from: ITG/GI-Fachtagung Entwurfsmethodik für Integrierte Schaltungen und Systeme, Darmstadt, 10.-11.2.1992

Schaltungseditor mit autmatisierter grafischer Visualisierung der textuellen (System)-Beschreibung

Peter Lüders
Technische Universität Braunschweig

Kurzfassung:

Mit den immer ausgereifteren Werkzeugen für die Systementwicklung und der fortschreitenden Technologie wird der Produktzyklus eines neuen Systems immer kürzer und der Faktor "time to market" wird immer mehr zum ausschlaggebenden Parameter für den Erfolg eines neuen Systems. Dagegen wird die Entwicklung eines neuen Systems immer schwieriger, bedingt einerseits durch die Benutzung verschiedener Werkzeuge auf unterschiedlichen Abstraktionsebenen und andererseits durch die ständig wachsende Komplexität der Systeme. Hier sind vor allem die reaktiven Systeme, charakterisiert durch ihre mannigfachen Interaktionen mit der Systemumgebung und ihrer starken Parallelität zu nennen. Weiterhin erschweren häufig zusätzliche Anforderungen nach Zuverlässigkeit und Echtzeitverhalten das Systemdesign.

Die Problematik der Werkzeugvielfalt wird mit der Entwicklung des Frameworks adressiert. Die Handhabung der verschiedenen Werkzeuge wird für den Entwickler vereinfacht. Unter einer grafischen Fensteroberfläche ist die Bedienung der Werkzeuge konsistent, eine Versionskontrolle wird automatisch durchgeführt und in großen Designs wird die Teamarbeit weitgehend unterstützt (Stichwort: CSCW), um nur einige wesentliche Punkte zu nennen.

Die Entwicklung komplexer Systeme und damit die Handhabung großer Mengen von Designdaten wird zwar durch ein Framework vereinfacht, es stellt dem Entwickler jedoch keine Funktionen für eine adäquate Darstellung der zur Zeit interessierenden Designdaten zur Verfügung. Er muß die gewünschte Darstellung der Information mühsam durch Öffnen, Schließen, Verschieben, Vergrößern oder Verkleinern von einzelnen Fenstern aufbauen; eine zeitraubende Arbeit, die ihn von der eigentlichen Entwicklungsproblematik ablenkt.

Der Einsatz einer geeigneten Benutzeroberfläche mit Schwerpunkt auf der Steuerung der Informationsdarstellung, der als Schnittstelle zwischen dem Framework und dem Entwickler fungiert, kann die Effizienz und damit die Produktiviät des Entwicklers steigern. Als wesentliche Problemstellungen sind mit den vorherigen Ausführungen zu nennen:

- Die Implementierung eines *Informationsfilters*. Die Auswahl der zur Zeit relevanten Information muß interaktiv durch den Entwickler steuerbar sein. Beispielsweise muß der Entwickler die Möglichkeit haben, in einem Schematic

sich *nur* den kritischen Pfad oder in einem Zustandsdiagramm sich *nur* Zustände mit speziellen Nachbarschaftsbeziehungen darstellen zu lassen. In den gegebenen Beispielen ändert sich - bedingt durch den Informationsfilter - die Struktur der dargestellten grafischen Information, die Grafik selbst darf daher nicht statisch sein (damit würden nur einfache Zoom- und Colorierungsfunktionen möglich sein), sondern sie muß sich dynamisch an den Informationsfilter anpassen.

→ Es wird daher ein *automatisches Layout* benötigt, das den Aufbau der grafischen Information auf dem Bildschirm automatisch berechnet. Hiermit wird sowohl die Verteilung und Größe der Fenster auf der Grafikoberfläche als auch die Berechnung der Darstellung von beispielsweise Flußgraphen oder Automatengraphen innerhalb eines Fensters angesprochen.

Der Informationsfilter wird als ein eher applikationsspezifisches Problem der Datenbanktechnik angesehen und ist bereits anderweitig ausreichend untersucht worden. In unserem Projekt beschäftigen wir uns vor allem mit der letztgenannten Problematik, der Berechnung des Bildschirmlayouts. Zur Untersuchung der Anforderungen, die an einen Schaltkreiseditor in Hinsicht auf das Bildschirmlayout gestellt werden müssen, haben wir das Teilproblem der Systemspezifikation herausgegriffen und in einem ersten Schritt einen Editor zur Beschreibung von Endlichen Automaten implementiert. Hier spezifiziert der Benutzer in einem Textfenster den Automaten mit einer rein textuellen Sprache. Diese Spezifikation wird automatisch in eine grafische Representation des Automaten umgesetzt und in einem Grafikfenster dargestellt.

