

4. KONZEPTIONELLE MODELLE

4.1 Lebenszyklus

Aktivitäten beim Entwicklungsprozess

- Identifikation der Anforderungen: **Was soll getan werden?**
- Entwurf einer Lösung: **Wie soll es getan werden?**
- Implementierung: **Herstellung des Systems**
- Test und Evaluation: **Funktioniert es wirklich?**

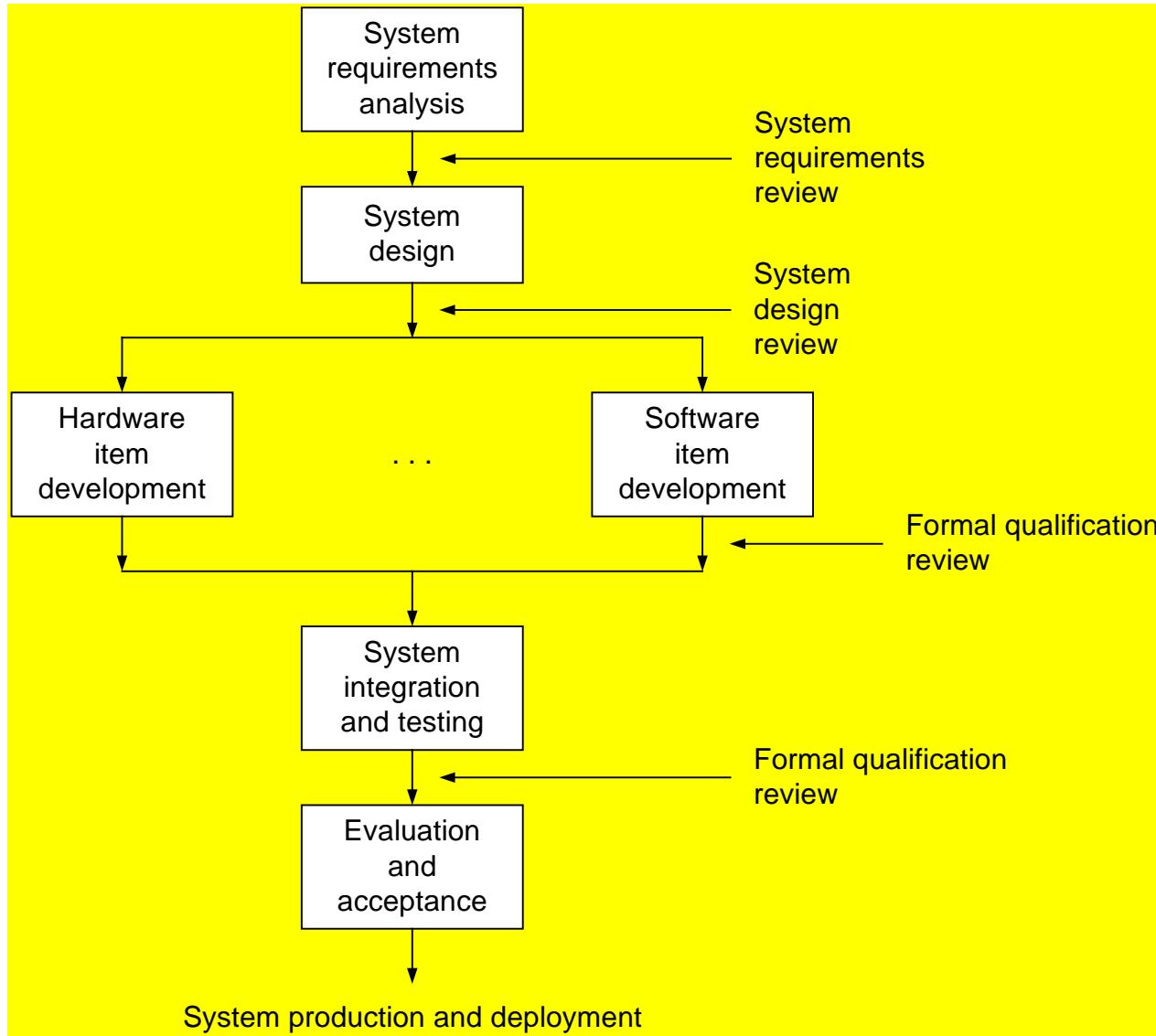
Eine Untereinteilung in mehrere zeitliche und räumliche Schritte

⇒ **Lebenszyklus**

Übergang von einem Schritt zum nächsten im Lebenszyklus klar definiert.
Dokumente beschreiben Ergebnis eines Schritts und sind Voraussetzung für den
nächsten.

