50 Jahre modellbasierter Entwurf:

Vom Modellieren mit Programmen zum Programmieren mit Modellen

Ringvorlesung
HU Institut für Informatik
Sommersemester 2005
W. Reisig

was ich erzählen will

1. Begriffe
   modellieren
   programmieren
2. Modellieren mit Programmen
   1955 – 1975
3. Modellieren mit andern Ausdrucksmitteln
   1955 – 1975
4. Modellieren verteilter und reaktivaktiver Systeme
   1975 – 1985
5. Temporale Logik
   seit 1980
6. Logik-basiertes Modellieren
   seit 1975
7. Requirements modellieren
   seit 1980
8. Programmieren mit Modellen
   seit 2000

1. Begriffe

Modellieren
Modell

Programmieren
Programm

Modellieren

- einige Aspekte der realen Welt herausgreifen (informal, intuitiv)
- diese Aspekte darstellen (formal, mit Mathematik)

Modell:
Ergebnis des Modellierens

Programmieren

- einen Algorithmus für die Ausführung am Rechner herrichten

Programm:
Ergebnis des Programmierens

Konsequenz

„Modellieren mit Programmen“:
Aspekte der realen Welt mit Mitteln darstellen,
die für die Ausführung mit einem Rechner geeignet sind.

„Programmieren mit Modellen“:
Aspekte der realen Welt darstellen wie man will
und dann von einem Rechner ausführen lassen

2. Modellieren mit Programmen

1955 – 1975

Ein Spiel von E. W. Dijkstra:

Gegeben eine Urne mit Pebbles (*Spielsteine*)
jedes Pebble ist entweder weiß oder schwarz

Schritt:
nimm 2 Pebbles aus der Urne.
Farben verschieden: ein weißes zurück
Farben gleich: ein schwarzes zurück.

Problem: Welche Farbe hat das letzte Pebble?

Lösungsverfahren:
Das Spiel modellieren, im Modell die Lösung ausrechnen

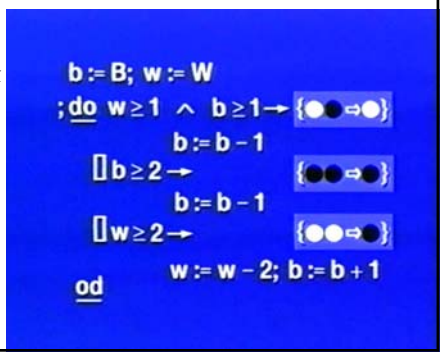
Dijkstra im Original

Dijkstra‘s Modell

was man alles machen soll:
take two pebbles out
look at the colours
return the white one
paint a pebble black

wo kommen diese Tätigkeiten vor im Modell?

nirgends !!!

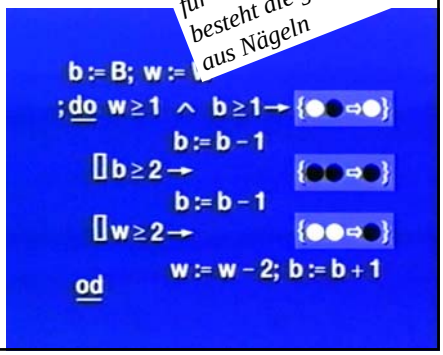


was kommt im Modell vor?

die Pebbles werden abgezählt
Auf den Zahlen wird gerechnet.
Warum das?

… weil Dijkstra als Modell ein “einfaches Programm” haben will!

für einen Hammer besteht die ganze Welt aus Nägeln



Programmieren 1960 - 1970

Numerik
Sortieren und Suchen
Betriebssysteme und Compiler
1968 Monstersprachen *PL/1*, *ALGOL 68*

Probleme wurden offensichtlich:

Garmisch-Partenkirchen-Tagung
„Softwarekrise“ „Software-Engineering“

Lösungsvorschläge

Dijkstra und andere:
Ein Programm ist ein Mathematisches Gebilde!