Bei der genaueren Betrachtung des Layoutproblems, d.h. hier der Plazierung von miteinander verbundenen Knoten unterschiedlicher Größe auf einer vorgegebenen Fläche, erkennt man die Ähnlichkeit zur Layoutproblematik aus dem VLSI-Layout (Stichwort Macro-Cell, Floorplanning). In unserem Ansatz übernehmen wir Algorithmen aus dem VLSI-Layout, passen sie unserer Problemstellung an und untersuchen sie auf ihre Leistungsfähigkeit für den Einsatz in einem interaktiven Editor.

Es zeigt sich, daß kein einzelner Algorithmus allein das Layoutproblem in einem Schaltkreiseditor lösen kann, eine Aufspaltung des Gesamtlayouts in die drei Schritte *globales Layout*, *lokales Layout* und *Postprocessing* jedoch sehr vielversprechend ist.

Die in den Untersuchungen erzielten Ergebnisse für die Layoutalgorithmen könnten dann wiederum in die Steuerung der gesamten Fensteroberfläche eines Schaltkreiseditors einfließen. Damit stände dem Entwickler eine Oberfläche zur Verfügung, die ihn bei der Validierung des von ihm spezifizierten Systems unterstützen und ihm damit helfen könnte, Fehler in frühen Schritten des Designprozesses zu finden und damit zeitraubende Redesigns zu vermeiden.

Schaltungseditor mit automatisierter, grafischer Visualisierung der textuellen (System-)Beschreibung

P. Lüders
R. Ernst

Institut für Datenverarbeitungsanlagen
Technische Universität Braunschweig

Problem:

- Steigende Komplexität der Systeme
  Besondere Problematik: Reaktive Systeme

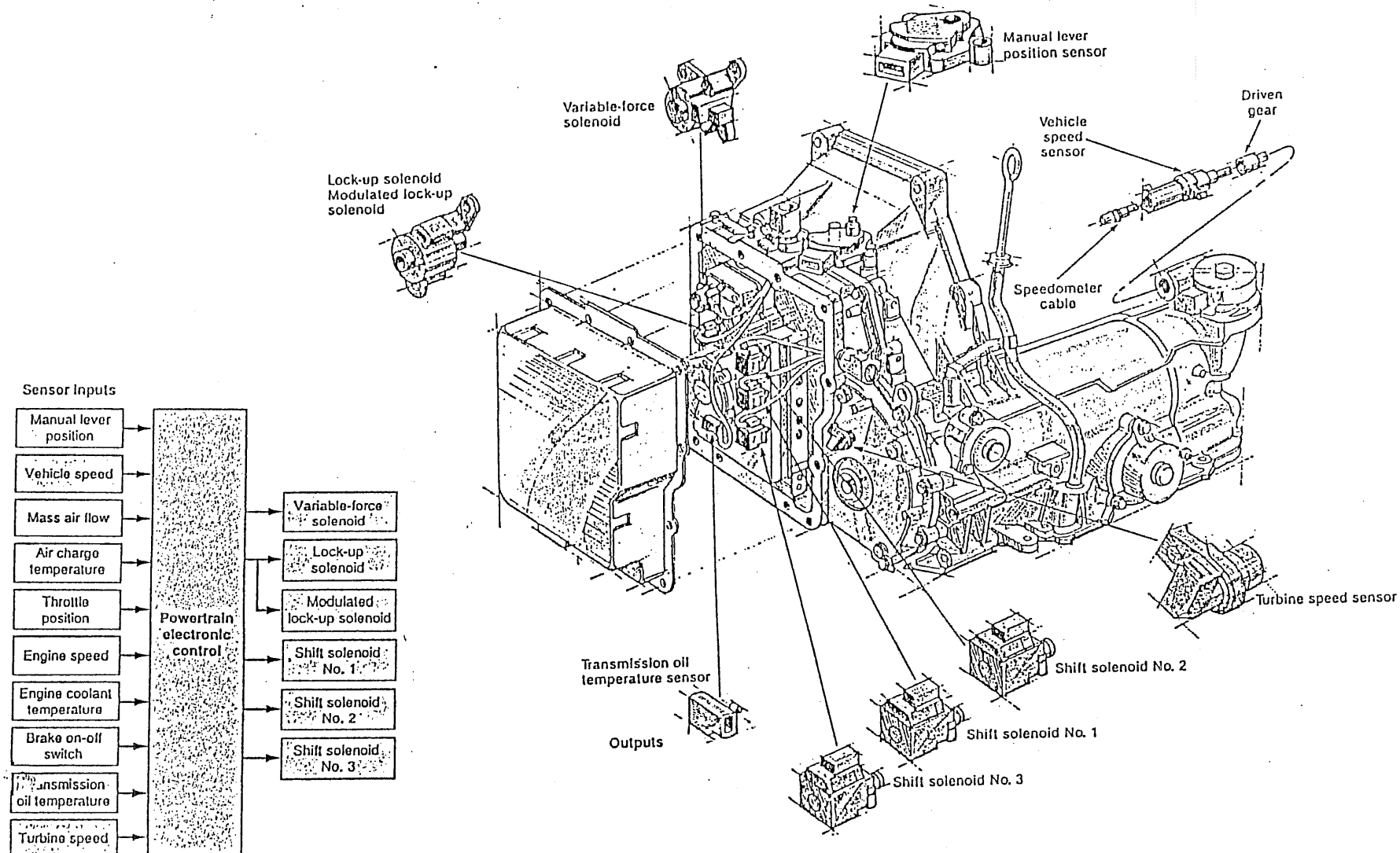
  Reaktive Systeme: Intensive Interaktion mit der Systemumgebung, starke Parallelität, Echtzeit, häufig Zuverlässigkeit.