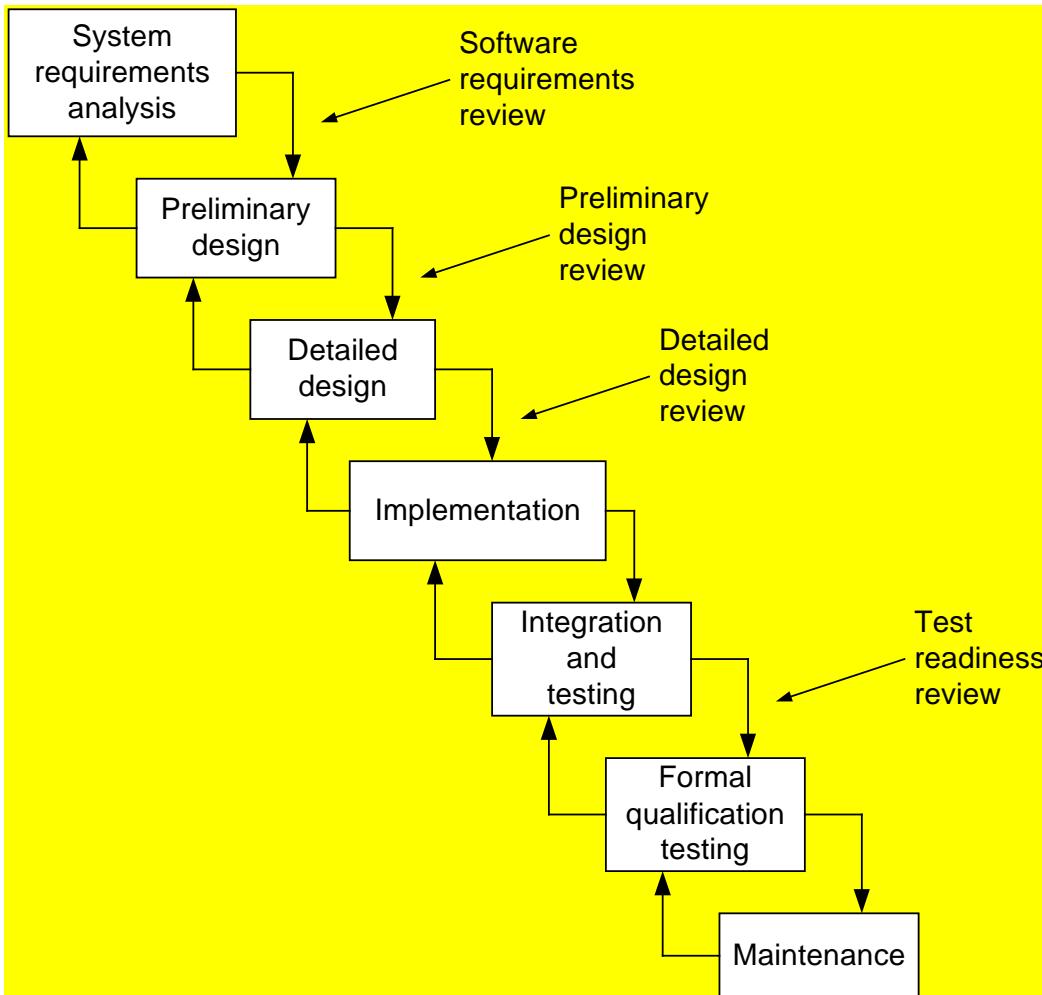
Rahmen für den systematischen Systementwurf

System-Lebenszyklusmodell



Software-Lebenszyklus

Beispiel: Wasserfall-Modell (Klassischer Ansatz)



- Dokumente beschreiben den aktuellen Status der Software und bilden die Basis für die weiteren Aktivitäten.
- Stellen die Schnittstelle zwischen den Schritten und zum Projekt-Management dar.
- Halten alle wichtigen Entscheidungen und Festlegungen fest und bilden die Basis für die Modifikation und Wartung der Software.

Unterstützung der Dokumentverwaltung durch CASE-Tools

Kosten

Typische Kostenverteilung über den Lebenszyklus (Entwicklungskosten = 100%)

(i)	Requirement analysis	15%
(ii)	Preliminary design	10%
(iii)	Detailed design	15%
(iv)	Implementation	20%
(v)	Integration and testing	20%
(vi)	Qualification	20%
(vii)	Maintenance	70-200%

Die meisten Kosten entstehen nach der Implementierung!!!

⇒ Fehler müssen schon in den frühen Entwurfsphasen weitgehend ausgemerzt werden. Software muss so konzipiert werden, dass sie leicht an sich ändernde Benutzeranforderungen anpassbar und erweiterbar ist.

Hilfsmittel:

Einsatz formaler Modelle und Methoden statt umgangssprachlicher Dokumente.
Weitgehend automatisierter, rechnergestützter Entwurf.

4.2 Konzeptionelle Sichtweisen und Modell-Taxonomie

Je nach Aufgabenstellung und Phase im Lebenszyklus unterschiedliche konzeptionelle Darstellung des Systems.

Beispiel: Aufzugssteuerung

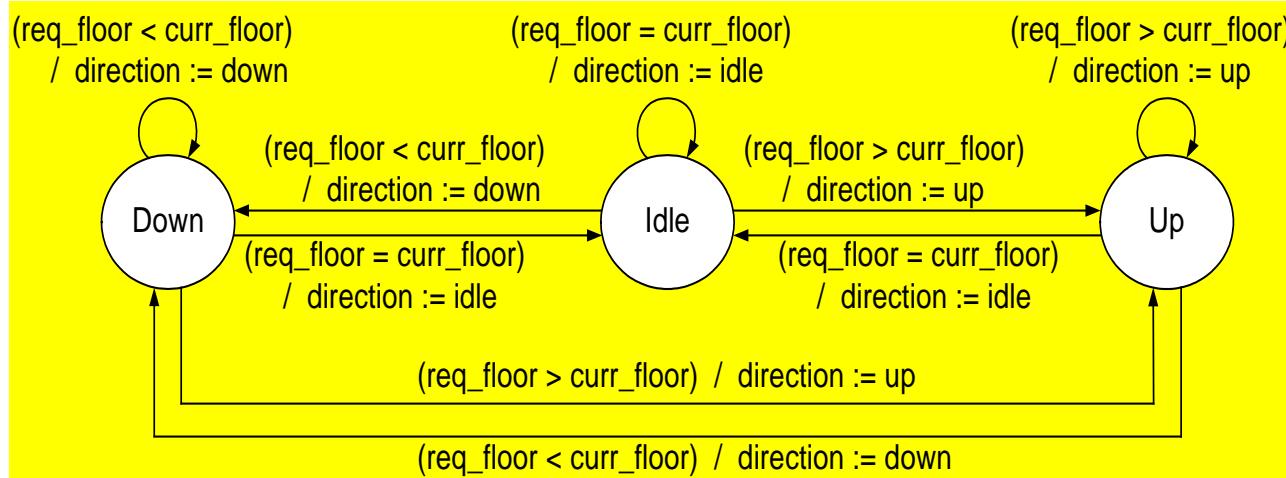
- Umgangssprachliche Systemspezifikation

„If the elevator is stationary and the floor requested is equal to the current floor, then the elevator remains idle.

