



TU Seminare Modellbildung und Simulation

Seminarunterlagen Course Material

EUROSIM'95 Seminare

- **Simulation in der Didaktik
13.9.1995**
- **COMETT Course Fuzzy Systems and
Control, 7.9.-8.9.1995**
- **COMETT Course OO Discrete Simulation,
28.6.1995**

**Editors: F. Breitenecker, I. Husinsky, N. Kraus, D.P.F. Möller,
D. Murray-Smith, M. Salzmann**

**ARGESIM Report AR 4 / AR 5 / AR 6
ISBN ebook 978-3-901608-71-1
DOI 10.11128/arep.04-05-06**

© 1995 ARGESIM

ARGESIM Report AR 4 / AR 5 / AR 6

ISBN ebook 978-3-901608-71-1

DOI 10.11128/arep.04-05-06

ARGE Simulation News (ARGESIM)
c/o Technical University of Vienna
Wiedner Hauptstr. 8-10
A-1040 Vienna, Austria
Tel: +43-1-58801 5386, 5374, 5484
Fax: +43-1-5874211
Email: argesim@simserv.tuwien.ac.at
WWW: <[URL:http://eurosim.tuwien.ac.at/](http://eurosim.tuwien.ac.at/)>

Seminare über Modellbildung und Simulation

Seit dem Frühjahr 1991 veranstaltet das EDV-Zentrum gemeinsam mit der Abteilung Regelungsmathematik und Simulationstechnik des Instituts für Technische Mathematik und der ARGE Simulation News (ARGESIM) Vortragsveranstaltungen zum Thema Modellbildung und Simulation (Simulationsseminare). Das Ziel ist, verschiedene Simulationswerkzeuge vorzustellen, über ihre Einsatzmöglichkeiten zu informieren und Erfahrungen auszutauschen. Ferner werden bekannte Simulationsfachleute eingeladen, Grundsatzvorträge zum Thema Simulation zu halten.

Im Vorlauf des Kongresses EUROSIM'95, der im September 1995 an der TU Wien stattfand, ergab sich die Möglichkeit zu drei Seminaren mit internationalen Vortragenden:

- Simulation in der Didaktik, 13.9.1995
- COMETT Course Fuzzy Systems and Control, 7.9.-8.9.1995
- COMETT Course OO Discrete Simulation, 28.6.1995

Die Unterlagen zu diesen Seminaren sind in den ARGESIM Reports AR 4, AR 5 und AR 6 zusammengefasst und als Sammelband mit ISBN veröffentlicht.



ARGESIM Report no. 4

Simulation in der Didaktik

Seminar Modellbildung und Simulation EUROSIM'95 Seminar

F. Breiteneker, I. Husinsky, M. Salzmann

in ISBN ebook 978-3-901608-04-9 (3-901608-04-4) DOI 10.11128/arep.4-5-6.ar4

© 1995 ARGESIM

in ISBN ebook 978-3-901608-04-9 (3-901608-04-4)
DOI 10.11128/arep.4-5-6.ar4

ARGESIM Report No. 4

ARGE Simulation News (ARGESIM)
c/o Technical University of Vienna
Wiedner Hauptstr. 8-10
A-1040 Vienna, Austria
Tel: +43-1-58801 5386, 5374, 5484
Fax: +43-1-5874211
Email: argesim@simserv.tuwien.ac.at
WWW: <URL:<http://eurosim.tuwien.ac.at/>>

VORWORT

SIMULATION ist die Nachbildung eines dynamischen Prozesses in einem Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind.

Simulation ist ein Verfahren zur Lösung eines Problems, das in einem Prozeß auftaucht bzw. dessen Auftauchen in einem geplanten Prozeß erwartet wird.

Mit vorhandenen Daten, die den Ist-zustand eines Prozesses oder den Soll-Zustand eines geplanten Prozesses beschreiben, wird ein (mathematisches) Modell erstellt. Dieses Modell ist dann zunächst in der Lage, Vorgänge im realen Prozeß nachzuvollziehen bzw. Vorgaben für einen geplanten Prozeß zu erfüllen (MODELLBILDUNG).

In der Folge wird dann das Modell mit neuen Eingangsdaten versorgt. Am Modell können nun verschiedene Experimente durchgeführt werden, die das Verhalten des Modelles (und damit des Prozesses) unter anderen Bedingungen zeigen (EXPERIMENTE).

Ziel dieses Seminares ist die Vermittlung von Kenntnissen über Grundlagen in der Simulationstechnik, wobei eine Orientierung an aktuellen Themen im Vordergrund steht.

Der Stoff wird dabei für eine didaktische Darstellung für den Unterricht in AHS und BHS und für Einstiegslehrveranstaltungen am Beginn des ersten Studienabschnittes bearbeitet.

Fünf international anerkannte Fachleute auf dem Gebiet der Simulationstechnik werden zu folgenden Themenkreise vortragen:

- Beschreibung diskreter Prozesse mit Petrinetzen (u.a. für Signalverarbeitung, Ablaufplanung)
- Anwendung von Fuzzy Logic (Erweiterung der klassischen binären Logik durch „Unschärfe“, neue Sichtweise in der Regelungstechnik)
- Beschreibung paralleler Prozesse mit Hilfe von Petrinetzen
- Human Computer Interfaces (Man-in-the-Loop Simulation, Interfaces mit WWW, etc.)
- Diskrete Simulation

Modellbildung und Simulation in der Lehre

Mittwoch, 13. September 1995

Technische Universität Wien
Wiedner Hauptstr. 8-10, 1040 Wien
HS 2, 2. Stock, gelber Bereich

Ziel dieses Seminares, das in Verbindung mit dem Kongreß „EUROSIM'95“ stattfindet, ist die Vermittlung von Kenntnissen über das Erstellen dynamischer Modelle (ohne Kenntnis von Differentialgleichungen bzw. Warteschlangen) und Experimentieren mit diesen Modellen in einfachen Simulationsprogrammen. Der Stoff wird dabei für eine didaktische Darstellung für den Unterricht in AHS und BHS und für Einstiegslehrveranstaltungen am Beginn des ersten Studienabschnittes bearbeitet.

Programm

- 9⁰⁰ Begrüßung, Vorstellung Seminarreihe und Didaktiktag**
F. Breitenecker und M. Salzmann, TU Wien
- 9¹⁵ Modellbildung und Simulation mit Petri-Netzen**
A. Javor, Ungarische Akademie der Wissenschaft, Budapest
- 10⁴⁵ Kaffeepause**
- 11⁰⁰ Introduction to Fuzzy Logic (in Englisch)**
D. Murray-Smith, University of Glasgow
- 12³⁰ Mittagspause**
- 13⁰⁰ Nebenläufigkeit (concurrency) - ein alltägliches Phänomen (in Deutsch)**
H. Fuss, Gesellschaft für Mathematik und Datenverarbeitung, Bonn
- 14³⁰ Kaffeepause**
- 14⁴⁵ Human Computer Interface (HCI) (in Englisch)**
P.C.P. Bhatt, Indian Institute of Technology at Dehli, New Dehli
- 16¹⁵ Diskrete Simulation (in Deutsch)**
F. Breitenecker, Technische Universität Wien

Der Didaktiktag wird vom Bundesministerium für Unterricht und kulturelle Angelegenheiten gesponsert.

About ARGESIM

ARGE Simulation News (ARGESIM) is a non-profit working group providing the infra structure for the administration of **EUROSIM** activities and other activities in the area of modelling and simulation.

ARGESIM organizes and provides the infra structure for

- the production of the journal **EUROSIM Simulation News Europe**
- the comparison of simulation software (**EUROSIM Comparisons**)
- the organisation of seminars and courses on modelling and simulation
- **COMETT Courses on Simulation**
- "Seminare über Modellbildung und Simulation"
- development of simulation software, for instance: **mosis** - continuous parallel simulation, **D_SIM** - discrete simulation with Petri Nets, **GOMA** - optimization in ACSL
- running a WWW - server on **EUROSIM** activities and on activities of member societies of **EUROSIM**
- running a FTP-Server with software demos, for instance
 - * demos of continuous simulation software
 - * demos of discrete simulation software
 - * demos of engineering software tools
 - * full versions of tools developed within ARGESIM

At present ARGESIM consists mainly of staff members of the Dept. Simulation Technique and of the Computing Services of the Technical University Vienna.

In 1995 ARGESIM became also a publisher and started the series **ARGESIM Reports**. These reports will publish short monographs on new developments in modelling and simulation, course material for COMETT courses and other simulation courses, Proceedings for simulation conferences, summaries of the EUROSIM comparisons, etc.

Up to now the following reports have been published:

No.	Title	Authors / Editors	ISBN
# 1	Congress EUROSIM'95 - Late Paper Volume	F. Breiteneker, I. Husinsky	3-901608-01-X
# 2	Congress EUROSIM'95 - Session Software Products and Tools	F. Breiteneker, I. Husinsky	3-901608-01-X
# 3	EUROSIM'95 - Poster Book	F. Breiteneker, I. Husinsky	3-901608-01-X
# 4	Seminar Modellbildung und Simulation - Simulation in der Didaktik	F. Breiteneker, I. Husinsky, M. Salzmann	3-901608-04-4
# 5	Seminar Modellbildung und Simulation - COMETT - Course "Fuzzy Systems and Control"	D. Murray-Smith, D.P.F. Möller, F. Breiteneker	3-901608-04-4
# 6	Seminar Modellbildung und Simulation -COMETT - Course "Object-Oriented Discrete Simulation"	N. Kraus, F. Breiteneker	3-901608-04-4
# 7	EUROSIM Comparison 1 - Solutions and Results	F. Breiteneker, I. Husinsky	3-901608-07-9
# 8	EUROSIM Comparison 2 - Solutions and Results	F. Breiteneker, I. Husinsky	3-901608-07-9

For information contact:

ARGESIM, c/o Dept. Simulation Techniques,
attn. F. Breiteneker, Technical University Vienna
Wiedner Hauptstraße 8-10, A - 1040 Vienna
Tel. +43-1-58801-5374, -5386, -5484, Fax: +43-1-5874211
Email: argesim@simserv.tuwien.ac.at

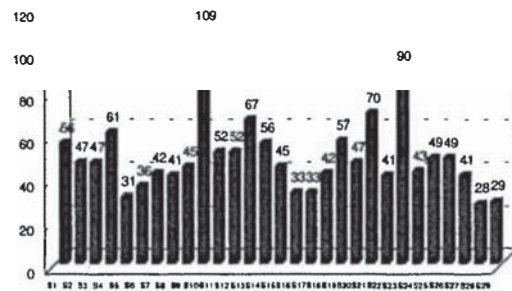
Seminare über Modellbildung und Simulation

Seit dem Frühjahr 1991 veranstaltet das EDV-Zentrum gemeinsam mit der Abteilung Regelungsmathematik und Simulationstechnik des Instituts für Technische Mathematik und der ARGE Simulation News (ARGESIM) Vortragsveranstaltungen zum Thema Modellbildung und Simulation (Simulationsseminare). Organisatoren sind I. Husinsky und F. Breitenecker. Das Ziel ist, verschiedene Simulationswerkzeuge vorzustellen, über ihre Einsatzmöglichkeiten zu informieren und Erfahrungen auszutauschen. Ferner werden bekannte Simulationsfachleute eingeladen, Grundsatzvorträge zum Thema Simulation zu halten. Im allgemeinen werden die Seminare teilweise von Firmen gesponsert oder über Simulationsprojekte mitfinanziert. Sie dauern einen halben oder einen Tag, es gibt schriftliche Unterlagen zu den Vorträgen und Softwareprodukten. Ein Buffet fördert die Kommunikation zwischen den Seminarteilnehmern in den Pausen.

Bis jetzt haben folgende Seminare stattgefunden:

S1	23. 4. 1991	ACSL
S2	4. 6. 1991	CTRL C XANALOG
S3	22. 10. 1991	SIMUL R
S4	5. 5. 1992	ACSL
S5	6. 5. 1992	MicroSaint
S6	17. 6. 1992	Objektorientierte Modellbeschreibung und qualitative Simulation (F. Cellier University of Arizona)
S7	1. 7. 1992	Diskrete Simulation und Analyse (D. Kelton, University of Minnesota)
S8	23. 10. 1992	GPSS/H (T. Schriber, University of Michigan)
S9	10. 12. 1992	SIMPLE
S10	2. 2. 1993	MATLAB und SIMULINK
S11	25. 3. 1993	Modellbildung mit Bondgraphen (D. Karnopp, University of California)
S12	24. 5. 1993	MicroSaint
S13	22. 6. 1993	ACSL
S14	21.10.1993	XANALOG SIMNON
S15	22.10.1993	GPSS/H (T. Schriber, University of Michigan)
S16	11.11.1993	IDAS
S17	7.12.1993	SIMPLE++
S18	14.12.1993	Petrinetze, D_SIM (R. Hohmann Magdeburg)
S19	4.2.1994	Modellbildung und Simulation in der Lehre
S20	14.3.1994	GPSS/H und Proof (T. Schriber, University of Michigan)
S21	13.4.1994	ACSL
S22	10.5.1994	SIMUL_R, Partielle Differentialgleichungen
S23	22. 11. 1994	MATLAB/SIMULINK
S24	14.12.1994	SIMPLE++
S25	31.1.1995	Parallele Simulation mosis
S26	28.3.1995	ACSL
S27	29.3.1995	MicroSaint
S28	13.6.1995	COMETT II, Part one, Discrete Simulation
S29	28.6.1995	COMETT II, Part two, Simulation and Automatisaton

Teilnehmer(angemeldet)



TU Wien
43.6%

andere Unis
11.2%

Die Teilnehmer, etwa 30 bis 110 je Seminar, kommen zum Großteil von der TU, aber auch von anderen Universitäten und aus der Industrie. Bei den bisherigen Seminaren waren etwa 20% der Teilnehmer aus der Industrie.

Das Programm eines Seminars setzt sich im allgemeinen aus einem oder zwei Grundlagenvorträgen, mehreren Anwendervorträgen, Produktpräsentationen, Vorführungen am Rechner und Diskussionen zusammen.

Die Teilnehmer werden um eine Anmeldung gebeten, daher können die Unterlagen (Seminarberichte), die zu Beginn des Seminars verteilt werden, schon eine Teilnehmerliste enthalten. Ab Herbst 1995 erscheinen die Unterlagen als ARGESIM Report. Alle, die bereits an einem Seminar teilgenommen haben, werden automatisch zu den weiteren Seminaren eingeladen.

Information:

I. Husinsky, EDV-Zentrum, Technische Universität Wien, Wiedner Hauptstr. 8-10, A-1040 Wien,
Tel: (0222) 58801 5484, Fax: (0222) 587 42 11,
E-Mail: husinsky@edvz.tuwien.ac.at

Prof.Dr. F. Breitenecker, Abt. Regelungsmathematik u. Simulationstechnik, Inst. 114, Technische Universität Wien, Wiedner Hauptstr. 8-10, A-1040 Wien,
Tel: (0222) 58801 5374, Fax: (0222) 587 42 11,
E-Mail: fbreiten@email.tuwien.ac.at

TABLE OF CONTENTS / INHALTSVERZEICHNIS

Vorwort	iii
About ARGESIM	iv
Seminare "Modellbildung und Simulation"	v
Modellbildung und Simulation mit Petrinetzen	1
Problem Organization for Control System Simulation	27
Simple Examples of Control System Simulation Using Common Simulation Tools	47
Was ist Fuzzy Logic	63
Systembeschreibung und - Simulation mit Petri - Netzen	71
Über Unmöglichkeiten, parallele Prozesse korrekt zu simulieren	79
Maximale Parallelverarbeitung und asynchrone Abläufe - oder: War der Golfkrieg unvermeidlich ?	85
Human Computer Interface	91
Diskrete Simulation	126

Simulation in der Didaktik, TU Wien, 13.9.1995, Teilnehmerliste

Dr.	Bayerl	Manfred	Büro für Technischen Umweltschutz		Morizgasse 4/2/4/14	A	- 1060 Wien
Mag.	Bierbaumer	Irma	BGR/BG	Albertgasse	Albertgasse 18-22	A	- 1080 Wien
Mag.	Dupor	Sanja	Resound Viennatone AG	R&D Department	Fröbelgasse 26-32	A	- 1164 Wien
Dipl.Ing.	Eder	Mathias	TU Wien	Inst. f. El. Maschinen	Gußhausstr. 27-29	A	- 1040 Wien eder@eemvs1.una.ac.at
Mag.	Fiala	Friedrich	BG/BRG	Wien 12	Rosasg. 1-3	A	- 1120 Wien
Dipl.Ing.	Fürst	Walter	Forstliche Bundesversuchsanstalt		Seckendorff-Gudent Weg 8	A	- 1131 Wien rjandl@email.tuwien.ac.at
Dipl.Ing.	Heil	Ernst	TU Wien	Inst.f. Allg. Maschinenlehre	Getreidemarkt 9	A	- 1060 Wien
Dipl.Ing.	Hennerbichler	Dietmar	TU Wien	Inst.f. Elektrische Maschinen	Gußhausstr. 25-29	A	- 1040 Wien hennerbich@eemvs1.una.ac.at
Dr.	Karigl	Günther	TU Wien	Inst.f. Algebra u.Diskrete Mathematik	Wiedner Hauptstr. 8-10	A	- 1040 Wien gkarigl@email.tuwien.ac.at
	Kienberger	Andreas	(TU Wien E 303)		Holocherg. 45/22	A	- 1150 Wien
	Kohant	Erwin	BG/BRG	Wien 12	Rosasg. 1-3	A	- 1120 Wien
DI	Koller	Hans			Gudrunstr. 11	A	- 1100 Wien
	Lämmerhofer	Thomas			Börnergasse 15/1/4	A	- 1190 Wien
Mag.	Landler-Gartmayr	Maria	GRG	Wien 4	Wiedner Gürtel 68	A	- 1040 Wien
Mag.	Ortner	Gerhard	TU Wien	Inst. f. Betriebswissenschaften	Theresianumgasse 27	A	- 1040 Wien ortner@ebwuv1.tuwien.ac.at
Dr.	Pillwein	Gerhard	BG u. BRG	Wien 3	Kundmannng. 22	A	- 1030 Wien
Dipl.Ing.	Punzengruber	Dieter	TU Wien	Inst. f. flexible Automation	Gußhausstr. 25-29	A	- 1040 Wien dieter@infa.tuwien.ac.at
Mag.	Rauch	Eva	PRIG	Strebersdorf	Anton Böckg. 37	A	- 1200 Strebersdorf
	Reichl	Herwig			Argentinioerstr. 11	A	- 1040 Wien
Mag.	Stowasser	Brigitte	BG u. BRG	Laa/Thaya	Martin Wachter Platz 6	A	- 2136 Laa/Thaya
Mag.	Stowasser	Horst	BG u. BRG	Laa / Thaya	Martin Wachter Platz 6	A	- 2136 Laa/Thaya

Simulation in der Didaktik, TU Wien, 13.9.1995, Teilnehmerliste

Mag.	Strohschneider	Petra	PRIG	Strebersdorf	Anton Böckgasse 37	A	- 1210 Wien
Dr.	Veitl	Mario	Klin. Institut f. med. u. chem.	Labordiagnostik	Währinger Gürtel 18-20	A	- 1090 Wien mario@ai.univie.ac.at
Dr.	Weigl	Heinz			Scheibengasse 5	A	- 1190 Wien

Petri-Netze, ein Überblick

abstrakte Modelle des Informations- und Objektflusses zur Beschreibung von Systemen und Prozessen auf unterschiedlichen Abstraktionsebenen in einheitlicher Sprache.

Hauptanwendungsgebiet: Systeme mit kausal unabhängigen (nebeneinander) Ereignissen.

Theorie setzt Struktur und Verhalten zueinander in Beziehung.

Anwendung der Theorie zum Korrektheitsnachweis der Modelle (Verifikation).

Softwarepakete zur Simulation und zur Analyse der Modelle.

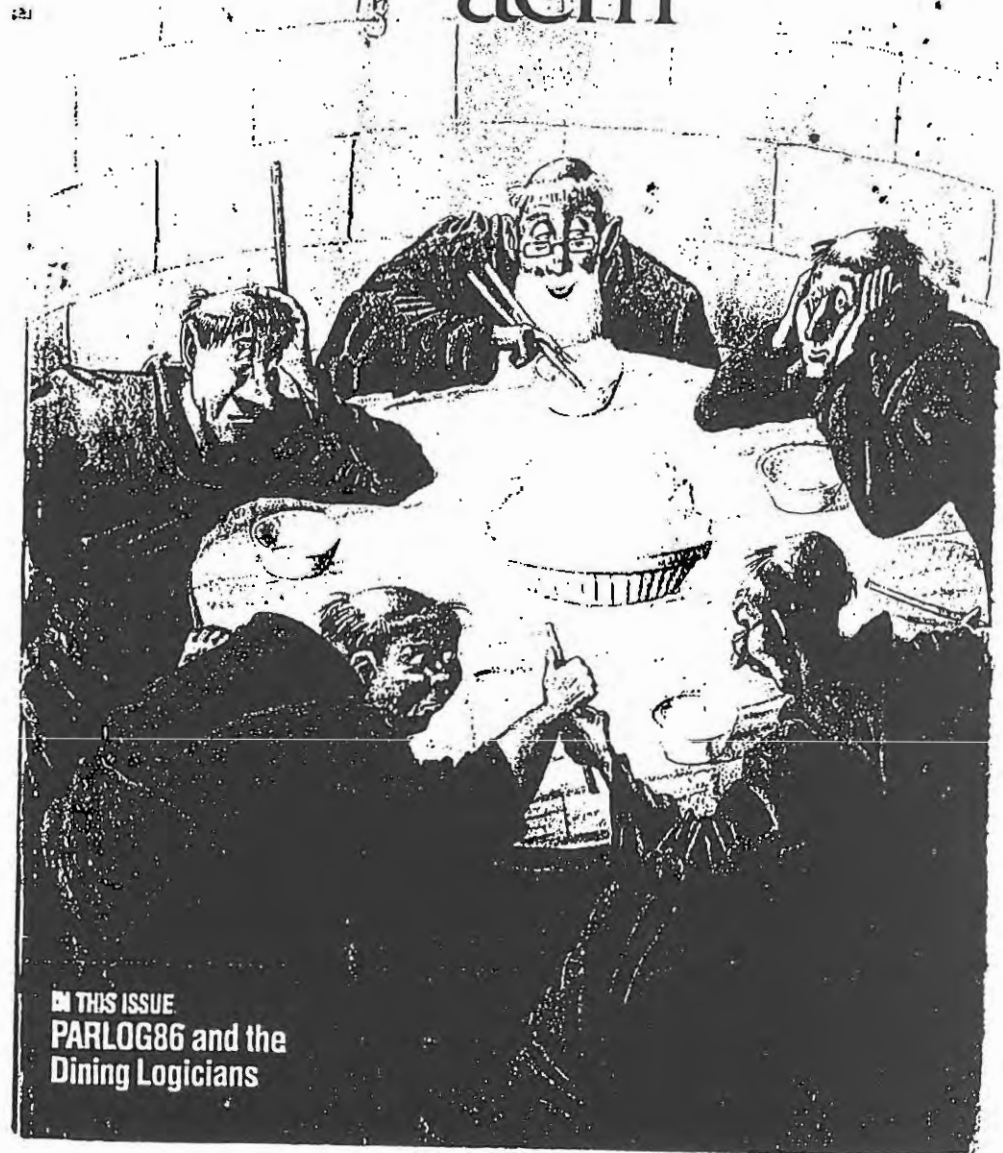
Beziehungen zwischen System und Modell:

- endliche viele Zustände → Beschränktheit des Netzes
- bestimmte Zustände → Repräsentation durch Markierungen
- verklemmungsfreies System → keine tote Markierung erreichbar
- rücksetzbares System → Anfangsmarkierung von jeder erreichbaren aus stets erreichbar
- Abschaltung von Systemteilen → Lebendigkeit bzw. Nichtlebendigkeit von Transitionen
- einseitige Lösung → unfaire Behandlung von Konflikten

VOLUME 31 NUMBER 1

JANUARY 1988

COMMUNICATIONS OF THE acm



Netzgraphen

grundlegende Struktur der Netztheorie; zunächst keine formalen Kanten- und Transitionsbeschriftungen sowie formale Netzdynamik (Markenspiel).

Netze und Teilnetze

Netz aus Rechtecken und Kreisen sowie Pfeilen zwischen Rechtecken und Kreisen.

Def.: Ein Netz (oder Netzgraph) ist ein Tripel

$$N = (S, T, F)$$

mit $S \cap T = \emptyset$ disjunkte Mengen

$$F \subseteq (S \times T) \cup (T \times S).$$

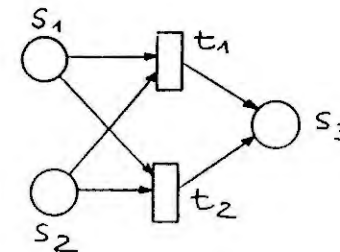
Die Elemente von S heißen Stellen, die von T heißen Transitionen. Beide zusammen werden als Knoten bezeichnet.

Die Elemente der Flußrelation F schließlich heißen Kanten.

Merkmale:



Ein Netz:



$$S = \{s_1, s_2, s_3\},$$

$$T = \{t_1, t_2\}$$

$$F = \{(s_1, t_1), (s_1, t_2), (s_2, t_1), (s_2, t_2), (t_1, s_3), (t_2, s_3)\}$$

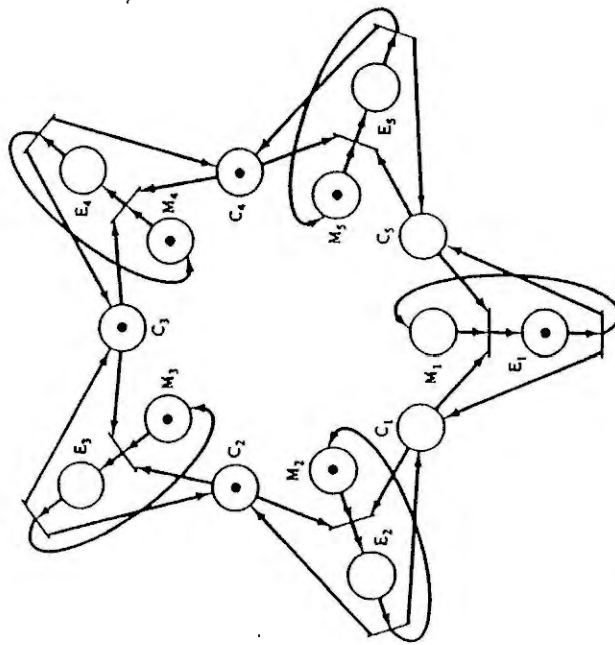


Figure 3.32 The dining philosophers problem. Each philosopher is modeled by two places, mediating (M_i) and eating (E_i).

- zwei bestimmte Mengen von Nachbarknoten eines Netzknotens x :

$$\bullet x := \{y \mid (y, x) \in F\} \text{ Vorbereich von } x$$

Menge aller Eingangs- bzw. Inputknoten (d.h. insbesondere Eingangsstellen falls x Transition ist usw.)

$$x^\bullet := \{y \mid (x, y) \in F\} \text{ Nachbereich von } x$$

Menge aller Ausgangs- bzw. Outputknoten (-stellen oder -transitionen)

Diese Definitionen sind leicht auf knotenmengen auszuweiten:

Für eine Menge $A \subseteq SUT$ ist

$$\bullet A := \bigcup_{x \in A} \bullet x \text{ und } A^\bullet := \bigcup_{x \in A} x^\bullet,$$

wobei analoge Bezeichnungen (Vorbereich usw.) gelten. im Netz:

$$\bullet \{t_1, t_2\} = \bullet t_1 = \bullet t_2 = \{s_1, s_2\}.$$

Insbesondere gilt also für $x, y \in N$: $x \in \bullet y \Leftrightarrow y \in x^\bullet$.

- Ist $|\bullet x| > 1$ bzw. $|x^\bullet| > 1$, so nennt man x Vorwärtsverzweigt bzw. Rückwärtsverzweigt.
- Ein Netz heißt schlicht, wenn keine zwei Knoten denselben Vor- und Nachbereich besitzen, d.h. wenn $\forall x, y: \bullet x = \bullet y \wedge x^\bullet = y^\bullet \Rightarrow x = y$.
- $x \in N$ heißt isoliert, falls $\bullet x \cup x^\bullet = \emptyset$
- Ein Netzteil der Form $(\{s\}, \{t\}, \{(s, t), (t, s)\})$ wird als Schlinge bezeichnet.
N heißt rein, falls es keine Schlingen enthält.
eine Schlinge:



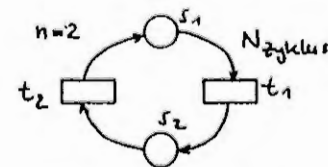
Spezielle Netzklassen

Zyklus:

Definition: Ein Zyklus ist ein Netz mit

$$n \in \mathbb{N}, S = \{s_1, s_2, \dots, s_n\}, T = \{t_1, t_2, \dots, t_n\}$$

$$F = \{(s_i, t_i) \mid 1 \leq i \leq n\} \cup \{(t_i, s_{i+1}) \mid 1 \leq i \leq n-1\} \cup \{(t_n, s_1)\}$$



Er gehört zu allen eingeführten Netzklassen

Zustandsmaschine:

Zustandsmaschinen (ZM) haben ausschließlich unverzweigte Transitionen, die außerdem nicht am absoluten Rand liegen.

Definition: Eine Zustandsmaschine ist ein Netz mit

$$\forall t \in T: |\bullet t| = |t^\bullet| = 1.$$

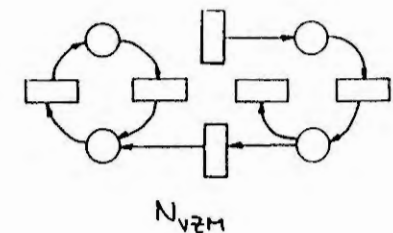
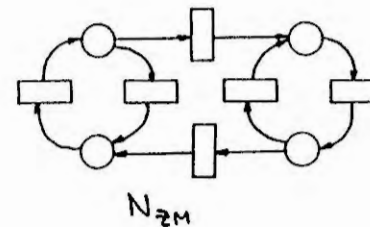
Verallgemeinerte Zustandsmaschine:

Bei Verallgemeinerten Zustandsmaschinen (VZM) sind Transitionen als absolute Randknoten zugelassen.

Definition: Eine Verallgemeinerte Zustandsmaschine ist durch

$$\forall t \in T: |\bullet t| \leq 1, |t^\bullet| \leq 1$$

definiert.



Synchronisationsgraph:

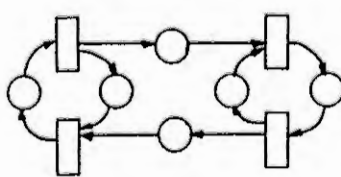
Synchronisationsgraphen (SG) haben ausschließlich unverzweigte Stellen, die außerdem nicht am absoluten Rand liegen:

Definition: Ein Synchronisationsgraph ist ein Netz mit
 $\forall s \in S: |s| = |s'| = 1$.

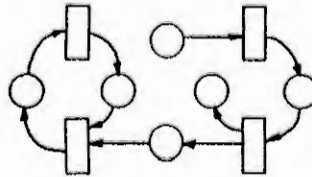
Verallgemeinerter Synchronisationsgraph:

Bei verallgemeinerten Synchronisationsgraphen (VSG) sind Stellen als absolute Randknoten zugelassen:

Definition: Ein verallgemeinerter Synchronisationsgraph ist durch
 $\forall s \in S: |s| \leq 1, |s'| \leq 1$
 definiert.



N_{sg}



N_{vsg}

Free-choice-Netz:

Definition: Ein Free-choice-Netz ist ein Netz mit
 $\forall (s,t) \in F \cap (S \times T): s' = \{t\} \vee \bullet t = \{s\}$.

mögliche Verzweigungen:



Die Outputtransitionen vorwärts verzweigt. Stellen sind nicht rückwärts verzweigt. Im Konfliktfall kann frei und unabhängig von anderen Stellen eine Transition zum Stellen ausgewählt werden.

Haben zwei Stellen eine gemeinsame Nachtransition, dann ist sie die einzige Nachtransition dieser Stellen.

Netztransformationen

Vergrößerung: Ein transitions- bzw. stellenbrauertes Teilnetz wird durch eine Transition bzw. Stelle ersetzt.

Formale Darstellung:

Seien $N = (S, T, F)$ ein Netz und $N' = (S', T', F')$ ein stellenbrauertes Teilnetz von N , d.h. mit $\text{Rand}(N', N) \subseteq S'$. Sei ferner $s_{N'} \notin S \cup T'$; dies wird die Stelle, die N' ersetzt.

Wir setzen für alle $(x, y) \in F \setminus F'$

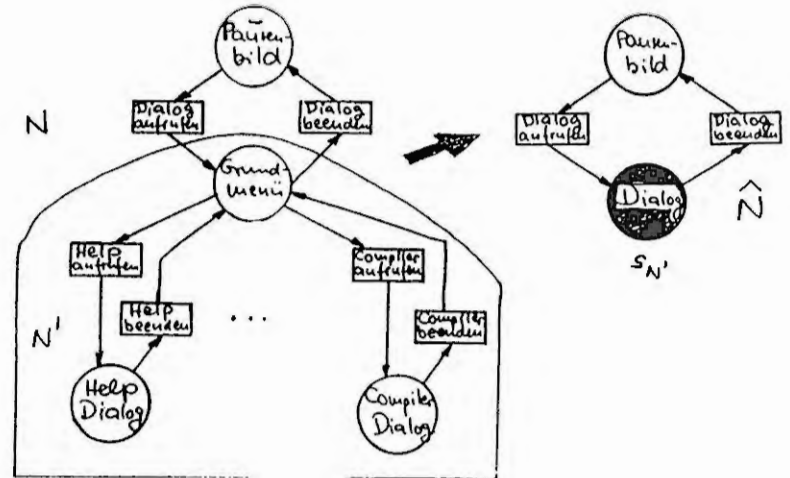
$$\varphi(x, y) = \begin{cases} (x, y) & \text{wenn } x, y \in S \cup T \setminus S' \cup T' \\ (x, s_{N'}) & \text{wenn } y \in \text{Rand}(N') \\ (s_{N'}, y) & \text{wenn } x \in \text{Rand}(N') \end{cases}$$

φ spiegelt die Operation wider, die alle Pfeile zwischen N' und seiner Umgebung auf $s_{N'}$ 'zusammenzieht'.

Dann ist das Netz $\hat{N} = (\hat{S}, \hat{T}, \hat{F})$ mit

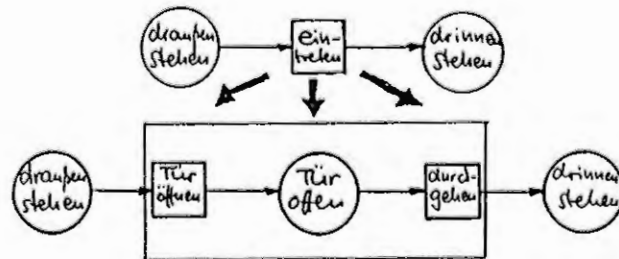
$$\begin{aligned} \hat{S} &= (S \setminus S') \cup \{s_{N'}\} \\ \hat{T} &= T \setminus T' \\ \hat{F} &= \varphi(F \setminus F') \end{aligned}$$

eine Vergrößerung von N ; entsprechend für $\text{Rand}(N', N) \subseteq T'$.



Verfeinerung: Umkehr der Vergröberung
 bei ihr wird eine Transition durch ein transitionsbrauertes Teilnetz bzw. eine Stelle durch ein stellenbrauertes Teilnetz ersetzt.

Verfeinerung bedeutet Detaillierung der 'inneren' Mechanismen von Zuständen oder Ereignissen.



Eintreten, summarisch und ausführlich

Einbettung: Ein Netz wird durch Hinzufügen von Kanten und Knoten erweitert, daß das erhaltene Netz das ursprüngliche als Teilnetz enthält.

Typische Ansatzpunkte für die Einbettung die absoluten Randknoten eines Netzes:

$$\text{Abs Rand}(N) := \{x \in \text{SUT} \mid x = \emptyset \vee x = \emptyset\}.$$

Restriktion ist die Umkehrung der Einbettung.

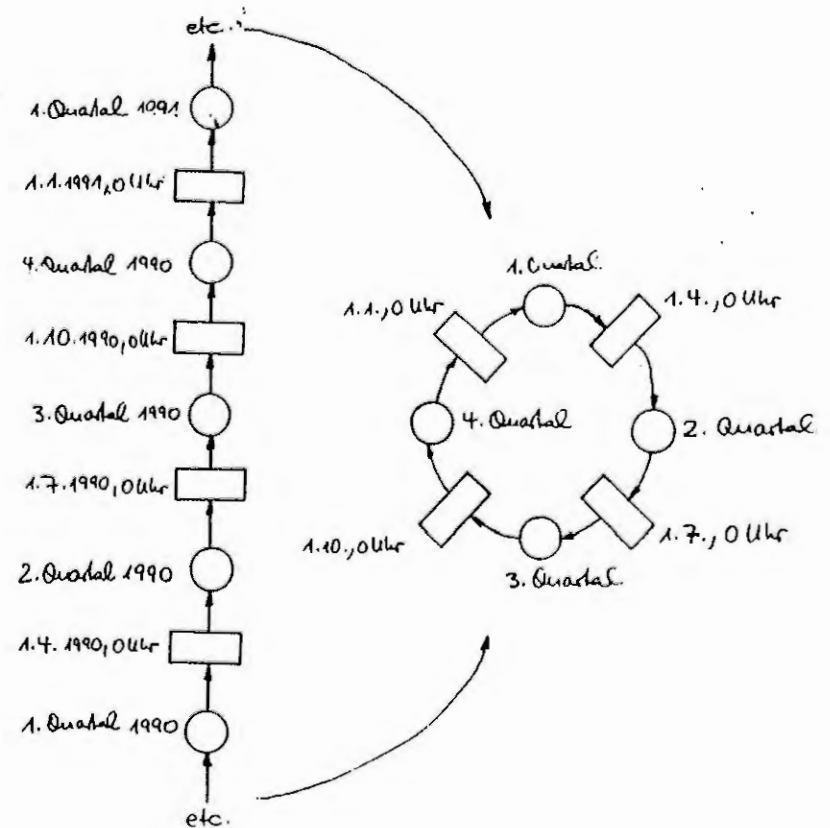
Faltung: Gleichartige Netzteile werden 'aufeinandergefaltet', wobei Knoten nur auf solche gleichen Typs gefaltet werden und die Aupnotation gewahrt bleibt.