- Große Anzahl von Design-Werkzeugen auf unterschiedlichen Abstraktionsebenen

Ziel:

Werkzeug zur textuell-grafischen Spezifikation

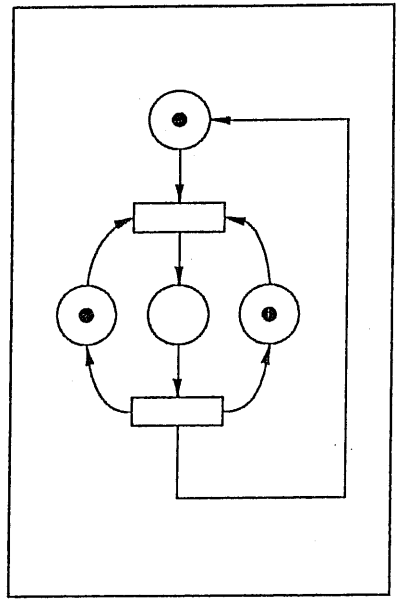
- Befreiung vom grafischen Edieren
- Unterstützung bei der Validierung des Systems

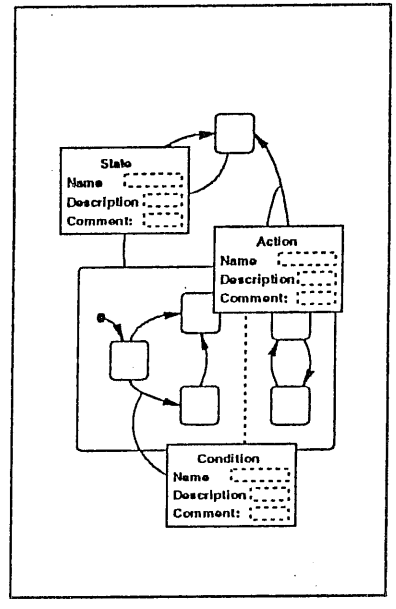


Quelle: IEEE Spectrum, Dec. 1990

Beispiel für den Einsatz eines reaktiven Systems

```
      .
      .
      .
proc: process
begin
  wait on a = '0';
  ctrl <= '1' after 10 ns;
  sig  <= '0' after 7 ns;
  if input = '1' then
    b <= '1' after 2 ns;
  else
    b <= '0' after 4 ns;
  endif
end
end process;
      .
      .
      .
```





Textuelle Spez.

Beispiel: *VHDL*

Grafische Spez.

Beispiel: *Petri-Netze*

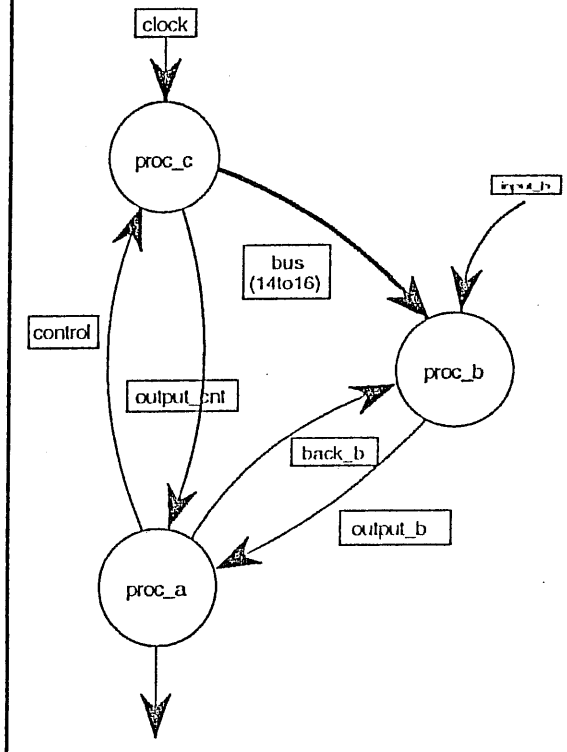
Grafisch-textuelle Spez.