If the elevator is stationary and the floor requested is less than the current floor, then lower the elevator to the requested floor.

If the elevator is stationary and the floor requested is greater than the current floor, then raise the elevator to the requested floor.”

- Endlicher Automat (FSM - Finite State Machine)



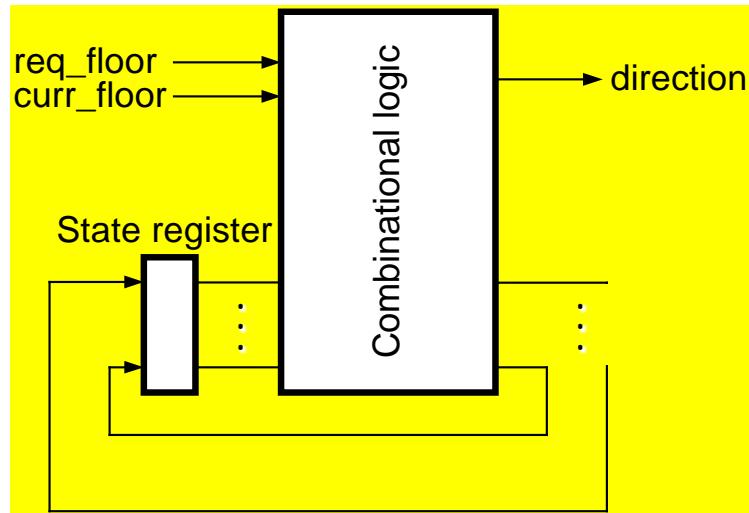
- Algorithmisches Modell

```
loop
    if (req_floor = curr_floor) then
        direction := idle;
    elseif (req_floor < curr_floor) then
        direction := down;
    elseif (req_floor > curr_floor) then
        direction := up;
    end if;
end loop;
```

Abbildung des konzeptionellen Modells auf eine geeignete Hardware-Architektur

Beispiel: Aufzugssteuerung

Schaltwerk



Wünschenswert:

Automatische Transformation zwischen Modellen

Automatische Abbildung auf Architektur

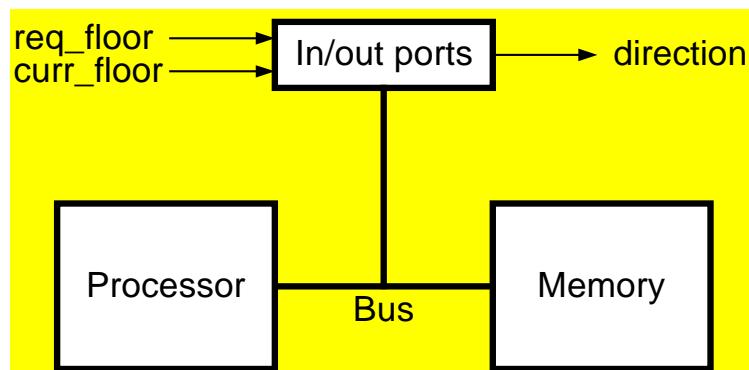
Vergleich Hardwareentwurf:

Automatische Synthese einer Hardwareschaltung aus Verhaltensmodell (FSM, VHDL) schon recht effizient möglich.

Voraussetzung:

Formale Modelle zur Verhaltenbeschreibung

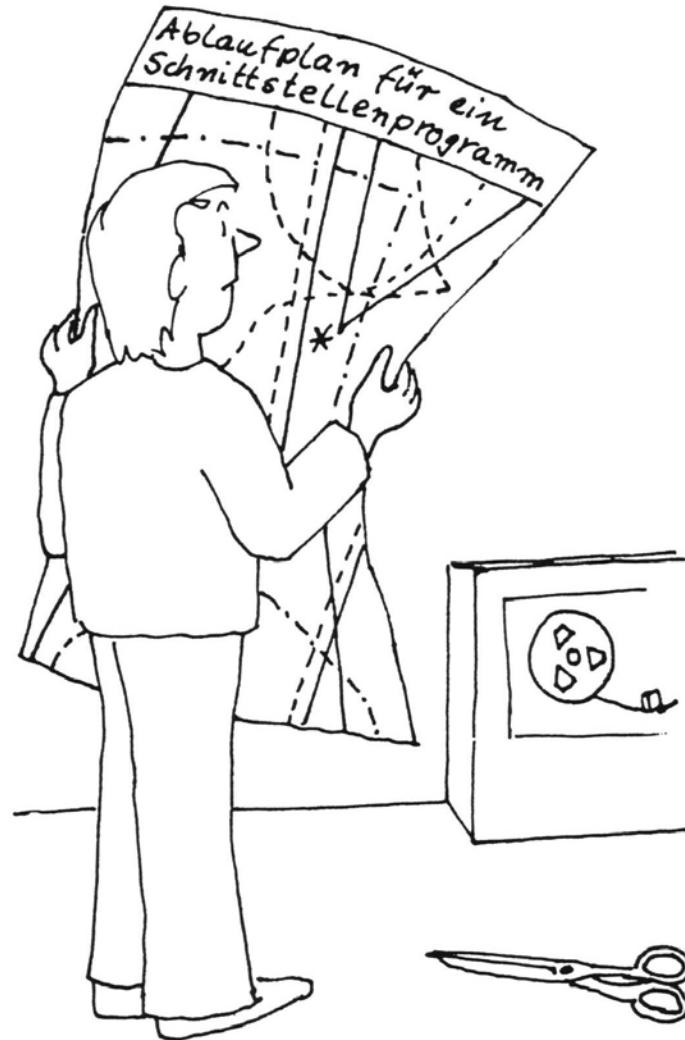
Mikrocontroller



Modell-Taxonomie [Gajski et al 94]