Wirth:
trotzdem muss das Programmieren die Intuition unterstützen!
Insbesondere mit Datenstrukturen

Wirth: PASCAL, OBERON
Dahl: Modula 2

Was aus heutiger Sicht damals gefehlt hat:
Idee der Modellierung als eigenständige Aufgabe

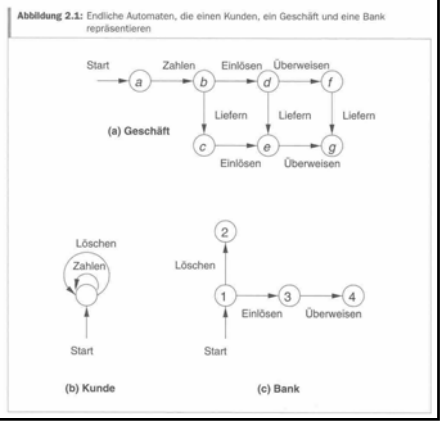
3. Modellieren mit anderen Ausdrucksmitteln 1955 – 1975

Wie beschreibt man Verhalten,

außer mit Programmen?

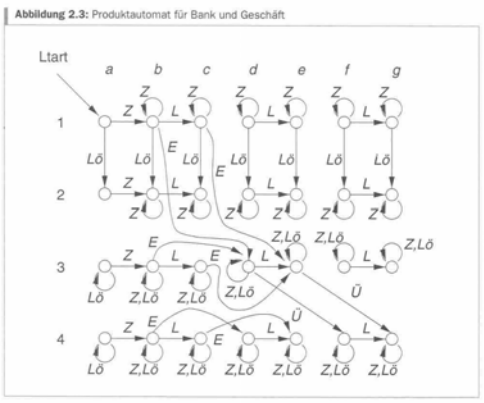
herrschende Lehre

mit Bildchen, insbesondere endlichen Atomaten



und Operationen darauf

z.B. Komposition



Rahmen solcher Ausdrucksmittel

Compiler braucht Keller

Chomsky-Sprachen

berechenbare Funktionen

„Turingmaschine ist adäquate Abstraktion für Computer

Der Außenseiter

C.A. Petri (1961) fragt:
Wie kann man eine allgemein rekursive Funktion *f* implementieren?

Die Schwierigkeit dabei:
... *für beliebige n:*
wie viele Ressourcen braucht man, um f(n) zu berechnen?
ist nicht beantwortbar

Also geht nicht: Erst die nötigen Ressourcen besorgen, dann losrechnen.

~~erster Lösungsvorschlag~~

Petri: es geht besser!

mit vorhandenen Ressourcen losrechnen.

Wenn sie reichen: Glück gehabt.

Wenn nicht: Mehr Ressourcen besorgen~~, das ganze von vorn~~.
anbauen, und weiterrechnen.

So oft wiederholen, bis die Ressourcen reichen.

Wenn *f(n)* überhaupt definiert ist, klappt das irgendwann.

Problem dabei: Wie baut man Ressourcen an?

... muß auch nach -zig Schritten noch möglich sein!

Anbauen braucht immer mehr Platz

Drähte werden immer länger

Signale brauchen immer mehr Zeit

fan-out wächst

Taktzeiten sind bald nicht mehr haltbar

wachsender fan-out verlangt mehr Strom

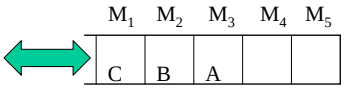
Das bricht mal zusammen!

Idee: Verwende keine langen Leitungen! Vergrößere den fan-out nicht! Baue „rein lokal“ an!

Das klappt tatsächlich!