Beispiel: *State Charts*

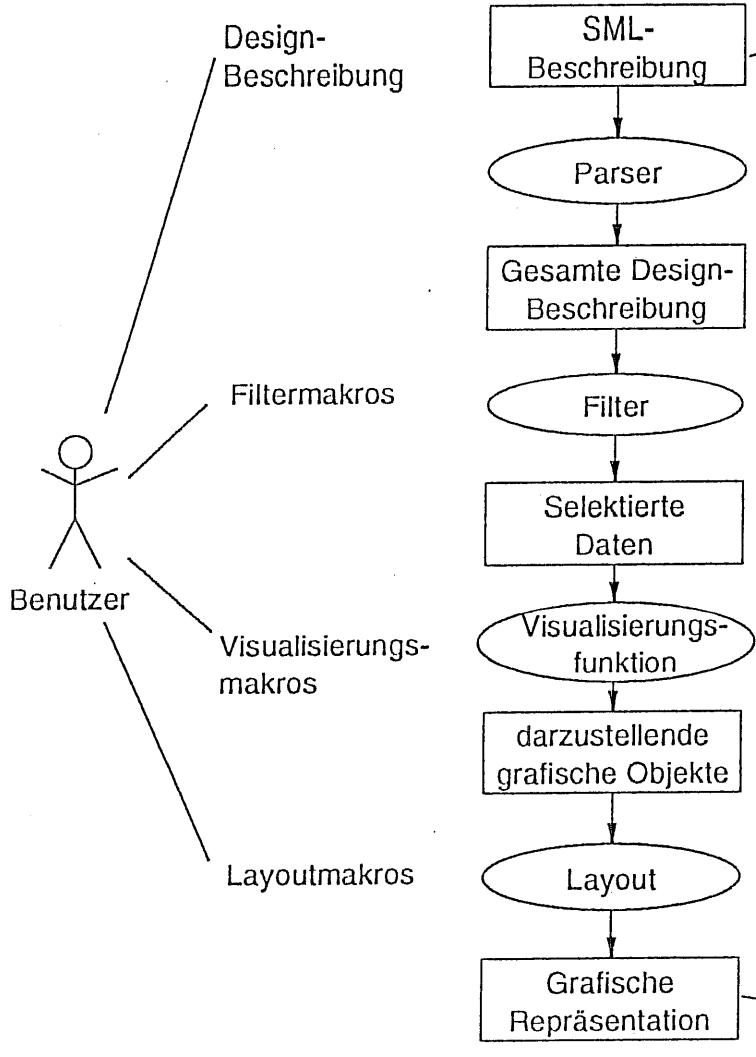
File: VHDL_Description

```
proc_c: process
begin
  wait on clock;
  bus(14 to 16) <= "101" after 5 ns;
  output_cnt <= '1' after 7 ns;
end
end process proc_c;

proc_b: process
constant a:bit_vector(0 to 2):="101";
begin
  wait on a = bus (14 to 16);
  if input_b = '1' then
    output_b <= '0' after 2 ns;
  else
    output_b <= '1' after 4 ns;
  endif
end
end process proc_b;

proc_a: process
begin
  wait on output_b = '0';
  control <= '1' after 7 ns;
  back_b <= '0' after 2 ns;
  last <= '1' after 5 ns;
end
end process proc_a;
```