- zustandsorientiert
- aktivitätsorientiert
- strukturorientiert
- (datenorientiert)
- heterogen

In heute üblichen Entwurfswerkzeugen oft Kombination verschiedener Modelle (z.B. Statemate: aktivitäts-, zustands- und strukturorientierte Teilmodelle).



4.3 Zustandsorientierte Modelle

4.3.1 Endliche Automaten (Finite State Machines - FSM)

Zeitliches Verhalten wird durch Zustände, Zustandsübergänge und Aktionen ausgedrückt.

Definition

Ein *endlicher Automat (FSM)* ist ein Quintupel

$$\langle S, I, O, f, h \rangle$$

mit

$$S = \{s_1, s_2, \dots, s_e\}$$

endl. Menge von *Zuständen*

$$I = \{i_1, i_2, \dots, i_m\}$$

Menge von *Eingaben*

$$O = \{o_1, o_2, \dots, o_n\}$$

Menge von *Ausgaben*

$$f: S \times I \rightarrow S$$

Zustandsübergangsfunktion

$$h: S \times I \rightarrow O$$

Ausgabefunktion (Mealy)

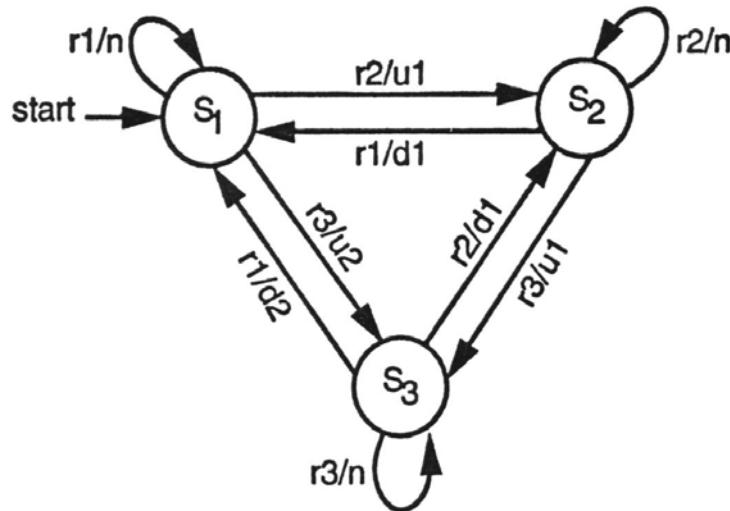
$$h: S \rightarrow O$$

Ausgabefunktion (Moore)

Varianten:

Mealy- bzw. Moore-FSM
(unterschiedliche Ausgabefunktionen)

Beispiele: (1) Aufzugssteuerung als Mealy-FSM



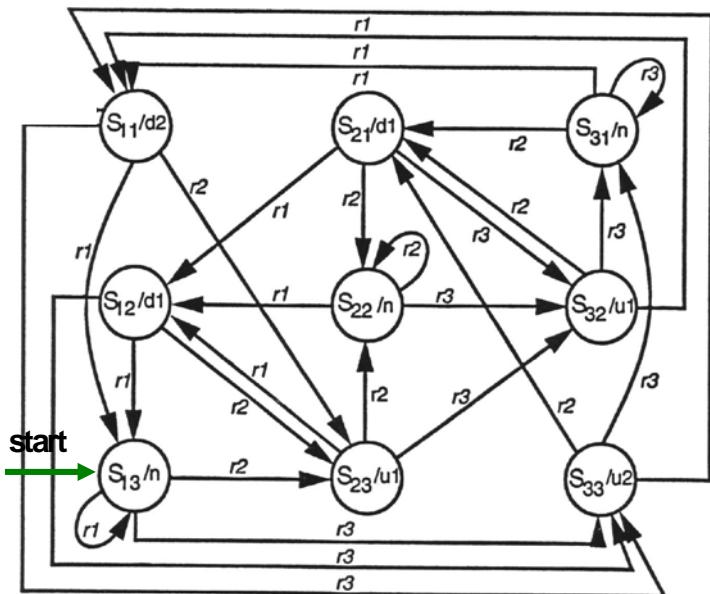
$$I = \{r1, r2, r3\}$$

Anforderung von Stockwerk $i = 1, 2, 3$

$$O = \{d2, d1, n, u1, u2\}$$

Richtung ($d = \text{down}$, $n = \text{neutral}$, $u = \text{up}$) und Anzahl Stockwerke $j = 1, 2$

(2) Aufzugssteuerung als Moore-FSM



Äquivalentes Verhalten – aber mehr Zustände

Problem bei FSMs:

Zustandsexplosion bei der Repräsentation von Integer- oder Floating-Point-Zahlen (je 1 Zustand pro Wert!)