Mit der Faltung führen wir eine globale Abstraktion durch.

Entfaltung ist die Umkehrung der Faltung.*

Eine wichtige Einbettung ist die zusätzliche Konstruktion des Komplements von Stellen.

* Die Semantik von komplexen Netztypen läßt sich durch Entfaltung auf die Semantik einfacher Netztypen zurückführen.



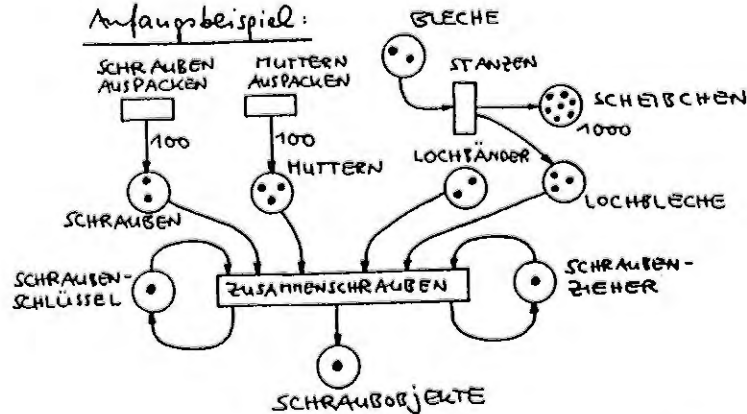
Standardbeispiel zur Darstellung der Faltungen ist der Vergleich zwischen der 'historischen' und der 'unklassischen' Sicht der Vierteljahre.

Alle historischen Jahresanfänge '1.1.1990', '1.1.1991', ... werden zu einer allgemeinen Stelle '1.1.' zusammengefaßt, u.z. (Aufwickeln des unendlichen Netzes auf den Zyklus).

STELLEN/TRANSITIONEN-Netze

Systeme mit anonymen Marken ('black token')

Anfangsbeispiel:



Es genügt das Wissen über die Anzahl der modellierten Objekte auf jeder Stelle, um die möglichen nächsten Ereignisse zu bestimmen.

Anstelle der Originalobjekte entsprechende Anzahlen von 'Spielmarken'.

Man kann nicht mehr von 'Bedingungen' und 'Ereignissen' sprechen. Stattdessen bezeichnen wir allgemeinere (abstrakte) Kreise als Stellen und Kästchen als Transitionen und erklären dynamische Veränderungen mit dem Schalten von Transitionen nach einer Regel.

Die Transition ZUSAMMENSCHRAUBEN 'schaltet', indem sie jeder Eingangsstelle eine Marke entzieht und eine Marke auf die Ausgangsstelle SCHRAUBOBJEKTE legt. SCHRAUBEN AUSPACKEN erzeugt bei jedem Schalten 100 Marken auf der Stelle SCHRAUBEN.

=> 'Markenspiel'

Netz ist ergänzt um Kapazitätsangaben an Stellen, z.B. 1000 Schreibeichen und Gewichtsangaben an Kanten, z.B. 100 Schrauben.

Kein 'Markenfluss'. Viel mehr werden sie auf den benachbarten Stellen erzeugt und verbraucht, wobei sich ihre Gesamtzahl im allgemeinen ändert.

Wir folgen der allgemein üblichen Konvention, daß Stellen ohne Kapazitätsangabe implizit die Kapazität ∞ (Bezeichnung ω) und Kanten ohne Gewichtsangabe implizit das Kantengewicht 1 haben.

Formale Definition der S/T-Netze:

Definition:

Ein 6-Tupel $N = (S, T, F, K, W, M_0)$ heißt Stellen/Transitionen-Netz (S/T-Netz), falls gilt:

(S, T, F) ist ein Netz aus Stellen S und Transitionen T .

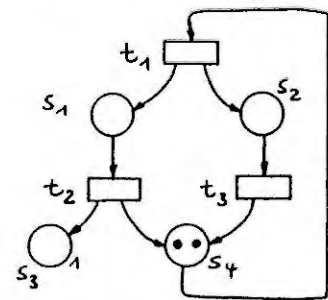
$K: S \rightarrow \mathbb{N} \cup \{\omega\}$ erklärt eine (möglicherweise unbeschränkte) Kapazität für jede Stelle.

$W: F \rightarrow \mathbb{N}$ bestimmt zu jedem Pfeil des Netzes ein Gewicht.

$M_0: S \rightarrow \mathbb{N}_0$ ist eine Anfangsmarkierung, die die Kapazitäten respektiert, d.h. für jede Stelle $s \in S$ gilt: $M_0(s) \leq K(s)$.

In der graphischen Darstellung wird die 'Markenzahl' $M(s)$ einer Stelle $s \in S$ unter einer Markierung M durch $M(s)$ Punkte (Marken) in s dargestellt. Kapazitäten $\neq \infty$ bzw. Gewichte $\neq 1$ werden an die betreffenden Stellen und Pfeile geschrieben.

Anwenden der formalen Begriffe auf das S/T-Netz Y-Beispiel



Y-Beispiel

Netzgraph $\hat{N} = (S, T, F)$

$S = \{s_1, s_2, s_3, s_4\}$

$T = \{t_1, t_2, t_3\}$

$F = \{(t_1, s_1), (t_1, s_2), (s_1, t_2), (s_2, t_3), (t_2, s_3), (t_2, s_4), (t_3, s_4), (s_4, t_1)\}$

Daraus wird ein S/T-Netz $N = (S, T, F, K, W, M_0)$ unter Einbeziehung der Kapazitäten (genauer: der Kapazitätsabbildung)

$K = \{(s_1, \omega), (s_2, \omega), (s_3, 1), (s_4, \omega)\}$,

der Kantengewichte (genauer: Gewichtsbildung)

$W = \{((t_1, s_1), 1), ((t_1, s_2), 1), ((s_1, t_2), 1), ((s_2, t_3), 1), ((t_2, s_3), 1), ((t_2, s_4), 1), ((t_3, s_4), 1), ((s_4, t_1), 2)\}$

und der Anfangsmarkierung $M_0 = \{(s_1, 0), (s_2, 0), (s_3, 0), (s_4, 2)\}$

Die Dynamik der S/T- Netze

Im folgenden sei N ein Stellen-Transitions-Netz.

Eine Abbildung $M: S \rightarrow \mathbb{N}_0$ mit $\forall s \in S: M(s) \leq K(s)$ heißt Markierung von N .

Die Anfangsmarkierung M_0 eines S/T-Netzes ist eine Markierung. $\mathcal{M}(N)$ bezeichnet die Menge aller Markierungen auf N .

Definition: Eine Transition $t \in T$ heißt aktiviert unter M , geschrieben $M[t >]$, wenn

$$\forall s \in {}^{\circ}t: M(s) \geq W(s,t), \quad (A)$$

$$\forall s \in t^{\circ}: M(s) \leq K(s) - W(t,s). \quad (B)$$

- (A) fordert, daß beim Schalten genügend Marken vorhanden sind, so daß die Markierung keiner Stelle unter Null sinkt.
(B) sichert, daß die Kapazität keiner Stelle überschritten wird.

Wir sagen t schaltet von M nach M' und schreiben $M[t > M']$, wenn t unter M aktiviert ist und M' aus M durch Entnahme von Marken aus den Eingangsstellen und Ablage von Marken auf die Ausgangsstellen gemäß den Kantenrichtungen entsteht:

$$M'(s) = \begin{cases} M(s) - W(s,t) & \text{falls } s \in {}^{\circ}t \setminus t^{\circ}, \\ M(s) + W(t,s) & \text{falls } s \in t^{\circ} \setminus {}^{\circ}t, \\ M(s) - W(s,t) + W(t,s) & \text{falls } s \in {}^{\circ}t \cap t^{\circ}, \\ M(s) & \text{sonst.} \end{cases}$$

M' heißt dann (unmittelbare) Folgemarkierung von M unter t und wird auch als M_t geschrieben.

Die Aktivierungsbedingungen und die vorstehende Definition der Folgemarkierung bezeichnet man auch als die Schaltregel.

Im Y_{Beispiel} ist t_1 aktiviert, denn ${}^{\circ}t_1 = \{s_4\}$, und mit

$$M_0(s_4) = 2 = W(s_4, t_1)$$

ist (A) erfüllt. Ferner ist $t_1^{\circ} = \{s_1, s_2\}$, und mit

$$M_0(s_1) = 0 = K(s_1) - W(t_1, s_1) \quad \omega = 1$$

$$M_0(s_2) = 0 = K(s_2) - W(t_1, s_2) \quad \omega = 1$$

ist (B) erfüllt. Schaltet t_1 , so erhalten wir als neue Markierung von N

$$M_0 t_1 = \{(s_1, 1), (s_2, 1), (s_3, 0), (s_4, 0)\}.$$

Im Y_{Beispiel} ist die Folge $w = t_1 t_3 t_2 t_1$ eine Schaltfolge, die M_0 in:

$$M'' = \{(s_1, 1), (s_2, 1), (s_3, 1), (s_4, 0)\}$$

überführt, d.h. es gilt $M_0[w > M'']$.

In der Regel werden Markierungen als Vektoren notiert. Die i -te Komponente gibt an, wieviele Marken auf der i -ten Stelle des Netzes liegen. Diese Notation setzt voraus, daß die Plätze 'durchnumeriert' sind, so z.B.:

$$s_1, s_2, s_3, \dots, s_6.$$

Die Markierung

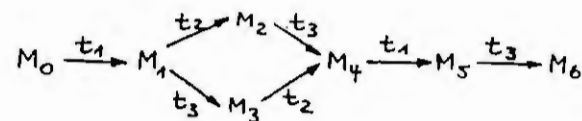
$$(0, 2, 1, 5, 0, 3)$$

ordnet z.B. der Stelle s_1 null und der Stelle s_6 drei Marken zu.

Die Erreichbarkeitsmenge $[M_0 >]$ von Y_{Beispiel} berechnen wir am besten tabellarisch durch systematisches Abprüfen aller Schaltmöglichkeiten.

Nr.	s_1	s_2	s_3	s_4	Schaltungen
M_0	0	0	0	2	$t_1 \rightarrow M_1$
M_1	1	1	0	0	$t_2 \rightarrow M_2$ $t_3 \rightarrow M_3$
M_2	0	1	1	1	$t_3 \rightarrow M_4$
M_3	1	0	0	1	$t_2 \rightarrow M_4$
M_4	0	0	1	2	$t_1 \rightarrow M_5$
M_5	1	1	1	0	$t_3 \rightarrow M_6$
M_6	1	0	1	1	

Es gibt 7 erreichbare Markierungen und auch nur endlich viele Schaltfolgen aus M_0 .



Der Erreichbarkeitsgraph von Y_{Beispiel}

Definition: Ein S/T-Netz $N = (S, T, F, K, W, M_0)$ heißt Kontaktfrei (contactless), wenn für alle erreichbaren Markierungen $M \in [M_0]$ eine Transition $t \in T$ genau dann aktiviert ist, wenn $M(s) \geq W(s, t)$ für alle $s \in \bullet t$ gilt.

Netzdynamik und Lineare Algebra

Beschreibung der Änderung von Markierungen beim Schalten von Transitionen durch Matrizen.

Netz aus Stellen $S = \{s_1, s_2, \dots, s_i, \dots, s_m\}$
und Transitionen $T = \{t_1, t_2, \dots, t_j, \dots, t_n\}$.

C^+ -Matrix

Matrix mit m Zeilen und n Spalten, d.h. $m \times n$ - oder $S \times T$ -Matrix.

$\forall 1 \leq i \leq m, 1 \leq j \leq n$

$$C_{ij}^+ := \begin{cases} W(t_j, s_i) & \text{falls } s_i \in t_j \bullet \text{ bzw. } (t_j, s_i) \in F \\ \emptyset & \text{sonst} \end{cases}$$

$$C^+ = \left(\begin{array}{ccc|c} C_{11}^+ & \dots & C_{1n}^+ & s_1 \\ \vdots & & \vdots & \vdots \\ C_{m1}^+ & \dots & C_{mn}^+ & s_m \end{array} \right) \begin{array}{c} t_1 \dots t_n \end{array}$$

- C_{ij}^+ Anzahl der Marken, die beim Schalten von t_j an s_i gemäß Kantengewicht $W(t_j, s_i)$ übergeben werden. (Spaltenvektor)
- C_{ij}^+ ist auch diejenige Platzkapazität, die in der Stelle s_i beim Schalten von t_j mindestens frei sein muß

C^- -Matrix

analog zur C^+ -Matrix

$$C_{ij}^- := \begin{cases} W(s_i, t_j) & \text{falls } s_i \in \bullet t_j \text{ bzw. } (s_i, t_j) \in F \\ \emptyset & \text{sonst} \end{cases}$$

- C_{ij}^- Anzahl der Marken, die beim Schalten von t_j aus s_i abgezogen werden.
- Schaltbarkeit von t_j , wenn an den Stellen s_i mindestens so viele Marken vorhanden sind, wie im Spaltenvektor stehen.

Mit der Angabe von C^+ und C^- ist ein Petri-Netz umkehrbar eindeutig beschrieben.

Zusammenfassung zur Inzidenzmatrix C :

$$C = C^+ - C^-$$

$$C_{ij} := \begin{cases} W(t_j, s_i) & \text{wenn } (t_j, s_i) \in F \setminus F^{-1} \\ -W(s_i, t_j) & \text{wenn } (s_i, t_j) \in F \setminus F^{-1} \\ W(t_j, s_i) - W(s_i, t_j) & \text{wenn } (t_j, s_i) \in F \cap F^{-1} \\ \emptyset & \text{sonst} \end{cases}$$

Schleife

C_{ij} zeigt an, wie sich die Markenzahl von s_i durch ein Schalten von t_j verändert.

Schlingen erscheinen nur mit der Differenz ihrer beiden Kanten Gewichte, bei Gleichheit also hier als \emptyset , d.h. nur ein schlingenfreies S/T-Netz ist durch die Inzidenzmatrix eindeutig bestimmt.

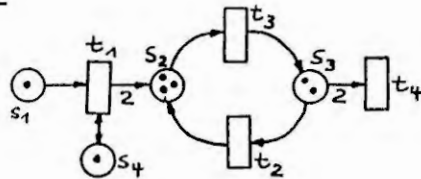
Die j -te Spalte der Inzidenzmatrix bildet den Spaltenvektor

$$C_j = \begin{pmatrix} C_{1j} \\ C_{2j} \\ \vdots \\ C_{mj} \end{pmatrix}$$

C_j entspricht der Markierungsänderung beim Schalten von t_j : $\Delta M = C_j^T = (C_{1j}, C_{2j}, \dots, C_{mj})$

$$M[t_j] > M' \Rightarrow M' = M + C_j^T$$

Beispiel:



Zugehörige Incidenzmatrix:

$$C = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 2 & 1 & -1 & 0 \\ 0 & -1 & 1 & -2 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Die angegebene Markierung könnte durch Schalten von t_2 entstanden sein:

$$(1, 3, 2, 1) = (1, 2, 3, 1) + (0, 1, -1, 0)$$

Anfangs-
markierung

C_2^T Transponierter
Spaltenvektor zu t_2

Bei Schaltfolgen w kann der Häufigkeitsvektor oder Parikh-Vektor definiert werden, der angibt, wie oft die verschiedenen t_j in w auftreten:

$$\underline{w} = \begin{pmatrix} \#(t_1, w) \\ \vdots \\ \#(t_n, w) \end{pmatrix}$$

Beispielsweise gilt im Fall $n=4$ (vier Transitionen)

$$w = t_1 t_2 t_1 t_4 t_1 \Rightarrow \underline{w} = \begin{pmatrix} 3 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

Wir schreiben Markierungen als Zeilenvektoren und Transitions- sowie Parikh-Vektoren als Spalten.

Satz: Ist w eine Schaltfolge auf einem S/T-Netz (N, M_0) , so gilt:

$$M_0[w] \Rightarrow M_0 \cdot w = M_0 + (C \underline{w})^T$$

In w ist nicht mehr die Reihenfolge der schaltenden Transitionen wiedergegeben, da die Endmarkierung häufig durch verschiedene Permutationen einer Schaltfolge erreicht werden kann.

Um entscheiden zu können, ob ein gegebenes n -Tupel der Häufigkeitsvektor einer Schaltfolge ist, sind bis zu $n!$ Transitionenfolgen zu untersuchen. Dagegen läßt sich durch schrittweise Überprüfung der Aktivität leicht feststellen, ob eine gegebene Transitionenfolge eine Schaltfolge ist.

Parikh-Vektor für einzelne Schaltungen t_j :

$$(\underline{t}_j)_i = \delta_{ij} \quad \text{mit } \delta_{ij} := \begin{cases} 1 & \text{wenn } i=j \\ 0 & \text{sonst} \end{cases}$$

i. Komponente von \underline{t}_j

Beispiel:

$$\underline{t}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

Nach einer einzelnen Schaltung:

$$C \underline{t}_j = C_j^T \quad \text{Markierungsänderung}$$

Schaltfolge $w = t_1 t_3 t_4 t_3 t_4$ im Beispiel-Netz

Parikh-Vektor

$$\underline{w} = \begin{pmatrix} 1 \\ 0 \\ 2 \\ 2 \end{pmatrix}$$

$$M_0 w = (1, 3, 2, 1) + \left[\begin{pmatrix} -1 & 0 & 0 & 0 \\ 2 & 1 & -1 & 0 \\ 0 & -1 & 1 & -2 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 2 \\ 2 \end{pmatrix} \right]^T = (1, 3, 2, 1) + (-1, 0, -2, 0) = \underline{(0, 3, 0, 1)}$$

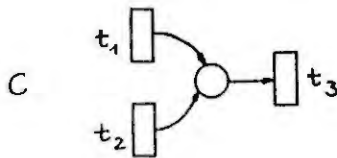
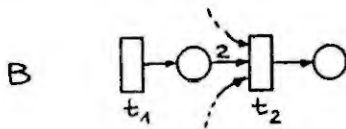
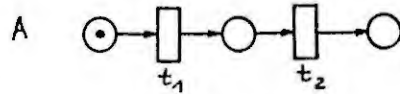
Grundsituationen

Kausalität, Nebenläufigkeit, (Konflikt, Kontakt), Synchronisation, Konfusion, Semaphore, markierte Netze, Deadlock, Trap

beschreiben Relationen zwischen modellierten Ereignissen in realen Systemen durch Schaltbeziehungen zwischen Transitionen.

Kausalität

Kausalität beschreibt, welche Ereignisse einem bestimmten Ereignis vorausgehen müssen.

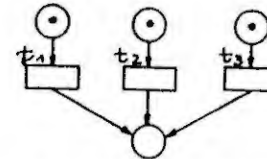


- A: t_1 muß vor t_2 geschaltet haben.
 B: Unter anderem muß t_1 dreimal schalten, bevor t_2 schalten kann (notwendige Voraussetzung).
 C: Jede Schaltung von t_1 oder t_2 ist hinreichend dafür, daß t_3 schalten kann.

Definition: In einem S/T-Netz (N, M_0) ist (das vorherige Schalten von) t_1 notwendig für (das Schalten von) t_2 , wenn in allen Schaltfolgen σ mit $M_0[\sigma t_2]$ die Transition t_1 vorkommt. ■

Nebenläufigkeit

Gegenteil von Kausalität

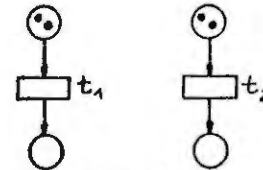


t_1, t_2 und t_3 können unabhängig voneinander schalten. Sie beeinflussen einander nicht.

Definition: Eine n -elementige Transitionenmenge kann (nebenläufig) schalten bzw. ist (nebenläufig) aktiviert wenn jede aus allen ihren Transitionen gebildete n -gliedrige Folge eine Schaltfolge ist. Ihre Folgemarkierung ist die durch alle diese Schaltfolgen erzeugte eindeutige Markierung. ■

S/T-Netze mit Synchronisationsgraphen als zugrundeliegendem Netz erlauben Nebenläufigkeit, auch mit nur einer Anfangsmarkierung.

Anstelle von Transitionenmengen können wir allgemeiner Transitionenmultimengen betrachten, wodurch ggf. eine einzelne Transition quasi nebenläufig zu sich selbst schalten kann:

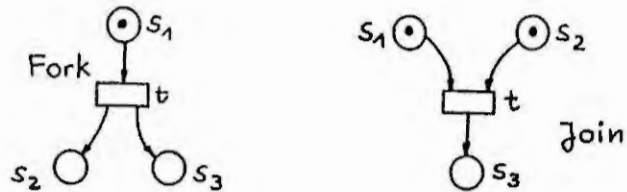


Es können alle sechs "Permutationen" von t_1, t_1, t_2, t_2 schalten. Es gibt sich leicht die Eindeutigkeit der erreichten Markierung.

Satz: In einem S/T-Netz erzeugen alle Schaltfolgen, die Permutationen einer festen Schaltfolge sind, die gleiche Folgemarkierung. ■

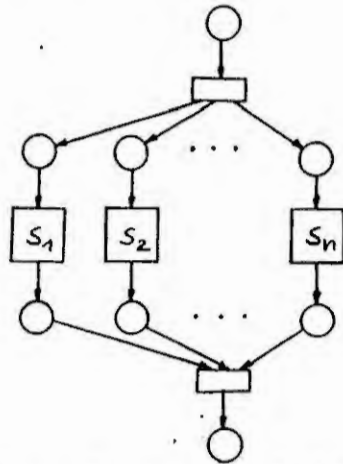
Anspruch für die Überprüfung der nebenläufigen Aktiviertheit: $n!$ Schaltfolgentests, also $n \cdot (n!)$ Einzeltests auf Aktiviertheit.

Synchronisation



Zweistellige Fork- und Join-Synchronisation

Ein 'Ereignisstrom' spaltet sich hier in zwei auf bzw. zwei solche fließen zu einem einzigen zusammen. Programmkonstrukte, die nebelaufige Prozesse starten bzw. wieder vereinigen, lassen sich i.a. auf diese Weise modellieren.



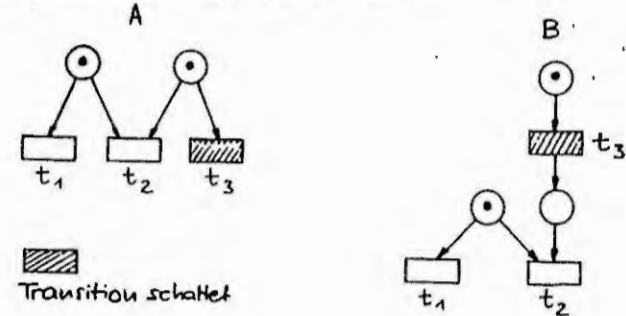
s_1, s_2, \dots, s_n Petri-Netz Darstellungen von Anweisungen (Statements)

Synchronisationsgraphen bestehen ausschließlich aus Synchronisationen.

Konfusion

Konfusion liegt vor, wenn ein Konflikt 'von dritter Seite' aufgelöst oder herbeigeführt werden kann.

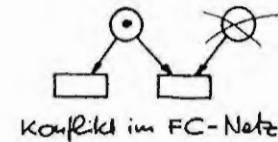
Bemerkung: In markierten Netzen ($K=W, W=1$) sind zwei Transitionen genau miteinander im Konflikt, wenn sie eine gemeinsame Eingangsstelle haben, die genau eine Marke trägt und beide Transitionen aktiviert sind.



Fall A: t_1 und t_2 im Konflikt, nach dem Schalten von t_3 aber nicht mehr. \Rightarrow Konfliktbereinigung oder konfliktbereinigende Konfusion.

Fall B: t_3 führt den Konflikt erst ein. \Rightarrow Konflikteinführung oder konflikteinführende Konfusion.

Markierte Free-Choice-Netze kennen keine Konfusion, da die alternativen Folgetransitionen einer vorwärts verzweigten Stelle definitionsgemäß nicht rückwärts verzweigt sein dürfen.

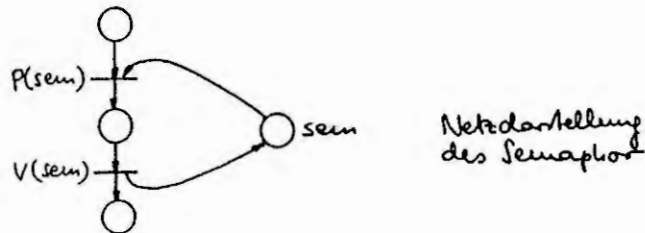


Semaphor

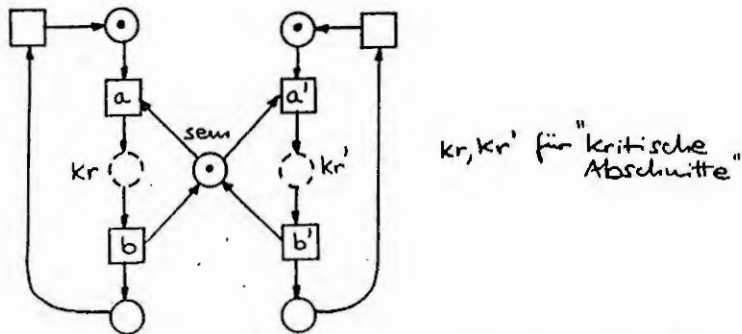
Definition: Ein Semaphor ist eine integer-Variable, auf die nach ihrer Initialisierung nur durch die folgenden Anweisungen $P(sem)$ und $V(sem)$ zugegriffen werden darf:

$P(sem): \text{if } sem > 0 \rightarrow sem := sem - 1$
 $V(sem): sem := sem + 1$

Durch Dijkstra (68) in die Literatur eingeführt. P und V sind abgeleitet von den niederländischen Wörtern "proberen" und "verhogen".



Wechselseitiger Ausschluss mittels Semaphor:



Die Transitionen a und a' bzw. b und b' entsprechen der Wirkung der P - bzw. V -Operation auf den Semaphor sem . Für alle erreichbaren Markierungen M gilt:

$$kr + kr' + sem = 1,$$

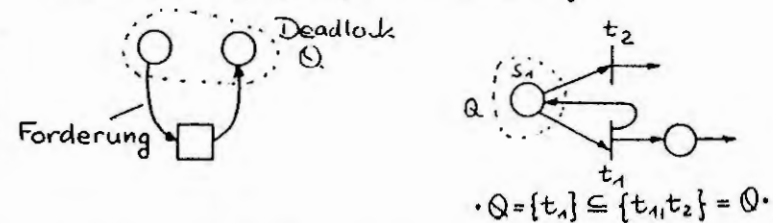
Woraus unmittelbar folgt, daß kr und kr' nicht gleichzeitig markiert sein können.

Markierte Netze

Definition: Ein markiertes Netz ist ein endliches, Kontext-freies S/T-Netz N , wobei für alle $s \in S: M(s) \in \mathbb{N}_0$ und für alle $p \in F: W(p) = 1$.

Deadlocks

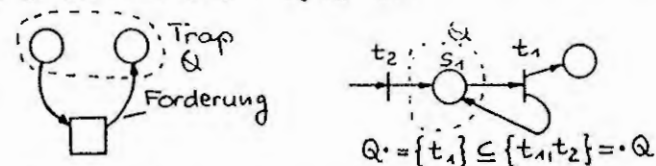
Eine nichtleere Menge von Stellen eines Netzes $N = (S, T, F)$ wird (statischer) Deadlock genannt, wenn jede Vortransition einer Stelle aus Q auch Nachtransition einer Stelle aus Q ist, d.h. $\bullet Q \subseteq Q \bullet$ gilt.



Gerät ein Deadlock in einen unmarkierten Zustand, dann bleibt er in diesem Zustand. Denn jede Transition, die eine Marke auf eine Stelle des Deadlocks legen könnte, müßte dazu mindestens eine Marke von einer Stelle des Deadlocks entfernen. Das ist aber nicht möglich, wenn er keine Marken trägt. (Platzmenge mit der Eigenschaft: einmal raus - immer raus).

Traps (Fallen)

Eine beliebige Menge Q von Stellen wird Trap genannt, wenn jede Nachtransition einer Stelle aus Q auch Vortransition einer Stelle aus Q ist: $Q \bullet \subseteq \bullet Q$.



Wenn Traps markiert sind, können sie niemals alle Marken verlieren. Denn jede Transition, die Marken aus Q entfernt, gibt auch mindestens eine Marke an Q zurück.

Dynamische Eigenschaften von S/T-Netzen

Sicherheit, Beschränktheit, konservative Netze, Lebendigkeit, Erreichbarkeit und Überdeckbarkeit.

Typische Eigenschaften, die man durch Analyse nachweisen kann.

Sicherheit

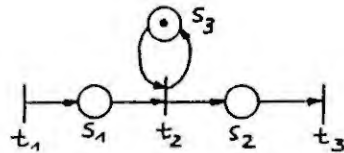
Definition: Eine Stelle $s \in S$ eines S/T-Netzes $N = (S, T, F, K, W, M_0)$ wird sicher (bei M_1) genannt, wenn

$$\forall M \in [M_1] : M(s) \leq 1 \text{ gilt.}$$

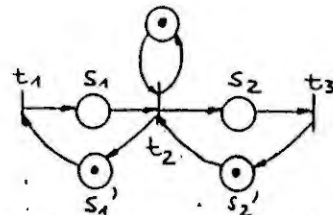
Das Netz N heißt sicher, wenn alle seine Stellen sicher bei M_0 sind.

$$\forall s \in S, \forall M \in [M_0] : M(s) \leq 1.$$

Beispiel:



Netz ist nicht sicher, s_3 ist sicher



Erweitertes Netz ist sicher

Beschränktheit

Es seien $N = (S, T, F, K, W, M_0)$ ein S/T-Netz und $B: S \rightarrow \mathbb{N}_0$ eine Abbildung, die jeder Stelle eine 'kritische Markenzahl' zuordnet. Eine Stelle heißt B-beschränkt, wenn

$$\forall M \in [M_0], s \in S: M(s) \leq B(s)$$

Die Stelle s heißt beschränkt, wenn sie B-beschränkt für eine Zahl B ist.

Eine 1-beschränkte Stelle ist sicher. Statt B-beschränkt wird auch B-sicher gesagt.

Wenn eine Stelle B-beschränkt ist, dann ist sie auch B' -beschränkt für alle $B' \geq B$.

Sei $B \geq \max_{s \in S} B(s)$, dann heißt das Netz N B-beschränkt bzw. beschränkt.

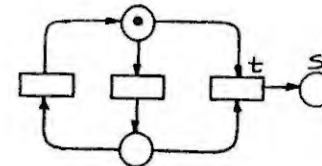
Unterschied zwischen Kapazität und Beschränktheit.

Durch eine Kapazitätsprüfung werden diejenigen Transitions-Schaltungen von vornherein ausgeschlossen, die zu einer Überschreitung der Kapazität führen würden.

Eine Sicherheitsprüfung (Beschränktheit) ist hingegen erst das abschließende Ergebnis einer Analyse oder Beobachtung, die dem Geschehen selbst keine Beschränkung auferlegt.

Kapazität ist Begrenzung a priori, als Vorschrift, Beschränktheit ist Begrenzung a posteriori, als Beobachtung.

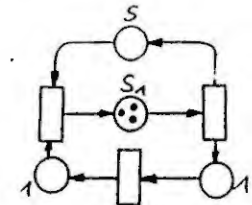
Durch Erweiterung des S/T-Netzes um geeignete Beobachtungsstellen lassen sich auch noch andere Eigenschaften durch Beschränktheit modellieren. Eine Beobachtungsstelle soll das Verhalten nicht ändern.



t schaltet wie.

Dies entspricht $B(s) = 0$.

(Nichts halten als Sicherheitsvariante)



s_1 trägt stets mindestens eine Marke.

$M(s_1) \geq 1 \Leftrightarrow M(s) \leq 2$,
d.h. $B(s) = 2$.

(Interpretation als Sicherheitsvariante)

Fakten

Zunächst Definition der toten Transition:

Definition: Eine Transition t eines S/T-Netzes $N=(S,T,F,W,K,M_0)$ heißt tot, wenn sie unter keiner erreichbaren Markierung aktiviert ist:

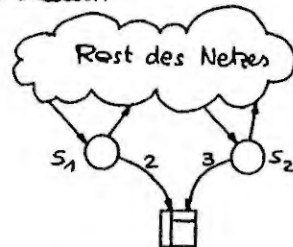
$$\forall M \in [M_0]: \neg M[t >$$

In S/T-Netzen läßt sich jede B-Sicherheit dadurch modellieren, daß für jede Stelle s mit $B(s)$ eine geeignet eingefügte fiktive Transition t_s tot ist.

Dazu wird die Kante (s, t_s) mit dem Kantengewicht $W(s, t_s) = B(s) + 1$ eingefügt.

Mit fiktiven toten Transitionen können auch komplexe Eigenschaften dargestellt werden \Rightarrow Fakten.

Fakten werden als Kommentare im Netz eingezeichnet und von den tatsächlichen Transitionen durch ein stilisiertes 'F' unterschieden.



Die Beobachtung

$$\forall M \in [M_0]: M(s_1) < 2 \vee M(s_2) < 3$$

ist als Fakt dargestellt.

Ein Fakt bezieht sich auf eine bestimmte Anfangsmarkierung M_0 . Bei Folgemarkierungen bleibt er Fakt, wird aber bei anderen Anfangsmarkierungen i.a. falsch.

Konservative Netze

Definition: Eine Transition t des Petri-Netzes N heißt konservativ, wenn die Zahl der Marken, die sie beim Schalten von ihren Vorplätzen abzieht, mit der Zahl der Marken, die sie dabei auf ihre Nachplätze aufbringt, übereinstimmt.

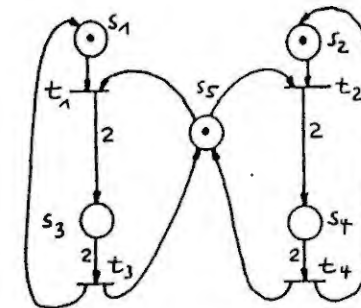
$$\sum_{\forall s \in t} W(s, t) = \sum_{\forall s \in t} W(t, s)$$

Ein Netz wird konservativ genannt, wenn alle seine Transitionen konservativ sind.

Die Gesamtzahl der Marken im Netz bleibt dann konstant:

$$\sum_{\forall s \in S} M(s) = \sum_{\forall s \in S} M_0(s)$$

Beispiel:



Ein konservatives Netz, jede Transition bewegt 2 Marken beim Schalten. Die Gesamtzahl der Marken im Netz ist drei.

Verallgemeinerung: gerichtete Markensumme

Definition: Sei $w: S \rightarrow \mathbb{N}$ eine Abbildung, die wir hier Gerichtsvektor nennen. Ein S/T-Netz heißt w -konservativ, wenn

$$\forall M \in [M_0]: w \cdot M = w \cdot M_0,$$

wobei

$$w \cdot M := \sum_{\forall s \in S} w(s) \cdot M(s).$$

Es heißt streng konservativ, wenn es 1-konservativ ist ($w(s)=1$) sonst konservativ (ohne Zusatz).

Synchronieabstände

Synchronieabstände liefern quantitative Aussagen über das dynamische Systemverhalten, auch ohne die Einführung eines Zeitbegriffs.

E_1, E_2 Ereignismengen:

Wir beobachten, wie oft die Ereignisse aus E_1 bzw. E_2 in jedem Prozess p des Systems eintreten.

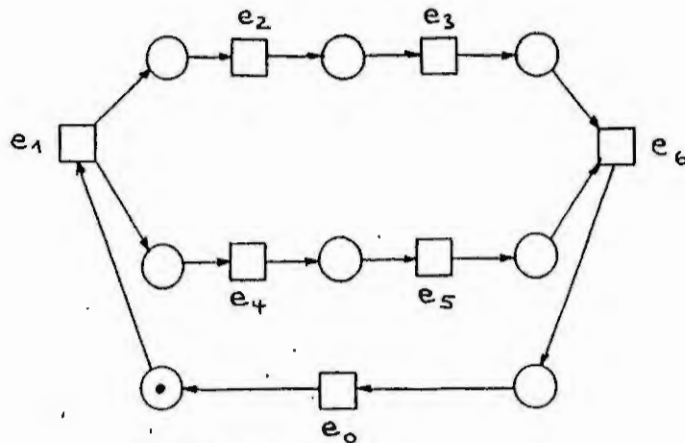
Die Differenz ihres Eintretens wird als die Varianz von E_1 und E_2 in p bezeichnet.

Das Supremum über die Varianzen aller Prozesse ist der Synchronieabstand

$$\sigma(E_1, E_2)$$

zwischen E_1 und E_2 .

$$\sigma(E_1, E_2) := \sup \{ \vartheta(p, E_1, E_2) \mid p \in \Pi_Z \}$$



$$\sigma(\{e_4\}, \{e_0\}) = 1$$

$$\sigma(\{e_2\}, \{e_4\}) = 2$$

$$\sigma(\{e_2, e_3\}, \{e_4, e_5\}) = 4$$

$$\sigma(\{e_2, e_4\}, \{e_3, e_5\}) = 2$$

$$\sigma(\{e_4, e_5\}, \{e_3\}) = \omega$$

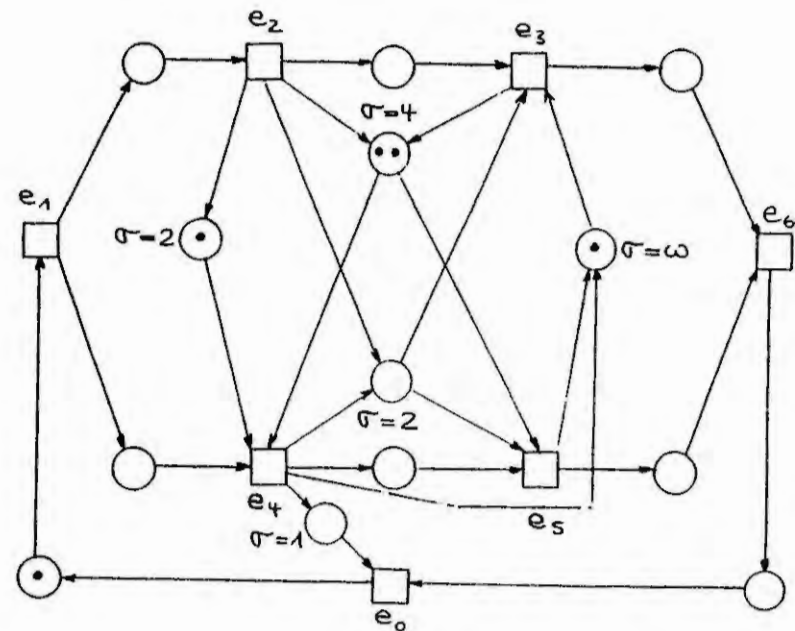
Graphische Darstellung von Synchronieabständen

Synchronieabstände $\sigma(E_1, E_2)$ zwischen Ereignismengen eines B/E-Systems Z veranschaulicht ein zusätzliches S-Element s mit $\bullet s = E_1$ und $s \bullet = E_2$; s kann beliebig viele Marken tragen.