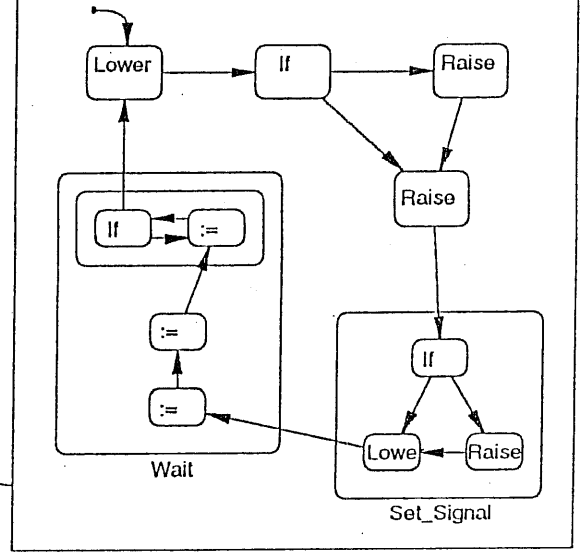
actual_process: display_1



Problemstellung	Auswirkung	Ansatz
Komplexität der Systembeschreibung	unübersichtlich Darstellung, hohe Rechenzeit	Informationsfilter (bedingt Automatisches Layout)
Topologische Inkonsistenz	ermüdendes und frustrierendes Arbeiten, nicht akzeptabel	Berücksichtigung alter Graphdarstellungen
Interaktives Softwaresystem	Antwortzeit auf Benutzeraktionen im Sekundenbereich	Beschränkung der dargestellten Information (Filter)
Visualisierung der Graphdarstellung	muß verständlich sein, sonst nicht akzeptabel	Regeln für Layoutalgorithmen




```
PROCEDURE WAIT
  SUM = 0
  MAX = 100
B:  IF (SUM>MAX) THEN SUM = SUM+1
      ELSE GOTO A: ENDIF
A: ENDROC

PROCEDURE SET_SIGNAL
  IF ( SIG_A = '1' ) THEN RAISE (SIG_B) ENDIF
  LOWER (SIG_C)
ENDPROC

/* Hauptschleife */
LOOP
  LOWER (SIG_B)
  IF ( TASTE = '1' ) THAN RAISE (ERR) ENDIF
  RAISE (SIG_A)
  SET_SIGNAL
  WAIT
ENDLOOP
```



Algorithmen zur Visualisierung von graphstrukturierten Daten

bislang:
- Graphlayoutalgorithmen (z.B. Sugiyama et.al.)
  - zumeist für azyklische Graphen
  - deterministisch, heuristisch (=> schnell)

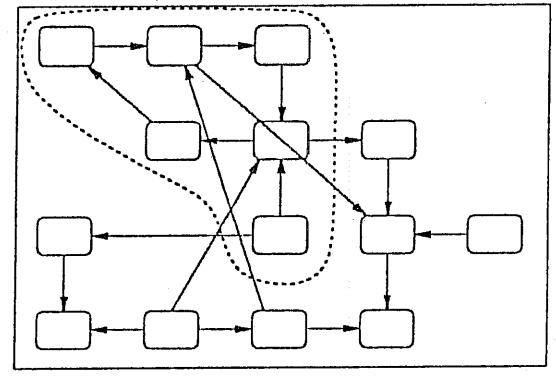
Ansatz:
- Algorithmen aus dem VLSI-Layout
  - implementiert: Simulated Annealing, Genetischer Algorithmus
    Min-Cut, Force-Directed-Placement
  - basieren auf einer Kostenfunktion, die eine Bewertung der "Güte" der Graphdarstellung berechnet.

=> wesentliche Fragestellung: Was ist "Güte"?

Elemente der Bewertungsfunktion objektivieren Eigenschaften eines "guten" Layouts

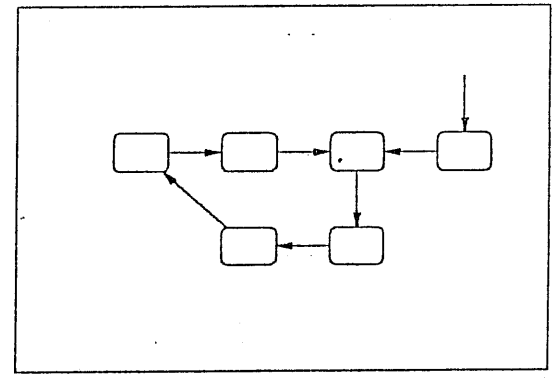
statisch, Einzelbild

- Kantenlänge ↓
- Kantenkreuzungen ↓
- Sichtbarkeit ↑
- homogene Verteilung ↑



dynamisch, Bildsequenz

- Knotenplazierung (Ändg.) ↓
- Kantenrichtung (Ändg.) ↓
- Visualisierung des Graphen (Ändg.) ↓



Algorithmus:

- *Simulated Annealing*

Arbeitsweise:

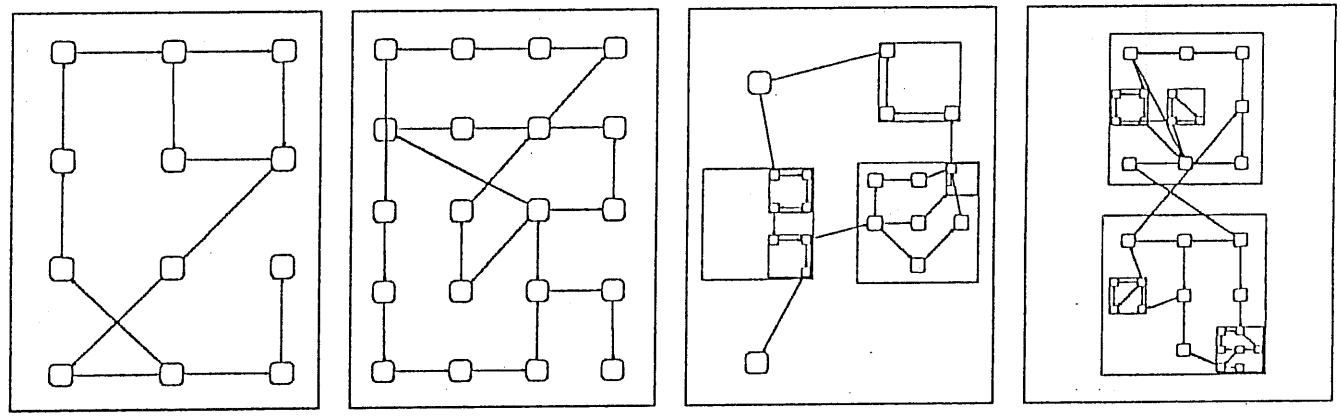
- Nachbildung des Kristallisierungsprozesses beim Erkalten von Metallen

Eigenschaften:

- stochastisch (hill-climbing)
- ein Layout wird iterativ verbessert

Anmerkungen:

- gleichzeitiges Layout aller Hierarchieebenen
- große Anzahl der Iterationsschritte
- starker Einfluß der Kostenfunktion auf Rechenzeit



Knoten / Kanten	12 / 13	20 / 22	26 / 22	37 / 39
Rechenzeit	1,92 s	3,58 s	1,95 s	3,38 s
Layouts	2366	6810	13088	20171
Kantenlänge	2510	4116	6028	4109
Kantenkreuzungen	1	1	0	3

Algorithmus:

- *Genetischer Algorithmus*

Arbeitsweise:

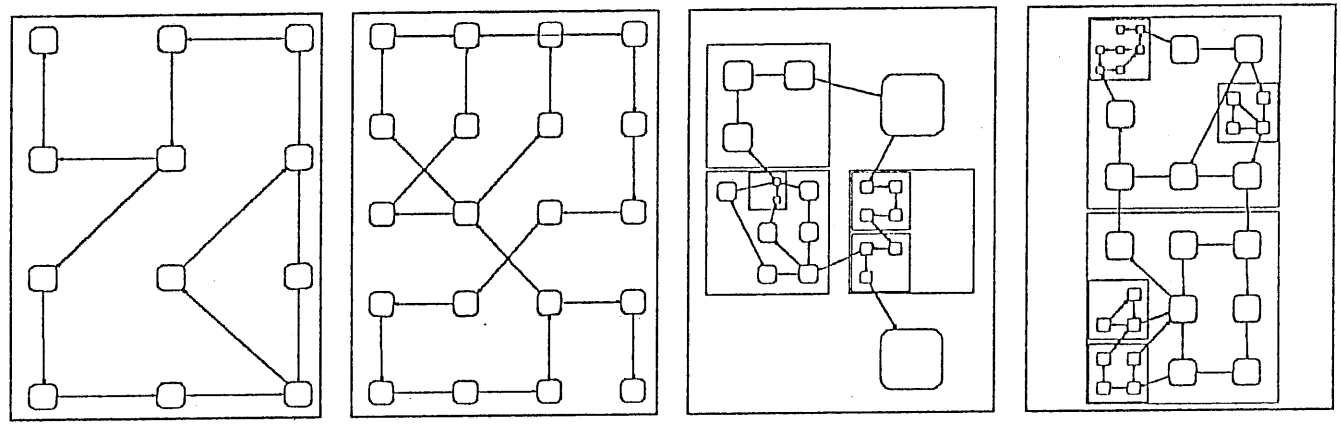
- Nachbildung der biologischen Evolution durch Kreuzung und Mutation

Eigenschaften:

- stochastisch (hill-climbing)
- eine Population wird iterativ verbessert
- parallelisierbar

Anmerkungen:

- gleichzeitige Berechnung aller Hierarchieebenen
- geringe Anzahl von untersuchten Layouts (relativ zum Simulated Annealing)



Knoten / Kanten	12 / 13	20 / 22	26 / 22	37 / 39
Rechenzeit	3,48 s	4,07 s	5,77 s	8, 65 s
Population / Gen.	25 / 100	25 / 100	25 / 100	25 / 100
Kantenlänge	2763	4001	2479	3425
Kantenkreuzungen	0	2	0	4