Lösung: Einführung von Variablen

FSMD: FSM mit Datenpfad

Definition

Seien

VAR: Menge von *Speichervariablen* (z. B. Integer)

$\text{EXP} = \{\text{Fun}(x, y, z, \dots) \mid x, y, z, \dots \in \text{VAR}\}$ eine Menge von *Ausdrücken* (Funktionen)

$A = \{x \Leftarrow e \mid x \in \text{VAR}, e \in \text{EXP}\}$ eine Menge von *Speicherzuweisungen*

$\text{STAT} = \{\text{Rel}(a,b) \mid a, b \in \text{EXP}\}$ eine Menge von logischen *Relationen* zwischen zwei Ausdrücken

Dann ist eine **FSMD** gegeben durch das Quintupel

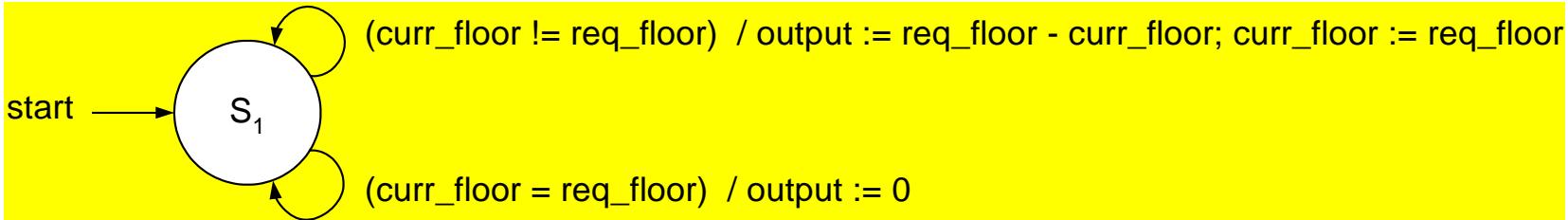
$\langle S, I \cup \text{STAT}, O \cup A, f, h \rangle,$

wobei die *Zustandsübergangsfunktion* f und die *Ausgabefunktion* h erweitert werden zu

$f: S \times (I \cup \text{STAT}) \rightarrow S$

$h: S \times (I \cup \text{STAT}) \rightarrow O \cup A$

Beispiel: Aufzugssteuerung



Nur noch ein Zustand erforderlich!

Aktuelles Stockwerk wird in Variable *curr_floor* gespeichert.

Ausgabe *Output* $\in O = \{-2, -1, 0, +1, +2\}$ gibt an, in welche Richtung (-: down, 0 : neutral, +: up) und um wie viele Stockwerke (1 oder 2) der Aufzug fahren soll.

Kritik des FSM und FSMD-Modells

- + intuitive Formulierung einfacher sequentieller Abläufe.
- bei höherer Zustandszahl schnell unübersichtlich (keine Hierarchisierung).
- Parallelle Abläufe führen zur Zustandsexplosion (bei k parallelen Prozessen mit je n Zuständen n^k Zustände des Automaten!!!).

4.3.2 Hierarchische nebenläufige endliche Automaten (Hierarchical Concurrent Finite-State Machines – HCFSM)

FSM mit Unterstützung von

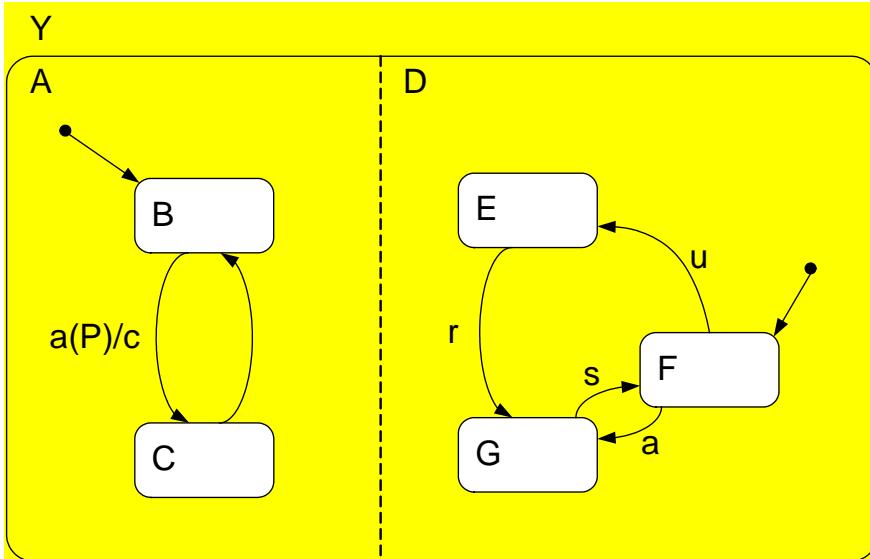
- *Hierarchisierung*
- *Nebenläufigkeit*

damit Gefahr der Zustands- und Kantenexplosion gebannt.

Konzept

- Jeder Zustand kann (rekursiv) aus Unterzuständen bestehen (Zustandshierarchien)
- Jeder Zustand kann in nebenläufige Unterzustände aufgeteilt sein (nebenläufige Sub-FSMs / Kommunikation über globale Variable)
- Transitionen (Kanten) innerhalb einer Hierarchieebene und zwischen Hierarchien zulässig

Beispiel: Statecharts (Teil von Statemate, UML)



**Weitaus komplexere Systeme
modellierbar als mit FSMS.**

Zustand Y aufgeteilt in zwei *nebenläufige* Zustände A und D.

Zustand A besteht aus Subzuständen B und C. B ist Startzustand.

Zustand D besteht aus Subzuständen E, F und G. F ist Startzustand.

