramme
- Aktivitätsdiagramme
- informell

Modellierung

- **ein Modell ist . . .**
- **keine Spezifikation!**
- "**abstrakte Implementierung**" der Spezifikation
  - oft basierend auf formalen Methoden
  - geeignet für die Verifizierung des Systems
  - geeignet für die automatische Erzeugung einer Implementierung
- kommt dem Verhalten des realen Systems sehr nahe

↝ **nicht Bestandteil dieser Veranstaltung**

Implementierung

- **In EZS verwendete Programmiersprache: C/C++**
- **Modulkonzept in C: schwach ausgeprägt**
  - hohe Disziplin des Entwicklers notwendig
  - Schnittstelle: Header
  - Implementierung: Übersetzungseinheit
  - private Programmstrukturen: Schlüsselwort `static`
  - globale Programmstrukturen: Schlüsselwort `extern`
- **muss überprüft / sicher gestellt werden**
  - Metriken
  - Coding-Guidelines
  - Code Reviews

↝ mehr dazu in der Vorlesung **Verlässliche Echtzeitsysteme**

Modularisierung – Zusammenfassung

- **minimiere Kopplung**
- **maximiere Kohäsion**
- **Unterstützung durch die Programmiersprache**
  - Modulkonzept in C: schwach ausgeprägt
  - besser: C++ - Klassen
  - explizites Modulkonzept: Ada

↝ orientiere dich am **realen System**

Randbedingungen für die Rahmenlänge

Lang genug und so kurz wie möglich halten...

Jobverdrängung vermeiden $\rightsquigarrow$ f hinreichend lang

1. ist erfüllt, wenn gilt: $f \geq \max(e_i)$, für $1 \leq i \leq n$
   $\rightsquigarrow$ jeder Job läuft in der durch f gegebenen Zeitspanne durch
2. f teilt die Hyperperiode H so, dass gilt: $\lfloor p_i/f \rfloor - p_i/f = 0$
   $\rightsquigarrow$ ermöglicht die zyklische Ausführung des Ablaufplans

- das Intervall H heißt großer Durchlauf
- das Intervall der Länge f heißt kleinster Durchlauf

Terminüberwachung unterstützen $\rightsquigarrow$ f hinreichend kurz

3. erfordert eine rechtzeitige Auslösung: $f \leq p_i$, für $1 \leq i \leq n$
4. ist möglich unter der Bedingung: $2f - \text{ggT}(p_i, f) \leq D_i$

- anstehenden Aufgaben „passend“ auf die Rahmen verteilen
  $\rightsquigarrow$ Jobs zwischen Auslösezeit und Termin erledigen

Beispielsystem

Aufgabe T_i	Periode p_i ms	WCET e_i ms	Termin D_i ms
T_1	9	2	5
T_2	18	3	8
T_3	45	3	45

Aufgabe 5 - Hinweise

Wichtige Hinweise

- Die Lösung erfolgt **konstruktiv**
  $\rightsquigarrow$ keine Implementierung notwendig
- Kern der Aufgabe: Auswirkung der Rahmenlänge
- Denksportaufgabe!
- Abgabe: Rechnerübungen in der Woche vom 08.12.2014