Beispiel: *Keller*

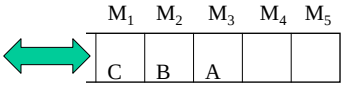
typische Konfiguration eines Kellers



M_i speichert *einen* Wert: *„Ruhezustand“*

Der Wert *D* wird in den Keller gelegt

push(D)



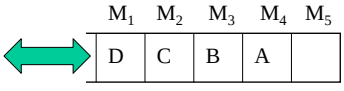
M_i speichert *zwei* Werte:

gibt den älteren nach rechts

Der letzte Modul verliert seinen älteren Wert

Der Wert *D* von M_1 wird entnommen

pop

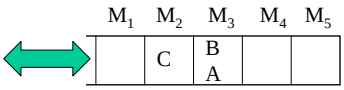


M_i speichert *keinen* Wert:

nimmt den von rechts

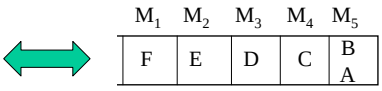
Der letzte Modul erzeugt sich ein .

Konfigurationen mit 0, 1, 2 Werten



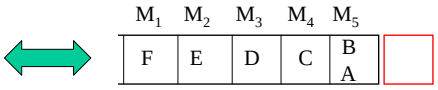
können durchaus entstehen

Problem



A droht herauszufallen!

Idee: Rechts anbauen!



und alten Wert in die neue Zelle schieben

lokal möglich

keine langen Leitungen!

kein größerer fan-out!

Der Preis dafür

Das System wird zwangsläufig asynchron!

damit:
asynchrone Algorithmen können mehr als synchrone.

… und wie modelliert man so was?

der Beginn der Petrinetz-Theorie!

Carl Adam Petri

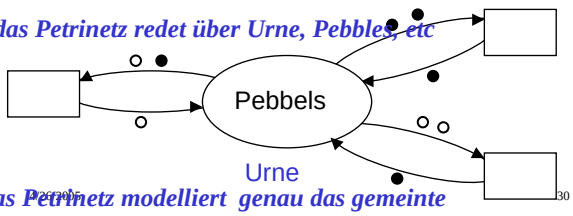
zurück zu Dijkstra‘s Problem

- was man alles machen soll:
- *take two pebbles out*
- *look at the colours*
- *return the white one*
- *paint a pebble black*

wo kommen diese Tätigkeiten vor im Modell?
nirgends

als Petrinetz:

das Petrinetz redet über Urne, Pebbles, etc



das Petrinetz modelliert genau das gemeinte

4. Modellieren verteilter und reaktiver Systeme

1975 – 1985

verschiedenste Modelle

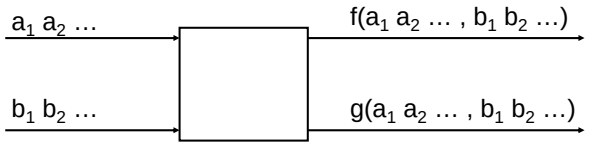
Datenfluss, Prozessalgebren, synchrone Sprachen, (Bi-) Simulation, Statecharts, ESTEL ESTEREL

Milner, Hoare, Berrie

z.B. Datenflussdiagramme

J. Dennis, MIT

G. Kahn, INRIA

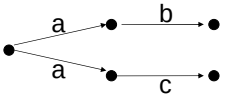


M. Broy, TUM

Prozessalgebren

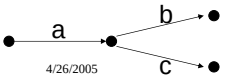
Robin Milner *CCS*

Beispiel: Automat mit Knöpfen a, b, c



Sprache {ab, ac}

das selbe System wie

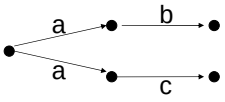


???

NEIN !!

Robin Milner *CCS*

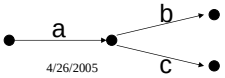
Beispiel: Automat mit Knöpfen a, b, c



Sprache {ab, ac}

… eine Sensation! im herkömmlichen Rahmen nicht formulierbar!

das selbe System wie



???

dieses ist *liberaler*

wählen zwischen b und c passiert später!