Trifft ein Ereignis aus E_1 bzw. E_2 ein, so erhöht bzw. vermindert sich die Anzahl um 1.

$\sigma(E_1, E_2)$ ist dann die maximale Differenz der Markenanzahl auf s .

In jedem Fall c von Z befindet sich auf s eine Anzahl Marken.



Lebendigkeit

Man unterscheidet zwischen der Lebendigkeit einer Transition t und des Netzes N .

Definition: Eine Transition t eines S/T-Netzes $N = (S, T, F, K, W, M_0)$ heißt aktivierbar, wenn sie mindestens unter einer Folgemarkierung aktiviert ist:

$$\exists M_1 \in [M_0] : M_1[t] > .$$

Sie heißt lebendig, wenn sie unter allen Folgemarkierungen aktivierbar ist:

$$\forall M_1 \in [M_0] : \exists M_2 \in [M_1] : M_2[t] > .$$

Für das Weitere wird ein Lebendigkeitsbegriff zugrunde gelegt, der für jede erreichbare Markierung die Aktivierbarkeit jeder Transition verlangt.

Definition: Ein S/T-Netz $N = (S, T, F, K, W, M_0)$ heißt lebendig, wenn alle seine Transitionen lebendig sind:

$$\forall t \in T, \forall M_1 \in [M_0] : \exists M_2 \in [M_1] : M_2[t] > ,$$

oder $\forall t \in T : t \text{ ist lebendig}.$

Es heißt deadlockfrei oder schwach lebendig, wenn es unter keiner Folgemarkierung tot ist:

$$\forall M_1 \in [M_0] : \exists t \in T : M_1[t] > .$$

Ein S/T-Netz ist genau dann tot, wenn keine Transition aktiviert ist:

$$\forall t \in T : \neg M_0[t] > \quad (\text{tot von Anfang an}).$$

Bemerkung: $[M_1] \subseteq [M_0]$

Lineare Analyse

Zustandsgleichung des Netzes:

$$M' = M + (C \cdot \vec{w})^T$$

C Incidenzmatrix
 \vec{w} Parikh-Vektor

$C \cdot \vec{w}$ Spaltenvektor

Wir schreiben Parikh-Vektoren (wie alle T-Vektoren) als Spalten und alle S-Vektoren (wie z.B. Markierungen) als Zeilen. x^T bzw. C^T berechnet den Transponierten des Vektors x bzw. der Matrix C .

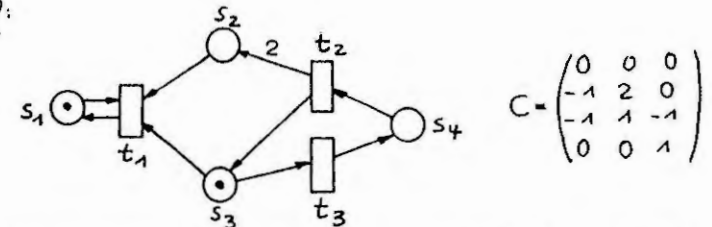
Erreichbarkeit

Notwendig dafür, daß die Markierung M' in N von der Markierung M erreicht werden kann, ist folglich, daß das lineare Gleichungssystem

$$C \cdot x = (M' - M)^T \quad x \text{ gesuchter Parikh-Vektor}$$

eine ganzzahlige Lösung x besitzt, bei der keine Komponente $x_i = x(t_i)$ negativ ist, denn der Parikh-Vektor jeder Schaltfolge w mit $M[w] > M'$ ist von dieser Art, aber hinreichend ist das nicht.

Beispiel:



Ist die Markierung $(1, 8, 0, 1)$ von $(1, 0, 1, 0)$ erreichbar?

$$C \cdot x = \begin{pmatrix} 0 \\ 8 \\ -1 \\ 1 \end{pmatrix} \quad \text{Lösung: } x = \begin{pmatrix} 0 \\ 4 \\ 5 \end{pmatrix}$$

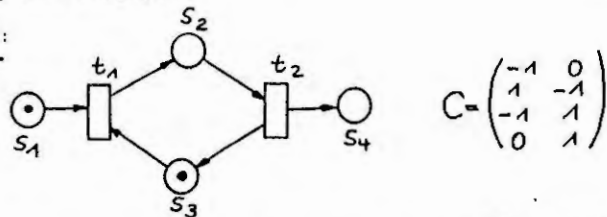
Mögliche Schaltfolge $w = t_1 t_2 t_3 t_2 t_3 t_2 t_3 t_2 t_3$

Dagegen ist die Markierung $(1, 7, 0, 1)$ nicht erreichbar, weil die Matrix-Gleichung

$$C \cdot x = \begin{pmatrix} 0 \\ 7 \\ -1 \end{pmatrix}$$

keinen Parikh-Vektor als Lösung hat. Die Nicht-Erreichbarkeit ist so entscheidbar.

Gegenbeispiel:



Ist die Markierung $(0, 0, 0, 1)$ von $(1, 0, 0, 0)$ erreichbar?

$$C \cdot x = \begin{pmatrix} -1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \quad \text{Lösung: } x = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Zwei Sequenzen $t_1 t_2$ oder $t_2 t_1$. Beide Schaltfolgen sind nicht realisierbar, weil weder t_1 noch t_2 unter der Anfangsmarkierung aktiviert ist.

Satz: Zu einem gegebenen ganzzahligen S-Vektor $\Delta = M' - M$ gibt es genau dann eine Markierung M , von der die Markierung $M + \Delta$ erreichbar ist, wenn das Gleichungssystem $C \cdot x = \Delta^T$ eine Lösung ohne negative Komponenten hat.

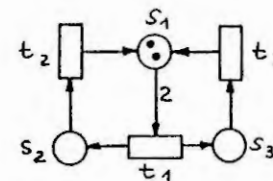
Zu obigen Beispiel würde der Übergang von $M = (1, 0, 1, 0)$ nach $M' = (0, 0, 1, 1)$ das gleiche Δ^T haben (grüne Marke).

Die Schaltfolge $t_1 t_2$ ist jetzt realisierbar.

Lineare Invarianten

T-Invarianten

Schaltfolgen σ , die eine vorgegebene Markierung (z.B. die Anfangsmarkierung M_0) reproduzieren, jedoch nicht unbedingt unter jeder Anfangsmarkierung realisierbar sind.



Die eingetragene Markierung $M_0 = (2, 0, 0)$ wiederholt sich nach Schaltungen von t_1, t_2 und t_3 :

$$M_0 [t_1 t_2 t_3] M_0.$$

Inzidenzmatrix

$$C = \begin{pmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \end{pmatrix},$$

Parikh-Vektor

$$\vec{w} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Die Zustandsgleichung des Netzes liefert ebenfalls

$$M' = (2, 0, 0) + \left[\begin{pmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right]^T = (2, 0, 0) + (0, 0, 0),$$

$$M' = M_0.$$

Definition:

- (1) Jede nichttriviale ganzzahlige Lösung x des homogenen linearen Gleichungssystems $C \cdot x = 0$ wird Transitionsinvariante (kurz: T-Invariante) von N genannt.

- (2) Eine T-Invariante wird als echte T-Invariante bezeichnet, wenn x keine negativen Komponenten hat: $x \geq 0$ (Parikh-Vektor).
- (3) Eine T-Invariante heißt realisierbar in N , wenn es eine Schaltfolge w mit $\bar{w} = x$ und eine in N erreichbare Markierung M mit $MCw > M$ gibt.
- (4) Wir nennen das Netz N von T-Invarianten überdeckt, wenn es eine T-Invariante x besitzt, die in allen Komponenten positiv ist.

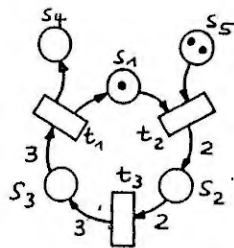
Realisierbare T-Invarianten beschreiben tatsächlich vorhandene Kreise im Erreichbarkeitsgraphen.

Satz: Wenn das Netz N bei irgendeiner Markierung M lebendig und beschränkt ist, dann ist N von T-Invarianten überdeckt.

Die minimalen T-Invarianten bilden ein Erzeugendensystem für alle echten T-Invarianten durch Linearkombinationen.

S-Invarianten

w -konservatives Beispielnetz, in dem die Gesamtmarkenzahl auf einer gewissen Stellenmenge unter geeigneter Wichtung konstant bleibt.



$$6M(s_1) + 3M(s_2) + 2M(s_3) = 6$$

Diese Eigenschaft ist unabhängig von der Anfangsmarkierung (bis auf die Konstante rechts).

Inzidenzmatrix:

$$C = \begin{pmatrix} 1 & -1 & 0 \\ 0 & 2 & -2 \\ -3 & 0 & 3 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix}$$

$$(6, 3, 2, 0, 0) \cdot C = 0$$

Gewichtsvektor

Definition:

- (1) Jede nicht-triviale ganzzahlige Lösung y des homogenen linearen Gleichungssystems $y \cdot C = 0$ wird Stelleninvariante (kurz: S-Invariante) von N genannt.
- (2) Eine S-Invariante y wird als echte S-Invariante bezeichnet, wenn y keine negative Komponente hat: $y \geq 0$.
- (3) Wir nennen das Netz N von S-Invarianten überdeckt, wenn es eine S-Invariante y besitzt, die in allen Komponenten positiv ist.

Satz:

Es sei y eine S-Invariante von N . Dann gilt für alle von M_0 erreichbaren Markierungen M : $y \cdot M^T = y \cdot M_0^T$.

Beweis:

Es sei w eine Schaltfolge mit $M_0[w] = M$, also $M = M_0 + (C \cdot \bar{w})^T$.

$$\begin{aligned} y \cdot M^T &= y \cdot [M_0 + (C \cdot \bar{w})^T]^T = y \cdot M_0^T + y \cdot (C \cdot \bar{w}) = \\ y \cdot M_0^T + (y \cdot C) \cdot \bar{w} &= y \cdot M_0^T + 0 \cdot \bar{w} = y \cdot M_0^T \end{aligned}$$

Im Beispiel: $(6, 3, 2, 0, 0) \begin{pmatrix} 0 \\ 0 \\ 3 \\ 0 \\ 1 \end{pmatrix} = (6, 3, 2, 0, 0) \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 2 \end{pmatrix} = 6$.

\uparrow $\bar{w} = t_2 t_3$ \uparrow M_0^T

Der obige Satz liefert einen Nichterreichbarkeitstest:

Wenn es eine (nicht notwendig echte) S-Invariante y von N gibt, für die $y \cdot M^T \neq y \cdot M_0^T$ ist, dann ist M nicht erreichbar von M_0 .

Beschränktheit

Satz:

1. Ist y eine echte S-Invariante von N und $y(s) > 0$, dann ist die Stelle s bei jeder Anfangsmarkierung beschränkt.
2. Wenn N durch S-Invarianten überdeckt ist, dann ist N bei beliebiger Anfangsmarkierung beschränkt.

Beweis: Aus der 1. Behauptung folgt offensichtlich die zweite. Unter den Voraussetzungen von (1) gilt für jede Anfangsmarkierung M_0 und jede von M_0 erreichbare Markierung M :

$$y \cdot M_0^T = y \cdot M^T \geq y(s)M(s) \geq M(s),$$

also ist die Stelle s bei der Anfangsmarkierung M_0 $y \cdot M_0^T$ -beschränkt.

Es ist eine besondere Aufgabe, echte Invarianten zu finden, die minimal sind. Jede S-Invariante kann aus minimalen S-Invarianten linear kombiniert werden.

Die Menge der Stellen einer echten S-Invariante y , deren Komponente positiv ist, bezeichnen wir als Träger von y .

Bei einer S-Invariante y ist also die mit y gerichtete Markierzahl invariant gegenüber dem Schalten. Diese Zahl kann in vielen praktischen Fällen unmittelbar als Zustandsvariable interpretiert werden.

Erreichbarkeit

Definition: Eine Markierung M ist in einem Netz erreichbar, wenn $M \in [M_0]$.

Es hat bis 1981 gedauert, die Entscheidbarkeit der allgemeinen Erreichbarkeitsfrage

1. Kann in einem gegebenen S/T-Netz eine gegebene Markierung erreicht werden?

zu beweisen. (komplizierter Algorithmus, der nicht in der schlichten Erreichbarkeitsanalyse besteht).

Weitere, ebenso komplexe Probleme:

2. Kann eine gegebene Markierung auf S' , $\emptyset \subseteq S' \subseteq S$, erreicht werden?

3. Kann die leere Markierung erreicht werden?

4. Kann eine feste Stelle $s \in S$ alle Markierungen haben?

Jeder Algorithmus, der eine der Fragen 1-4 in allen Fällen beantwortet, beantwortet mit geringen Veränderungen auch die drei anderen in allen Fällen.

Überdeckbarkeit

Definition: Eine Markierung M eines S/T-Netzes $N = (S, T, F, K, W, M_0)$ heißt überdeckbar, wenn mindestens eine von M_0 erreichbare Markierung M' die vorgegebene überall erreicht oder übersteigt:

$$\exists M' \in [M_0] : \forall s \in S : M'(s) \geq M(s).$$

Bemerkung: Eine Transition eines S/T-Netzes ist genau dann tot, wenn ihre Schwellenmarkierung nicht überdeckbar ist.

Erreichbarkeitsanalyse

Erreichbarkeitsbaum (reachability tree)

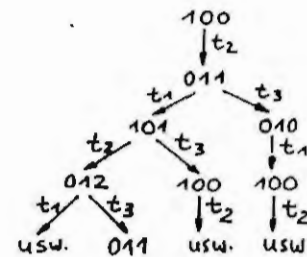
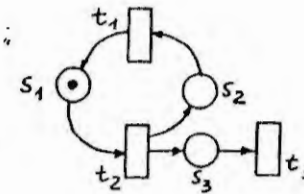
- Knoten sind die Markierungen (in Vektorform)
- Wurzel ist die Anfangsmarkierung M_0
- Seine Zweige sind die von der Wurzel M_0 ausgehenden Pfade
- Die Kanten zwischen den Knoten sind mit den geschalteten Transitionen etikettiert
- Knoten ohne Nachfolger werden Blätter genannt

Konstruktion ('breadth first')

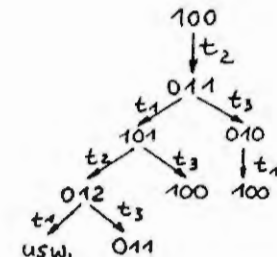
1. Wir tragen die Anfangsmarkierung M_0 als Wurzel ein. M_0 ist jetzt aktuelle Markierung M_a .
2. In der aktuellen Markierung M_a untersuchen wir in der 'Standardreihenfolge' alle Transitionen t darauf, ob sie aktiviert sind:
 - (a) Wenn nein, sind wir momentan mit ihr fertig.
 - (b) Wenn ja, berechnen wir die Folgemarkierung M_f , tragen sie als neuen Knoten M_f ein und verbinden beide durch die Kante t .
3. Wenn alle Transitionen auf diese Weise behandelt worden sind, ist M_a erledigt. Besitzt M_a keine Folgemarkierung M_f , so ist sie ein Blatt und wird nicht weiter betrachtet.
4. Es folgt die nun 'älteste' zu untersuchende Markierung als aktuelle Markierung M_a und Fortsetzung bei Punkt 2.

Der Algorithmus bricht ab, wenn alle Markierungen Blätter sind, d.h. keine Transition mehr aktiviert ist. Der volle Erreichbarkeitsbaum wird (bei endlicher Transitionenmenge) unendlich, sobald eine unendliche Schaltfolge existiert.

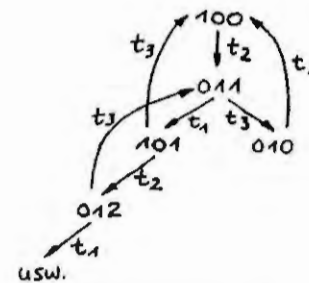
Beispiel:



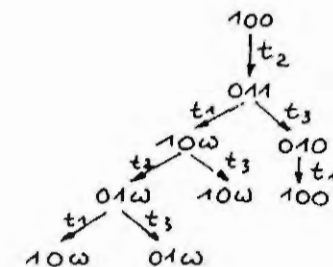
voller Erreichbarkeitsbaum



beschnittener Erreichbarkeitsbaum



Erreichbarkeitsgraph



Überdeckungsbäum

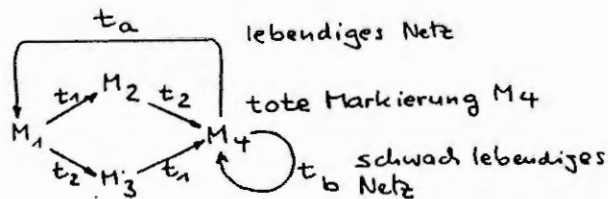
Erreichbarkeitsgraph und Lebendigkeit

Ein S/T-System ist genau dann stark lebendig, wenn sein Erreichbarkeitsgraph von jedem Knoten ausgehend für jedes $t \in T$ einen Pfad besitzt, in dem t als Etikett vorkommt.

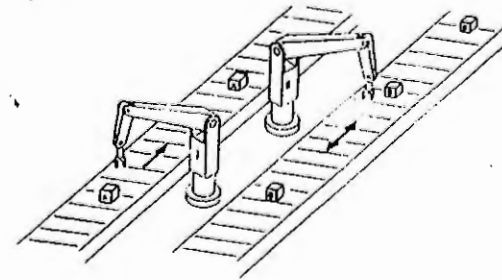
Eine erreichbare Markierung ist genau dann tot, wenn von ihr im Erreichbarkeitsgraph keine Kante ausgeht.

Eine Transition ist genau dann lebendig, wenn der Erreichbarkeitsgraph von jedem Knoten ausgehend einen Pfad besitzt, in dem t als Etikett vorkommt.

Ein S/T-System ist genau dann schwach lebendig, wenn in seinem Erreichbarkeitsgraphen von jedem Knoten ein Pfad weiterführt.



Fertigungsprozeß



Zwei Varianten:

In der ersten Variante soll jedes Werkstück A und B nacheinander von den Robotern I und II bearbeitet werden (Abb. 3.3).

Alternativ dazu erfolgt in der zweiten Variante zur Bearbeitung der Werkstücke B der Einsatz der Roboter in umgekehrter Reihenfolge (Abb. 3.4).

Bedingungen und Ereignisse von Variante 1

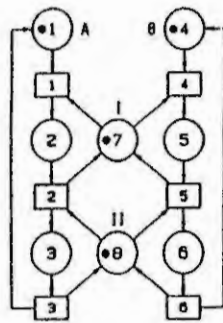
Ereignis	Vorbedingungen	Nachbedingungen
1	1,7	2
2	2,8	3,7
3	3	4,8
4	4,7	5
5	5,8	6,7
6	6	4,8

Ereignisse (Transitionen)

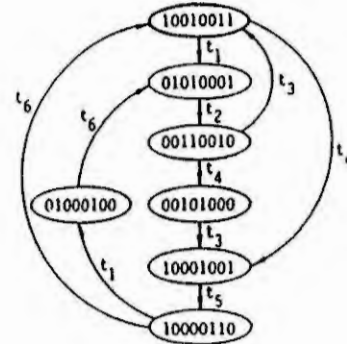
1 (4):	Start Bearbeitung Werkstück A(B), Rob. I
2 (5):	Ende Bearbeitung Werkstück A(B), Rob. I
3 (6):	Start Bearbeitung Werkstück A(B), Rob. II
4 (7):	Ende Bearbeitung Werkstück A(B), Rob. II

Bedingungen (Stellen)

1 (4):	Werkstück A(B) vor Bearbeitung
2 (5):	Werkstück A(B) in Bearbeitung Rob. I
3 (6):	Werkstück A(B) in Bearbeitung Rob. II
7:	Roboter I frei
8:	Roboter II frei



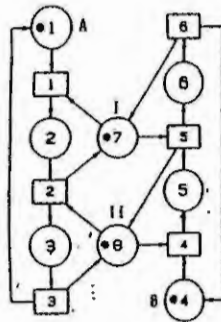
a) Petri-Netz



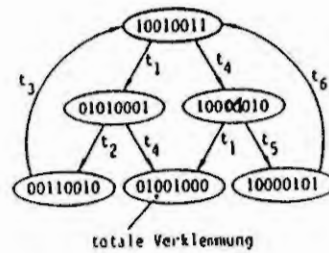
b) Erreichbarkeitsgraph

Abb. 3.3: Zwei Fertigungsstraßen mit gleicher ...

P	M	PRE	POST	NETZ	2: y_beisp2.pnt
1	1	3:1	1:1		
2	0	1:1	2:1		
3	0	2:1	3:1		
4	1	6:1	4:1		
5	0	4:1	5:1		
6	0	5:1	6:1		
7	1	2:1	5:1	1:1 4:1	
8	1	3:1	6:1	2:1 5:1	
e					
		1:	p1		
		2:	p2		
		3:	p3		
		4:	p4		
		5:	p5		
		6:	p6		
		7:	p7		
		8:	p8		
e					
		1:	t1		
		2:	t2		
		3:	t3		
		4:	t4		
		5:	t5		
		6:	t6		
e					



a) Petri-Netz



b) Erreichbarkeitsgraph

Abb. 3.4: ... und mit unterschiedlicher Bearbeitungsfolge

PAN session report:

Current options are:

firing rule: normal (for Place/Transition nets)
time option: no times
fire mode : single
line length: 80

Net read from y_beisp2.pnt

Analysis

Information on elementary structural properties:

Properties of net 2 'y_beisp2':

Places: 8 [1...8]

Transitions: 6 [1...6]

The net is pure.

The net is not statically conflict-free.

Static conflicts:

1: 4.t4
2: 5.t5
4: 1.t1
5: 2.t2

The net is not conservative.

Maximal in/out-degree: 2

The net is an ordinary (hence, homogeneous) one.

The structure [P,T,F] is not EFC, but extended simple.

BND LIV CFF DTR DHA REV ORD HOH Ft0 tF0 Fp0 pF0 SM EFC ES DTP SMC SMD SMA
? ? ? ? ? ? Y Y N N N N N N Y ? ? ? ?

Current analysis options are:

transition names not to be written
print the markings
with format = complete table
output of the graph matrix
computation of the dynamic conflicts
resetability (reversability) test
liveness test

LIVENESS, RESETABILITY AND CONFLICTS OF NET NR. 2 'y_beisp2':

KEY: 1 2 3 4 5 6 7 8

1 0 0 1 0 0 1 1 initial

Conflicts:

1: 1/4;

The net is not dynamically conflict-free.

Reachable markings:

KEY: 1 2 3 4 5 6 7 8

1: 1 0 0 1 0 0 1 1
2: 0 1 0 1 0 0 0 1
3: 1 0 0 0 1 0 0 1
4: 0 0 1 1 0 0 1 0
5: 1 0 0 0 0 1 1 0
6: 0 0 1 0 1 0 0 0
7: 0 1 0 0 0 1 0 0

Arrows:

KEY: 1 2 3 4 5 6

1: 2 . . 3 . .
2: . 4 . . .
3: . . . 5 .
4: . . 1 6 . .
5: 7 . . . 1
6: . . 3 . . .
7: 2

The net is bounded.

The net is safe (1-bounded).

The net is reversible.

The net has no dead reachable markings.

The net is live.

The net has no dead transitions at the initial marking.

The net is covered by semipositive transition-invariants.

BND LIV CFF DTR DHA REV ORD HOH Ft0 tF0 Fp0 pF0 SM EFC ES DTP SMC SMD SMA
Y Y N N N Y Y Y N N N N N N Y ? ? ? ?

Net written to #.SSI

Current options written to options.pan

End of PAN-session.

Arrows:

KEY: 1 2 3 4 5 6

```

1: 2 . . 3 . .
2: . 4 . 5 . .
3: 5 . . 6 . .
4: . . 1 . . .
5: . . . . . dead
6: . . . . . 1

```

The net is bounded.

The net is safe (1-bounded).

The net is not live.

The deadlock-trap-property is not valid.

The net is not reversible.

The net has no dead transitions at the initial marking.

BND LIV CFF DTr DMA REV ORD HOH FcO tFO FpO pFO SM EFC ES DTP SHC SHD SHA
Y N N N Y N Y Y N N N N N Y N ? ? ?

Current options written to options.pan

End of PAN-session.

Höhere Netze

Als Auswahl werden Gefärbte Netze und Prädikat/Transitionsnetze betrachtet.

Gefärbte Netze

Verschiedene, individuelle Marken im Netz. Jede Markensorte hat eine sogenannte Farbe, d.h. sie kann über Namen (rot, grün, blau, ...), Buchstaben (A, B, C, ...) oder Zahlen (0, 1, 2, ...) spezifiziert werden. In der Marke keine strukturierte Information.

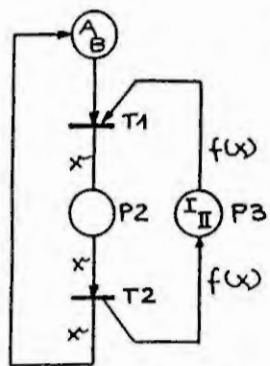
Es wird jedem Platz des Netzes eine endliche Menge von Markensorten (Platzfarben) und jeder Transition eine endliche Menge von Schaltmodi (Transitionsfarben) zugeordnet. Zu jeder Kante des Netzes ist eine Abbildung gegeben, die sich als Übergangsfunktion darstellen läßt. Eine weitere Möglichkeit der mathematischen Beschreibung kann als Netzmatrix erfolgen.

Gegenüber einer Platz/Transitionsnetzmatrix enthält sie pro Element eine Blockmatrix, die für jede Transitionsfarbe die Übergänge der individuellen Marken beschreibt.

Beispiel:

Zwei unabhängige Prozesse benötigen für die Abarbeitung eines bestimmten Programmabschnitts gewisse exklusive Betriebsmittel. Prozeß A benötigt die Betriebsmittel I und II, während Prozeß B nur Betriebsmittel II benötigt. Nach Beendigung dieses Abschnitts werden die Betriebsmittel wieder freigegeben und die Prozesse befinden sich in einer wiederholenden Schleife.

Analyse des Gefärbten Netzes mit INA (Inkpristr Netz-Analysator) möglich! (Starke, HU Berlin)



$x \in \{A, B\}$
 $f(A) = I + II$
 $f(B) = II$

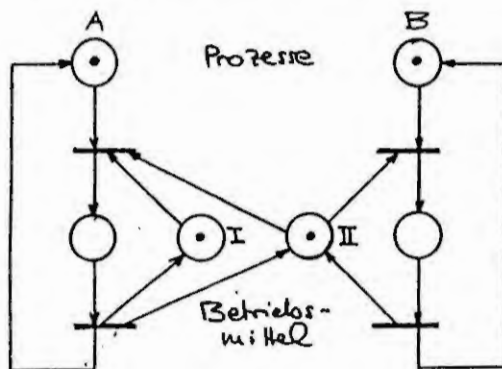
Transitionen Plätze	Farben	1		2	
		Mod1	Mod2	Mod1	Mod2
1	A	-1	0	1	0
	B	0	-1	0	1
2	A	1	0	-1	0
	B	0	1	0	-1
3	I	-1	0	1	0
	II	-1	-1	1	1

Mod 1: $x = A$
 Mod 2: $x = B$

Matrix

Die erste Transitionsfarbe entfernt vom Betriebsmittelplatz beide Marken, die zweite nur II.

Entfaltung zu einem Platz-Transitions-Netz

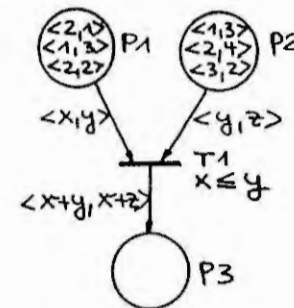


Um das gefaltete Netz zu bekommen, werden die beiden gleichartigen Teilaetze übereinander gelegt ("gefaltet"). Die beiden Betriebsmittel können über einen weiteren Platz abgebildet werden.

Prädikat/Transitionsnetze

Die individuellen Marken sind hier strukturiert. Auch die Bedingungen und Übergänge beim Schalten der Transitionen sind nur noch komplex beschreibbar. Die Beschreibung erfolgt in einer Sprache aus der mathematischen Logik oder Algebra. Ein Prädikat stellt einen über eine Beschreibungssprache festgelegten Ausdruck dar, der die Struktur der Plätze mit ihren Marken (Strukturen) charakterisiert (Platzprädikate) oder Transitionsprädikate einerseits über eine entsprechende Kantenbeschriftung sowie über Schaltbedingungen an den Transitionen definiert.

Beispiel:

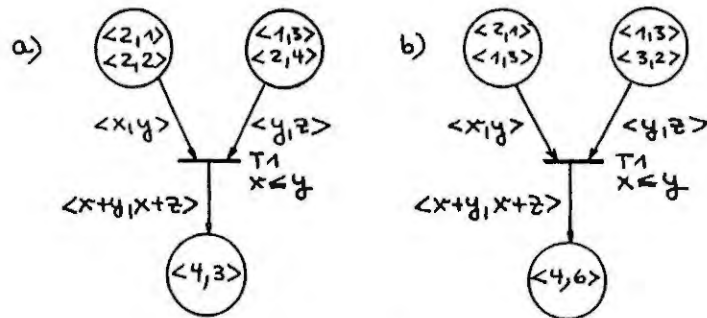


Strukturierung der Marken in Form von Tupeln. Jedes Element der Tupel repräsentiert dabei eine Zahl. Die Transition hat zweimal Konzeption:

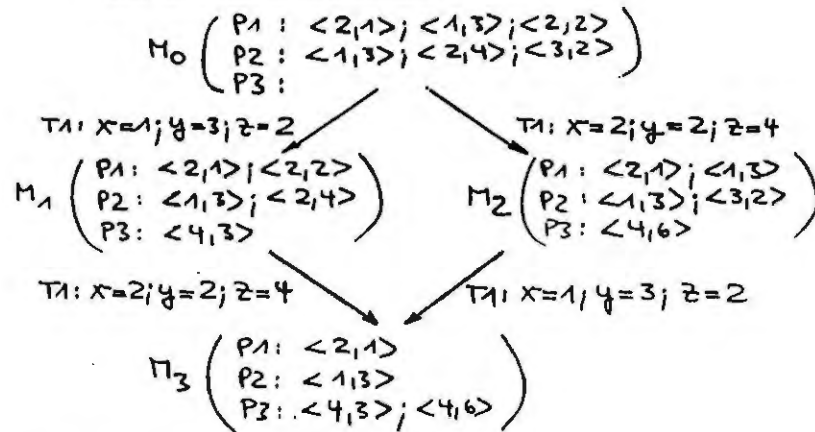
- Marke von P1 mit der Belegung $\langle 1, 3 \rangle$ und Marke von P2 mit der Belegung $\langle 3, 2 \rangle$.
- Marke von P1 mit $\langle 2, 2 \rangle$ und Marke von P2 mit $\langle 2, 4 \rangle$.

Nicht möglich P1 mit $\langle 2, 1 \rangle$ und P2 mit $\langle 1, 3 \rangle$, da die erste Tupelzahl der P1-Marken \leq der zweiten sein muß (Schaltbedingung an T1).

Folgemarkierungen



Erreichbarkeitsgraph



Die Kanten repräsentieren nicht nur eine Transition, sondern einen Schaltmodus von ihr, der deshalb mit aufgeschrieben werden muß.

Die Knoten beschreiben die individuelle Belegung der Plätze mit Marken. Die Darstellung ist noch nicht standardisiert.

Problem Organisation For Control System Simulation

1. Introduction

Many methods exist for describing a mathematical model. Sets of ordinary differential equations and algebraic equations are clearly of central importance in lumped parameter dynamic models. In the case of linear models used for control system design there are other forms of description which may be more natural, including transfer functions, block diagrams, signal flow graphs and bond graphs. All of these can be used to show, in diagrammatic form, the mathematical operations to be performed and the order in which information must be processed.

2. Descriptions for continuous-variable models: reduced and state variable forms

Two of the most widely used forms of mathematical description for dynamic systems are the so-called **reduced form** and the **state variable form**. An example of a simple model expressed in reduced form is the equation

$$M\ddot{y} + R\dot{y} + Ky = f(t) \quad (1)$$

which is the differential equation commonly used to represent the mechanical system of Figure 1 involving the displacement, $y(t)$, of a mass M which is subjected to an external force $f(t)$ while suspended by means of a linear spring of stiffness K and damping element having viscous resistance R .

The corresponding description in state variable form involves two first order equations in place of the single second order equation. It takes the form

$$\dot{x}_1 = x_2 \quad (2)$$

$$\dot{x}_2 = -\frac{K}{M}x_1 - \frac{R}{M}x_2 + \frac{f(t)}{M} \quad (3)$$

where the new variable x_1 is the displacement, $y(t)$, and the new variable x_2 is the velocity, dy/dt . These are known as the **state variables** of the system. An n th order description in reduced form is equivalent to a set of n first order ordinary differential equations in state variable form. There is no unique set of state variables for a given model in reduced form.

Physical reasoning is often used in selecting quantities to be defined as state variables. In the system of Figure 1 it is clear that displacement and velocity have advantages over other possible sets of state variables. Displacement is a variable of particular interest in the model and is also likely to be a measured quantity in the real system. Velocity is also likely to be a measurable quantity.

The two-dimensional vector having components x_1 and x_2 is known as the **state vector**. The importance of the state vector is that if the initial state is defined and the system inputs

are known all future states are defined. The two equations defining the state-space model above can be rewritten as a single vector matrix equation

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{K}{M} & -\frac{R}{M} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{M} \end{bmatrix} f(t) \quad (4)$$

or, more concisely, as

$$\dot{\mathbf{x}} = \mathbf{Ax} + \mathbf{bu} \quad (5)$$

where \mathbf{x} is the state vector, \mathbf{A} is a 2×2 square matrix and \mathbf{b} is a two-element column vector. This equation relates the rate of change of the state to the present state and the input. It is a form which is particularly convenient for simulation purposes since the numerical solution can then be obtained for each equation within the state-space model simply by a process of integration. A second order system, with two state variables, thus requires two integration operations in the corresponding simulation program; a sixth order model, with six state variables, would require six integrations.

In the case of a nonlinear model the state space model would have the form

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}(t), u(t)) \quad (6)$$

where \mathbf{f} now denotes the vector of derivative functions and $u(t)$ is the input.

3 Conversion from reduced form to state-variable form

If the model is given in the form

$$\frac{d^n y}{dt^n} = f\left[t, y, \frac{dy}{dt}, \dots, \frac{d^{n-1}y}{dt^{n-1}}, u(t)\right] \quad (7)$$

where $u(t)$ represents an input forcing function and f is a given linear or nonlinear function, it is always possible to convert the model to state space form by selecting the following as the state variables

$$x_1 = y$$

$$x_2 = \frac{dy}{dt}$$

$$x_n = \frac{d^{n-1}y}{dt^{n-1}}$$

$$x_n = \frac{d^{n-1}y}{dt^{n-1}} \quad (8)$$

In the case of a mechanical system, this could involve selecting position, velocity, acceleration etc. as the set of state variables. The resulting state space description has the form

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = x_3$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\dot{x}_{n-1} = x_n$$

$$\dot{x}_n = f[t, x_1, x_2, \dots, x_n, u(t)] \quad (9)$$

Since the choice of state variables is never unique the state variables may be numbered in any order.

4 Transfer function descriptions

Linear lumped parameter models can also be written very conveniently as transfer functions, defined as the ratio of the Laplace transform of the model output variable to the Laplace transform of the input when all initial conditions are zero. Most introductory texts dealing with control systems engineering (e.g. [1], [2]) provide a detailed account of transfer function models and their derivation.

The assumption that all initial conditions are zero when using a transfer function type of model allows a given differential equation to be transformed into the Laplace domain simply by replacing d/dt by s , d^2/dt^2 by s^2 and so on for higher derivatives. Subsequent analysis is all carried out in terms of algebraic relationships which both simplifies the mathematical operations which have to be performed and can also provide additional physical insight. For example, the mass-spring-damper system defined in Equation (1) transforms to

$$Ms^2Y(s) + RsY(s) + KY(s) = F(s) \quad (10)$$

where $Y(s)$ is the Laplace transform of $y(t)$, $F(s)$ is the Laplace transform of $f(t)$ and all

initial conditions on $y(t)$ and its derivatives are zero. The quantity $Y(s)$ can then be manipulated algebraically to give

$$(Ms^2 + Rs + K) Y(s) = F(s) \quad (11)$$

and thus

$$\frac{Y(s)}{F(s)} = \frac{1}{Ms^2 + Rs + K} \quad (12)$$

In general a transfer function $G(s)$ may be written in the form

$$\frac{Y(s)}{U(s)} = G(s) = \frac{A(s)}{B(s)} \quad (13)$$

where $Y(s)$ is the Laplace transform of the output variable, $U(s)$ is the Laplace transform of the input and where $A(s)$ and $B(s)$ are polynomials in s .

Many properties of a transfer function depend upon the denominator $B(s)$. The roots of the **characteristic equation** $B(s)=0$ largely determine the form of the output when the transfer function is subjected to a given input. These roots are the **poles** of the transfer function. The number of poles is, of course, equal to the order of the system as discussed above in the context of state variable descriptions.

5. Block diagram and signal flow graph representations

Linear systems can be described using block diagrams or signal flow graphs. Block diagrams and signal flow graphs are important for simulation since these both provide a simple means of expressing the structure of a complex model. Many simulation packages are block diagram orientated and a thorough understanding of these diagrams is essential.

In the block diagram approach to the description of a system each element is described by a single block. The arrow entering the block represents the input variable and the arrow leaving the block represents the output variable. The block contains an expression, usually in transfer function form, which relates the output to the input. Signal flow diagrams incorporate exactly the same information as the block diagram. In this case input and output variables are represented by nodes and the line connecting two nodes is the equivalent of the block in the block diagram approach. A given system can be represented at many different levels using a block diagram or signal flow graph. For example Figure 2 shows valid block diagram and signal flow graph representations for the mass-spring-damper system of Equation (1). In these diagrams the only variables which appear are the input variable $F(s)$ and the output $Y(s)$. The diagrams describe only the relationship between the chosen input variable and the chosen output variable. Figure 3 shows another level of representation. Here the diagrams involve a number of blocks or signal flow graph elements in combination and the structure provides additional information about the system. It should, of course, be noted that the diagrams of Figures 2 and 3 are exactly equivalent and Figure 2 may be derived from Figure 3 using standard rules for block diagram or signal flow graph manipulation and reduction.