Algorithmus:

- *Look-up-table*

Arbeitsweise:

- Auswahl der Plazierung aus einer Look-up-table anhand der Charakeristik des Graphen

Eigenschaften:

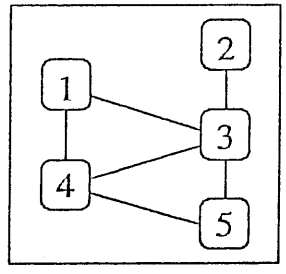
- deterministisch

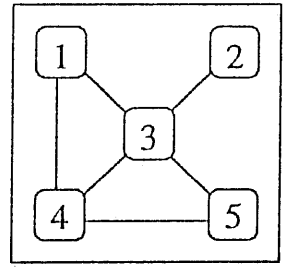
Anmerkungen:

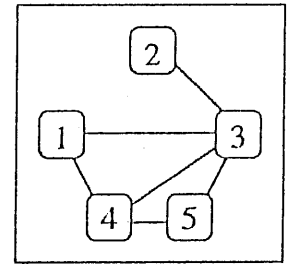
- kein Raster => kein orthogonales Layout
- subjektives Empfinden der interaktiven Zuordnung
- schnell
- (nahezu) unabhängig vom Plazierungsraster
- pro Notation mehrere Plazierungen möglich
- geeignet für kleine Graphen (7)

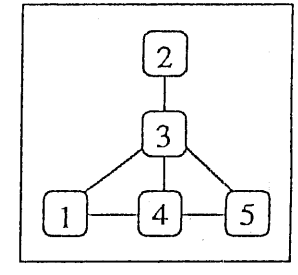
Knoten 1: 2 Kanten
Knoten 2: 1 Kanten
Knoten 3: 4 Kanten
Knoten 4: 3 Kanten
Knoten 5: 2 Kanten

Notation: " 1 2 2 3 4 "









Knoten / Kanten	5 / 6
Rechenzeit (1)	8 ms
Rechenzeit (2)	2 ms

Algorithmus:

- *Nichtlinearer, kräftegerichteter Algorithmus*

Arbeitsweise:

- Einführung von Kräften (Federn) zwischen den grafischen Elementen, Federkräfte bewirken Verbesserung des Layouts

Eigenschaften:

- deterministisch
- ein Layout wird iterativ verbessert
- beliebige Größen der Objekte möglich
- kein / sehr feines Grid-Raster

Anmerkungen:

- Iteration kann dem Benutzer grafisch dargestellt werden ("natürlicher Fluß von Objekten")

Berechnung eines Graphlayouts, Beispiel:

Interne Datenbasis

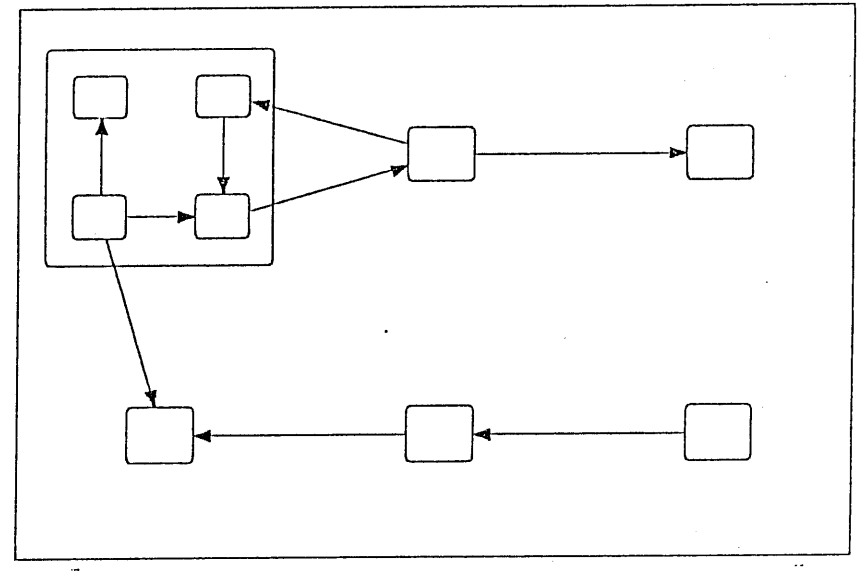
- Hierarchischer, gerichteter und zyklischer Graph
- Geringe Anzahl von graphischen Elementen
- Variable Größe der Knoten