Weiteres

CSP

Verteilte Algorithmen

parallele Programme

Selbststabilisierung

Netzwerkalgorithmen

synchrone Sprachen ESTELLE ESREREL

Statecharts

…

5. Temporale Logik

seit 1980

Modale Logik *Prior 1962*

Operatoren für Aussagen p:
„es ist möglich, dass p gilt“ : ◇p
„es ist sicher, dass p gilt“ : □p

Operatoren für Tätigkeiten p:
„man darf p tun“ : ◇p
„man muss p tun“ : □p

zusammen mit Aussagenlogik gelten Gesetze:
¬◇¬p gdw □p
◇(p∨q) gdw ◇p ∨ ◇q

Kripke und Hintikka

Operatoren für Zustands-Eigenschaften in Transitionssystemen

„p gilt irgendwann“ : ◇p
„p gilt immer“ : □p

temporale Logik

zusammen mit Aussagenlogik gelten Gesetze:
¬◇¬p gdw □p
◇(p∨q) gdw ◇p ∨ ◇q

Pnueli 1979: prima zu brauchen!

Kripke-Strukturen sind die angemessene Abstraktion reaktiver Systeme,
(insbesondere unendlich lange Abläufe).

„Wann immer ein Prozess kritisch werden will,
wird er irgendwann kritisch“ :
□ (*pending* → ◇*critical*)

…wurde eine besonders eindrucksvolle Erfolgsstory “

model checking
zum verifizieren
zum systematisch Fehler finden
…liefert im Fehlerfall ein Gegenbeispiel

Temporale Logik zum Modellieren

Aufpassen: „linear time“ vs „branching time“

Sicherheit: es passiert nie was schlechtes.
Lebendigkeit: es passiert mal was gutes.

Theorem (Alpern, Schneider)
Jede Eigenschaft ist komponierbar aus Lebendigkeits- und Sicherheitseigenschaften

prominentes Beispiel:
klassische *Partial Correctness* und *Termination*

Lamport: TLA

Idee: Wertzuweisung *x:= x+1* logisch beschreiben:
spendiere zu jeder Variablen noch eine: zu *x* auch *x‘* .
x: Wert im gegebenen Zustand
x‘: Wert im nächsten Zustand
aus *x:= x+1* wird *x‘ = x+1*
!Jetzt kann man rechnen! z.B. *x‘-1 = x*
hingegen sinnlos: *x-1 := x*

Verfeinerung und Kompositionalität

Abadi, Lamport:

Implementierung (Verfeinerung) ist Implikation

Komposition ist Konjunktion

TLA
FOCUS

!Gerade wenn‘s groß wird, muss es einfach sein!

6. Logik-basiertes Modellieren *seit 1975*

Algebraische Spezifikationen, VDL, Z,
LARCH, Focus, ASM

Modell

im Sinne der Modelltheorie der Logik:

schreibe logische Formal hin
„gemeint“ ist alles, was diese Formel erfüllt

jede Struktur, die

Algebraische Spezifikation

Tarski:
Signatur: Menge von Symbolen,
jedes mit einer „Stelligkeit“.
Konstruiere *Terme* z.B. *f(g(a), b)*
und *Formeln* z.B. $\forall x \exists y\ f(g(x), y)$
Interpretiere Terme und Formeln über *Strukturen*
z.B. $\mathbb{Z} =_{def} (\mathbb{Z}, 0, -, +)$

diese Idee herrichten für große Systeme

statische Strukturen:

Algebraische Spezifikationen

VDM, Z, LARCH, RAISE, FOCUS

dynamische Strukturen:

ASM

ASM: Die Idee der „Semantik“

klassische Semantik von Programmiersprachen:

operationell: mathematische Zustandsräume,
denotationell: Funktionen,
logisch: Prädikate

mühselig, artifiziell, unintuitiv

Semantik von Modellierungssprachen:
???