6. Block diagram and signal flow graph methods for transfer function simulation

Several methods exist for deriving a set of equations in state variable form from a given transfer function. The methods given here are based on block-diagrams or signal-flow graphs. more than one approach is presented because some methods, which can be entirely satisfactory with low order models, present problems of numerical robustness when applied to higher order problems.

6.1 The direct construction approach

Most transfer functions of practical importance can be manipulated into a general form involving a ratio of two polynomials in s and a gain factor. Consider the transfer function

$$\frac{Y(s)}{U(s)} = \frac{K(s^m + a_{m-1}s^{m-1} + \dots + a_2s^2 + a_1s + a_0)}{s^n + b_{n-1}s^{n-1} + \dots + b_2s^2 + b_1s + b_0} \quad (14)$$

where $n > m$ and K is a simple gain factor. This is in the required form and the restriction that the order of the denominator should be greater than that of the numerator is connected with conditions for physical realisability.

The only transfer functions which are of general practical importance for control systems applications but which cannot be manipulated into the form shown in Equation (14) involve pure delay elements involving factors $\exp(-sT)$ or distributed delay elements involving factors $\exp(-\sqrt{s}T)$. However, such cases can be handled without difficulty if the delay elements are simple multiplicative factors which can be treated as a separate block in cascade with a block described by a transfer function involving a ratio of polynomials in s .

Dividing all terms in the numerator and denominator of the right hand side by s^n gives

$$\frac{Y(s)}{U(s)} = \frac{K(s^{m-n} + a_{m-1}s^{m-1-n} + \dots + a_2s^{2-n} + a_1s^{1-n} + a_0s^{-n})}{1 + b_{n-1}s^{-1} + \dots + b_2s^{2-n} + b_1s^{1-n} + b_0s^{-n}} \quad (15)$$

It is now possible to rewrite the relationship between the transfer function output $Y(s)$ and the input $U(s)$ to involve an intermediate variable $E(s)$. This new variable need not have any obvious physical significance and is defined from the following equation

$$\frac{Y(s)}{U(s)} = \frac{Y(s)}{E(s)} \cdot \frac{E(s)}{U(s)} \quad (16)$$

The transfer function of Equation (15) may now be split into two parts as follows

$$\frac{Y(s)}{E(s)} = K (s^{m-n} + a_{m-1}s^{m-1-n} + \dots + a_2s^{2-n} + a_1s^{1-n} + a_0s^{-n}) \quad (17)$$

and

$$\frac{E(s)}{U(s)} = \frac{1}{1 + b_{n-1}s^{-1} + \dots + b_2s^{2-n} + b_1s^{1-n} + b_0s^{-n}} \quad (18)$$

Equation (18) may be re-arranged to give

$$E(s) = U(s) - (b_{n-1}s^{-1} + \dots + b_2s^{2-n} + b_1s^{1-n} + b_0s^{-n}) E(s)$$

which corresponds to a signal flow graph of the form shown in Figure 4. Here the new variable $E(s)$ is the input to a sequence of integrator elements, the output of each of which is fed back to the input through coefficient elements. The outputs of each of the integrators, taken in order from the left, are thus $s^{-1}E(s)$, $s^{-2}E(s)$, $s^{-3}E(s)$, ..., $s^{-n}E(s)$. However from Equation (17) the output $Y(s)$ is seen to be a weighted sum of $m+1$ quantities such as these and this gives a signal flow graph such as that shown in Figure 5.

Variables from each of the integrator blocks in Figure 5 may be assigned as state variables equations in state variable form may be written down from the signal flow graph or block diagram by inspection. If the output of the final integrator is x_1 , with the other state variables taken as the outputs of each of the remaining integrator elements. The set of state equations is then as follows

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = x_3$$

$$\vdots$$

$$\dot{x}_{n-1} = x_n$$

$$\dot{x}_n = -b_0x_1 - b_1x_2 - \dots - b_{n-1}x_n + u(t) \quad (19)$$

This set of equations is similar to Equations (9). A single algebraic equation relating the output y to the state variables must now be added. It is clear from the block diagram, that this equation is

$$y = K(a_0x_1 + a_1x_2 + \dots + a_{m-1}x_{n-m} + x_{n-m+1}) \quad (20)$$

Equations (19) and (20) together provide a simple way for implementing the given transfer function within a simulation. The only disadvantage of this direct construction approach is that the properties of the resulting state space model can be highly sensitive to small numerical inaccuracies in coefficient values, especially in the case of high-order models.

6.2 The parallel construction approach

The parallel programming approach is appropriate when the given transfer function has a denominator in factored form as shown below

$$\frac{Y(s)}{U(s)} = \frac{K(s^m + a_{m-1}s^{m-1} + \dots + a_2s^2 + a_1s + a_0)}{(s + \beta_1)(s + \beta_2) \dots (s + \beta_n)} \quad (21)$$

Using partial fractions it is then possible to express the right hand side of this equation as a sum of terms each of which involves a single pole. The resulting equation is

$$\frac{Y(s)}{U(s)} = \frac{\alpha_1}{s + \beta_1} + \frac{\alpha_2}{s + \beta_2} + \dots + \frac{\alpha_n}{s + \beta_n} \quad (22)$$

Each term on the right hand side of the equation has the same form and the corresponding signal flow graph involves a parallel structure as shown in Figure 6. The state variables are taken to be the outputs of each integrator. The state equations describing the signal flow graph are then of the form

$$\begin{aligned} \dot{x}_1 &= -\beta_1 x_1 + u(t) \\ \dot{x}_2 &= -\beta_2 x_2 + u(t) \\ &\vdots \\ \dot{x}_n &= -\beta_n x_n + u(t) \end{aligned} \quad (23)$$

An additional algebraic equation relates the output y to the n state variables. This has the form

$$y = \alpha_1 x_1 + \alpha_2 x_2 + \dots + \alpha_n x_n \quad (24)$$

A model with a parallel structure having first-order (or second-order) blocks is generally more robust in terms of numerical sensitivity to coefficient inaccuracies than the equivalent model in direct construction form. The higher the order of the original transfer function description the greater are the benefits of the parallel approach.

6.3 The iterative construction approach

If a transfer function has only real poles and zeros the numerator and denominator may both be factorised into products of first-order factors to give a description of the following form:

$$\frac{K(s+\alpha_1)(s+\alpha_2)\dots(s+\alpha_m)}{(s+\beta_1)(s+\beta_2)\dots(s+\beta_n)} \quad (25)$$

where K , α_i and β_j are all real constants. Grouping the factors into products of terms of the form

$$\frac{(s+\alpha_i)}{(s+\beta_j)} \quad (26)$$

it is possible to construct a sub-model block diagram or signal flow graph which represents a single factor of the complete transfer function. This sub-model diagram has the form shown in Figure 7. Since the order of the denominator is always equal to or greater than the order of the numerator of the transfer function for physically realisable systems. Hence, when the process of grouping these factors in pairs has been completed, there may be some additional terms involving denominators of the form $(s + \beta_k)$ which cannot be combined with any numerator factors. In such cases the additional factors are dealt with by means of the sub-model signal flow graph element in Figure 8. The form of the diagram for the complete transfer function for the special case where the numerator is of the same order as the denominator is shown in Figure 9. The state variables are the output variables of the integrator blocks and the resulting state equations are as follows:

$$\dot{x}_n = -\beta_n x_n + u \quad (27)$$

$$\dot{x}_{n-1} = -\beta_{n-1} x_{n-1} + \alpha_n x_n + \dot{x}_n = -\beta_{n-1} x_{n-1} + (\alpha_n - \beta_n) x_n + u \quad (28)$$

.
.

$$\dot{x}_{n-j} = -\beta_{n-j} x_{n-j} + (\alpha_{n-j+1} - \beta_{n-j+1}) x_{n-j+1} + \dots + (\alpha_n - \beta_n) x_n + u \quad (29)$$

.
.

$$\dot{x}_1 = -\beta_1 x_1 + (\alpha_2 - \beta_2) x_2 + \dots + (\alpha_n - \beta_n) x_n + u \quad (30)$$

As with the parallel construction approach the iterative method is often superior to the direct construction method in terms of coefficient sensitivity. A cascaded arrangement of first or second order transfer functions tends to be more robust to coefficient errors than the

equivalent structure involving multiple feedback loops.

6.4 An example of block diagram construction from a transfer function

Consider the following transfer function:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{s^2 + 9s + 20}{s^3 + 6s^2 + 11s + 6} \quad (31)$$

Direct construction method

First of all the numerator and denominator polynomials on the right hand side of the equation should be divided by the highest power of s to give

$$G(s) = \frac{Y(s)}{U(s)} = \frac{s^{-1} + 9s^{-2} + 20s^{-3}}{1 + 6s^{-1} + 11s^{-2} + 6s^{-3}} \quad (32)$$

Introducing a dummy variable $E(s)$ gives

$$\frac{Y(s)}{E(s)} = s^{-1} + 9s^{-2} + 20s^{-3} \quad (33)$$

and

$$\frac{E(s)}{U(s)} = \frac{1}{1 + 6s^{-1} + 11s^{-2} + 6s^{-3}} \quad (34)$$

That is

$$E(s) = U(s) - (6s^{-1}E(s) + 11s^{-2}E(s) + 6s^{-3}E(s)) \quad (35)$$

and

$$Y(s) = s^{-1}E(s) + 9s^{-2}E(s) + 20s^{-3}E(s) \quad (36)$$

Equations (35) and (36) may be expressed in block diagram form by a model involving three integrators connected in cascade as shown in Figure 10. Taking the outputs of integrator blocks as state variables allows the following set of simultaneous first order differential equations to be established:

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = x_3$$

$$\dot{x}_3 = -6x_1 - 11x_2 - 6x_3$$

or, in matrix notation,

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6 & -11 & -6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u \quad (37)$$

The output equation is

$$y = 20x_1 + 9x_2 + x_3$$

or

$$y = [20 \ 9 \ 1] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad (38)$$

Parallel construction method

In this case the first step involves factorising the denominator of the given transfer function to give:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{s^2 + 9s + 20}{s^3 + 6s^2 + 11s + 6} = \frac{s^2 + 9s + 20}{(s+1)(s+2)(s+3)} \quad (39)$$

Expressing the right hand side of this equation in partial fraction form gives:

$$\frac{Y(s)}{U(s)} = \frac{6}{s+1} - \frac{6}{s+2} + \frac{1}{s+3} \quad (40)$$

and this may be represented as a block diagram with the parallel structure shown in Figure

9. Again the outputs of the integrator blocks are taken as the state variables to give the following set of first-order differential equations

$$\dot{x}_1 = -x_1 + u; \quad \dot{x}_2 = -2x_2 + u; \quad \dot{x}_3 = -3x_3 + u$$

and an algebraic equation relating the output, $y(t)$, to the three state variables $x_1(t)$, $x_2(t)$ and $x_3(t)$ which has the form

$$y = -6x_1 - 6x_2 - x_3$$

In matrix form the state and output equations are thus

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} u \quad (41)$$

and

$$y = [6 \quad -6 \quad 1] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad (42)$$

Iterative construction method

There are two zeros and three poles in the given transfer function. Hence the two zeros can be associated with two of the poles in the standard way for the iterative approach and one pole must be treated separately, as shown in the transfer function below

$$\frac{Y(s)}{U(s)} = \frac{s^2 + 9s + 20}{s^3 + 6s^2 + 11s + 6} = \left(\frac{s+5}{s+1} \right) \left(\frac{s+4}{s+2} \right) \left(\frac{1}{s+3} \right) \quad (43)$$

The resulting block diagram thus has two stages involving elements of the type shown in Figure 7 and one stage of the type shown in Figure 8. Figure 12 shows the complete block diagram. Taking outputs of integrator blocks as state variables gives a set of first order differential equations

$$\begin{aligned} \dot{x}_1 &= -3x_1 + 4x_2 + \dot{x}_2 \\ &= -3x_1 + 2x_2 + 4x_3 + u \end{aligned} \quad (44)$$

$$\dot{x}_2 = -2x_2 + 5x_3 + \dot{x}_3$$

$$= -2x_2 - 4x_3 + u \quad (45)$$

$$\dot{x}_3 = -x_3 + u \quad (46)$$

and an algebraic output equation

$$y = x_1 \quad (47)$$

7. Modelling of distributed parameter elements

7.1 Introduction

Distributed parameter models arise in the study of systems in which quantities are transmitted from one point to another, as in the case of electrical signals in a transmission line, flow of fluid in a pipe, phenomena associated with the conduction of heat etc.. The "signals" in such systems are distributed in space as well as time and the mathematical models are described by partial differential equations.

In situations in which the system reduces essentially to a single input variable and a single output variable of interest at one point in space, as often applies in control systems applications, it is possible to derive a transfer function type of description which involves a distributed parameter. Such distributed parameter transfer function models may then be used in the same way as conventional transfer functions for lumped parameter models. The two types of distributed parameter element considered in this section are the pure time delay and the distributed time delay.

7.2 Pure Time Delay

A pure time delay (known also as a transportation lag, or "dead time") arises in systems in which quantities are transmitted at finite velocity from one point to another. In the case of a pure delay this transmission does not involve any frequency dependent change of signal amplitude.

A system involving a pure time delay (due perhaps to physical phenomena such as heat flow but where losses can be neglected), may be described approximately by a partial differential equation of the form

$$\frac{\partial z}{\partial t} = v \frac{\partial z}{\partial x} \quad (48)$$

where $z(x,t)$ is the state of a point at position x and time t . The parameter v is the velocity of transmission.

Applying the Laplace transformation to this equation gives, assuming zero initial conditions, a new equation as follows

$$s Z(x, s) = v \frac{\partial Z(x, s)}{\partial x} \quad (49)$$

where

$$Z(x, s) = \int_0^{\infty} \exp(-st) z(x, t) dt \quad (50)$$

The solution of this equation is

$$Z(x, s) = A \exp\left(-s \frac{x}{v}\right) \quad (51)$$

where A is an arbitrary constant.

If $U(s)$ is the Laplace transform of the input of the system at point x_1 , and $Y(s)$ is the transform of the output of the element at point x_2 then it follows that

$$U(s) = Z(x_1, s) = A \exp\left(-s \frac{x_1}{v}\right) \quad (52)$$

and

$$Y(s) = Z(x_2, s) = A \exp\left(-s \frac{x_2}{v}\right) \quad (53)$$

Thus the transfer function relating the output transform $Y(s)$ to the input transform $U(s)$ is

$$\frac{Y(s)}{U(s)} = \exp\left(-s \frac{x_2 - x_1}{v}\right) \quad (54)$$

That is

$$\frac{Y(s)}{U(s)} = \exp(-sT) \quad (55)$$

where T is the transmission time from point x_1 to point x_2 . The parameter T is thus the duration of the pure time delay.

7.3 Distributed time delay

A distributed time delay arises where quantities are transmitted from one point to another with a finite velocity but with an attenuation which varies with frequency. examples include the conduction of heat and the transmission of electrical signals in a medium which is not loss free. The partial differential equation describing this situation has the form

$$\frac{\partial z}{\partial t} - a \frac{\partial^2 z}{\partial x^2} \quad (56)$$

where $z(x,t)$ represents the value of the quantity concerned at point x and time t . The parameter a is a constant and could represent, for example, thermal conductivity. Applying the unilateral Laplace transform for zero initial conditions gives

$$sZ(x, s) = a \frac{\partial^2 Z(x, s)}{\partial x^2} \quad (57)$$

where

$$Z(x, s) = \int_0^{\infty} \exp(-st) z(x, t) dt \quad (58)$$

The solution of this equation has the form

$$Z(x, s) = A \exp\left(-\sqrt{\frac{s}{a}} x\right) + B \exp\left(\sqrt{\frac{s}{a}} x\right) \quad (59)$$

It is now assumed that at infinite distance the value of the variable $z(x,t)$ is always zero. This allows one of the terms in the above equation to be eliminated, giving

$$Z(x, s) = A \exp\left(-\sqrt{\frac{s}{a}} x\right) \quad (60)$$

for $x \geq 0$.

If $U(s)$ is the Laplace transform of the input to the system being modelled (at point x_1) and $Y(s)$ is the Laplace transform of the output of the system (at point x_2) then it follows that

$$U(s) = Z(x_1, s) = A \exp\left(-\sqrt{\frac{s}{a}} x_1\right) \quad (61)$$

Similarly

$$Y(s) = Z(x_2, s) = A \exp\left(-\sqrt{\frac{s}{a}} x_2\right) \quad (62)$$

Hence the transfer function relating $Y(s)$ to $U(s)$ has the form

$$\frac{Y(s)}{U(s)} = \exp\left(-\sqrt{\frac{s}{a}} (x_2 - x_1)\right) \quad (63)$$

Thus

$$\frac{Y(s)}{U(s)} = \exp\left(-\sqrt{s \frac{(x_2 - x_1)^2}{a}}\right) = \exp(-\sqrt{sT}) \quad (64)$$

where the parameter T is an equivalent time.

It should be noted that elimination of the term involving the positive exponent in this derivation has physical significance. It is equivalent to rejecting the possibility of reflected waves in the system under consideration. The use of the transfer function resulting from this analysis is therefore restricted to cases in which reflected waves are not expected.

7.4 Simulation models involving pure and distributed delay elements

Most simulation languages incorporate facilities for the representation of pure delays. Pure delays also present few difficulties in a simulation which is developed using a general purpose high level language, provided the delay does not itself vary with time. Cases involving variable delays present additional problems [3]. An alternative approach, and more approximate representation for a pure delay, is based upon the properties of Padé approximations. These approximations, based upon truncated series for the exponential function, were widely used for the representation of pure delays in analog simulations and it is equally possible to use this approach in a digital simulation.

More complex problems involving distributed parameter elements can be approached in a very general way using finite differences or finite element methods or other numerical techniques for the solution of partial differential equations. Detailed treatments of these specialised topics can be found in appropriate textbooks and a review of some of the problems of simulation of such systems may be found in the book by Spriet and Vansteenkiste [4]. Simulations involving distributed parameter elements can be numerically intensive and for time-critical applications distributed parameter problems are often reduced to quite simple lumped-parameter approximations. One example of this type can be found in the work of Bryce et al. [5] which is concerned with the real-time simulation of a water pipeline as part of an investigation of water-turbine governing systems.

References

- [1] Palm, W.J., "Control Systems Engineering", John Wiley & Sons, New York, U.S.A., 1986.
- [2] Golten, J. and Verwer, A., "Control System Design and Simulation", McGraw-Hill, London, U.K., 1991.
- [3] Doebelin, E.O. "System Modelling and Response", pp. 193-201, J. Wiley & Sons, New York, U.S.A., 1980.
- [4] Spriet, J.A. and Vansteenkiste, G.C. "Computer-aided Modelling and Simulation", Academic Press, London, U.K., 1982.
- [5] Bryce, G.W., Foord, T.R., Murray-Smith, D.J. and Agnew, P.W. 'Hybrid simulation of water-turbine governors', Simulation Councils Proceedings, Vol. 6, Part 1, pp. 35-44, 1976.

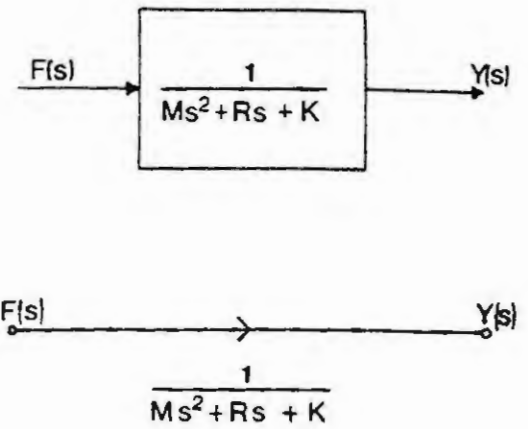
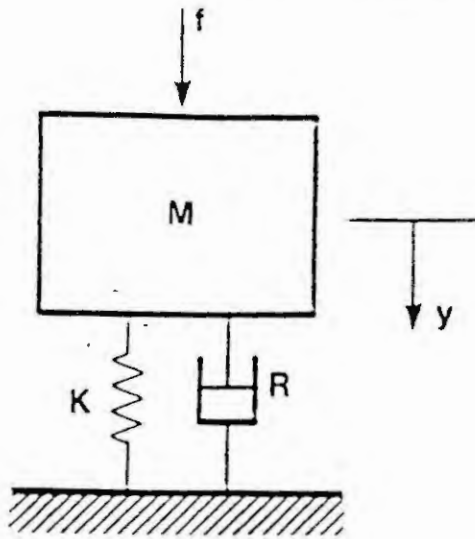


Figure 1: Mechanical system involving mass, spring and viscous damping elements.

Figure 2: Transfer function based block diagram and signal flow graph for system of Figure 1.

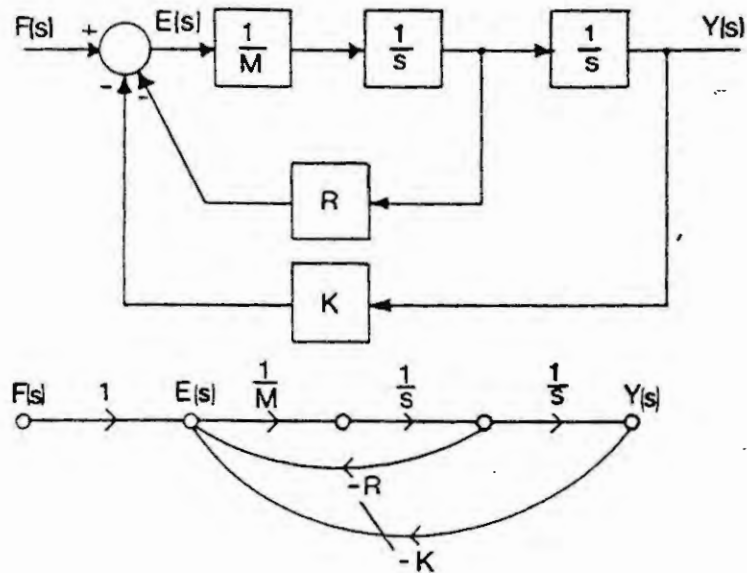


Figure 3: Detailed block diagram and signal flow graph for system of Figure 1.

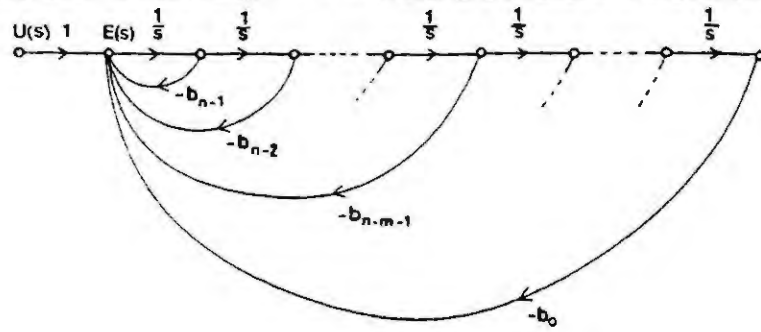


Figure 4: Direct construction signal flow graph for denominator terms.

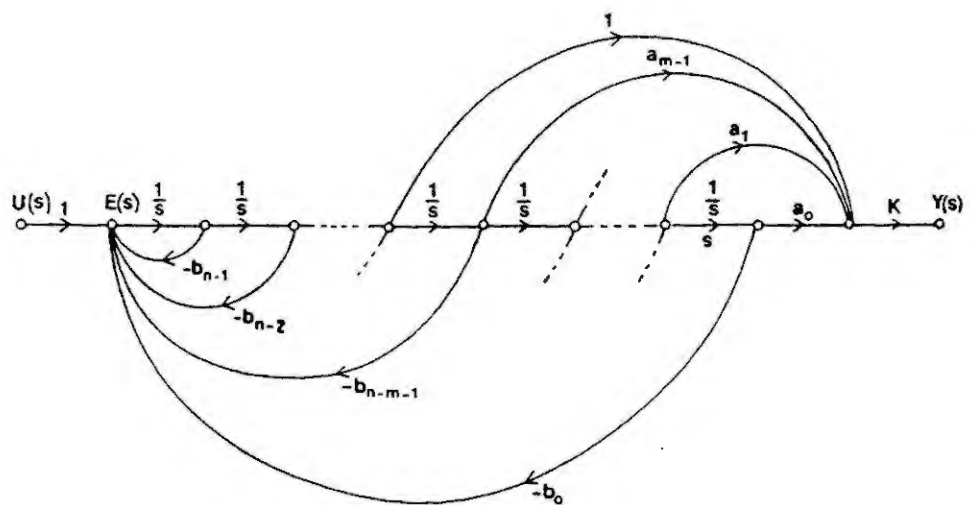


Figure 5: Complete signal flow graph for direct construction representation.

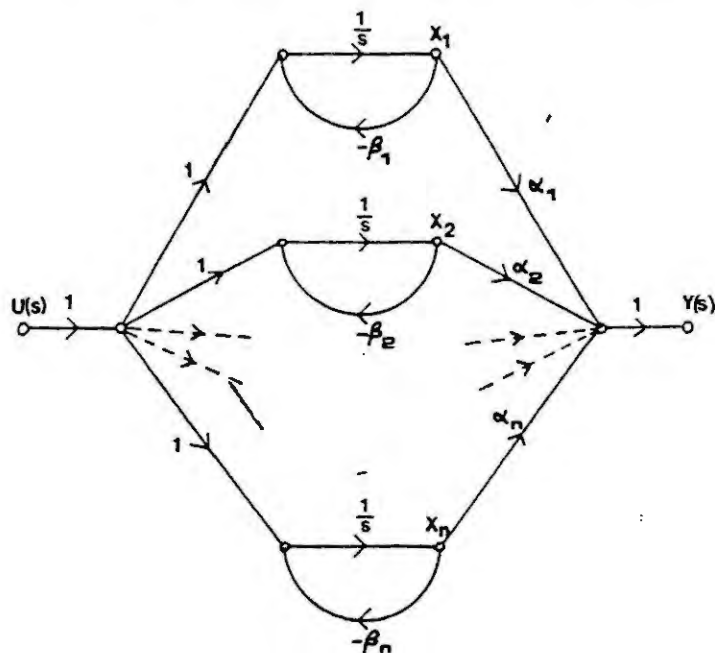


Figure 6: Signal flow graph illustrating parallel construction method.

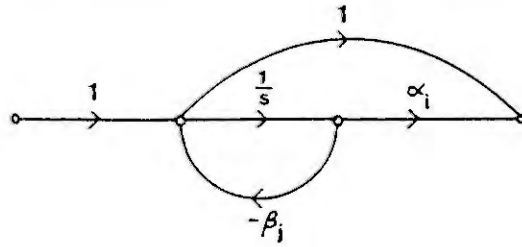


Figure 7: Signal flow graph for single pole-zero pair.

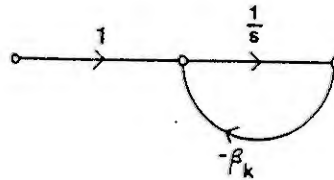


Figure 8: Signal flow graph for single pole.

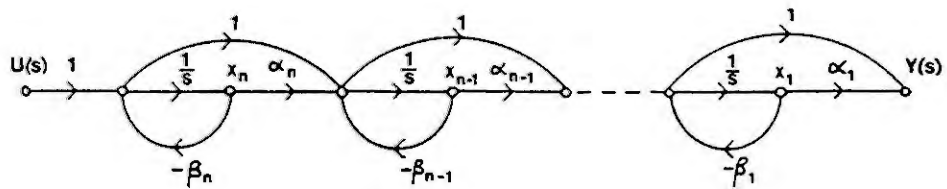


Figure 9: Signal flow graph illustrating iterative construction method.

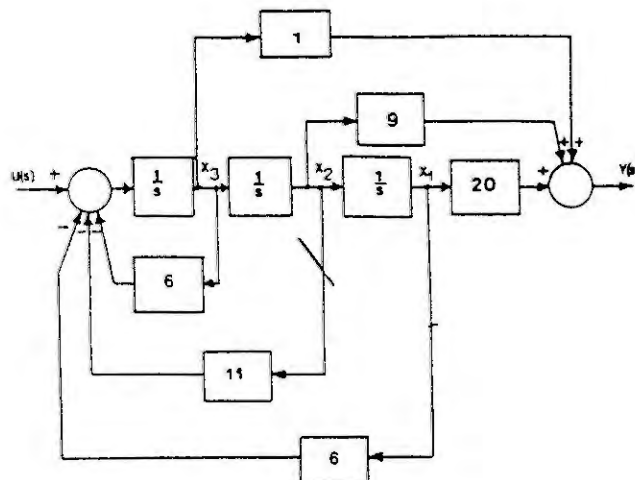


Figure 10: Block diagram for example system by direct construction method.

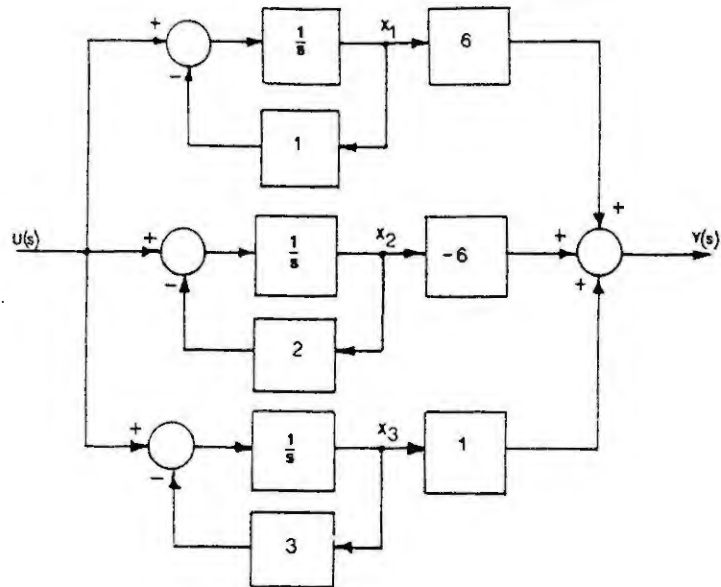


Figure 11: Block diagram for example system by parallel construction method.

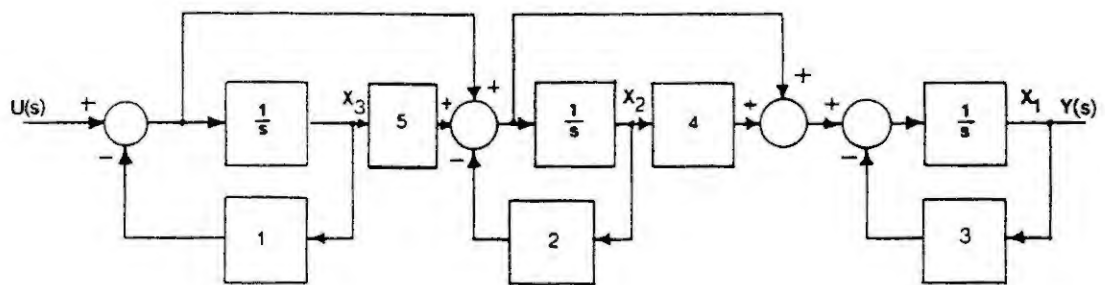


Figure 12: Block diagram for example system by iterative construction method.

Simple Examples of Control System Simulation Using Common Simulation Tools

1. Introduction

This section of the course provides examples of a number of different types of simulation problem involving control system applications. The models upon which the simulations are based are relatively simple and have been chosen to illustrate features of some typical simulation tools which are currently available. Further details of the SLIM simulation language may be found in the Users' Manual and in an associated textbook [1] which also provides some additional linear and nonlinear simulation examples. The underlying principles which can be applied both in simulation studies of systems developed using classical methods of control system design and systems designed using fuzzy control principles. Particular simulation techniques have been selected but this choice is not intended to suggest that other tools could not have provided an equally appropriate solution but strengths and weaknesses of some methods of approach are discussed. In each case suggestions are made of further investigations which could be made with these simple models.

2. A control system simulation: speed control of a water turbine

Control systems investigations form one of the most common application areas for continuous system simulation tools. This example is concerned with a simulation of a speed control system for a water turbine used for the generation of electricity. Figure 1 is a block diagram of a highly simplified description of a system of this kind. The turbine, which is of the impulse type, is represented by a linear transfer function which also incorporates dynamics of the pipeline system. Control of the turbine is accomplished through feedback of a signal proportional to turbine shaft speed and comparison with the desired (reference) speed signal. Any difference between the desired and actual speed provides an error signal which is processed by the controller block to provide the signal applied to the turbine actuator input to change the water flow in such a way that the error is continuously minimised. The controller may be implemented either in analogue (continuous) or digital form. The model is linear throughout, apart from backlash in the mechanical linkages associated with the turbine inlet actuator. The purpose of the model is to allow the performance of the model of the closed loop system to be investigated using both continuous and digital controllers. This is a useful illustration of continuous and discrete simulation techniques being used together to approach an important class of practical engineering problem.

A number of controller transfer functions can be considered for this type of application. Detailed consideration of possible forms of controller are not of primary interest in the present context but any readers interested in this aspect, or in the underlying model, may find further details elsewhere (e.g. [1-2]). For the present purposes the controller considered is of the "temporary droop" type implemented in continuous form. This type of controller has a transfer function of the form:-

$$C(s) = \frac{1 + sT_x}{\sigma + (\sigma + \mu) T_x s + T_x T_y s^2} \quad (1)$$

In this controller transfer function only the parameters σ and μ may be regarded as adjustable quantities. The other parameters (T_i and T_d) are fixed quantities and are not available for controller tuning.

Appropriate parameter values for the plant and controller are as follows:

$T_i = 7.0$ sec. (inertial time constant)

$T_w = 1.1$ sec. (water time constant)

$T_s = 0.2$ sec. (actuator servo time constant)

All model variables, such as turbine speed are expressed as normalised (per unit) quantities.

Figure 2 shows the DYNAMIC segment of a SLIM program (TURB.SLI) for the simulation model and the full source may be found on the diskette. Parameter values for the nominal set of conditions are provided in the program listing. The simulation experiment in this case involves investigation of the response of the control system to a step change of the reference speed. It can be seen from Figure 3 that the response is stable but oscillatory in nature for the controller parameters values used. A program TURB2.SLI, which allows input from a data file, is also included in the Appendix to this section of the notes. This version of the simulation program, together with the appropriate data files in the same format as the test file TURB1.IN, provides a convenient means of experimenting with controller parameter values.

One interesting extension to this simulation program involves the inclusion of mechanical backlash between the servomotor and the turbine inlet. This is an important feature of real mechanical systems of this kind and is known to have a destabilising effect on the overall control system. Backlash is a double-valued form of nonlinearity which has a steady-state input-output characteristic of the form shown in Figure 3. It generally arises because of 'slack' in mechanical linkages and gears. With backlash present movement of the input in one direction produces a proportional movement of the output, but any reversal of the direction of the input will cause the output to stop before following the input again.

The representation of backlash and other similar double-valued nonlinearities which involve a form of "memory", such as hysteresis, can be difficult. One approach is to use principles first established for the modelling of backlash elements using analogue computers [3]. This involves the use of an integrator with feedback to provide the memory element. Figure 4 shows a block diagram for a representation of backlash by this type of method. Essentially the system works by comparing the integrator output, which is the output variable for the backlash element, with the input. The integrator input is controlled through comparator elements and logic blocks and when the input reverses direction (at the extremes of travel) the integrator input becomes zero. The integrator input remains zero until the input variable has changed by an amount equal to the width of the backlash element. It is then switched back to the output of the summing element. This causes the integrator output to remain fixed in value for a period of time and this corresponds to the flat top and bottom sections on the diagram showing the input-output characteristics for the backlash. Although presented here in block-diagram form the implementation of a backlash model of this kind is quite simple using the equation-oriented methods. Figure 5 shows the DYNAMIC segment of a SLIM program TURB2.SLI which is a version of the turbine speed control system model with backlash incorporated. This program and the corresponding data file TURB2.IN are listed in the Appendix.

The oscillation observed on the response shown in Figure 2 is influenced by the backlash parameter a_b as well as by the parameters of the controller. A well damped step response in the absence of backlash ($a_b=0.0$) can become a maintained oscillation if the backlash is increased sufficiently (Figure 6).

Figure 7 is an ACSL program listing for this example. Note that a special statement BCKLSH is available in ACSL to represent backlash effects and use has been made of this facility. Figure 8 shows results obtained from this ACSL program. In modern block-oriented tools such as XANALOG and SIMULINK special facilities are also available for the simulation of nonlinear elements such as backlash. Figure 9 shows an XANALOG block diagram for this problem.

Further investigations which could be carried out using this model and the SLIM simulation programs include determination of the minimum value of the backlash parameter b which gives rise to a maintained oscillation in terms of turbine speed. This "limit cycling" type of behaviour is clearly an undesirable situation in a speed control system and one that should be avoided in practice. For those with an appropriate level of understanding of control systems analysis techniques the simulation result for the critical value of the backlash parameter may be compared with predictions from theory, based upon describing function analysis. It is interesting to consider the reasons for any differences between the simulation result and the value predicted by theory. It should be remembered that the describing function approach involves some important simplifying assumptions and approximations; the significance of some of these can be investigated easily using the simulation.

3. Simulation of a simple digital control system

Figure 10 is a block diagram of a simple closed-loop digital control system. The difference between the reference signal and the plant output forms an error which is sampled by the digital-to-analogue converter. The digital processor carries out some form of numerical operation on the error samples and provides an actuating signal to the plant input through the digital-to-analogue converter. Figure 11 shows part of the corresponding SLIM program listing for the simplest possible situation in which the control computer simply samples the error signal and outputs the sampled error values periodically as input to the plant. The complete program file, named DIGCON.SLI, and the necessary input data file DIGCON.IN, can be found on the diskette. Examination of the DIGCON.SLI shows that, because SLIM does not have special facilities for mixed continuous and discrete system simulation, the sampling period is defined as a multiple of the communication interval parameter and the facilities of the DYNAMIC segment and DERIVATIVE section are used to emulate the discrete action of the controller. The sampled variable is held constant in a zero-order hold type of action by a loop within the DYNAMIC segment. Any discrete calculations representing the action of a control algorithm within the control computer must also be performed within the DYNAMIC segment. Values of variables of interest in the simulation may be written to the output file in the usual way, at times set by the communication interval.

Figure 12 shows results obtained for three different sampling periods. It can be seen that the system shows an increasing tendency to oscillate as the sampling period is increased and eventually becomes unstable. Sampled data theory (see, for example [14]) predicts that for this system instability occurs when the sampling period is greater than 0.549 sec.. This is consistent with the results presented in Figure 12 and detailed investigations using the

simulation program can confirm the theoretical result more precisely.