Stochastische Anfangsplazierung

Globales Layout mit *Simulated Annealing*

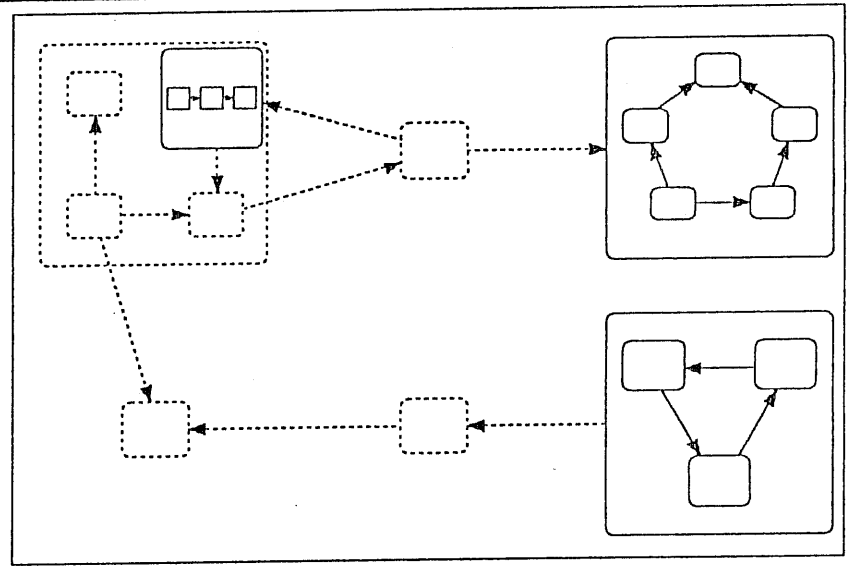
- gleichzeitige Berechnung der hierarchischen Ebenen
- Objekte der gleichen Größe in gleicher Last
- grobes Raster => orthogonale Plazierung

  (auch Genetischer Algorithmus, Min-Cut)



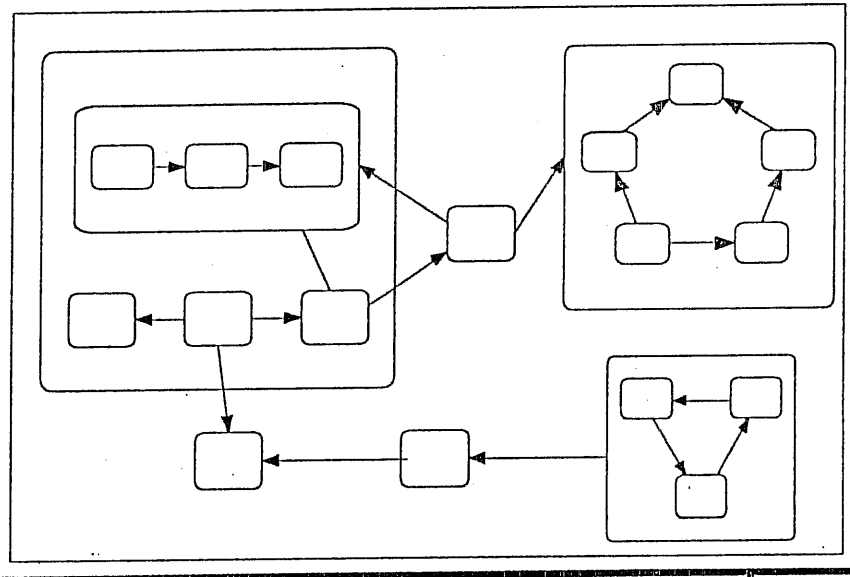
Lokales Layout mit *Look-up-table* :

- Verbundene Knotengruppen werden plaziert
- Deterministischer Algorithmus



Nachbearbeitung mit nicht-linearer, kräftegerichteter Plazierung

- Eindruck des "natürlichen Flusses" von Objekten
- Benutzung des Bildschirms



Zusammenfassung

- Schaltungseditor
- Informationsfilter
- Automatisches Layout
- Layoutalgorithmen aus VLSI-Layout
- Berechnung des Layouts in drei Phasen:
  - Globales Layout (Simulated Annealing, Genetischer Algorithmus)
  - Lokales Layout (Look-up-table)
  - Nachbearbeitung (Kraftgerichteter Algorithmus)

Weitere Schritte

- Applikationen: Hypertext, Multimedia,
  VHDL, Schematic Generation
  CASE - Tools

- Untersuchung des dynamischen Anteils der Kostenfunktion

- Routing Algorithmen

Textuelle System-
beschreibung

Dialogfenster

Ergebnisausgabe

Zweite grafische
Sicht

Grafische Anzeige