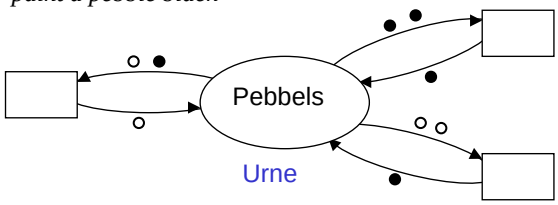
Gurevich: von der Logik lernen!

noch mal Dijkstra‘s Beispiel

was man alles machen soll:
- *take two pebbles out*
- *look at the colours*
- *return the white one*
- *paint a pebble black*

Vorschlag:
genau das symbolisch beschreiben!

als Petrinetz:



ASM machen’s entsprechend mit Wertzuweisung

7. Requirements Modellieren
seit 1980

SA SADT ER Jackson RAISE LARCH MSC ….

aktuell: UML

8. Programmieren mit Modellen
seit 2000

zur Erinnerung : Modellieren

„Aspekte der realen Welt darstellen wie man will und dann von einem Rechner ausführen lassen“

Wie darstellen?
bequem, intuitiv, selbsterklärend, ...

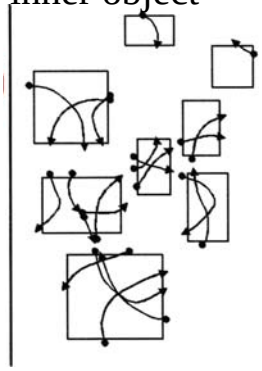
Ansätze dazu

Gurevich:
ASM : Ein Symbol wird in der realen Welt interpretiert

Lamport:
TLA: Logik-Spezifikation verfeinern bis hin zu Gleichungen der Gestalt $x' = f(x,y)$.
Die implementieren als $x := f(x,y)$.

... Modelle von Szenarien interaktiv herstellen:
David Harell:
„Come let‘s play“

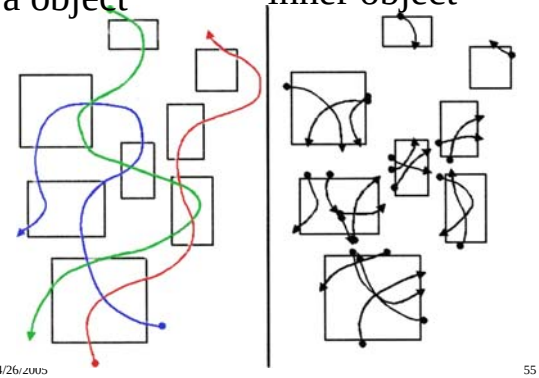
Szenarien
inner object



Szenarien

intra object

inner object



Beispiel: eine Schalttafel

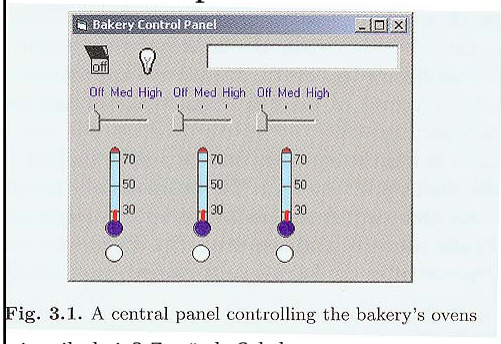


Fig. 3.1. A central panel controlling the bakery's ovens

beschrieben in einer speziellen Sprache

jeweils drei 3-Zustände-Schalter,
Thermometer,
Warnleuchten.

Ende

50 Jahre modellbasierter Entwurf:

Vom Modellieren mit Programmen zum Programmieren mit Modellen

Ringvorlesung
HU Institut für Informatik
Sommersemester 2005
W. Reisig

danke für Ihr Interesse

Zusammenfassung

Von Beginn an wurden in der Informatik Systeme modelliert; zunächst informell , mit Pseudocode oder mit Programmen. Im Laufe von fast 50 Jahren wurde das Modellieren zentraler Bestandteil des Systementwurfs; in der industriellen Praxis derzeit beispielsweise häufig auf der Basis der UML.

In der Vorlesung wird die Entwicklung einer Reihe verschiedener Modellierungstechniken nachgezeichnet, insbesondere Techniken für verteilte und reaktive Systeme.