The DIGCON.SLI program can be modified easily to allow for more complex controller action. For example, if one wanted to simulate the system with a controller which implemented a difference equation of the form

$$O(kT) = O((k-1)T) + I(kT) - 0.5I((k-1)T) \quad (2)$$

the changes to the simulation program would all be made in the initial part of the DYNAMIC segment. Figure 13 shows the relevant part of the listing and the complete program is listed in the Appendix as the file DIGCON1.SLI. Note how the discrete input and output variables are stored for one sample period and updated. The transfer function of the controller of Equation (2), expressed in terms of z transforms, is as follows:

$$\frac{O(z)}{I(z)} = \frac{1 - 0.5z^{-1}}{1 - z^{-1}} = \frac{z - 0.5}{z - 1} \quad (3)$$

In the special case when the sample period is 0.347 sec. the controller should, from sampled-data theory, act as a "dead beat" compensator [4]. In such a situation the plant output should exhibit zero steady state error and should rise to its final value, in response to a step change of system reference input, in one sampling period. Figure 14 shows results from the simulation program which are consistent with theory for this special case.

Some other equation-oriented simulation languages, such as ACSL, include special facilities which can be very useful for the simulation of digital control systems. In ACSL, for example, DISCRETE sections representing the difference equations or z-transfer function of a digital controller may be inserted within the DYNAMIC segment. Such DISCRETE sections are thus similar to DERIVATIVE sections but communicate with the continuous parts of the simulation at regular predetermined times. Figures 15 and 16 show XANALOG and SIMULINK block diagrams for this digital control problem and illustrate some more of the specialised blocks and icons available with these simulation tools.

This example offers any reader interested in automatic control systems many opportunities for experimentation. It is clear from the listing of the SLIM program DIGCON1.SLI that, with some minor changes to the DYNAMIC segment, it would be very easy to replace the dead beat compensator by some other form of controller. Similarly any other form of plant transfer function could be used in place of the one given in Figure 10, with only some simple changes to the DERIVATIVE section of the program being necessary.

References

- [1] Murray-Smith, D.J. "Continuous System Simulation", Chapman and Hall, London, 1995.
- [2] Bryce, G.W., Foord, T.R., Murray-Smith, D.J. and Agnew, P., 'Hybrid simulation of water turbine governors', Simulation Council Proceedings, Vol.6, Part 1, pp. 35-44, 1976.
- [3] Ricci, F.J. "Analog-Logic Computer Programming and Simulation", Spartan Books 1972.
- [4] Leigh, J.R., "Applied Digital Control", Prentice-Hall Intl., Englewood Cliffs, N.J., U.S.A., 1984.

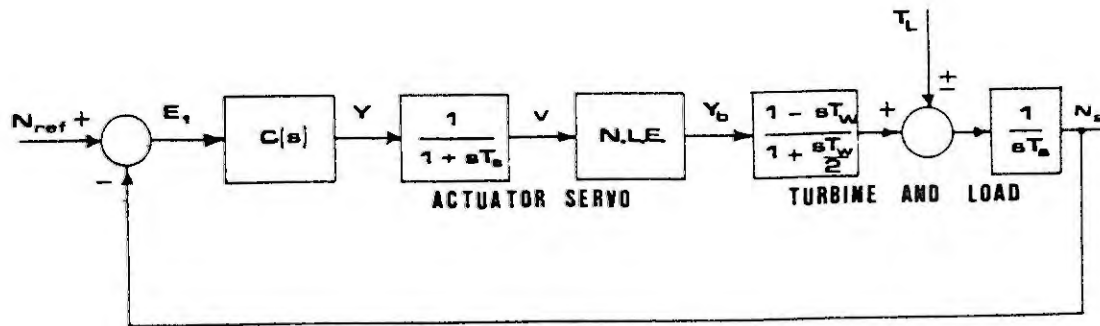


Figure 1: Block diagram of closed-loop system for automatic control of speed of water turbine connected to an electrical generator.

```

C
C *****Start of Dynamic Segment*****
C
C   DYNAMIC
C
C *****Start of Derivative Segment*****
C
C   DERIVATIVE
C     DT1=(V-T1)*(2.0/TW)
C     T1=INTEG(DT1,T10)
C     T0=T1-TW*DT1
C     TA=T0-TL
C     DNS=TA/TIA
C     ANS=INTEG(DNS,ANS0)
C     E1=REF-ANS
C     DDY1=(E1-SIG*Y1-((SIG+AMU)*TX+TY)*DY1)/(TX*TY)
C     DY1=INTEG(DDY1,DDY10)
C     Y1=INTEG(DY1,Y10)
C     Y=Y1+TX*DY1
C     DV=(Y-V)/TS
C     V=INTEG(DV,V0)
C   DERIVATIVE END
C
C *****End of Derivative Section*****
C
C   Values of t, ref, tl and ans for current communication
C   interval output to results file
C
C     TYPE T,REF,TL,ANS
C
C   Test for end of simulation run
C
C     IF(T-TMAX)10,10,12
C   10   DYNAMIC END
C
C *****End of Dynamic Segment*****
C
C *****Terminal Segment*****
C   12   STOP
C       END

```

Figure 2: Part of SLIM program TURB.SLI for simulation of the turbine speed control system with a temporary-droop type of governor.

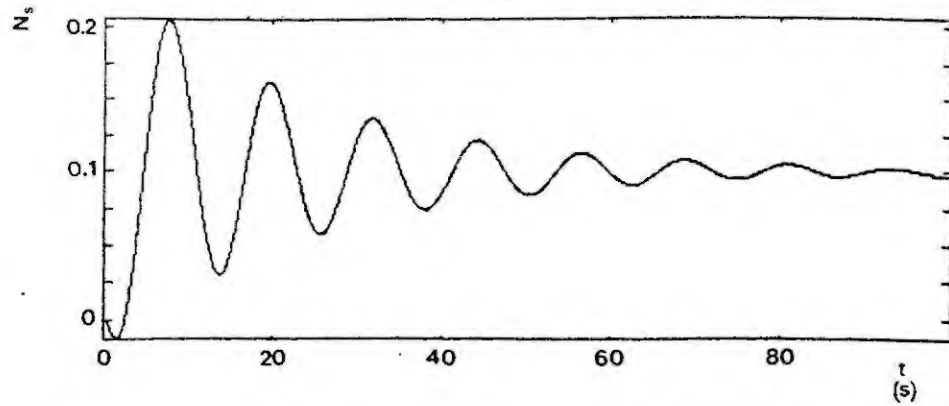


Figure 3: A typical simulated response of the speed control system to a step change in reference speed.

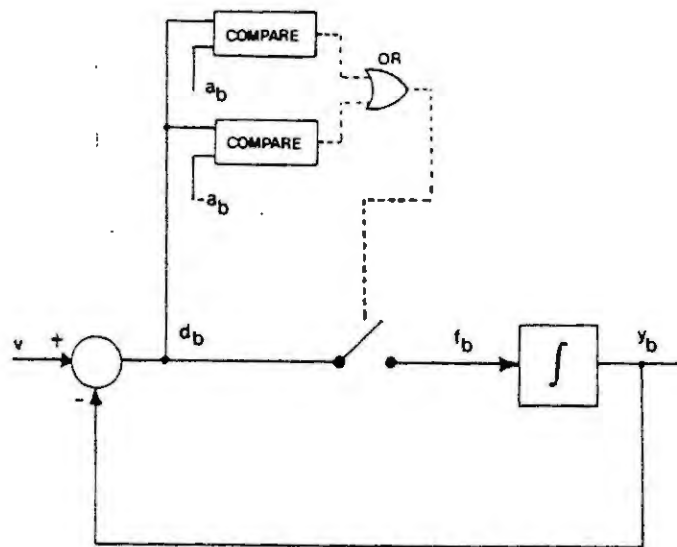


Figure 4: Block diagram illustrating one technique for representation of backlash element.

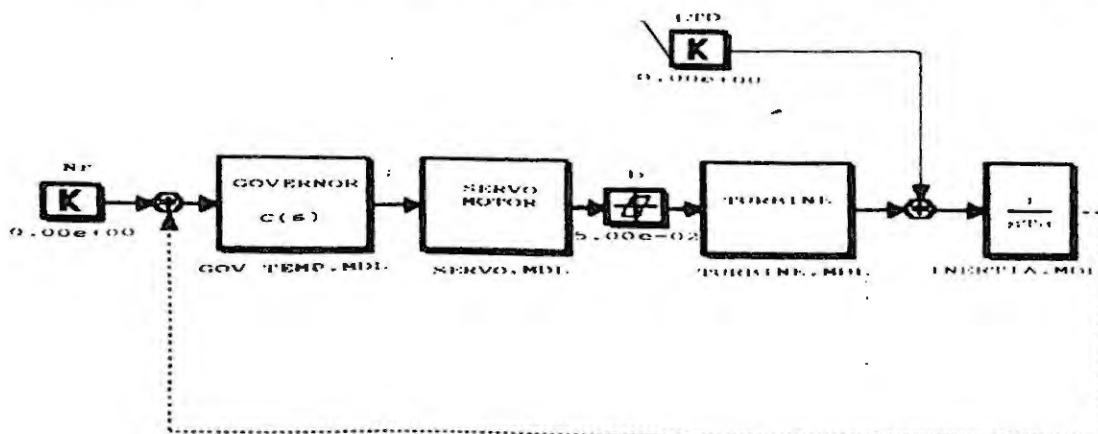


Figure 9: XANALOG block diagram of speed control problem.

```

C          SIMULATION UNDER DIDAKTIK
C *****Start of Dynamic Segment*****
C
C      DYNAMIC
C
C *****Start of Derivative Segment*****
C
C      DERIVATIVE
C
C      Section for representation of backlash element
C
C          IF(AB-0.001)40,45,45
45      DB=V-YB
C          IF(DB+AB)60,50,50
60      FB=DB
C          GOTO 100
50      IF(DB-AB)80,70,70
70      FB=DB
C          GOTO 100
80      FB=0.0
100     YB=INTEG(10.0*FB,YB0)
C          GOTO 110
40      YB=V
C
C      End of section for backlash
C
110     DT1=(YB-T1)*(2.0/TW)
C          T1=INTEG(DT1,T10)
C          T0=T1-TW*DT1
C          TA=T0-TL
C          DNS=TA/TIA
C          ANS=INTEG(DNS,ANS0)
C          E1=REF-ANS
C          DDY1=(E1-SIG*Y1-((SIG+AMU)*TX+TY)*DY1)/(TX*TY)
C          DY1=INTEG(DDY1,DDY10)
C          Y1=INTEG(DY1,Y10)
C          Y=Y1+TX*DY1
C          DV=(Y-V)/TS
C          V=INTEG(DV,V0)
C      DERIVATIVE END
C
C *****End of Derivative Section*****
C
C      Values of t, ref, tl, ans and yb for current communication
C      interval output to results file
C
C          TYPE T,REF,TL,ANS,YB
C
C      Test for end of simulation run
C
C          IF(T-TMAX)10,10,12
10      DYNAMIC END
C
C *****End of Dynamic Segment*****
C
C *****Terminal Segment*****
12      STOP
C          END

```

Figure 5: Part of SLIM program TURB2.SLI for simulation of the turbine speed control system with a temporary-droop type of governor and backlash between the servomotor and the turbine inlet valve.

PROGRAM turb2.csl

"Simulation of governed hydro-turbine system with"
 "a temporary-droop governor and with backlash between"
 "servo-motor and turbine inlet actuator"

INITIAL

"Data for plant and governor transfer functions"

CONSTANT tw=1.1, ts=0.2, tia=7.0

CONSTANT ab=0.06 \$ "backlash parameter"

CONSTANT tx=16.0, ty=0.3, sig=0.03, mu=0.25

"Data for reference input and disturbance input"

CONSTANT ref=0.1, tl=0.0

"Data for experiment duration"

CONSTANT tend=99.9

ALGORITHM IALG=4 \$ "Runge-Kutta second order"

CINTERVAL CINT=1.0

"Initial conditions"

tl=0.0

v=0.0

y1=0.0

dyl=0.0

yb0=0.0

END \$ "of INITIAL"

DYNAMIC

DERIVATIVE

"Turbine"

dtl=(yb-tl)*(2.0/tw)

tl=INTEG(dtl,0.0)

t0=tl-tw*dtl

"Load"

ta=t0+tl

dns=ta/tia

ns=INTEG(dns,0.0)

"Governor"

e1=ref-ns

ddyl=(e1-sig*y1-((sig+mu)*tx+ty)*dyl)/(tx*ty)

dyl=INTEG(ddyl,0.0)

y1=INTEG(dyl,0.0)

y=y1+tx*dyl

"Servo motor"

dv=(y-v)/ts

v=INTEG(dv,0.0)

yb=BCKLSH(yb0,ab,v)

END \$ "of DERIVATIVE"

TERMT (t.GE.tend)

END \$ "of DYNAMIC"

TERMINAL

END \$ "of TERMINAL"

END \$ "of PROGRAM"

Figure 7: ACSL Program listing for the speed control problem.

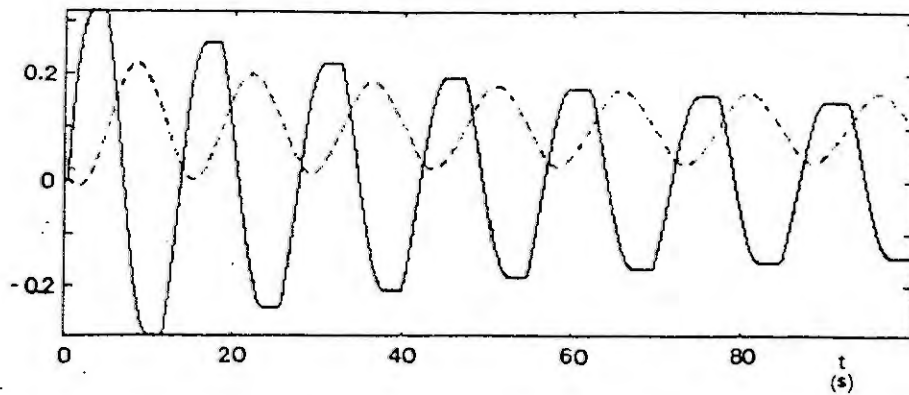


Figure 6: Response of the simulated speed control system with sufficient backlash to cause limit cycle oscillations. The continuous line is the control valve position while the dashed line is the output speed.

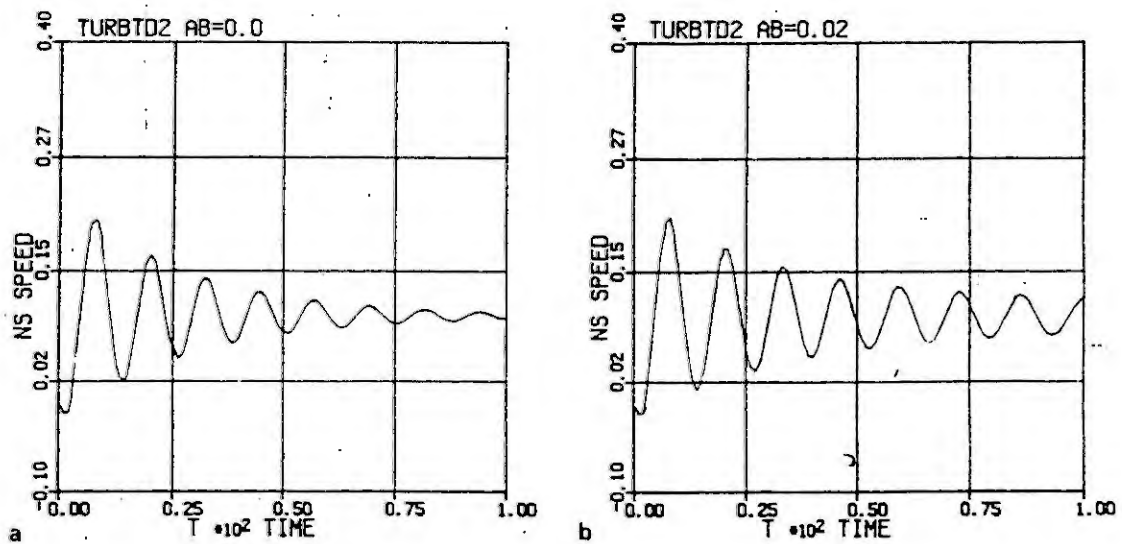


Figure 8: Results from ACSL simulation, a) with no backlash and b) with backlash of 0.02

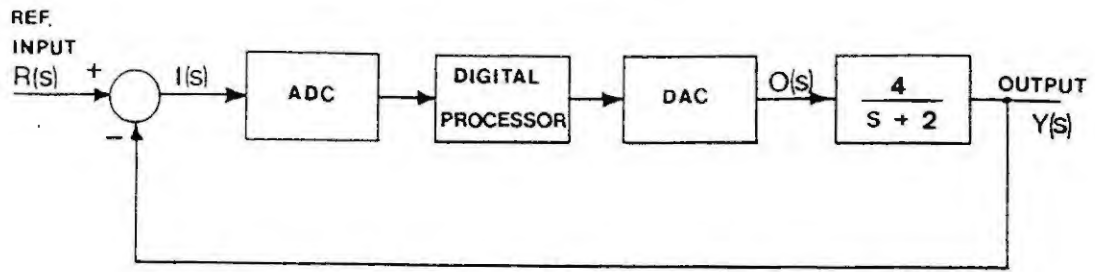


Figure 10: Block diagram of simple digital control system

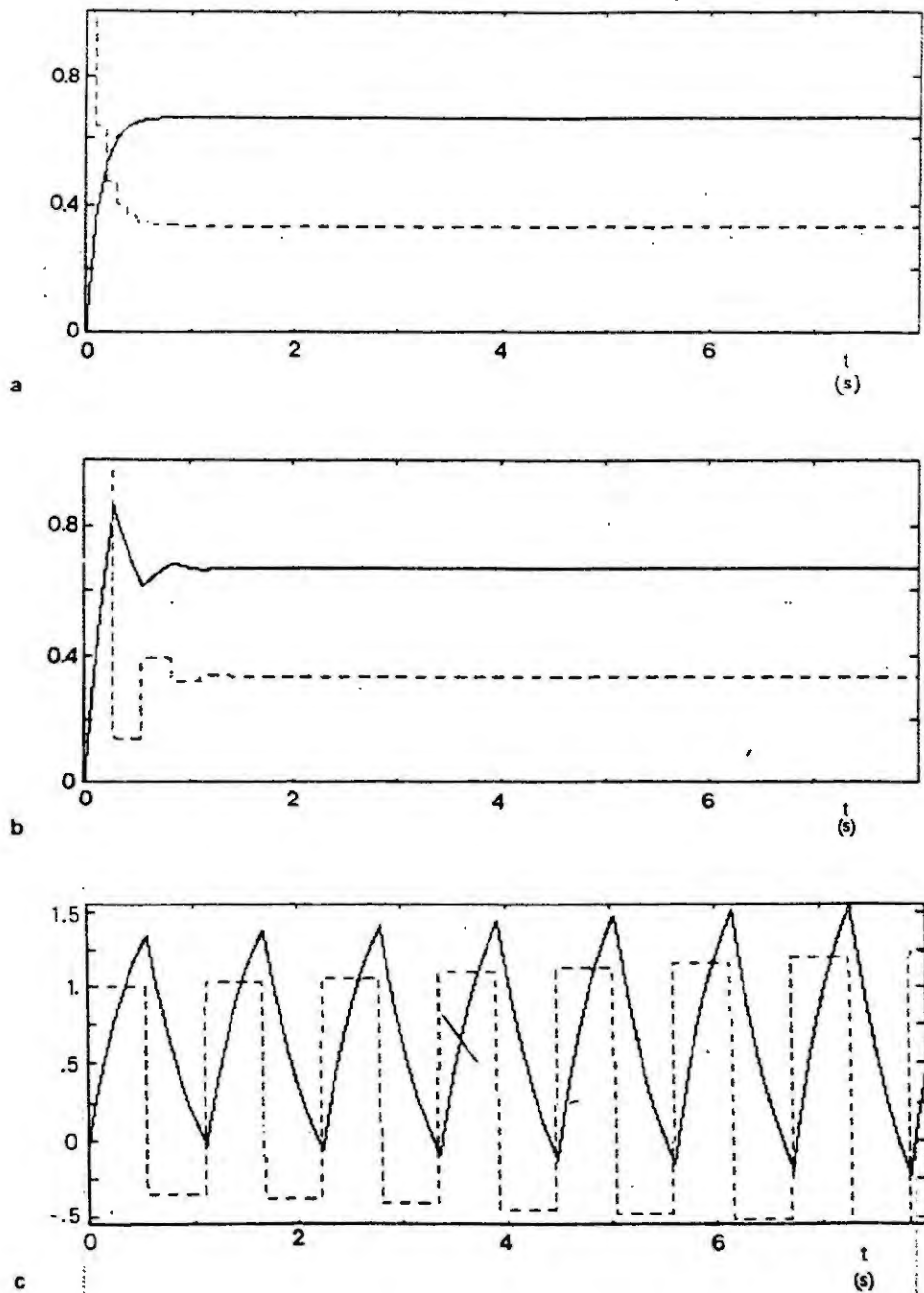


Figure 12: Responses {output (continuous line) and error (dashed line)} of digital control system for 3 values of controller sampling interval ($a=0.1s$, $b=0.28s$, $c=0.56s$).

```

      DYNAMIC
      I=I+1
      IF(I)15,25,15
15     IF(I-MULT)20,25,25
25     E=R-Y
      AIN=E
      OUT=OUT1+AIN-0.5*AIN1
      OUT1=OUT
      AIN1=AIN
      I=0
20     TYPE T,V,E,Y
C
C *****Start of Derivative Section*****
C
      DERIVATIVE
      DERIV=-(Y-AK*OUT)/TAW
      Y=INTEG(DERIV,0.0)
      DERIVATIVE END
C
C *****End of Derivative Section*****
C
C Test for end of simulation run
C
      IF(T-TMAX)10,10,12
10     DYNAMIC END
C
C *****End of Dynamic Segment*****
C
C *****Terminal Segment*****
C
12     STOP
      END

```

Figure 13: Part of SLIM program DIGCON1.SLI for simulation of digital control system.

```

C
C *****Start of Dynamic Segment*****
C
      DYNAMIC
      I=I+1
      IF(I)15,25,15
15      IF(I-MULT)20,25,25
25      AIN=R-Y
      OUT=AIN
      I=0
20      TYPE T,R,AIN,Y
C
C *****Start of Derivative Section*****
C
      DERIVATIVE
      DERIV=-(Y-AK*OUT)/TAW
      Y=INTEG(DERIV,0.0)
      DERIVATIVE END
C
C*****End of Derivative Section*****
C
C Test for end of simulation run
C
      IF(T-TMAX)10,10,12
10      DYNAMIC END
C
C *****End of Dynamic Segment*****
C
C *****Terminal Segment*****
C
12      STOP
      END

```

Figure 11: Part of SLIM program DIGCON.SLI for simulation of digital control system.

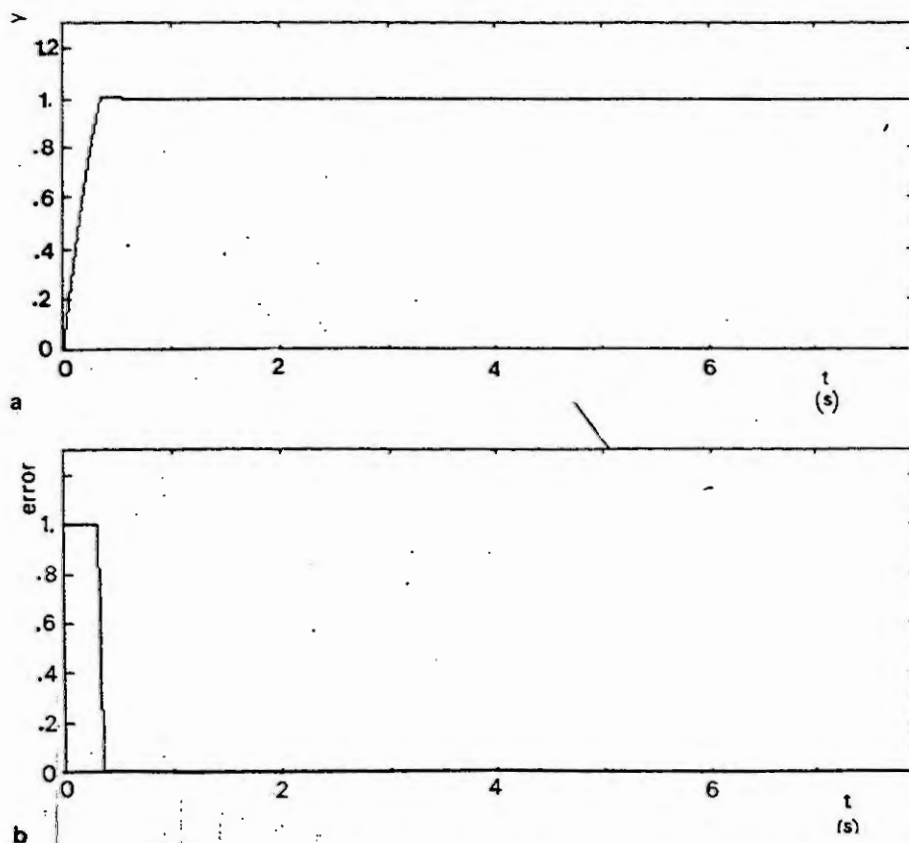


Figure 14: Responses from system with dead-beat control applied for step change of reference.

```

DYNAMIC
  I=I+1
  IF(I)15,25,15
15  IF(I-MULT)20,25,25
25  E=R-Y
    AIN=E
    OUT=OUT1+AIN-0.5*AIN1
    OUT1=OUT
    AIN1=AIN
    I=0
20  TYPE T,V,E,Y
C
C *****Start of Derivative Section*****
C
    DERIVATIVE
      DERIV=-(Y-AK*OUT)/TAW
      Y=INTEG(DERIV,0.0)
    DERIVATIVE END
C
C *****End of Derivative Section*****
C
C Test for end of simulation run
C
    IF(T-TMAX)10,10,12
10  DYNAMIC END
C
C *****End of Dynamic Segment*****
C
C *****Terminal Segment*****
C
12  STOP
    END

```

Figure 13: Part of SLIM program DIGCON1.SLI for simulation of digital control system.

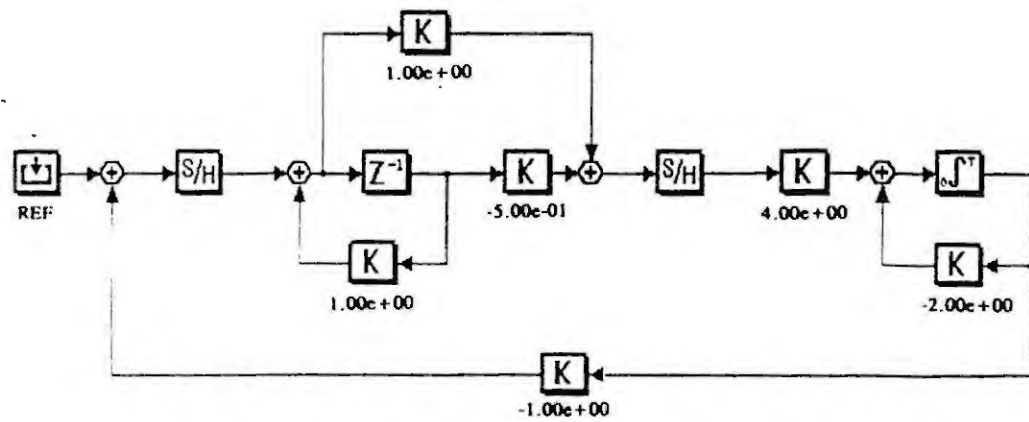


Figure 15: XANALOG block diagram for digital control system.

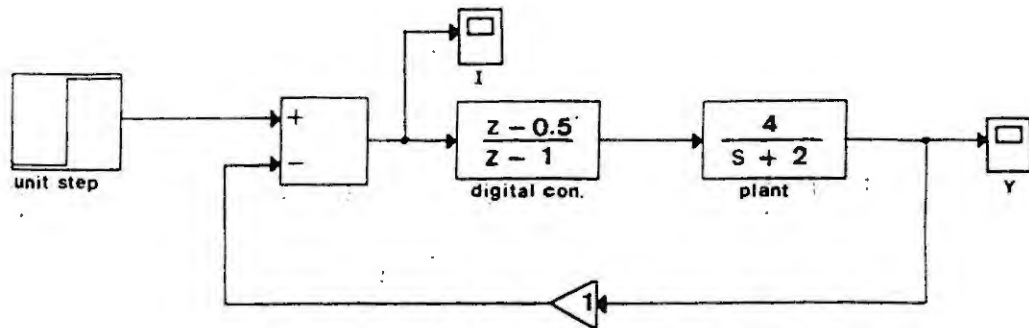
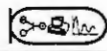


Figure 16: SIMULINK diagram for digital control system.



1. Was ist Fuzzy-Logik?

FUZZY-LOGIK

Methodik des sog. unscharfen
Schließens

Subjektive Unbestimmtheit von
Begriffen

Mathematisch fundierte Methode;
erlaubt graduierbare oder vage
Prädikate

So far as the laws of mathematic
refer to reality they are not cer-tain;
And so far as they are certain
they do not refer to reality
(A.Einstein)

1. Beispiel:

Tatsache: Sokrates war ein Mensch

Wissen: Menschen sind sterblich

Schlußfolgerung: Sokrates ist ge-
storben

2. Beispiel:

Tatsache: Die Tomate hat eine leicht rötliche Farbe

Wissen: Eine rote Tomate ist reif

Schlußfolgerung: Die Tomate ist nicht ganz reif

Klassische Logik versus Fuzzy Logik

Scharfe Menge charakterisiert durch zweiwertige Funktion

$$f_A(x) \rightarrow \{0,1\}$$

Unscharfe Mengen charakterisiert durch Zugehörigkeitsfunktion

$$\mu_A(X)$$

Die Funktion $f_A(x)$ wird als zweiwertig bezeichnet, da sie entweder den Wert 0 oder 1 annimmt, je nachdem ob das Element x zur Menge A gehört oder nicht. Die Menge $\{0,1\}$ ist die Wertemenge von A .

Ist als Wertemenge das gesamte Intervall $[0,1]$ zugelassen, geht die scharfe Menge A über in die unscharfe Menge A .

Eine unscharfe Menge A wird durch eine verallgemeinerte charakteristische Funktion μ_A gekennzeichnet: $X \rightarrow [0,1]$, die **Zugehörigkeitsfunktion** von A genannt wird und über einen (von Fall zu Fall geeignet festzulegenden) Grundbereich X definiert ist.

Je dichter der Wert $\mu_A(X)$ zu dem Wert 1 tendiert, desto mehr entspricht x der Charakteristik der Menge A .

DEFINITION:

Sei X eine unscharfe Menge.

Eine unscharfe Menge (fuzzy set) A über X wird charakterisiert durch eine Zugehörigkeitsfunktion $\mu_A(X)$, die jedem Element aus X eine reelle Zahl aus dem Intervall $[0,1]$ zuordnet.

Der Wert von μ_A an der Stelle X wird *Zugehörigkeitsgrad* von X zur Menge A bezeichnet.

3. Beispiel:

Sie wollen sich ein Cello kaufen. Die von Ihnen favorisierten Instrumente liegen zwischen DM 3.500,00 und 8.000,00.

GRUNDMENGE:

$$X = \{3500, 4500, 5600, 7200, 8000, 8500\}$$

UNSCHARFE MENGE:

$$A = \{(3500, 0.3), (4500, 0.6), (5600, 0.9), (7200, 0.4), (8000, 0.3), (8500, 0.1)\}$$

FUZZY-LOGIK

ist ein expandierender Markt!

1988 begann in Japan die Verbreitung des industriellen Einsatzes einer neuartigen Logik, die nicht mehr auf den absoluten Aussagen der monokontextuellen Logik „wahr“ und „falsch“ bzw. „ja“ und „nein“ beruhte. Für diesen neuen Logiktyp wurde der Begriff FUZZY-LOGIK eingeführt, der am treffendsten mit „unscharfe Logik“ übersetzt werden kann.

FUZZY-LOGIC Marktanteile

- Es ist zu erwarten, daß 1995 für mehr als 2,5 Mrd DM Geräte im Bereich der Konsumelektronik verkauft werden, z. B. Audio, Video, Haushalt, etc., mit einem Anteil von 2 - 15 DM an Bauteilen der Mikroelektronik
- Es ist zu erwarten, daß 1995 der Gesamtmarkt von Geräten und Systemen mit FUZZY-LOGIK-Bausteinen eine Größenordnung von ca. 6 Mrd DM haben wird.
- Im Jahr 2000 wird der Weltmarktanteil für „FUZZY-LOGIK-HALBLEITER“ auf ca. 15 Mrd DM geschätzt

Der Erfolg der FUZZY-LOGIC erklärt sich aus den vielen Vorteilen, die diese Logik mit sich bringt. Die mit FUZZY-LOGIC realisierten Geräte sind beispielsweise wesentlich

- benutzerfreundlicher als Geräte, die auf den bisherigen klassischen Verfahren basieren
- robuster und erfordern weniger Entwicklungszeit, bevor sie in den Markt eingeführt werden könnten, so daß eine Just-in-Time-Entwicklung möglich ist
- weniger Regeln als vergleichbare Expertensysteme z. B. bei Diagnoseanwendungen (Performancevorteil)

FUZZY-LOGIC basierte Regelung

- FUZZY-LOGIC wird z. Z. in hochentwickelten Systemen eingesetzt und repräsentiert bereits heute den nächsten Schritt der Entwicklung des sogenannten „Embedded Control“
- Hochentwickelte Regler sind das erste Ziel für State-of-the-Art-Controller die auf FUZZY-LOGIC basieren
- Bereits für 1994 wird von einer Kostenreduzierung für FUZZY-LOGIC-Bauteile erwartet, die es erlauben, auch das Segment für die Routine-Regelung einzubeziehen, sogenanntes LOW-END-CONTROLLER-Segment
- Insgesamt kann der FUZZY-LOGIC-Anteil im Controller-Segment bereits heute auf ca. 40 % abgeschätzt werden

Systembeschreibung und –Simulation mit Petri-Netzen

Dr. H. Fuss

Institut für Methodische Grundlagen (F 1)

Gesellschaft für Mathematik und Datenverarbeitung (GMD)

Bonn – Birlinghoven

0. Einleitung

Petri-Netze sind inzwischen weltweit bekannt und werden häufig angewandt, doch wird in einem Großteil der Anwendungen lediglich die dazugehörige Graphik als eine Art Flußdiagramm-Sprache benutzt. Die zugrundeliegende Idee der Netze – und das ist das Phänomen der Nebenläufigkeit (engl: concurrency) – ist aber viel umfassender, und damit bedeutender.

Nebenläufigkeit tritt in fast allen realen Systemen auf. Sie ist aber meist erst dann von Bedeutung, wenn das System räumlich verteilt ist und die Übertragungsgeschwindigkeiten von Nachrichten und Kommandos im System von der gleichen Größenordnung sind wie seine (materielle) interne Arbeitsgeschwindigkeit, d.h. wie die Reaktionszeiten des Systems.

1. Systembeschreibung: Nebenläufigkeit als Systemeigenschaft

Reale Systeme sind verteilt im Raum und agieren in der Zeit; jede Aktion zwischen den System-Teilen, auch die Informationsübertragung innerhalb des Systems, benötigt dafür eine gewisse Zeitdauer; jedoch werden diese physikalischen Eigenschaften eines Systems oftmals bei seiner Beschreibung und Simulation, häufig gar bei seiner Konstruktion, nicht genügend beachtet.

Bei der Beschreibung eines realen Systems durch ein DGL-System z.B. werden die Zustandsübergänge, d.h. die neuen Werte für *alle* Variablen für *denselben Zeitpunkt* berechnet. Das unterstellt jedoch dem Original-System (s.u. bei 'Bi-Simulation'), daß die Systemzustände aller Teilsysteme jedem anderen Systemteil zu jeder Zeit (insbesondere zur Taktzeit) gleichzeitig bekannt sind, es unterstellt ihm also eine unendlich große Informations- bzw. Signal-Übertragungsgeschwindigkeit.

Schon Minkowski hat in seinem Raum-/Zeit-/Ereignis-Modell¹ durch die Idee der 'Vorkegel' bzw. 'Nachkegel' gezeigt, (daß nicht nur nicht alle Information immer überall sein kann, sondern darüber hinaus auch:) daß die kausale Ordnung der Ereignisse in Raum und Zeit keine Vollordnung (Kette) bildet, wie man sie für die reellen Zahlen kennt und wie man sich das auch für die Zeit gedacht hatte, sondern daß Ereignisse eine *Halbordnung* bilden.

Diese Erkenntnis führte schließlich auch zu einer Auffassung von einer *Halbordnung der Zeit* und ermöglichte es, von *lokaler Zeit* sprechen zu können. Die Halbordnung ('*partial order*') impliziert, daß 'Gleichzeitigkeit' *nicht transitiv* ist; somit ist 'Zeit' ein ungeeigneter Sortierbegriff.

Halb-geordnet-sein, und damit teilweise Unvergleichbarkeit, gilt überall da, wo genau genug gemessen, verglichen und sortiert wird, also u.a. *in der gesamten experimentellen Physik*; denn es handelt sich hier um ein generelles Phänomen aus der Theorie des Messens. Es ist nämlich ein konzeptioneller Irrtum davon auszugehen, daß beim Messen (von Länge, Zeit oder anderen Größen) ein Vergleich zwischen Objekt und Meßplatte/Meßskala auf *Gleichheit* gemacht wird. In Wirklichkeit geht es beim Meßvorgang um das (Nicht-mehr-)Wahrnehmen von Differenzen, also um Unterscheidbarkeit, genauer: um (reale, experimentelle) *Un-Unterscheidbarkeit*, und nicht um (ideelle) Gleichheit.

Zur Veranschaulichung mögen die (paarweise) 'Un-Unterscheidbarkeits-Balken' dienen:

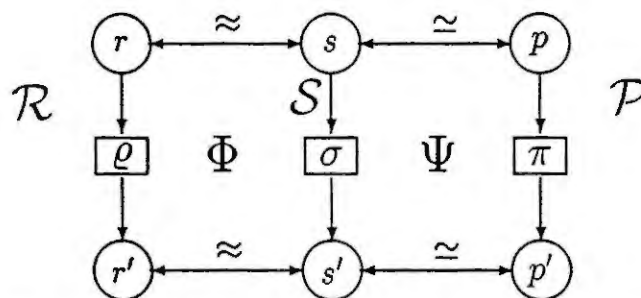
¹ Ein Ereignis ist 'früher', wenn es auf ein anderes eine kausale Wirkung haben kann

2. System-Simulation; Prozesse, Abläufe; Verhalten

Zunächst scheint eine Präzisierung der Ausdrucksweise angebracht: obwohl vielfach so (verkürzt) genannt, sind es nicht Systeme, die simuliert werden, sondern die in ihnen ablaufenden Vorgänge, ihre Prozesse, d.h. das *Verhalten* des Systems im Verlaufe der Zeit.