Eingaben (Ereignisse) können durch Prädikate bedingt zu Zustandsübergängen führen (Übergang von B nach C wird nur ausgeführt, wenn Ereignis a eintritt und Prädikat P erfüllt ist).

Ausgaben (Aktionen) bei Transitionen möglich (z.B. c bei Transitionen von B nach C).

4.4 Aktivitätsorientierte Modelle

4.4.1 Datenfluss-Graphen (Dataflow Graphs)

Zustandsorientierte Modelle vorwiegend für *reaktive, ereignisgetriebene* Systeme geeignet.

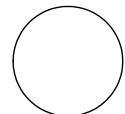
Datenflussmodelle bieten sich mehr für Systeme zur Transformation *kontinuierlicher Datenströme* (z. B. Signalverarbeitung) an.

Datenflussgraph (DFG)

Knotentypen



Eingabe bzw. Ausgabe



Aktivitäten (Prozesse zur Datentransformation oder -manipulation)
Bsp.: Prozeduren, Funktionen, Operationen,...



Datenspeicher (Datei, Variable, Register,...)

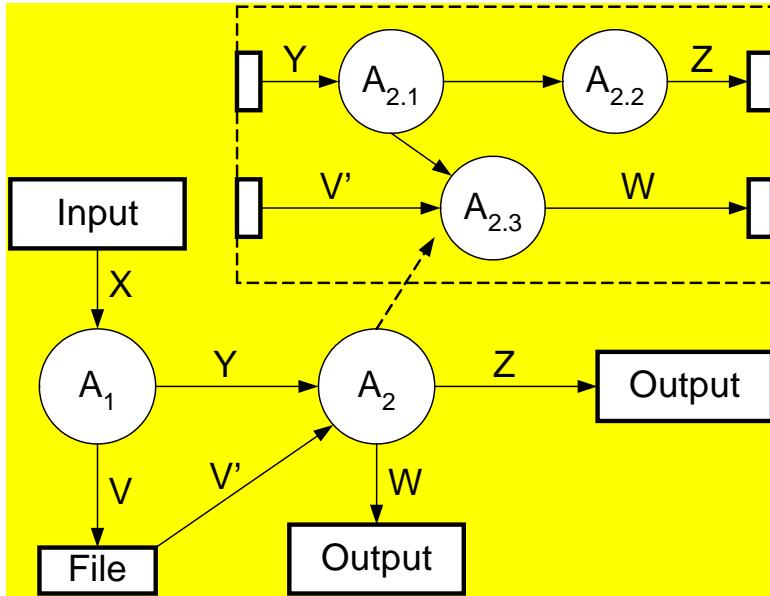
Kanten



geben *Datenfluss* an, markiert mit zugehörigem Datum

Hierarchien: Aktivität knoten können wiederum DFGs enthalten.

Beispiel



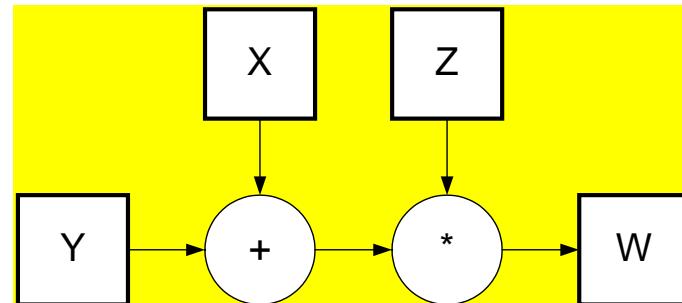
Aktivität A_1 verarbeitet Eingabedatum X und erzeugt Ausgabedatum V zur Ablage in einem File sowie Y zur Weitergabe an Aktivität A_2 .

A_2 besteht aus Sub-DFG mit Aktivitäten $A_{2,1}$, $A_{2,2}$ und $A_{2,3}$.

Keine Angaben über zeitliches Verhalten, nur statische Repräsentation der Datenflüsse.

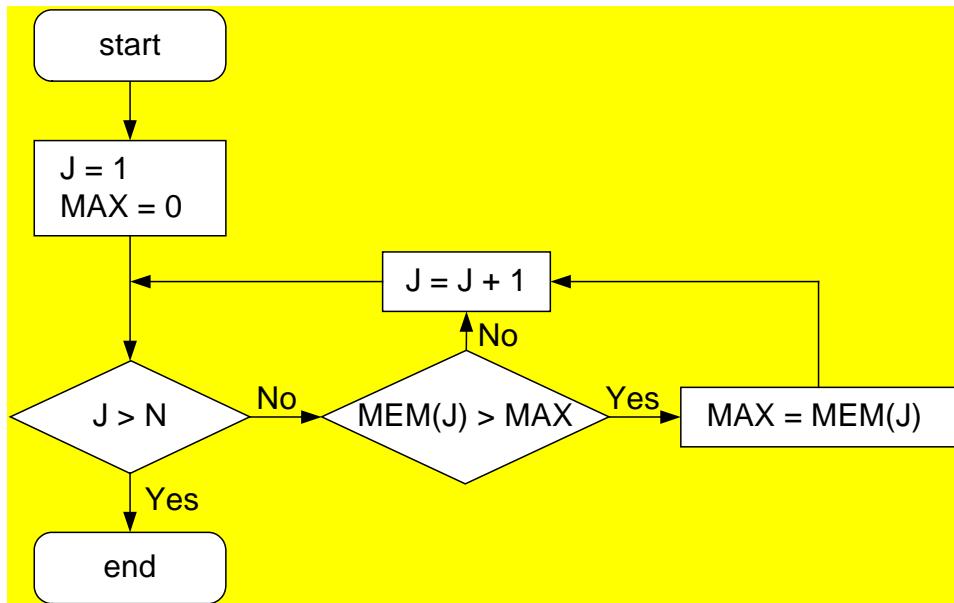
Unabhängig von der Implementierung (für Spezifikationsphase geeignet).