Eine Verhaltens-Simulation sollte *kongruent in beiden Richtungen*, Realität \rightarrow Simulations-Modell ($\mathcal{R} \rightarrow \mathcal{S}$) und Simulations-Modell \rightarrow Realität ($\mathcal{S} \rightarrow \mathcal{R}$) sein.

Gemeint ist folgendes: Wenn das reale System \mathcal{R} , in der Situation (im 'Zustand') r befindlich, durch eine Aktion ϱ in die Situation r' gerät, und dieses modelliert wird mit Φ in ein Simulations-Modell \mathcal{S} , in dem der Übergang von \mathcal{S} aus der Situation ('Zustand') s durch die modellierte Aktion σ in die Situation s' abgebildet wird, dann soll auch selbstverständlich r' durch die Modellierungsabbildung Φ in s übergehen, und die Abbildung Φ^{-1} führt s in r und s' in r' zurück, d.h. die Rollen von \mathcal{R} und \mathcal{S} könnten genausogut vertauscht sein: Bi-Simulation.



Desgleichen: Wenn das Simulations-Modell \mathcal{S} , in der Situation ('Zustand') s befindlich, ... und die dazugehörige numerische Berechnung durch die Abbildung (Programmierung) Ψ des Modells \mathcal{S} in ein Programm \mathcal{P} aus der Situation ('Zustand') p durch den Programmablauf π die Folgewerte p' errechnet, dann auch $\Psi(s') = p'$, und $\Psi^{-1}(p) = s$ und $\Psi^{-1}(p') = s'$.

Da meist noch mehr Abbildungsschritte bei der Simulation dazwischengeschaltet sind (z.B. Zielvorstellungen über die Simulationsergebnisse, graphische Darstellungen etc), muß man diese Forderung für alle Zwischenschritte erheben. (Tri-Simulation ...).

Durch ungeeignete Wahl der Simulationssprache wird bewirkt, daß diese Forderung – zumindest für die in der Einleitung angegebenen Fälle – garantiert *nicht* erfüllt werden kann.

3. Petri-Netze

In der *concurrency*-Theorie sind die Petri-Netze (PN) das mathematisch bzw. systemtheoretisch am weitesten bearbeitete Modell. Sie sind – zumindest ihre graphischen Symbole \square (Kästchen) und \circ (Kreise), die durch \rightarrow (Pfeile) verbunden werden – allgemein bekannt genug, so daß sich eine längere Beschreibung hier erübrigt.

$\text{PN} = \{ S, T, F; M_0; \mathcal{R} \}$ (Stellen, Transitionen, Flußrelation; Anfangsmarkierung; Schaltregel)

Dabei stehen die Kästchen für dynamische Objekte (Veränderungen, Ereignisse, System- bzw. Zustandsübergänge), die Kreise für statische Objekte (Dinge, Variablen, ihre Werte, Zustände...).

Die Pfeile geben Richtungen an: a) Flüsse von physischen Objekten (Materialien, Substanzen ...),

von ideellen Objekten (Geld, Zeit) – oder b) Einflüsse, Wirkungen, Informationsweitergabe (Unterschrift, Autorisierung, Kenntnisnahme, Geheimnummer/Code/Schlüssel etc.).

Die Marken auf einer Stelle bezeichnen das momentane Vorhandensein eines Objektes (auf diesem Platz), die Gültigkeit einer Bedingung, den Wert u.ä. (Ggf. ist bei der Darstellung von Nachrichten zu unterscheiden zwischen dem materiellen Datenträger und der immateriellen Information selbst.)

Die Schaltregel beschreibt die dem Netz innewohnende Dynamik, die Veränderungs-Automatik.

Zwar ist die formal korrekte Anwendung der PN-Graphik schon allein für sich eine nützliche Bereicherung der System-Beschreibungs- und -Darstellungsmöglichkeiten, erzwingt sie doch i.a. eine korrekte Darstellung von Flußstrukturen und kausalen Abhängigkeiten (der Relationen nacheinander/nebeneinander). Aber die gezielte inhaltliche Anwendung der *concurrency*-Theorie, darstellbar durch diese PN-Graphik, kann noch zu einigen Verbesserungen bei der Konstruktion von Systemen (realen und Simulations-Systemen) beitragen. Das wäre der Fall, wenn man die logisch bzw. räumlich verteilte Struktur des Original-Systems genauso verteilt durch Teil-Netze darstellt, und wenn man nebenläufige Abläufe im Original gezielt in ebensolche des Modell-Systems abbildet.

Unter den PN gibt es verschiedene Klassen und Spezialisierungen, in denen unterschiedliche Eigenschaften gelten; je nach beabsichtigten Zweck werden geeignete Netz-Typen benutzt.

4. Konkurrente Systeme, Parallel-Verarbeitung, Sequentialisierung

Nebenläufigkeit prägt das Verhalten vieler Systeme in signifikanter Weise; ohne ihre Betrachtung bleiben bestimmte Verhaltensweisen mancher realer Systeme unerklärlich. Sie ist allerdings aus den eingangs erwähnten Gründen (zentrale/hierarchische Steuerung, Steuerungs-Geschwindigkeit ähnlich der Arbeitsgeschwindigkeit) lange unbemerkt geblieben. Das änderte sich schlagartig, als die Objekte, die in einem System verarbeitet wurden, selber Nachrichten und Kommandos waren, also mit der Erfindung der Computer, und wurde noch einmal erheblich wichtiger mit der Erfindung der Mehrprozessor-Maschinen. Vorher konnten meistens noch fehllaufende Verarbeitungsmechanismen durch immer schnellere Steuerungssignale eingeholt und korrigiert werden.

Im Computer jedoch wurde mit derselben (Höchst-)Geschwindigkeit gesteuert wie verarbeitet.

Konkurrente Systeme sind dadurch gekennzeichnet, daß sie keine zentrale Steuerung enthalten (das sind z.B. alle Systeme, in denen Menschen bestimmend mitarbeiten, ferner viele Mehrprozessorsysteme, Datennetze, mailing-Systeme). Die system-typische partielle Autonomie bedeutet lokale Entscheidungsbefugnis und Verfolgen eigener Interessen, die allerdings denen anderer Systemteile zuwiderlaufen können. Zum Funktionieren eines konkurrenten Systems ist erheblicher organisatorischer Aufwand neuerer Art nötig, damit nicht unerwünschte Zustände wie Systemstillstand, unendliche Schleifen, unbeabsichtigtes Auslöschen von später noch benötigter Information u.dergl. eintreten.

Besonderes Augenmerk ist der Sequentialisierung von nebeneinander ('parallel') ablaufenden Prozessen zu widmen; da ja das Ergebnis in beeindruckender Weise von der Reihenfolge (Beispiel: update/lesen) abhängen kann. Unglücklicherweise werden im Umgang mit Simulations-Programmen viele dieser Konflikt-Entscheidungen dem jeweiligen Programm bzw. Betriebssystem überlassen, so daß der Kunde nicht einmal von der Existenz, geschweige denn von der Lösung solcher Konflikte erfährt. Damit hat der Benutzer die Kontrolle über die Simulationsergebnisse verloren.

5. Zusammenfassung

Halb-Ordnungen treten in realen Systemen viel häufiger auf, als man sich weithin bewußt ist, z.B. in der ganzen experimentellen Physik. Wohlgemerkt: hier wird nicht behauptet, daß die zu messenden Objekte bzgl. der zu messenden Eigenschaft nicht vollständig geordnet sind bzw. sein könnten, sondern, daß man eine Voll-Ordnung *nicht feststellen kann*, d.h. daß die *Meßwerte* halb-geordnet sind. Diese Halb-Ordnungen werden i.a. auf eine Voll-Ordnung (z.B. auf die rationalen Zahlen) abgebildet. Inwieweit dadurch Simulationsergebnisse verfälscht werden, oder gar bei der Systemkonstruktion etwas Unmögliches versucht wird, muß von Fall zu Fall geprüft werden.

Hier scheint es noch einen erheblichen Forschungs- und Aufklärungs-Bedarf zu geben.

LITERATUR:

- H.Fuss: *Reversal Simulation with Place-Transactor-Nets*.
in: H.Wedde (Ed.): *Adequate Modeling of Systems*. pp.222-232, Springer (1983)
- H.Fuss: *Zur Simulation von Zufall und Verlässlichkeit*.
in: D.P.F.Möller (Hrsg.): *Simulationstechnik*. pp.141-146; Inf.FB 109, Springer
- H.Fuss: *Simulating 'Fair' Random Numbers*.
in: Proc. 2nd Europ. Simulat. Congress ESC '86 (Antwerp), pp.250-257. SCS (1986)
- H.Fuss: (Hrsg.): *Fachgespräch Simulation mit Petri-Netzen und verwandten Methoden*.
in: R.Valk (Hrsg.): Proc. 18. GI-Jahrestg., pp. 533-606; IFB 187, Springer (1988)
- H.Fuss: *Is Parallel Processing a Temporal Issue Only?*
in: D.Murray-Smith/D.Stephenson/R.Zobel (Eds.). Proc. 3rd Europ. Simulat. Congress ESC '89 (Edinburgh), pp.35-40. SCS Simulation Councils Inc., (1989)
- H.Fuss: *Wie lang ist ein Meter? - Reflexionen über Daten in Simulationsmodellen*
in: Breitenacker/Troch/Kopacek (Hrsg.): *Simulationstechnik*. Proc. 6.ASIM-GI-Sympos. (Wien), pp.102-106, Fortschritte in der Simulationstechnik, Bd.1, Vieweg (1990)
- B.Baumgarten: *Petri-Netze Grundlagen und Anwendungen*. 369 S., B.I.Verlag 1990
- C.A.Petri: *Kommunikation mit Automaten*. IIM, Uni Bonn; Schrift Nr. 2, 1962
- C.A.Petri: *Fundamentals of a Theory of Asynchronous Flow*.
in: Proc. IFIP Congr. 62, München, pp.386-390. N.Holl. 1963
- C.A.Petri: *Concepts of Net Theory*. Proc. H.Tatr. pp.137-146. 1973
- C.A.Petri: *Interpretations of Net Theory*. GMD-ISF Bericht 75-07
- C.A.Petri: *Nichtsequentielle Prozesse*. IMMD Erlangen Ber. Vol.8, Nr.8 pp.57-82 (1976)
GMD-ISF 76-06.3 (Revid. Aufl. 1977) Engl. transl.: GMD-ISF 77-05
- C.A.Petri: *General Net Theory*.
in: B.Shaw (ed.) Proc. IBM & Newcastle Seminar Sept.76, pp.131-169
- C.A. Petri: *Concurrency* in: W.Brauer (ed.): *Net Theory and Applications*, Proc. Advanced Course, Hamburg LNCS 84, pp.251-260. Springer (1980)
- C.A.Petri: *State-Transition Structures in Physics and in Computation*.
in: Int.J.Th. Physics, Vol. 21, No. 12, pp.979-992, (1982)
- C.A.Petri: *Concurrency Theory* in: W.Brauer/W.Reisig/G.Rozenberg (eds.): *Advances in Petri Nets 1986*, Proc. Adv.Course, Bad Honnef LNCS 254, pp.4-24. Springer (1987)
- C.A.Petri, E.Smith: *Concurrency and Continuity*.
in: LNCS Vol. 266, pp.273-292. Springer (1987)
- W.Reisig: *Petri-Netze, Eine Einführung*. 158 S. Springer 1982
- W.Reisig: *Petri Nets; An Introduction*. EATCS Vol.4, 161 p., Springer 1985
- W.Reisig: *Systementwurf mit Netzen*. 125 S. Springer 1985
- E.Smith: *Zur Bedeutung der Concurrency-Theorie für den Aufbau hochverteilter Systeme* (Dissertation) GMD-Bericht Nr. 180. (164 S.) R.Oldenbourg (1989)
- K.Voss/H.J.Genrich/G.Rozenberg (eds.) *Concurrency and Nets*. 622 p., Springer 1987
- H.Plünnecke, W.Reisig: *Bibliography of Petri Nets 1990*
(4099 Einträge) in: *Advances in PN 1991*, S.317ff, LNCS 524, Springer 1991
- Petri Net Newsletter* der GI-FG 1.1.2 'Petri Nets and Related System Models', 3 Hefte/Jahr
- Diverse Modell-Generatoren und Simulatoren von Petrinetzen, z.T. als Tools mit ganzen Entwicklungs-Umgebungen und großer Software-Ausstattung, sind vielfältig erhältlich. Übersicht in:
- F.Feldbrugge: *Petri Net Tools Overview*. in: LNCS 424, pp.151-178; Springer 1990

C.A.Petri

The self-adjusting fire brigade

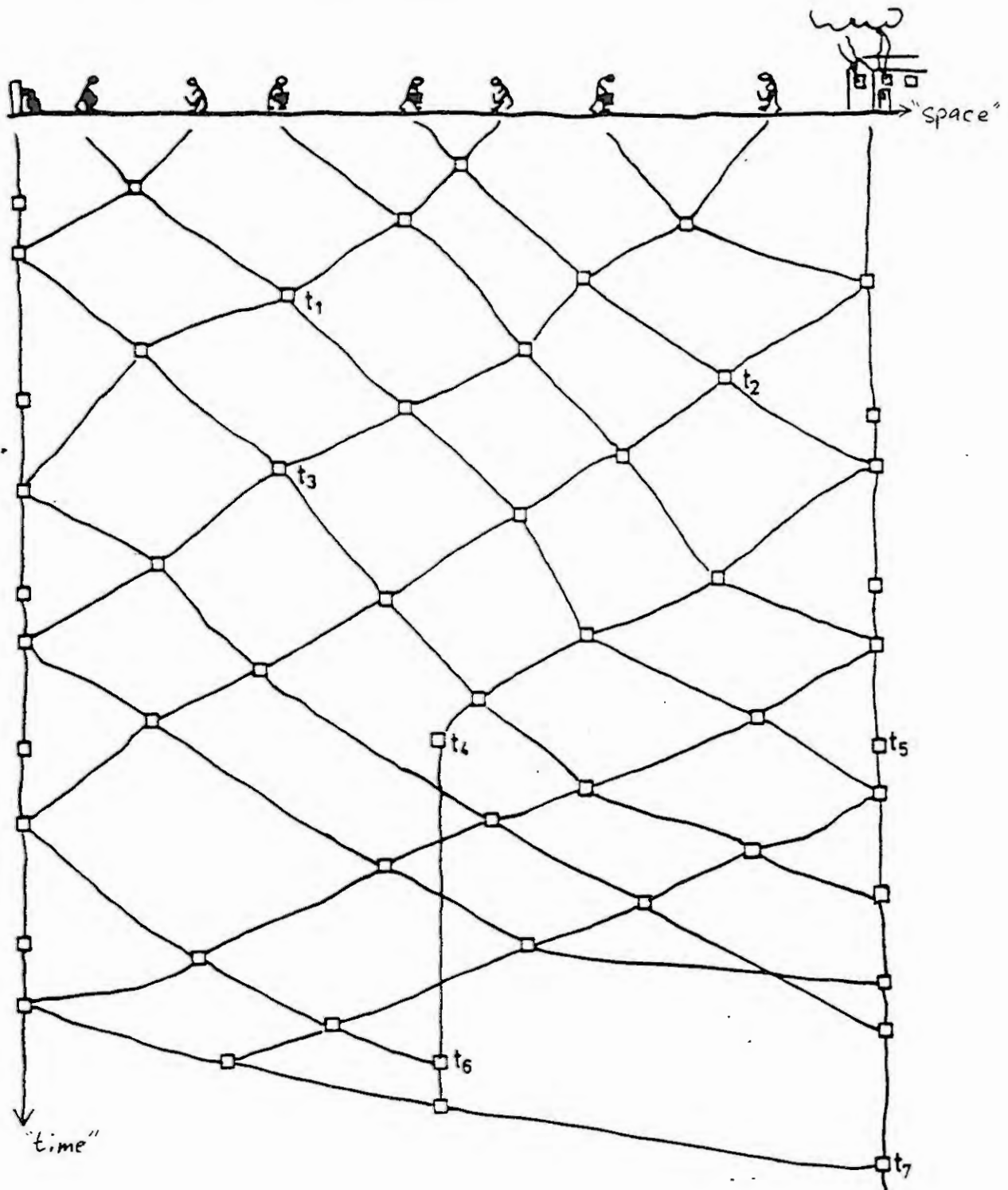
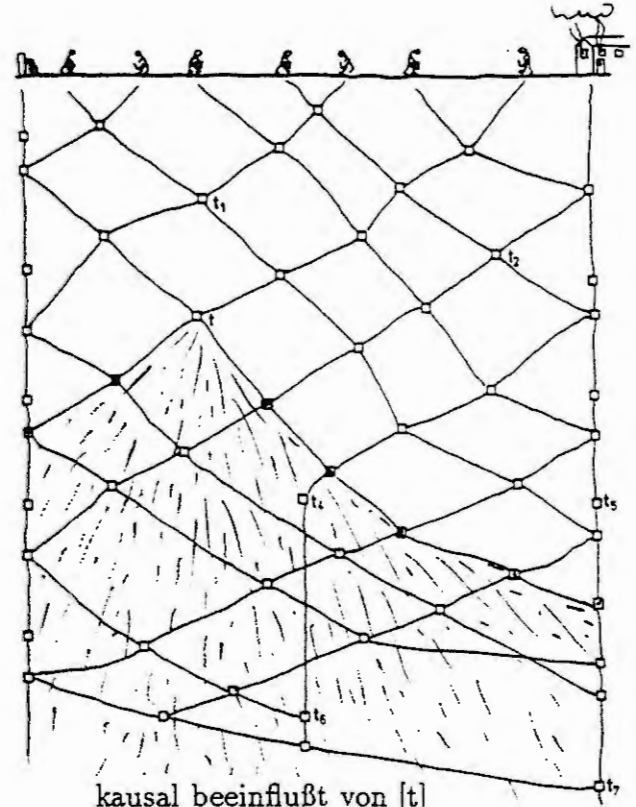
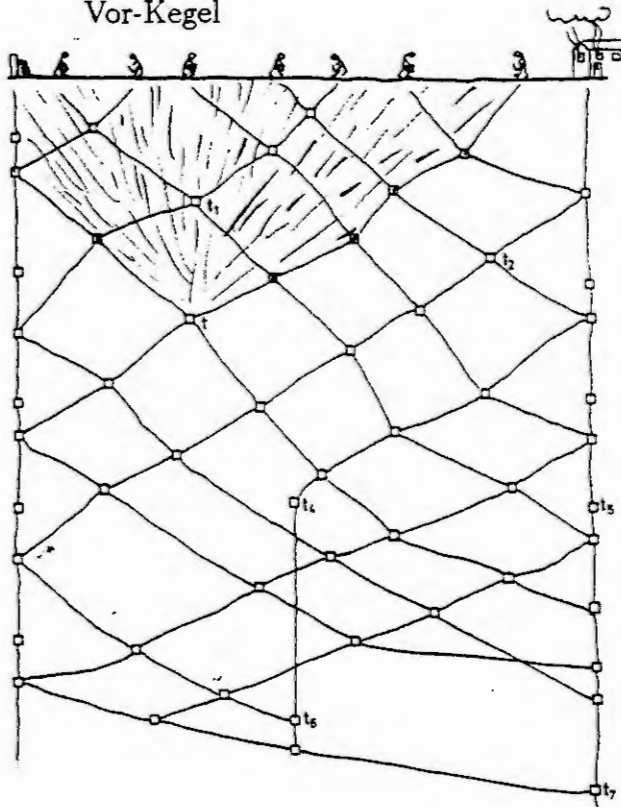


Fig 1 : Partial history of execution of a plan

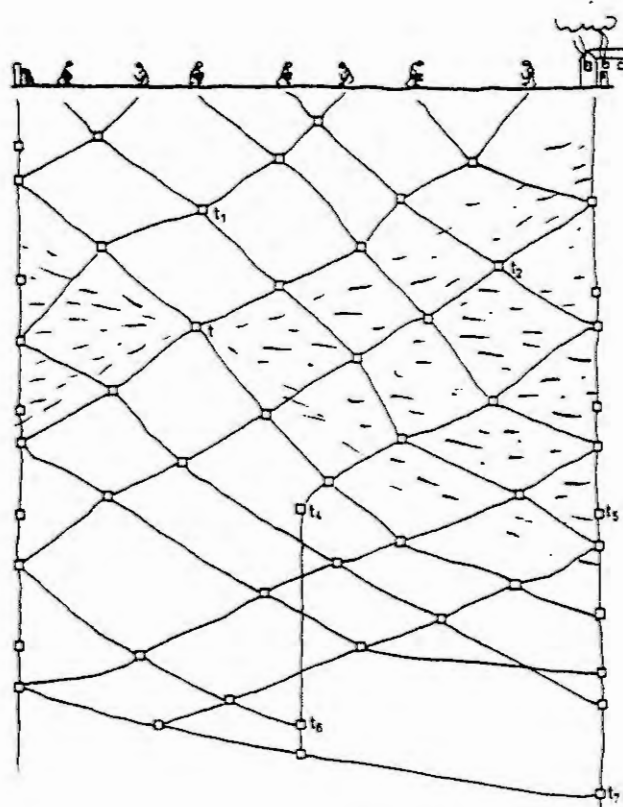
from: Introduction to General Net Theory

LNCS 84, pp.1-24 (1980)

ursächlicher Einfluß auf [t]
früher
Vor-Kegel



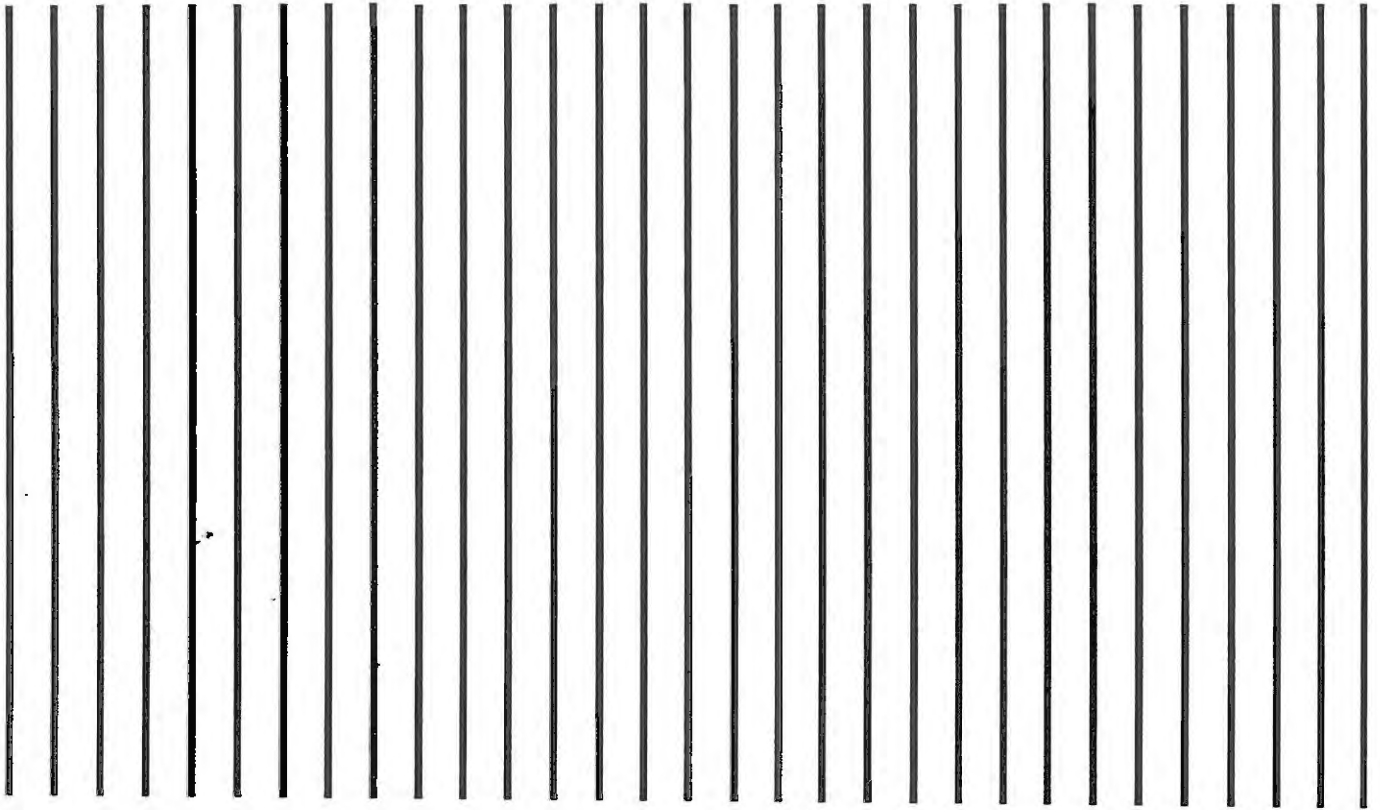
kausal beeinflusst von [t]
später
Nach-Kegel



kausal keine Beeinflussung
weder vorher – noch nachher
nebenläufig, engl.: concurrent
Indifferenz-Raum

'parallel' zu [t],
'gleichzeitig' mit [t]

Un-Unterscheidbarkeit (räumlich)



aus: G.Kampe, M.Zeitz: Simulationstechnik;
 Tagungsband 9. ASIM-Symposium Okt. 1994 in Stuttgart; pp.355-360;
 Fortschritte in der Simulationstechnik Band 9, Vieweg-Verlag (1994)

Über Unmöglichkeiten, parallele Prozesse korrekt zu simulieren

H. Fuss

GMD

Bonn – Birlinghoven

Zusammenfassung: Es ist bisher in verschiedenen Fällen ausgeführt worden, daß es ungleich komplizierter ist, verteilte Systeme darzustellen als sequentielle, daß ihre parallelen bzw. nebenläufigen Prozesse zu simulieren um ein Vielfaches schwieriger ist, und daß die Resultate recht unbefriedigend bleiben.

Hier wird das Resümee vieler Vorarbeiten gezogen, welches lautet: eine Simulation eines parallelen Prozesses (d.h. eine Computer-Modell-Rechnung) sagt nur wenig aus über das mögliche Geschehen im realen System. In einem verteilten System sind – wegen Konflikt-Entscheidungen und Zeitbedingungen (wer kommt vor wem?) – noch andere, vielleicht gar unendlich viele andere Varianten möglich als die bei der Modellbildung beobachteten. Die Simulation produziert nur eine Variante des realen Ablaufs, oder kann darstellen, warum ein Geschehen im realen System gerade so abgelaufen ist, aber für die Zukunft (z.B. bei einem Planungs- oder Prognose-Modell) kann man sagen: solange im realen System selbst noch Entscheidungen offen sind, kann die Simulation nicht klüger sein. Es ist vielleicht gar jedes beliebige Ergebnis möglich.

Da im Prinzip alle realen Systeme räumlich verteilt und die darin ablaufenden Prozesse also alle nebenläufig sind, hat die Simulation somit ein prinzipielles Problem...

1 Vorarbeiten

In [1] (Bälle eines Jongleurs, Linienbusse im Stadtverkehr, Rennwagen – alle auf geschlossenen Umläufen) wurde dargestellt, wie Abläufe, die nach unterschiedlichen Prinzipien erzeugt wurden (zentral- gegenüber individual-gesteuert), trotzdem zu gleichartigen Darstellungen (Bildpunkten auf einem Monitor) führen können – der Rückschluß vom Simulationsergebnis auf die Realität ist also ohne weitere Information nicht möglich.

In [2] und [3] wurden die enge Verknüpfung von räumlicher Verteiltheit und zeitlichem Nebeneinander aufgezeigt. In [4] wurde noch einmal die überragende Bedeutung einer zentralen, übergeordneten, konfliktentscheidenden Instanz für den Ablauf von parallelen Aktionen herausgearbeitet. In [Liste 5] wurde an verschiedenen Beispielen herausgearbeitet, daß die Darstellung durch Netze (nach Petri) ein geeignetes Mittel ist, dem Phänomen 'Nebenläufigkeit' bzw. 'Parallelverarbeitung' näherzukommen.

2 Die Schlüssel-Arbeiten

Auf dem 7. Workshop des AK 1.1 "Simulations-Methoden und -Sprachen für verteilte Systeme und parallele Prozesse" [6] wurde u.a. eine den Teilnehmern vorher gestellte Diskussionsaufgabe behandelt: der simpel erscheinende Fall 'Parallele Berechnung des Mittelwerts zweier Zahlen'. Wie einerseits theoretisch nachgewiesen, andererseits durch eine Simulation tatsächlich verifiziert, kommt das Unvorstellbare heraus, nämlich jedes beliebige Ergebnis –

und das lediglich in Abhängigkeit von der Geschwindigkeit der parallel laufenden Prozessoren. (Ergänzt auf dem 8. WS des AK 1.1 in Dresden 1992, s.a. ASIM-Heft 29).

Auf dem gemeinsamen Workshop der AKs 1.1 (s.o.) und 2.2 (Sim. technischer Systeme) im Jan. 1994 in Wien [7] wurde ein Uralt-Thema, das auch schon mehrfach gar Titelgeschichte der 'EUROSIM Simulation News Europe' (u.a. von Heft 3/91) war, wieder aufgegriffen: das Problem der 5 dinierenden Philosophen. Daran wurde nachgewiesen: Simulationen paralleler Probleme machen es sich i.a. bequemer, indem sie die Originalfragestellung umformulieren, stellen damit aber keine Lösung der ursprünglichen Aufgabe dar.

Hier noch einmal kurz die Hauptpunkte der beiden o.a. Betrachtungen:

2.1 Parallele Mittelwert-Berechnung

Die Aufgabe (Fragestellung nach Dijkstra) war: die jetzigen Werte eines Zahlenpaars x und y (z.B. $x_0 = 0$ und $y_0 = 1$) sollen von zwei parallel arbeitende Prozessoren P_1 und P_2 durch das arithmetische Mittel $x_{m,1} = (x + y)/2$ und $y_{m,2} = (y + x)/2$ ersetzt werden.

Nun kann man realistischerweise bei parallel arbeitenden Prozessoren keineswegs voraussetzen, daß sie gleichschnell arbeiten, erst recht natürlich nicht, daß sie, von einem dritten Prozessor geführt, im Gleichtakt arbeiten (letzteres würde ja dem Prinzip der Parallelverarbeitung (s. Kap 3.2) direkt widersprechen). Im Gegenteil, man muß Parallelverarbeitung so betreiben, daß man einzelne Prozessoren (die räumlich verteilt auf verschiedenen Chips, Karten oder Einschüben sitzen) für Wartungsarbeiten abschalten kann, und daß ein Austausch gegen andere, insbesondere schnellere Prozessoren, möglich ist. (Bei anderen Fällen der parallelen Zusammenarbeit, z.B. Aktenverarbeitung in einer Behörde, Halbfertigprodukt-Anlieferung für eine Fabrik etc. ist es unmittelbar einsichtig, daß in verschiedenen Bereichen verschieden schnell gearbeitet wird.)

Nehmen wir einmal an, in der obigen Aufgabe sei einer der Prozessoren, beispielsweise P_2 , etwa 100-mal schneller als der andere. Während P_1 dann noch in den ersten Schritten seiner Berechnung, vielleicht gar noch beim Einlesen des Wertes von x_0 ist, wird P_2 schon viele Male zugegriffen, berechnet und y mit seinem y_m überschrieben haben, also dem P_1 längst ein anderes y als y_0 zur Berechnung vorgelegt haben. Und wenn er statt 100-mal nur 13-mal schneller, oder wenn er für andere Zwecke kurzzeitig unterbrochen war (z.B. für 'interrupt-handling'), dann wird es wieder ein ganz anderer Zahlenwert für y sein, mit dem P_1 seine Berechnung ausführt. Und der Test, ob der Mittelwert von x und y derselbe ist wie der von y und x , fällt nie positiv aus – d.h. das parallele Programm terminiert nicht.

2.2 Die 5 dinierenden Philosophen

Es sind in der Literatur Dutzende, wenn nicht Hunderte von Lösungen des Problems bekannt. Die Darstellungen des Problems sind in großer Mehrheit richtig, die Lösungen i.d.R. untauglich. Denn es wird – unerklärlicherweise – nach einer *Lösung* gesucht, einer Lösung, die Deadlocks vermeiden soll, eine möglichst gleichmäßige oder anderweitig 'faire' Versorgung der Philosophen mit Nahrung (starvation-free) versucht, oder was der Ziele mehr sind.

Wie auch immer die Lösung aussieht – danach ist beim originären Philosophenproblem nicht gefragt. Dort sitzen die Herrschaften um einen Tisch und verhalten sich so, wie *sie*, die Philosophen es wollen. Jedes Simulationsprogramm dagegen verteilt ihnen die Löffel, wie *der Simulator*, vielleicht ein Zufallsgenerator, es will, und nimmt sie ihnen genauso wieder weg ('time out'). Der ins Spiel gekommene Simulator ist gerade die zentrale, steuernde, über den Philosophen schwebende Instanz, die den simulierten Philosophen seine Willensentscheidung aufzwingt. Dadurch werden die Philosophen zu Marionetten und können in der Tat durch Programmstücke ersetzt werden – dann aber ist die Original-Fragestellung umformuliert

worden. Erst ein Simulationssystem mit 5 unabhängigen Prozessoren, geschaltet nach den Regeln des Philosophenproblems, also eine Kopie des Philosophen-Tisches, würde sich so verhalten können, wie sich die Philosophen verhalten können. Man beachte die Potential-Formulierung: 'können'. Ob die Prozessoren sich auch so verhalten wie die Philosophen, steht dagegen auf einem ganz anderen Blatt.

3 Terminologie

3.1 Räumliche Verteiltheit

Es ist eigentlich unnötig, darauf hinzuweisen, daß jeder physisch reale Körper einen gewissen Raum ausfüllt. Wenn darin – und das hängt vom Detaillierungsgrad der Betrachtungsweise ab – mehrere aktive Systemteile enthalten sind, so sind diese Akteure voneinander entfernt. Ebenso unnötig zu betonen, daß die Lichtgeschwindigkeit (bzw. die elektrische Signalgeschwindigkeit) die größte Geschwindigkeit überhaupt ist.

Daraus folgt trivialerweise, daß für die Kooperation zwischen realen Komponenten *Zeit* verbraucht wird. Ob das für das Funktionieren des Systems von Bedeutung ist, hängt vom Detaillierungsgrad der Betrachtungsweise ab. Falls ja (d.h. bedeutsam), kann allerdings nicht mehr die Automatentheorie als theoretische Grundlage gelten (und derartige Differentialgleichungssysteme als Simulationsmodelle konstruiert werden), sondern dann befinden wir uns in einer Situation, wo die Theorie der *concurrency*, der *Nebenläufigkeit* gilt. Noch einmal betont: die *simultane Bestimmung von Zustandsgrößen* eines Systems (z.B. durch ein Gleichungssystem) ist dann schlichtweg falsch. Das ist leicht einsichtig auch für Nichtfachleute: simultane Bestimmung hieße unendlich große Abstimmungsgeschwindigkeiten zwischen den Variablen, alle Information verbreitet sich unendlich schnell und ist immer überall gleichzeitig (simultan) vorhanden. Realität jedoch ist: noch während X seinen Wert an Y übermittelt und während Y darauf reagiert, kann sich der Wert bei X schon geändert haben, und Y antwortet dann auf eine Situation an X, die dort nicht mehr vorliegt.

3.2 Parallelverarbeitung / Concurrency

Hier werden die Begriffe 'parallel' und 'nebenläufig' bzw. 'concurrent' so gut wie synonym gebraucht. 'Parallelverarbeitung' bedeutet hier immer eine Aktionsweise von Systemteilen, bei denen diese auf weite Strecken unabhängig (autonom) voneinander agieren, bis sie wieder an bestimmten Zeiten und Orten ihre Synchronisations-Informationen austauschen. Es ist lediglich abhängig vom Detaillierungsgrad der Betrachtungsweise, ob man einen Teilprozeß als *einen* Schritt oder als mehrere, neben- oder hintereinander, auffaßt.

Man könnte meinen, daß 'parallel' zu sagen eher in Fällen wie bei zwei Eisenbahnwaggons angebracht wäre, die nebeneinander auf zwei Schienen geführt werden, und 'nebenläufig' bei parallelen Fertigungsstraßen einer Fabrik. Man redet aber typischerweise auch von Parallelverarbeitung von Programmen in einem Computersystem – so daß eine weitere sprachliche Unterscheidung hier nicht sinnvoll erscheint.

4 Folgerungen

Bei den im Laufe der Jahre zusammengetragenen Betrachtungen könnte man zunächst vielleicht denken, daß es sich in jedem Einzelfall um typische Probleme dieses speziellen Falles handle. Diese Ansicht kann nicht mehr aufrechterhalten werden.

Das Mittelwert-Beispiel war ein konstruierter Fall, ein technisches Problem. Es hat den Status eines Original-Systems. Für das Resultat, also die neuen Werte von x und y , waren

Dinge maßgebend, die eigentlich nichts mit Berechnen zu tun hatten (wie etwa Addieren, Dividieren, Speichern), sondern das Ergebnis hing von Zeitbedingungen ab, vom Zeitpunkt des Zugreifens auf Werte, genauer: Lesen aus Speicherplätzen (und Hineinschreiben), d.h. von der Arbeitsgeschwindigkeit. Aber die kann gerade durch systemfremde Parameter (z.B. Interrupt-Verarbeitung) beeinflusst werden, und 'Einfluß systemfremder Parameter' bedeutet: 'Zufälligkeit des Ergebnisses'.

Das Philosophen-Problem ist ein natürliches Beispiel, mit Menschen als Akteure; es hat auch den Status eines Original-Systems. Auch hier hängt der Fortgang des Geschehens nicht von der Sache ab (also hier von der philosophischen Materie, die in den Köpfen der Philosophen gewälzt wird), sondern wiederum nur von Zeitbedingungen, vom Zeitpunkt des Zugreifens. Der springende Punkt, nämlich wann und weshalb sie zu den Löffeln greifen wollen, taucht in der Aufgabenbeschreibung gar nicht auf. Also auch hier: diese Zeitpunkte sind durch systemfremde Parameter gesteuert, und deshalb ist auch hier das System-Verhalten ein zufälliges Ergebnis; und wenn sie blind und taubstumm zugleich wären, d.h. nicht miteinander kommunizieren könnten, würde manch einer von ihnen wohl verhungern. Technische Systeme (s.o.: Mittelwert-Beispiel) haben allerdings nicht die Fähigkeit, von sich aus Kommunikation miteinander aufzunehmen, deshalb müssen sie 'abstürzen'.

Wir sprechen hier noch nicht von Simulationen, beide Beispiele sind Original-Systeme, in dem einen agieren nur Automaten (Programmteile), in dem anderen Menschen. Wir können hier generalisieren, d.h. von den speziellen Inhalten der Problemstellungen abstrahieren, und schließen: in allen Fällen paralleler Prozesse ist deren Verlauf bestimmt durch Zeitbedingungen, durch die Synchronisationspunkte. Dementsprechend sind viele, häufig genug unendlich viele Verhaltensweisen, unendlich viele Resultate möglich, und das wohlgemerkt bei dem Original-System. Wenn diese unendlich viele Resultate in der Realität nicht beobachtet werden, so mag das drei Gründe haben:

- o wir haben nicht genügend Zeit, alle Resultate zu erfassen;
- o vielleicht läuft (würfelt) dieser Prozeß nur einmal (wie z.B. in der Vererbung);
- o wahrscheinlicher jedoch ist, daß das System unvollständig beschrieben ist, daß es also noch geheime Regeln gibt, die die Kooperation steuern.

Wie viele Verhaltensweisen (Resultate) in einem parallelen, konkurrenten Prozeß möglich sind, muß aus den sachlichen Gegebenheiten ermittelt werden. Diese Methode der System-Betrachtung steckt noch in den Kinderschuhen. Ganz gewiß müssen wir die Geradeaus-Vorstellung aufgeben, nach der jede Aufgabe ein eindeutiges Resultat hat.

Zwar bedeutet Modellbauen immer: Vergrößern. Aber wenn schon das Original-System noch viele Verhaltensmöglichkeiten hat und wir seine Wirkungsweise nicht kennen, können wir erst recht kein besseres, 'korrekteres' Simulations-Modell für dieses System bauen.

Wir sollten aber beim Vergrößern nicht so weit gehen, daß wir das essentiellste Charakteristikum der parallelen Prozesse, nämlich den Zeitpunkt und die Art der Synchronisation (d.h. welche Dinge/Informationen wann und wo ausgetauscht werden), mit herausvergrößern, nur weil wir dann schöne, reproduzierbare Resultate bekommen – das wäre wie Schiffbau zu betreiben, ohne die für das Schwimmen bedeutsamste Voraussetzung, nämlich daß Wasser die Dichte 1 hat (Salzwasser ein wenig mehr), zu beachten.

Um noch einer Verwechslung vorzubeugen: Erst hinterher, wenn man die Historie eines parallelen Prozesses kennt, kann man sie als eine sequentielle Lebenslinie dieses Prozesses darstellen. Auch das in der Literatur wohlbekannte Beispiel der Petrischen Feuerbrigade beschreibt einen Prozeß, wie er abgelaufen *ist*, nicht wie er ablaufen *könnte* oder *wird*.

Literaturangaben

Eigene Arbeiten mit unmittelbarem Bezug (hier einzeln zitiert):

- [1] H.Fuss: *Simulation of Distributed Systems – A Competitive 3-Body Case Study*
in: W.Ameling (Ed.): Proc. 1st Europ. Simulation Congress ESC'83 (Aachen), Informatik-Fachberichte 71, pp. 323–328; Springer (1983)
- [2] H.Fuss: *Is Parallel Processing a Temporal Issue Only?*
in: D.Murray-Smith/D.Stephenson/R.Zobel (Eds.) Proc. 3rd Europ. Sim. Congr. ESC '89 (Edinburgh), pp. 35–40; SCS Simulation Councils Inc., (1989)
- [3] H.Fuss: *Temporal and Spatial Aspects in Parallel Processing*
in: Proc. 5.Internat. Symposium CLANP, Bulgar.Academy Sc. (1989)
- [4] H.Fuss: *Maximale Parallelverarbeitung und asynchrone Abläufe – oder: War der Golfkrieg unvermeidlich?* in: Dj. Tavangarian (Hrsg.): *Simulationstechnik*. Proc. 7. ASIM-GI-Sympos. (Hagen), Fortschritte in der Simulationstechnik, S. 102–106; Vieweg (1991)

[5]. Eigene Arbeiten mit mittelbarem Simulations-Bezug (hier zusammengefaßt zitiert):

- H.Fuss: *Wie gut können Simulations-Programme sein?*
in: W.Glatthaar (Hrsg.) Kurzvorträge, S. 61, Proc. 6. GI-Jahrestagung, Stuttgart, 1976
- H.Fuss: *Verteilte Simulationen*
in: M.Goller (Hrsg.): *Simulationstechnik*. Proceedings 1. ASIM-GI-Symposium Simulationstechnik (Erlangen); Informatik-Fachberichte 56, S. 283–288; Springer (1982)
- H.Fuss: *Simulation paralleler und parallelisierter Prozesse*
in: F.Breitenecker, W.Kleinert (Hrsg.): *Simulationstechnik*. Proc. 2. ASIM-GI-Sympos. Simulationstechnik (Wien), Informatik-Fachberichte Nr. 85, S. 344–348; Springer (1984)
- H.Fuss: *Improvement of Simulations by Place-Transactor-Nets*
in: A.Javor (Ed.): *Simulation in Research and Development*. Selected papers of IMACS Europ. Simulation Meeting, Aug.'84; Eger, Hungary. pp. 85–91; N.Holl. Publ.Co. (1985)
- H.Fuss: *General Aspects in Modelling Modular Systems*
in: Proc. 3.Internat. Symposium CLANP, pp. 38–49, Bulgar.Academy Sc. (1985)
- H.Fuss: *Nebenläufigkeit in Programmsystemen*
in: H.H.Witte (Hrsg.) ASIM-Mitteilungen Heft 21, ASIM (1990)
- H.Fuss: *Systembeschreibung und -Simulation mit Petri-Netzen*
in: D.Krönig/M.Lang (Hrsg.): *Physik und Informatik – Informatik und Physik*. Proc. Arbeitsgespräch DPG/ITG/GI, Nov. 1991, München. IFB Nr. 306, S. 224–227; Springer, 1992

Unmittelbare Vorarbeiten zu dieser Arbeit:

- [6] H.Fuss/G.Scheschonk (Hrsg.): *ASIM-Mitteilungen aus den Arbeitskreisen Heft 26*
7. Workshop Simulations-Methoden und -Sprachen für verteilte Systeme und parallele Prozesse (22./23. April 1991, Berlin). Zusammenfassungen d. Vorträge, 79 S., ASIM (1991)
- [7] H.Fuss: *Wie simuliert man am bequemsten halbgeordnete Prozesse?!*
in: ASIM-Mitteilungen Heft 40, TU Wien (1994)

Andere eigene Arbeiten mit mittelbarem Bezug

- H.Fuss: *Zur Simulation von Zufall und Verlässlichkeit*
in: D.P.F.Möller (Hrsg.): *Simulationstechnik*. Proc. 3. ASIM-GI-Symp. Simulationstechnik (Ebernburg); Inf.F.Ber. Nr. 109, S. 141–146; Springer (1985)

- H.Fuss: *Simulating 'Fair' Random Numbers*
in: Proc. 2nd Europ. Simulat. Congress ESC '86 (Antwerpen), pp. 250-257; SCS (1986)
- H.Fuss: *Parallelism in Representation, Design, and Simulation of Automatic Production Systems*; in: IEEE Comp.Soc. Press, Vol.797, pp.121-123; Washington DC, USA (1987)
- H.Fuss: *Petri-Netze und Meßgenauigkeit*
in: G.Kampe/F.Breitenecker (Hrsg.) ASIM-Mitteilungen Heft 18, S. 60-66. ASIM (1990)
- H.Fuss: *Wie lang ist ein Meter-Stab? - Reflexionen über Daten in Simulationsmodellen*
in: F.Breitenecker/I.Troch/P.Kopacek (Hrsg.): *Simulationstechnik*. Proc. 6. ASIM-GI-Sympos. (Wien), Fortschritte in der Simulationstechnik, Bd.1, S. 102-106; Vieweg (1990)
- Literatur zur Theorie der Parallelen Abläufe / Concurrency / Petri-Netze**
- B.Baumgarten: *Petri-Netze*, Grundlagen und Anwendungen.
369 S. B.I.Wissenschaftsverlag, Mannheim 1990
- E.Best, C.Fernández C.: *Nonsequential Processes. A Petri Net View*.
112 p., EATCS Vol.13, Springer 1988
- H.Fuss: *P-Ta-Netze zur Simulation von asynchronen Flüssen*
in: LNCS Vol.26, S. 326-335; Springer (1975)
- H.Fuss: *AFMG - Ein asynchroner Fluss-Modell-Generator*
Berichte der GMD Nr.100. 127 S. GMD (1975)
- H.Fuss: *Reversal Simulation with Place-Transactor-Nets*
in: H.Wedde (Ed.): *Adequate Modeling of Systems*. (Proc. Internat. Working Conf. on Model Realism, Bad Honnef/Bonn, April 1982), pp. 222-232; Springer (1983)
- H.Fuss: *Petri Net Languages for Automation of Distributed Systems and Processes*
in: IEEE Comp. Soc. Press (Vol.506, pp.159-162), Silver Spring, Md, USA (1983)
- H.Fuss: *Numerical Simulations with Place/Transactor-Nets*
in: K.Voss/H.Genrich/G.Rozenberg (Eds.): *Concurrency and Nets*, pp.187-199; Springer 1987
- W.Reisig: *Petri-Netze*. Eine Einführung. 158 S.; Springer 1982
- W.Reisig: *Systementwurf mit Netzen*; 125 S.; Springer 1985
- G.Rozenberg (ed.) *Advances in Petri Nets 19xx*. LNCS-Reihe; Springer (jährl. mind. 1 Bd.)
- E.Smith: *Zur Bedeutung der Concurrency-Theorie für den Aufbau hochverteilter Systeme* (164 S., Dissertation) GMD-Bericht Nr. 180; R.Oldenbourg (1989)
- C.A.Petri: *Kommunikation mit Automaten*. IIM, Uni Bonn; Schrift Nr. 2, 1962
- C.A.Petri: *Introduction to General Net Theory*; in: LNCS Vol. 84, pp. 1-19; Springer (1980)
- C.A.Petri: *Concurrency*; in: LNCS Vol. 84, pp. 251-260. Springer (1980)
- C.A.Petri: *State-Transition Structures in Physics and in Computation*
in: Int.J.Th. Physics, Vol. 21, No. 12, pp. 979-992, (1982)
- C.A.Petri: *Concurrency Theory*; in: LNCS Vol. 254, pp. 4-24. Springer (1987)
- C.A.Petri, E.Smith: *Concurrency and Continuity*; in: LNCS Vol. 266, pp. 273-292. (1987)
- C.A.Petri: *On technical Safety and Security*; in: PN NewsL 33 (8/89) und 35 (4/90)
- Petri Net Newsletter (3 pro Jahr) der GI-FG 1.1.2 'Petri-Netze und verwandte System-Modelle', Gesellschaft für Informatik (GI), Bonn.
- Eine umfassende Bibliographie dieses Gebiets findet sich ab 1987 von Zeit zu Zeit in dem jeweiligen Jahresband der Petri-Netz-Tagung 'Advances in Petri Nets' in der Springer-Reihe 'Lecture Notes in Computer Science', zuletzt 1991 mit über 4000 Einträgen.

Maximale Parallelverarbeitung und asynchrone Abläufe oder: War der Golfkrieg unvermeidlich?

Dr. H. Fuss

Institut für Methodische Grundlagen (F1)
Gesellschaft für Mathematik und Datenverarbeitung (GMD)

Zusammenfassung

Anhand eines einfachen parallelen Programms wird dargelegt, welche stillschweigenden – und z.T. realitätsfernen – Annahmen über den Ablauf von parallelen Prozessen, u.a. auch über die Funktionsweise von Mehrprozessor-Maschinen, bestehen.

Ziel ist es, durch erhöhte Einsichten in die tatsächlichen Abläufe konkurrierender realer Systeme, bessere Programme (Betriebssysteme, Simulationsprogramme etc.) schreiben zu können; oder zumindest nicht von Resultaten überrascht zu werden, deren Möglichkeit einzutreffen gar nicht in Betracht gezogen worden ist.

Die Ergebnisse der Betrachtungen einer asynchronen Berechnung werden auf andere Systeme (organisatorische, ökonomische, politische etc.) übertragen.

1. Aufgabenstellung

Was macht das folgende parallele Programm \mathcal{P} ? Was wird für x und y berechnet?

```

initial (x:=0, y:=1);
P while x ≠ y
do   x :=  $\frac{(x+y)}{2}$  || y :=  $\frac{(x+y)}{2}$    od;
print (x, y); end;
```

Man könnte meinen, hier handele es sich um eine trivial einfache Aufgabe: Nach Initialisierung stellt das *while*-Statement Ungleichheit fest, die beiden parallelen Zweige werden einmal ausgeführt (und errechnen für $x=0.5$ und $y=0.5$), dann stellt das *while* Gleichheit fest, und damit ist das parallele Programm fertig, es werden die Werte für x und y ausgegeben.

Doch: dieser Ablauf ist nicht zwingend so, im Gegenteil: für *unabhängig* nebeneinander ablaufende Programme – was ja einen erhöhten Grad an Parallelität bedeutete – wäre das Ergebnis $x=y=0.5$ höchst unwahrscheinlich. Das Ergebnis der Berechnung ist abhängig von dem Grad der Kopplung bzw. Entkopplung, also von dem Maß an Freiheit, den die beiden parallelen arithmetischen Statements haben.

Natürlich kann man Maschinen bauen oder so programmieren, daß sie $x=0.5, y=0.5$ in einem Parallelschritt errechnen; eine derartige Version des Programms sei mit \mathcal{P}_0 bezeichnet.

Aber wir wollen hier den Fall und die daraus sich ergebenden Folgerungen untersuchen, wenn x und y auf zwei unabhängig voneinander arbeitenden Prozessoren P_x und P_y berechnet würden – und welcher Synchronisierungsbedarf sich dann ergäbe.

2. Problemfeld

2.1 Zentralsteuerung

Daß die Ausführung des Programms \mathcal{P} (d.h. der Prozess \mathcal{P}) immer $x=y=0.5$ ergibt, ist nur dann möglich, wenn eine übergeordnete Zentrale die Prozessoren¹ P_x und P_y beaufsichtigt, synchronisiert und steuert (und auch noch das \parallel -Zeichen als enggekoppelt interpretiert wird). Um anzuzeigen, daß man Zentralsteuerung meint, könnte man \mathcal{P} umschreiben zu \mathcal{P}_1 :

$$\mathcal{P}_1 \quad \begin{array}{l} \text{initial } (x:=0, \ y:=1); \\ \text{when } x = y \text{ stop;} \\ \text{else do } \quad x := \frac{(x+y)}{2} \parallel y := \frac{(x+y)}{2} \quad \text{od;} \end{array}$$

wobei hier völlig unklar bleibt, wie das *gleichzeitige* Beaufsichtigen von x und y und das anfallende *sofortige* Abstoppen *beider* Prozesse¹ P_x und P_y zu realisieren wäre. Diese Version steht hier nur als eine Arbeitsbeschreibung, auf sie wird nicht weiter eingegangen.

2.2 Nebenläufigkeit (Asynchronie)

Bei der hier beabsichtigten Erweiterung der Auffassung von Parallelverarbeitung kann natürlich auch das Testen parallelisiert werden; dann schreibt sich das Programm \mathcal{P} als \mathcal{P}_2 so:

$$\mathcal{P}_2 \quad \begin{array}{l} \text{initial } (x:=0, \ y:=1); \\ \text{while } (x \neq y) \text{ do } (x := \frac{(x+y)}{2}) \parallel \text{while } (x \neq y) \text{ do } (y := \frac{(x+y)}{2}) \\ \text{else exit;} \quad \quad \quad \text{else exit;} \end{array}$$

Unnötig zu sagen, daß die Synchronisierung der Prozesse über einen globalen Speicher für x und y erfolgt, zu dem beide Prozessoren P_x und P_y ständig Zugriff haben – wobei wir hier auf die in der Informatik bekannten (und dort vielfältig abgehandelten) Probleme des gleichzeitigen/exklusiven Lese-/Schreib-Zugriffs nicht eingehen wollen.

Uns interessieren hier nur diejenigen Fragen, die durch die Erweiterung der Parallelverarbeitung zu Prozessen, die *unabhängig* voneinander gleichzeitig laufen, hinzukommen.

Einige davon sind, sobald wir uns die im tatsächlichen realen Geschehen auftretenden Zeitbedingungen vergegenwärtigen, wie folgt:

Wann liest das 'while' die Werte x und y ? Einzeln, gleichzeitig? Wie lange dauert das?

Wie lange dauert die Ausführung der rechten Seite des Statements, wann geschieht die Zuweisung zur linken Seite, wann ist sie beendet? Was macht in der Zeit der andere Prozessor?

Wann beginnen die Prozessoren, das arithmetischen Statement abzuarbeiten? sofort?

Muß ein Prozessor überhaupt beginnen/rechnen/zuweisen, kann er nicht gerade – in einer *multi-tasking*-Umgebung – mit einer anderen Arbeit (z.B. Fehlerkorrektur) beschäftigt sein?

Wie oft kommen die beiden arithmetischen Statements dran (gleich oft? warum sollten sie??), und wer überprüft das? Was passiert, wenn die Uhren der beiden Prozessoren unterschiedlich schnell gehen: a) im konstanten Verhältnis unterschiedlich, b) unregelmäßig schnell?

(Prüfungsfragen an Informatik-Studenten: Konvergieren x und y ? Wenn ja: wogegen? wann? ('eventually'?) Terminiert das Programm überhaupt?)

¹ Da kaum Verwechslungen möglich sind, werden hier sowohl die Prozessoren, die x und y berechnen, mit P_x und P_y bezeichnet, als auch die (Berechnungs-)Prozesse selbst. Ebenso wenig scheint eine typographische Unterscheidung zwischen den Variablen x und y , ihren Speicherplätzen, und ihren gegenwärtigen Werten nötig.

3. Maximaler Parallelitäts-Ansatz

Wir bezeichnen mit P_x das Programm: $x := \frac{(x+y)}{2}$, mit P_y das Programm: $y := \frac{(x+y)}{2}$.

Dann schreibt sich der Kern von P_2 als

P_2 while (x \neq y) do P_x
else exit; || while (x \neq y) do P_y
else exit;

Wir wollen hier, um die Situation deutlich zu machen, beispielhaft *einen* Lösungsweg verfolgen (es gibt derer, wie sich zeigen wird, unendlich viele; und falls wir die 'echte' Zeit auch noch in Betracht ziehen, gibt es gar überabzählbar viele Lösungen).

Es ist erfahrungsgemäß frapierend und verwirrend, wenn man zum erstenmal die gewohnten Wege der Informatik verläßt und nicht mehr den alle Schritte zwangsweise gleichschaltenden Begriff von 'Parallelität' verwendet. Deshalb wollen wir hier ein Ergebnis tatsächlich schrittweise vorrechnen.

Dazu versetzen wir uns in die Lage eines Beobachters, der die beiden weitestgehend unabhängig voneinander agierenden Prozesse gleichzeitig beobachten kann. Wir unterstellen einen Ablauf, der langsam genug ist, daß dem Beobachter nichts verloren geht, und bei dem nicht zu viele Dinge gleichzeitig geschehen, so daß er immer die Übersicht behalten kann.

Wir führen ein Protokoll unserer Beobachtungen. Darin gibt es folgende Einträge:

1. Beobachtungs-Nummer (gleichbedeutend mit einem Zeitpunkt, von dem wir aber nicht seinen Wert (also die Uhrzeit), sondern nur die Tatsache, daß etwas stattgefunden hat, notieren.
2. Ereignis; 3., 4. gegenwärtige Werte von x bzw. y; 5. Bemerkungen.

Wir beginnen unsere Beobachtungen nach erfolgter Initialisierung von $x=0$ und $y=1$

B-Nr	Aktion	x	y	Sonstiges
1	P_x liest x und y, Test	0	1	P_y ruht
2	P_x rechnet	0	1	P_y ruht
3	P_y liest x und y	0	1	
4	P_x speichert ab	1/2	1	P_y liest immer noch (y)
5	P_x liest x und y	1/2	1	P_y hat $x=0, y=1$ gelesen
(5')	...	1/2	1	Variante: P_y hat $x=0.5, y=1$ gelesen)
6	P_x testet ($x=0.5, y=1$)	1/2	1	P_y testet ($x=0, y=1$)
7	P_x ist schnell	3/4	1	
8	P_x ist schnell	7/8	1	
9	P_x ist schnell	15/16	1	P_y rechnet
10	P_x liest	15/16	1	
11	P_y speichert	15/16	0.5	
12	P_x rechnet, speichert	31/32	0.5	
13	P_x liest	31/32	0.5	
usw.				

Wie schon oben gesagt, dies ist *ein* möglicher Ablauf von vielen. Mögliches Ergebnis ist, wie man sieht, im maximalen Unabhängigkeitsfall *jeder beliebige*² Wert zwischen 0 und 1. Wenn man dagegen $x=y=0.5$ als den berechneten Wert haben will, muß man die Maschinen dementsprechend enggekoppelt konstruieren oder Programme mit gleichgetakteter Parallelität schreiben. Reale räumlich verteilte Systeme (Maschinen) ohne Synchronisation, und unabhängige Menschen bzw. Gruppen verhalten sich a priori nicht wie P_1 , sondern wie hier in P_2 bzw. P_2' vorgerechnet wurde.

²genauer: jeden beliebigen Wert approximierende Binärzahl

4. Vom Wesen der übergeordneten Instanz (arbiter)

4.1 Kooperierende Systeme

Es ist ein fundamentaler Unterschied, ob die parallele Berechnung unseres Beispiels *immer* $x=y=0.5$ oder einen beliebigen (und dazu noch bei wiederholtem Programmablauf normalerweise jedesmal einen anderen) Wert ergibt; es kommt dabei auf die Steuerung³ und Synchronisation der parallelen Prozesse an. Da man von Automaten *reproduzierbare* Ergebnisse erwartet, hat sich in der Informatik die Version P_0 (enge Kopplung) als die Auffassung von Parallelität durchgesetzt; etwas anderes erscheint folgerichtig als 'Fehler'.⁴

Diese vereinfachende Sicht von 'Parallelität' mag in der klassischen Informatik ihre Berechtigung haben, aber in den Fällen, in denen keine übergeordnete Instanz Rechenzeiten vergibt oder zum Gleichtakt zwingt – und das ist z.B. in allen sozio-technischen Systemen der Fall – werden die Abläufe in den Systemen mehr oder minder wie im Programmfall P_2 sein. Das ist z.B. in jedem Filialbetrieb so – und es ist so gewollt! –, daß jede Unterabteilung in einem gewissen Rahmen ihre Entscheidungsfreiheit hat (und, um in der obigen Sprechweise zu bleiben, ihre Uhren so schnell gehen können, wie sie es will); das gewünschte Funktionieren muß durch geeignete *organisatorische* Maßnahmen (z.B. gegenseitige Benachrichtigung, Lieferung gegen Quittung etc.), und eben nicht: Zeitbedingungen, gewährleistet werden.

Beim Modellieren und Schreiben von Simulationsprogrammen für derartige Systeme sollte man sich des (Ent-) Kopplungsgrades bewußt sein, damit die Simulationsläufe realitätsgerechte (und nicht in erster Linie: reproduzierbare) Ergebnisse⁵ liefern.

4.2 Konkurrierende Systeme

Viel schwieriger zu beschreiben und zu modellieren sind konkurrierende Systeme, wie sie z.B. im Sport oder auf dem Markt auftreten. Dort ist es ja gerade das Bestreben der Teilnehmer, *nicht* zu kooperieren, sondern durch ihr eigenes Verhalten das angestrebte Ziel des Konkurrenten zu vereiteln. Tennisspieler versuchen normalerweise nicht, den Ball möglichst lange im Spiel zu halten, Boxer tänzeln nicht 12 Runden gestikulierend herum; Anbieter können den Markt nicht vergrößern, so daß genügend großes und gutes Geschäft für alle übrigbleibt – der Beispiele gibt es viele. Wollte man solche Systeme in der Art von P_0 oder P_1 modellieren, so müßte man sich an, über den Ausgang des Spiels vorher Bescheid zu wissen oder Schicksal spielen zu dürfen. Trotzdem entsteht in der Realität kein Chaos, obwohl nach dem jetzigen Stand unserer Betrachtungen keine Kooperation stattfinden kann; sie ist ja auch gar nicht beabsichtigt.

Daß trotzdem in der Realität auch solche Systeme funktionieren, i.d.R. sogar einigermaßen gut und teilweise sogar vorhersagbar (d.h. prognostizierbar, auch ohne formales Simulationsprogramm), liegt an der Koordination durch die jeweiligen Spielregeln.

4.3 Hierarchien von Systemen

Jedes System ist ja eingebettet in seine Umgebung, mit der es in einer (geregelten) Input-/Output- oder Kommunikations-Beziehung steht. So ergeben sich Hierarchien von Systemen und damit Hierarchien von Spielregeln. Regelverletzungen eines Fußballspielers werden innerhalb des Spiels vom Schiedsrichter geahndet, und wenn sie von über das Spiel hinausgehender Bedeutung waren, wird er vom Verband gemäßregelt; sollte es sich um über den Verband hinausragender Bedeutung handeln, werden die ordentlichen Gerichte bemüht. Falls der Schiedsrichter ihn des Platzes verwiesen haben, er aber nicht gehen sollte, wird er durch Ordner, notfalls mit Polizeigewalt entfernt. So die Hierarchie der Maßregelungen.

³ Eine besondere Rolle spielt dabei die Sequentialisierung durch den einen *instruction counter*.

⁴ Man denke dabei an nicht reproduzierbaren Situationen in *multi tasking operating* Systemen.

⁵ Hier ist Skepsis vor der (unüberlegten) Benutzung von Zufallsgeneratoren angezeigt, denn auch sie steuern Abläufe so, als ob sie über zukünftige System-Informationen verfügten.

Bei Auseinandersetzungen zwischen Geschäftspartnern werden Schlichtungsstellen oder die Gerichte angerufen, kommt ein Partner seinen Verpflichtungen nicht nach, wird ggf. der Titel zwangsvollstreckt; im kriminellen Fall kommt das Strafrecht zum Zuge. Aber in bisher allen betrachteten Fällen endet die höchste Einbettung beim Gewaltmonopol des Staates. (Auch wenn der Internationale Gerichtshof einen Fall entschieden hat oder die UNO einen Beschluß gefaßt hat, muß dieser erst in nationales Recht umgesetzt werden.)

Die Situation könnte sich noch dadurch verkomplizieren, daß sich verschiedene Systeme (Einflußgebiete) überlagern, z.B. geschäftliche und politische, so daß 'Spiele' zwischen denselben Partnern auf unterschiedlichen Ebenen und nach unterschiedlichen Regeln ablaufen; aber trotzdem enden Spielhierarchien an den nationalen Grenzen – selbst wenn es sich um multi-nationale Konzerne handelt.

4.4 Ungeregelte Systeme

Nicht alle konkurrierenden Systeme haben einen (ggf. mehrere) Schiedsrichter, der Konflikte entscheiden und der die Entscheidung auch durchsetzen kann. Ohne Schiedsrichter spielen z.B. die Leute Schach (aber nicht, wenn es um die Weltmeisterschaft geht!) und kicken die Kinder Fußball auf der Straße. Nicht alle Entscheidungshilfen sind für die Gerechtigkeit von Wert: bei der Linienrichter-Maschine wird lediglich die 'Verantwortlichkeit' für eine Entscheidung in einer nicht reproduzierbaren Situation auf einen (im Detail) nicht überprüfbaren Automaten verlagert (dessen Entscheidungsfindung nicht reproduzierbar ist).

Aber auch solche Systeme können ihre Grenzen und Regelungen durch ihre Einbettung in ihrer Umgebung finden. Konfliktentscheidung erfolgt dann durch Informationszufuhr von 'außen', also durch eine höhergeordnete Instanz.

5. Analogien, Versuch einer Klärung

Unser eingangs genanntes Programm \mathcal{P} mit den beiden konkurrierenden Akteuren P_x und P_y in der Verständnis-Version \mathcal{P}_2 enthält alle betrachteten Aspekte. Es läßt alle Betrachtungsvarianten, und daraufhin alle möglichen Ergebnisse, zu. Es kann deshalb als Vehikel für Analogieschlüsse auch ausserhalb der Informatik dienen. Ziel dieser Betrachtungen war es, darauf hinzuweisen, daß nebenläufige Prozesse (höchst-parallele, konkurrierende Prozesse) nicht nach denselben einfachen Regeln ablaufen, wie sie in der klassischen Informatik unter dem Stichwort 'Parallelität' betrachtet werden. Außerdem kommen noch jedesmal die Spielregeln des den Prozess \mathcal{P} umhüllenden Prozesses dazu. Deshalb ist es so besonders schwierig, reale nebenläufige Prozesse richtig zu verstehen, strukturerhaltend abzubilden und adäquat zu simulieren.

Um nun auf die Titelfrage zurückzukommen: sie läßt sich aus informationstechnischer Sicht (nach den Kenntnissen aus der Theorie der Nebenläufigkeit, der *concurrency theory*) nur indirekt beantworten: Bei parallelen Prozessen ohne übergeordnete Instanz läßt sich kein Mechanismus angeben, der ein Teilsystem dazu zwingen könnte, sich mit dem anderen zu koordinieren. Jedes Ergebnis wäre möglich (s. Kap.3). Oder anders gesagt: die Analogie aus der Informatik zeigt, daß wegen des Fehlens einer übergeordneten Instanz jedes Ergebnis drohte, also der Golfkonflikt nur auf *politischer* Ebene hätte gelöst werden können. Das aber fällt in eine andere Wissenschaft als die hier unter dem Informations-Verarbeitungs-Aspekt behandelte Aufgabe aus dem Gebiet der Theorie der verteilten Systeme und parallelen Prozesse (Petri-Netze).

6. Literatur

- S.Drees, D.Gomm, H.Plünnecke, W.Reisig, R.Walter: *Bibliography of Petri Nets 1988* (2634 Einträge) 146 S, Arbeitspapiere der GMD Nr.315; GMD (1988)
 H.Plünnecke, W.Reisig: *Bibliography of Petri Nets 1990* (4099 Einträge) in: *Advances in Petri Nets 1991*, S.317 ff, Springer (im Druck)

HUMAN COMPUTER INTERFACE

P.C.P. BHATT

INDIAN INSTITUTE OF TECHNOLOGY
DELHI (INDIA)

The Lecture Plan

Part - 1 : * Some history and definitions

Part - 2 : * Software System Architecture

to Support HCI

Part - 3 : * Application Scenarios

Multi-Media Applications

Net-Sufring (Information Kiosk)

Co-ficiance and Co-operative work

The History of the Craft of Communication 1

Writing	Painting	Film
3300 B.C. Egyptian Heliographics	20,000 B.C. Cave Paintings Lascaux	1884 A.D. Thomas Ed. Kinetoscope
800 B.C. Library of Assyria	1100 AD Oil Paint	1903 The Great Train Robbery
1440 A.D. Gutenberg's type	1497 A.D. Da' Vinci's Last Supper	1983 Attenborough Gandhi
1719 Robinson cru- soe (Novel)	1937 Picasso's Guernica	1993 Spielberg's Jurrassic Park

The History of the Craft of Communication 2

Drama	Photography	Architecture	Music
600 B.C.	1839	2700 B.C.	2000 B.C.
Greek Amphitheater	Daguerre	Pyramids Sakkara	Egyptian Harp
1580 A.D.	1882 Eastman	1400 A.D. Chatres Cathedral	1520 A.D. First Violin
First Drama Company	Kodak	1936 Frank Llyod Wright	1720 J.C. Bach
1603 A.D. Shakespear's Hamlet	1947 A.D. Yosemite Photograph	Falling Water	Clavier
	Life Magazine Vietnam war pictures	1980s Charles Correa Low Cost Housing	1824 Beethoven's 9th Symphony
			199x Fusion Music

The History of the Craft of Communication 3

COMPUTERS

- * 450 B.C. First Abacus
- * 1864 A.D. Charles Babbage
- * 1955 First Commercial Computer
- * 1975 First Personal Computer
- * 1978 Visical 1979 Wordstar 1983 Lotus
- * Late 80s Story Board and Flip-books;
Touch screen learning
- * Early 90s Interactive Video and Video on demand
Multi-media tools for creative design; authoring
Virtual reality with visualisation and simulation

HCI Definitions

* Hewlett et.al. 1992 (ACM Curriculum Dev.)

.... a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of the major phenomenon surrounding them.

* Dix et.al. (HCI Book by PHI 1993)

.... the study of people, computer technology and the ways these influence each other. We study HCI to determine how we can make this comp. techonology more usable by people

* Preece et.al. (HCI Book AW 1994)

.... about designing computer systems that support people so that they can carry out their activity productively and safely

WHICH IS A GOOD HCI DESIGN ?

Norman (Technology watcher)

" The Psychology of Everyday Things "

- * Affordances : ... how it can be used
use of buttons, menu descriptors etc.
- * Constraints : ... Physical; semantic; logical factors;
encourage proper action, prevent errors
- * Conceptual Model : ... mental models of models ...
to enable them to predict the effects of
their actions.
- * Mappings : ... description of controls; their relationship
correspondence with command such as move
up, down left right with appropriate visibility

WHICH IS A GOOD HCI DESIGN ?

Contd...

Visibility : ... Make it easy to determine what actions are possible at any time (Use constraints)

Feed-back : ... Must follow the user's conceptual model
Its a good idea to provide alternatives based on plausible conformity with the model

State : ... Make it easy to let the user find out his current state. Helps in navigations.

Back-track : ... not always easy but one may provide an escape route from every where to a known state.

i.e. let the user know 1. what to do 2. what is going on

THE DESKTOP METAPHOR

.... the translucent screen on which material can be projected for convenient reading
there may be a set of 'buttons'; 'lev. ls' or key board for the input, control and navigation.

Memex : provides facilities for

- * Books * Current periodicals * News-paper
- * Correspondence * Graphics

Xerox (April 1981; Smith Irby etal.)