**Für eingebettete Systeme wegen fehlender Zeitangaben nur bedingt geeignet
(Ausnahme kontinuierliche Datenströme in der Signalverarbeitung).**



4.4.2 Flussdiagramme (Flowcharts)

Kontrollflussgraphen (Control-Flow Graphs - CFG)



Geben den Kontrollfluss eines Programms wieder.

Schon relativ nah an der Implementierung des Programms (ziemlich detailliert).

Sequentielles Ausführungsmodell wie bei FSMs, aber keine Ereignistriggerung, sondern einfacher sequentieller Ablauf (von-Neumann-Modell).

Sehr weit verbreitet, zur Modellierung eingebetteter Systeme aber nur bedingt geeignet. Entsprechendes gilt für Struktogramme (Nassi-Shneiderman-Diagramme).

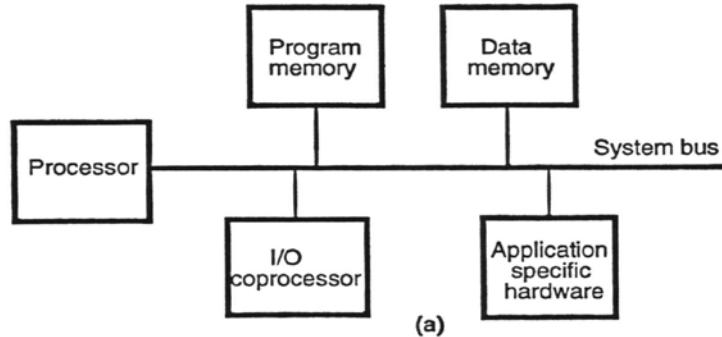
4.5 Strukturorientierte Modelle

4.5.1 Komponentenverbindungsdiagramme

Geben die Struktur eines Systems auf verschiedenen Abstraktionsebenen wieder. Keine Angaben über dynamisches Verhalten.

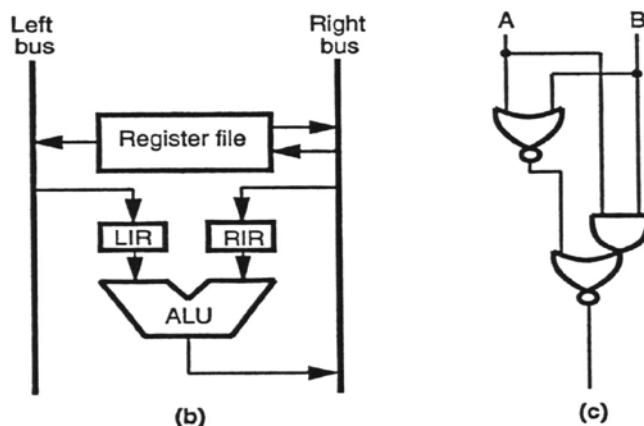
Beispiele

(a) Blockdiagramm; (b) Register-Transfer-Diagramm; (c) Gatterlaufplan



Verschiedene Knotentypen (Symbole) für Komponenten.

Kanten stellen Verbindungen von Komponenten dar.



Besonders geeignet für spätere Phasen des Entwurfsprozesses.



4.6 Heterogene Modelle

Kombinieren mehrere der oben ausgeführten Modellierungsparadigmen, um eine bessere Ausdrucks Kraft zu erreichen.

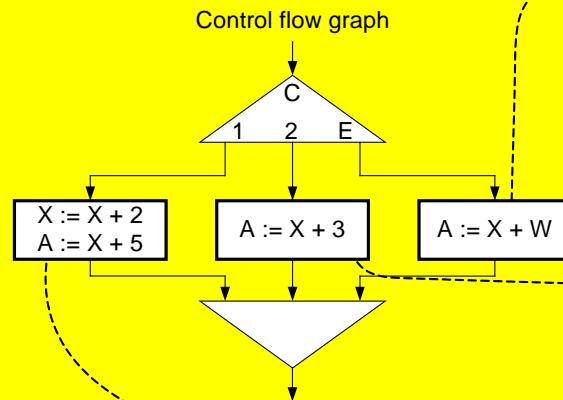
Verschiedene Varianten bekannt und in der Praxis im Einsatz.

Hier nur einige typische Beispiele

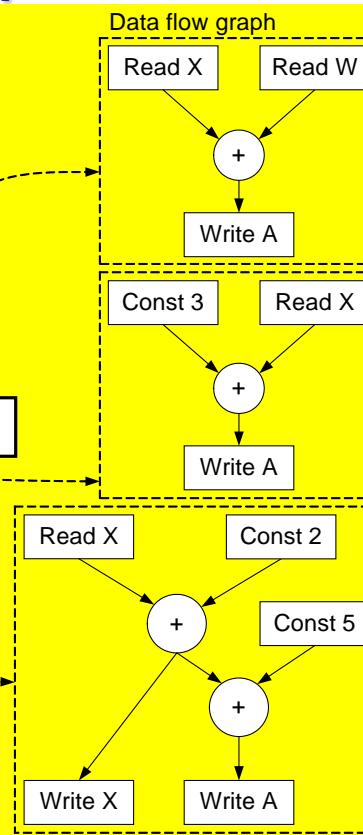
4.6.1 Kontroll-/Datenflussgraphen

```
case C is
  when 1 => X := X + 2;
              A := X + 5;
  when 2 => A := X + 3;
  when others => A := X + W;
end case;
```

(a)



(b)

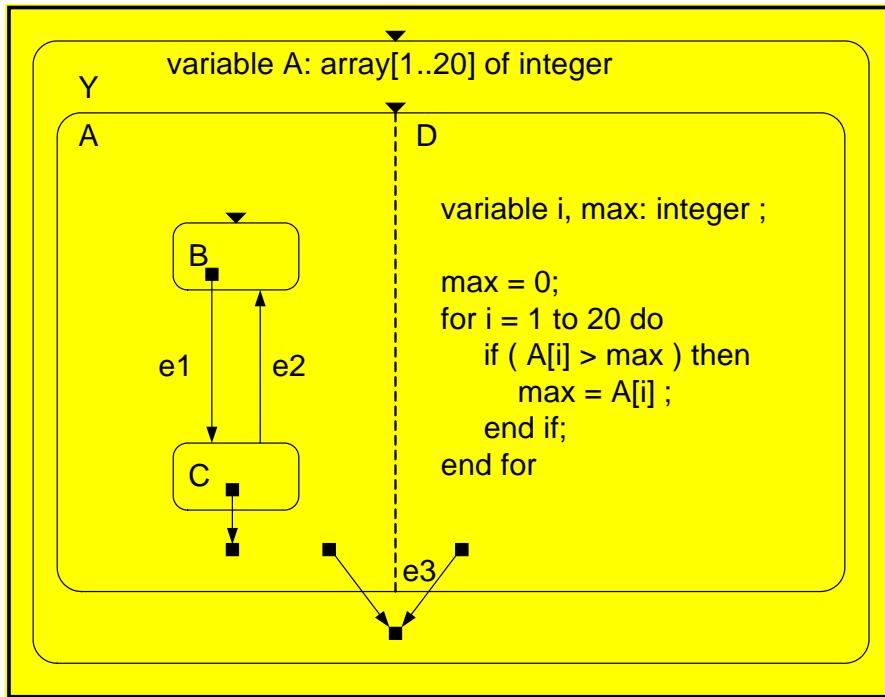


Kontrollfluss wird durch Kontrollflussgraphen modelliert, Datenabhängigkeit durch Datenflussgraphen.

Verbindung beider Darstellungen durch gestrichelte Kanten.

4.6.2 Program-State-Machine

Kombiniert FSM mit programmiersprachlicher Darstellung.



Darstellung von sequentiellen und nebenläufigen (A, D) Zuständen sowie Unterstützung von Hierarchien (vgl. Statecharts).

Blattzustände (tiefste Stufe in Hierarchie) werden durch Anweisungen in Programmiersprache dargestellt.

Anwendungsmöglichkeiten ähnlich Statecharts (etwas mächtiger).

Verschiedene Kantentypen

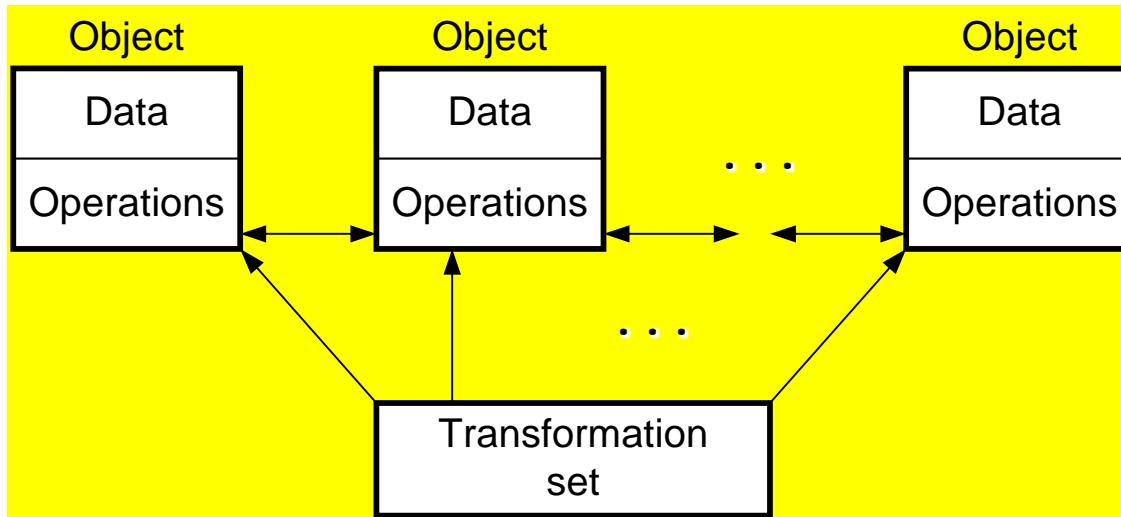
TOC (Transition-on-Completion): Übergang wird erst nach vollständiger Abarbeitung eines Zustands (dargestellt durch kleines schwarzes Quadrat am Pfeilanfang, z. B. e1).

TI (Transition Immediately): Übergang sofort bei Eintreten des Ereignisses, unabhängig von Beendigung des Zustands (z. B. e2).

4.6.3 Objektorientierte Modelle

System wird in Objekte zerlegt, die Daten **und** Operationen (Methoden) enthalten (Datenabstraktion, Information Hiding).

Objekte kommunizieren untereinander über Methodenaufrufe.



Problem:

Der zeitliche Ablauf der Berechnung muss adäquat dargestellt werden.

Objektorientierte Modelle bieten für Echtzeitsysteme gute Perspektive, daher zunehmende Verbreitung.