Xerox STAR

- | | |
|--|-------------------------------------|
| * Simulated desktop | * Direct Manipulations |
| * WYSISWYG | * ¹ Universal comds. for |
| * <i>Bitmapped Windows Display</i> | Move, update, delete |
| * <i>Windows and excellent file management</i> | |

THE DESKTOP METAPHOR

Apple LISA (Steve Jobs Contribution 1983)

- * Network Interface
- * Both an office and a personal productivity tool
- Interface with spreadsheet and other applications like business forecasting models

Apple Macintosh (1985)

- * essentially an enhanced LISA, a personal computer
- * Excellent graphics and laser print output
- * Ideal desktop tool with all the other adv. of LISA

CONTRIBUTIONS FROM ACADEMIA

Sketchpad (MIT 's contribution)

- * Internal representation of a picture may be defined in terms of sub-pictures object orientation
- * Icons and copies : iconise and when needed copy
- * Picture manipulation in realtime : transformation of pictures with no jitters (carried forward by program packages like pro-engr. and IDEAS)
(*Concurrent Engineering WS; surface tracking from models*)

Others : Sutherlands; Foley in USA

CONTRIBUTIONS FROM ACADEMIA

Collaborative Design (Aarhus University Denmark)

(Sussane Bodekar, Kaj Gronbaek and Morten Kyng)

- * Supports a team of designers to work on a common design project

- * Co-operative prototyping as in Project Imagine
(8 univ. in Europe and Siemens Germany)

- * Supports contextual query

(Note in GG project it may be gestural query)

(Imagine (siemens); Codiscovery project BNR)

AUGMENTING HUMAN INTELLECT

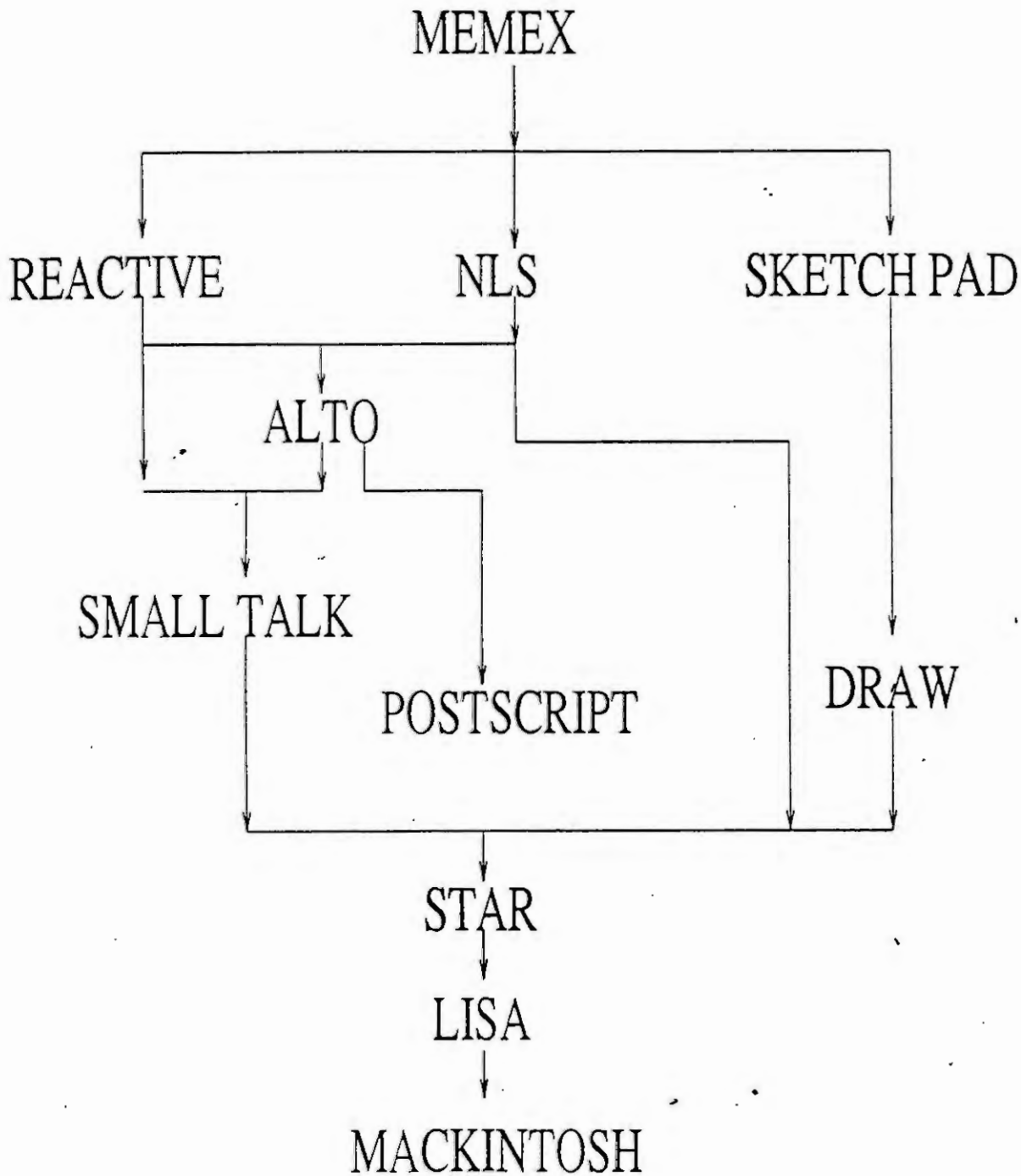
* " METHODS MATTER " Joseph McGrath

Language : *... the way an individual classifies his world into
the concepts that his mind uses to model
the symbols he uses and the meaning he associates with
them*

Methodology : *... the methods he uses in his ' goal-centered '
activity the procedures and strategies*

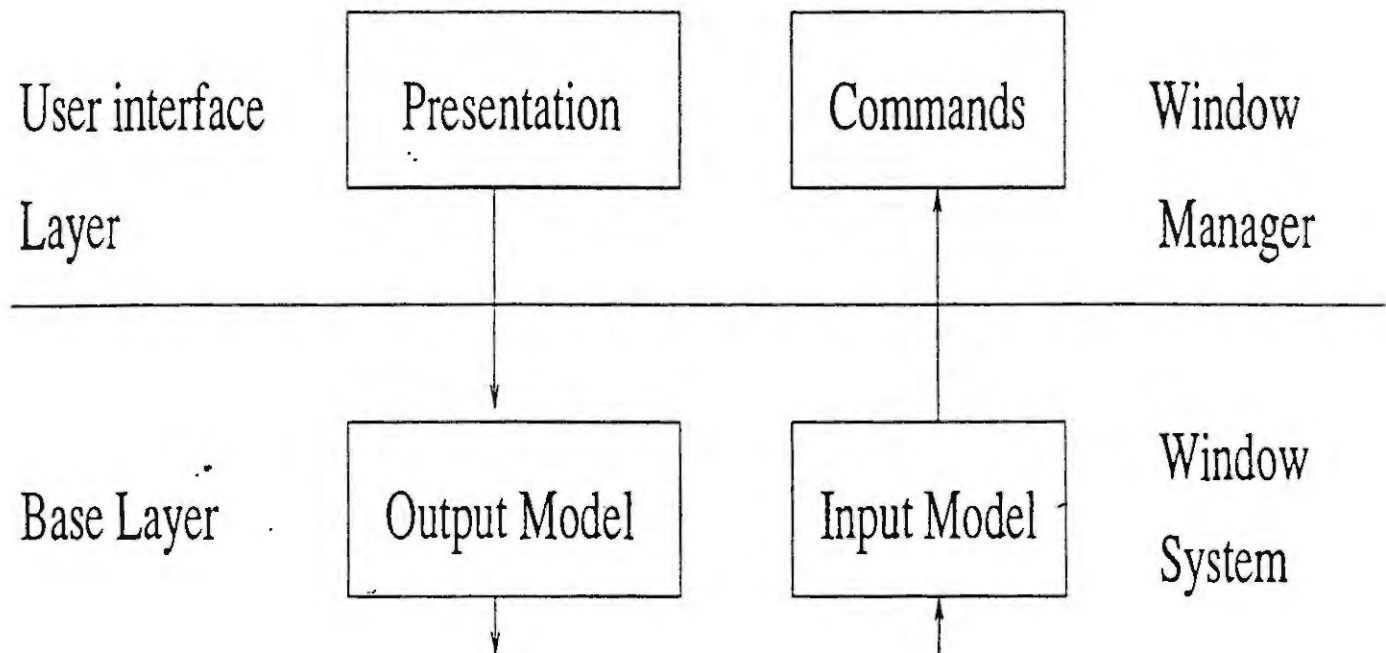
- * Minimise memorisation Selection of items ; not data entry
- * Engineer for errors besides error messages provide
an 'emergency exit ' at all times
help to recognise, diagnose and recover

SOFTWARE TOOLS AND ARCHITECTURE ... 1



SOFTWARE TOOLS AND ARCHITECTURE ... 2

The General Principle



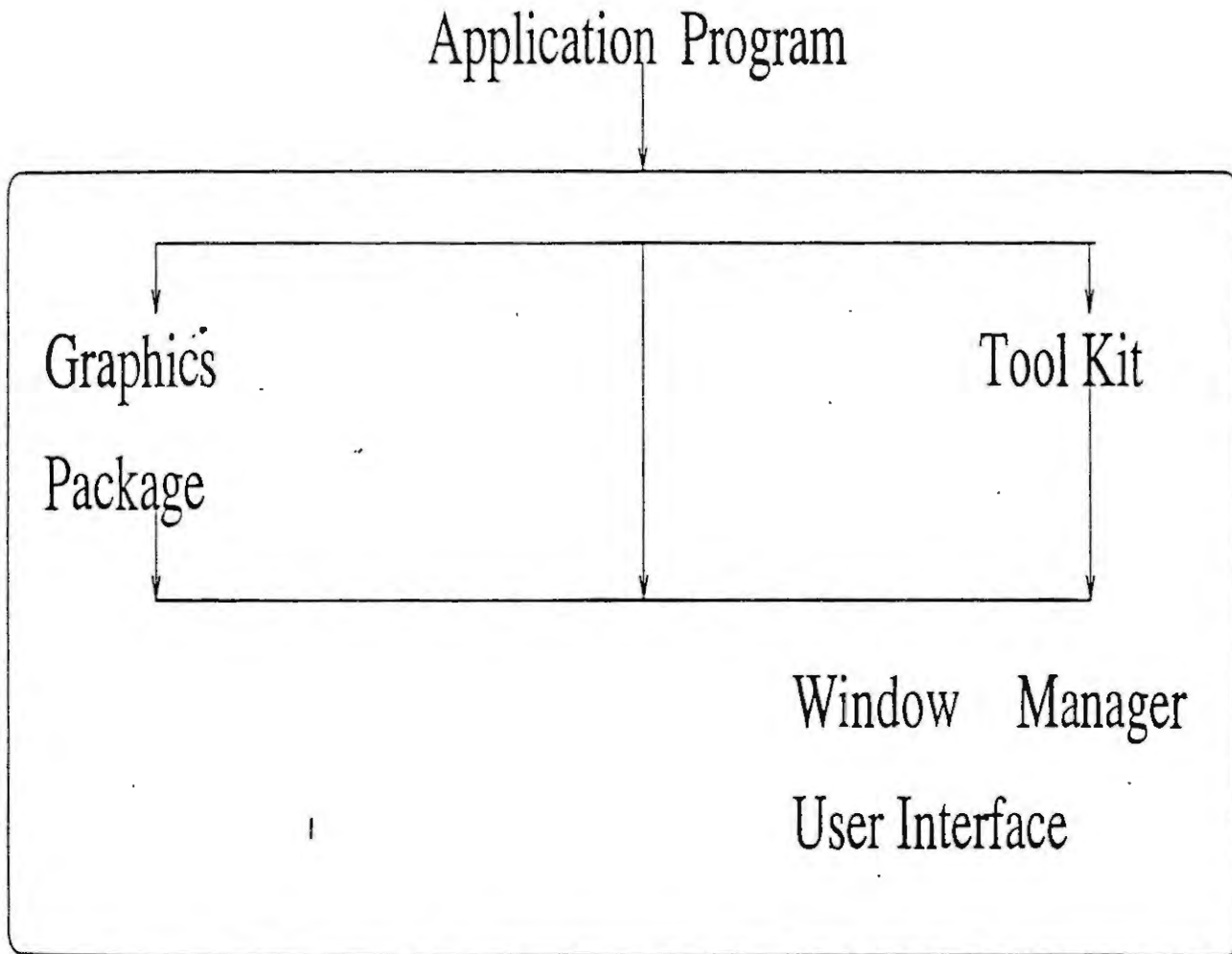
" A Window system is a software package that helps the user monitor and control different contexts by separating them onto different parts of one or more display screens "

Myers (1988)

" A Window Manager provides for input perception i.e. recognise event in the window; manage output graphics; flexibility in mgmt of window (like size; location etc.) and presentation styling (like Apple mac. toolbox),"

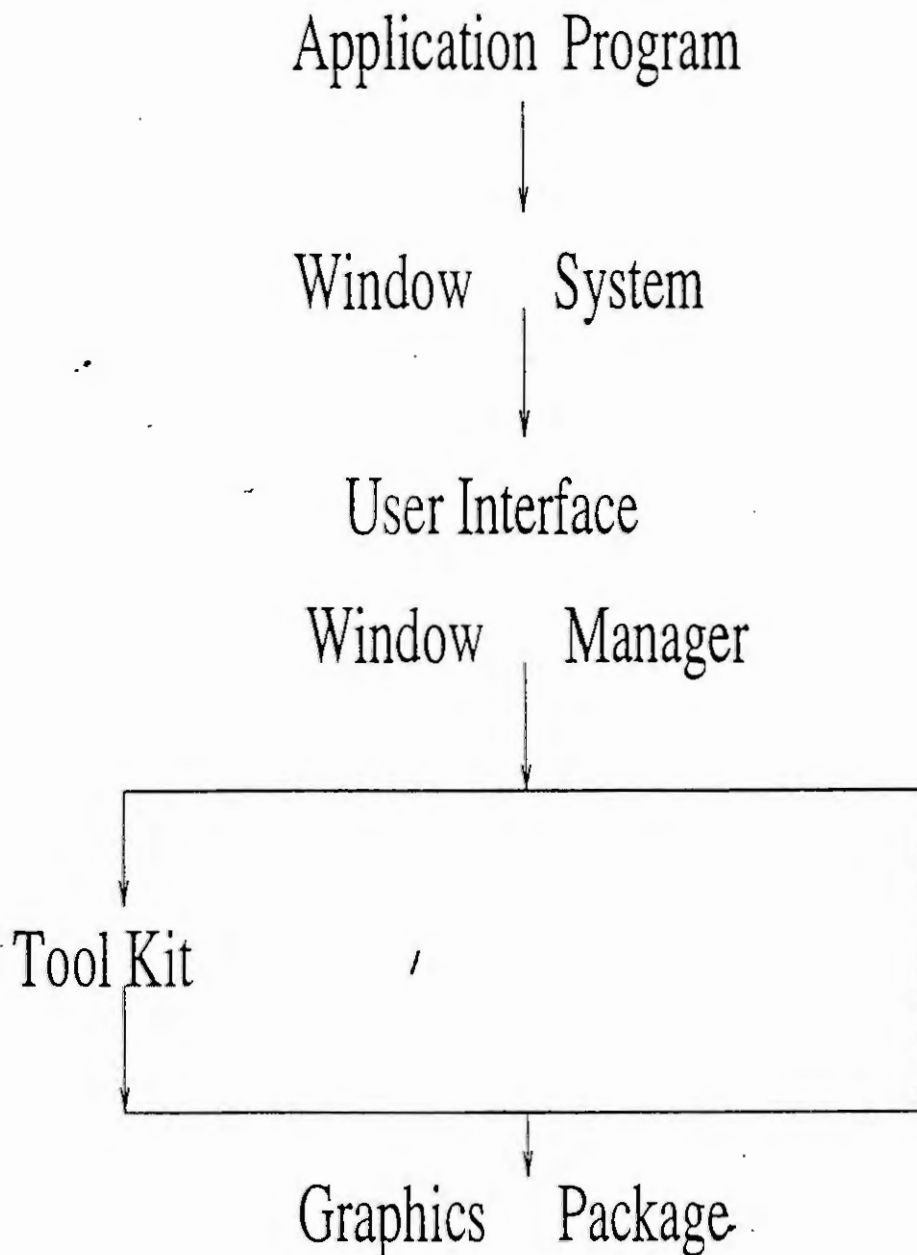
SOFTWARE TOOLS AND ARCHITECTURE ... 3

The SUN windows ...

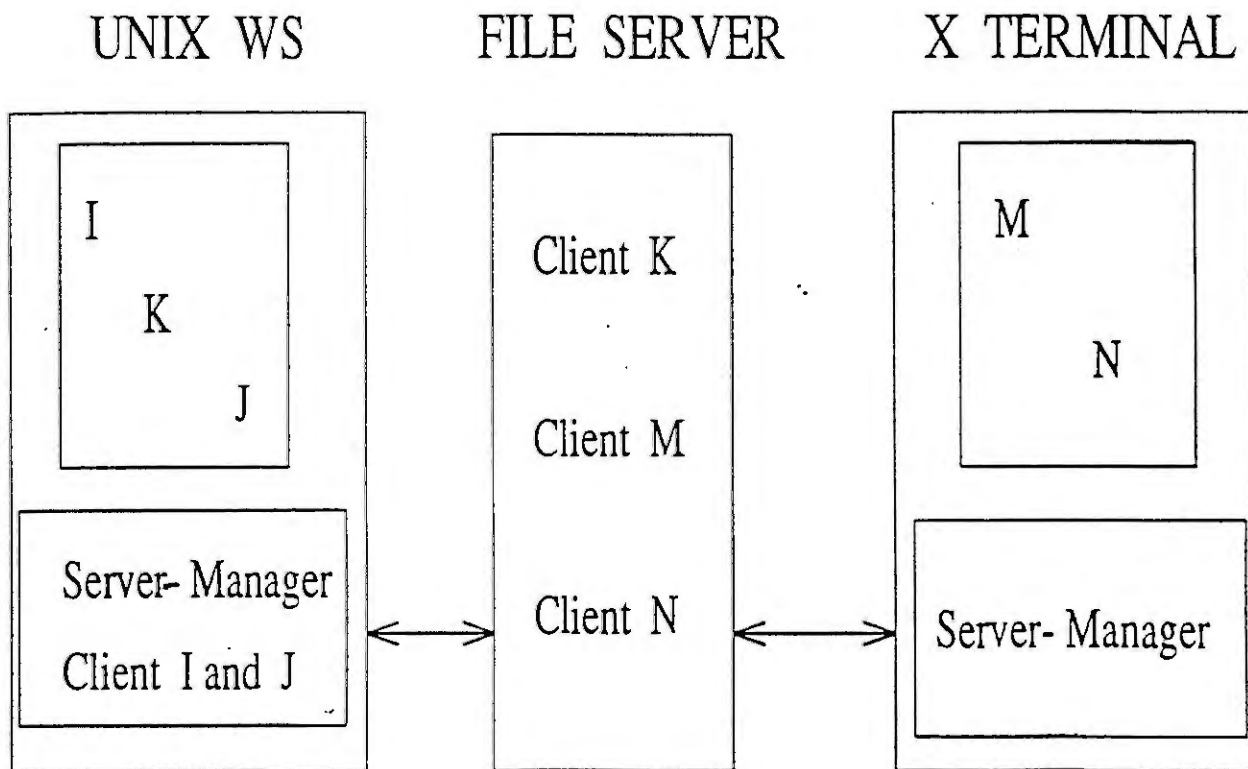


Window System

SOFTWARE TOOLS AND ARCHITECTURE ... 4

CEDAR... MACKINTOSH... NeXT

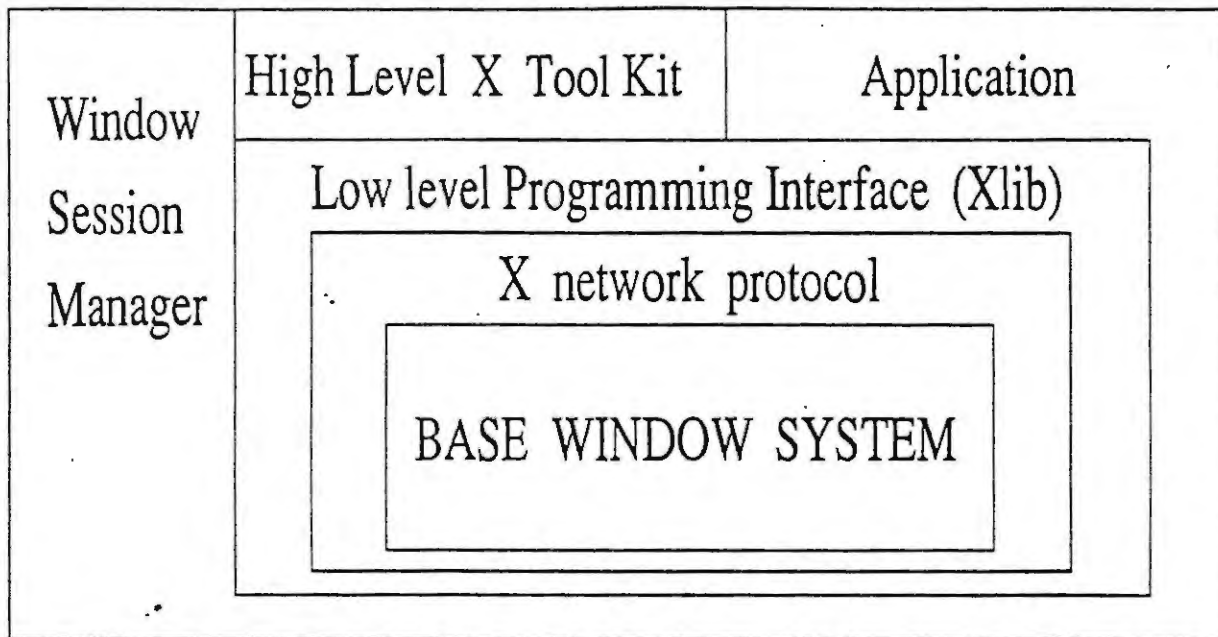
SOFTWARE TOOLS AND ARCHITECTURE ... 5



A configuration of Unix WS; file server and X terminal is shown. The WS as also the X terminal shall have X windows environment with their own servers. The windows managers may reside on the local machines or elsewhere(!). Note the client processes (I, J, K) are shown to be distributed. The main point is that regardless of their locations all clients receive their inputs and display output.

SOFTWARE TOOLS AND ARCHITECTURE ... 6

THE X ENVIRONMENT



Tool Kits : Tool kits is a reference to a collected set of interface building blocks (Often referred as Widgets). A programming language facilitates building the IIF.

Some well known Tool_Kits are : MOTIF, Inteviews, Mac-App

Typical widgets found in tool kits are :

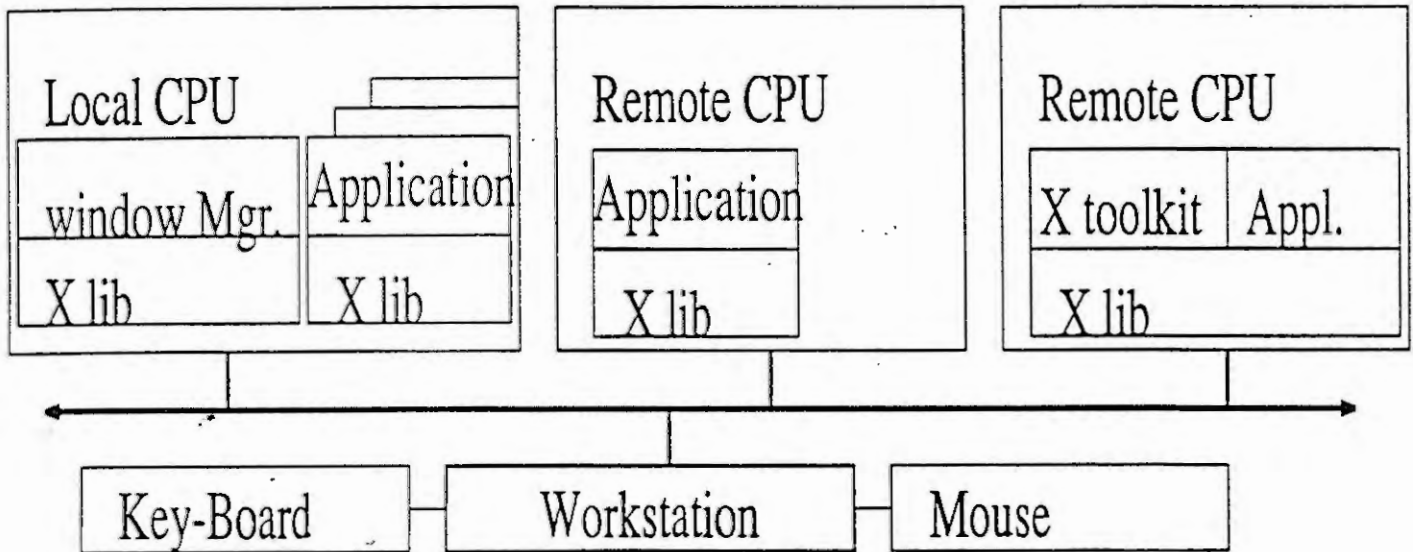
*BUTTONS, MENUS, SLIDERS, SCROLL-BARS, CONTROL PANELS
CANVASSES, DIALOGUE BOXES, TERMINAL WINDOWS
MAGIC LENSES ,*

*The NeXT IIF Builder provides a Pallette instead of having to write a prog.
Create and mark links with attributes assigned for multi media operations
i.e. for sound-stream with On/Off, Volume control etc.*

SOFTWARE TOOLS AND ARCHITECTURE ... 7

THE X ENVIRONMENT

The network transparency

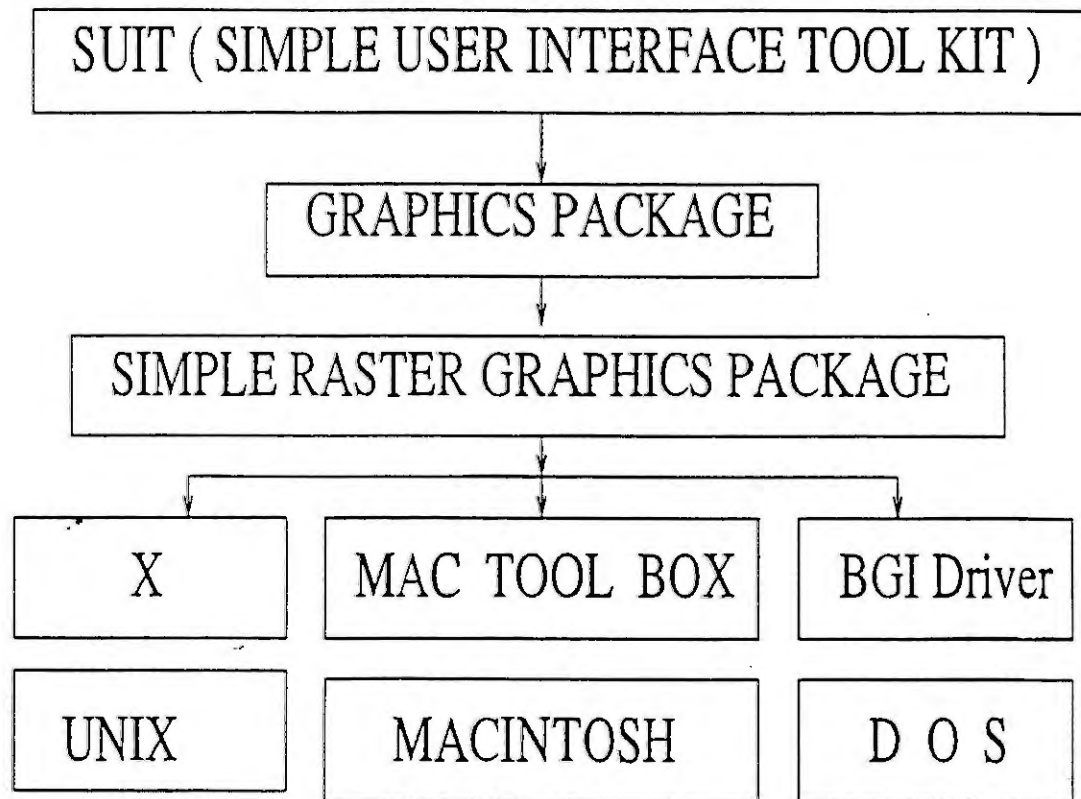


- Features*
- * An X application, called a client, may run on any m/c on N/W
 - * A server on the local m/c mediates all the i/p and display o/p
 - * A Window manager is a special kind of client and is responsible for the overall arrangement, look and user controls.

The beauty of the X system lies in its transparency on the network offering flexibilities with regard to the portability as long as the protocols are met.

SOFTWARE TOOLS AND ARCHITECTURE ... 8

SUIT A simple User Interface Toolkit Portable interface



- * *The attempt is to build on top of graphics package that supports DrawLine; DrawFilledCircle etc.*
- * *SRGP was chosen because of its availability on both DOS and Mac*
- * *GP on top of SRGP allows one to use arbitrary flt. pts in place of screen Coordinates and uses strict event model as opposed to sample / event*
- * *Easy to understand syntax : @ i (text) : italicised text;
@ b (text) for bold and @ u (text) for underlined text*

The Multi-Media Model

(The Dexter Model) ... 1

The Run time Layer

HyperText presentation

User Interaction

Presentation Specifications

Storage Layer

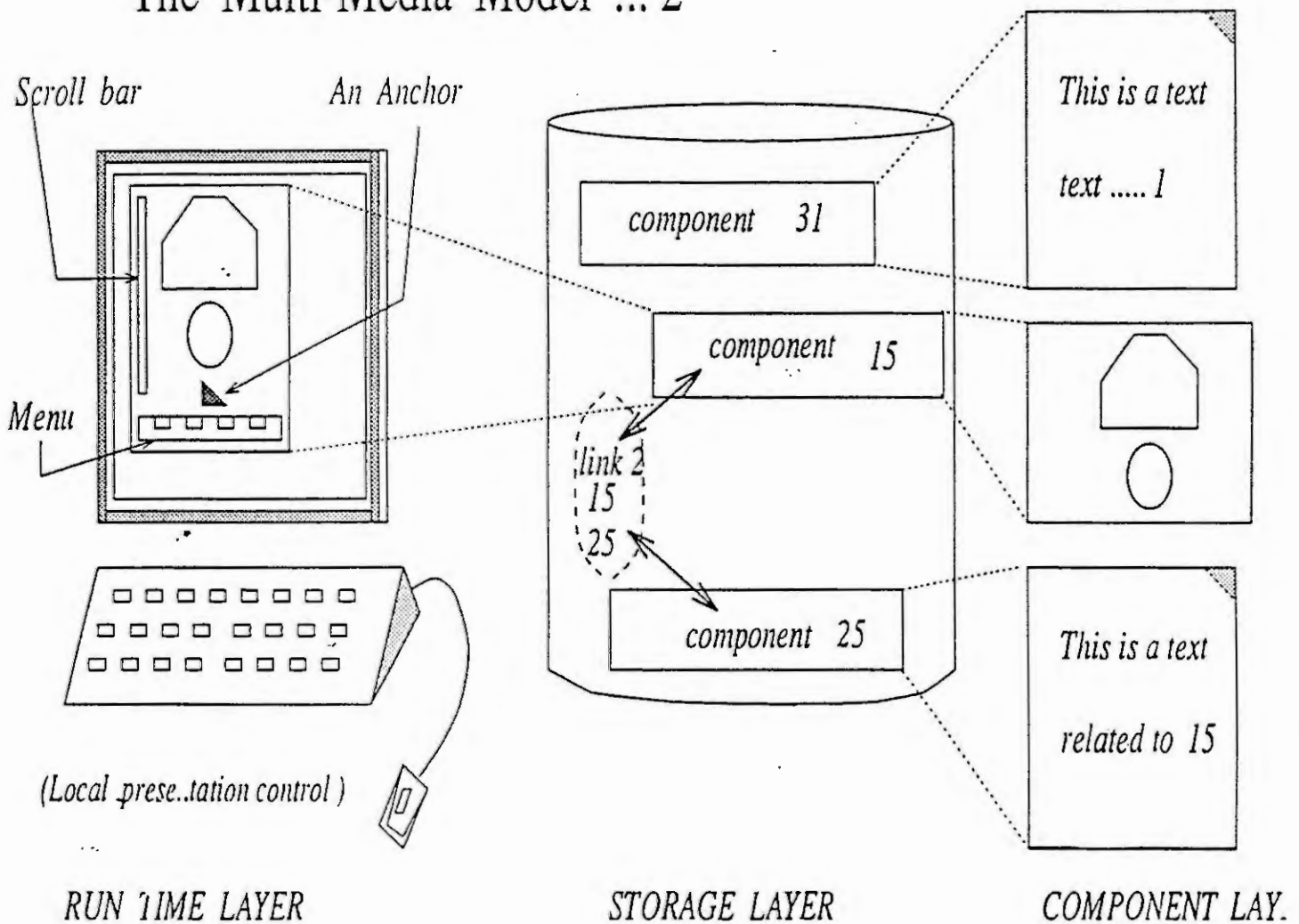
*A data-base containing
a network of nodes*

ANCHORING

Components

: The Physical structures (data files)

The Multi-Media Model ... 2



- * Every component has an unique identifier captured by UID
- * An anchor denote the correspondence that is on the hyperlink
- * Components are related only hierarchically (form a DAG)

The Multi-Media Model ... 3

Functionality of storage layer

- * Create : Atomic and structured components; links*
- * Delete : Delete and Modify components;*
- * Get : component with its attributes*
- * Set : attribute values; links to anchor (help resolver in access)*

Functionality of Run-time Layer

- * Open Session; open components*
- * Follow link offered by anchor (help in creating customised files)*
- * Present componets; unPresent (remove) components*
- * Edit instantiation; save edited instantiated component*
- * Delete components;*
- * Close session*

The Multi-Media Model ... 4

HyperText MultiMedia and HyperMedia

HyperText : Basically an interlinked component NW

"visits" bring forth the text, graphics, video
for a user determined duration by using an
anchor. i.e. click on it and navigate

Multi-media : A media combination for presentations

with audio-cassette kind of user control
for fast forward and some media control
Note that synchronisation is important

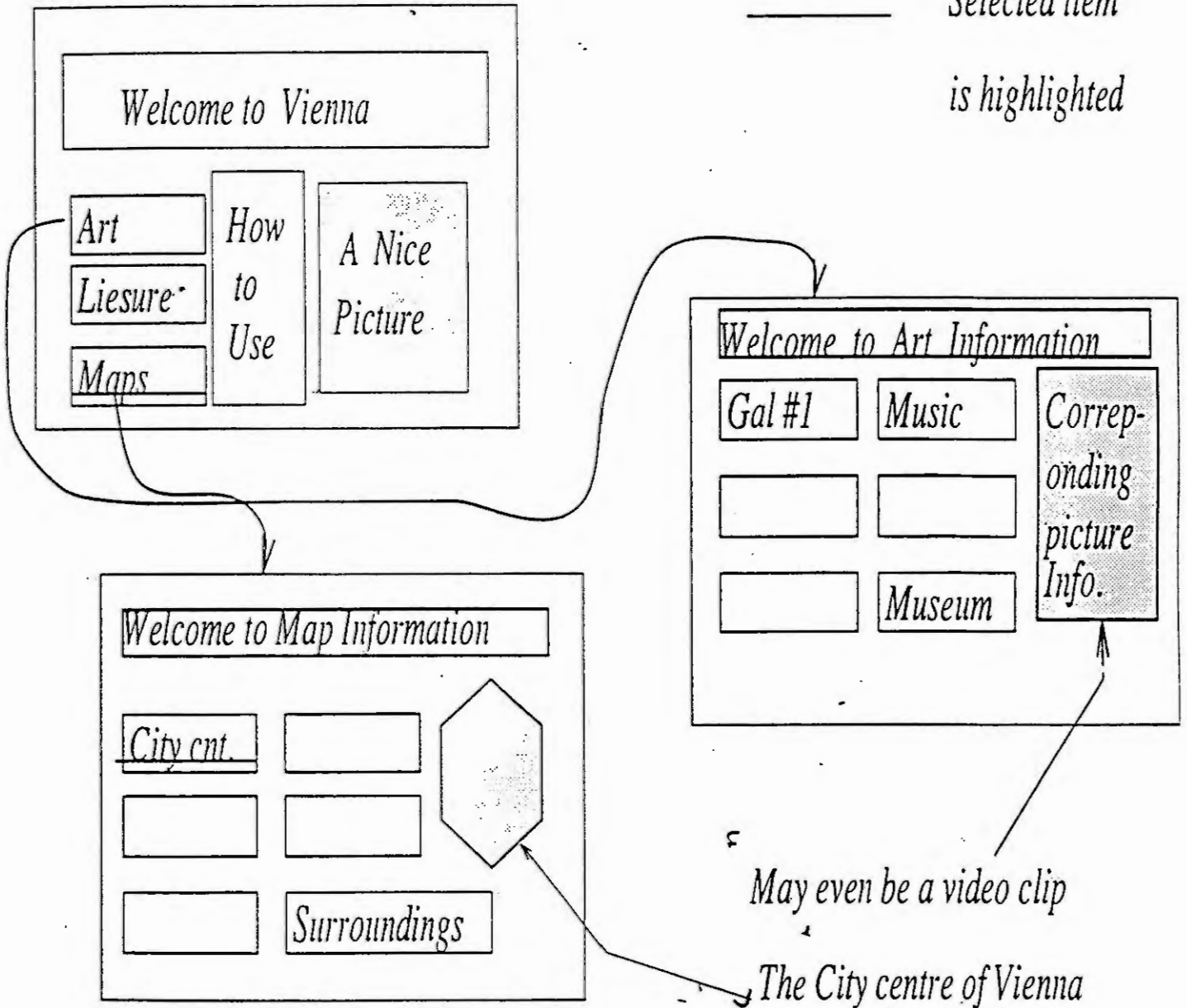
Hypermedia : Combination of the two; Note that this
is done by making each component a MM !!
and then making a NW of these components

The Multi-Media Model ... 5

HyperMedia Example

(note the navigation and synchronisation)

→ a hyperlink
from an anchor
Selected item
is highlighted



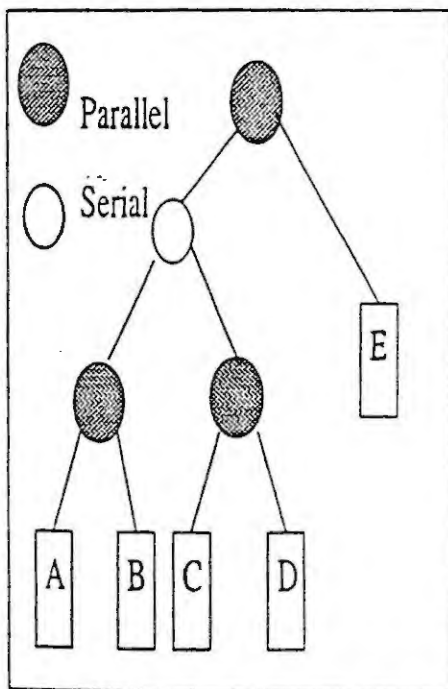
The Multi-Media Model ... 6

Authoring in MM ...

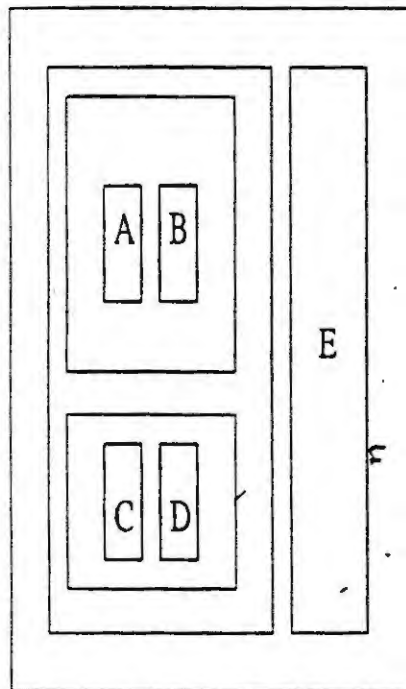
"Authoring" : to "compose"

and "evolve" a customised document.

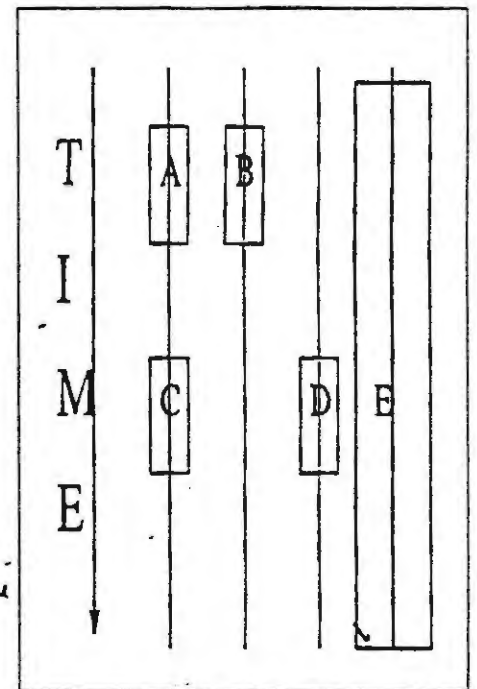
Document
Tree



Presentation
Order



The
Channel View



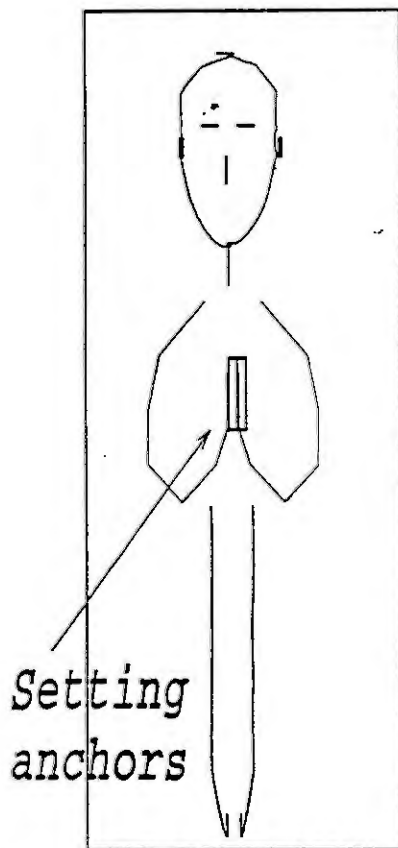
The Multi-Media Model ... 7

Authoring in MM ...

"Authoring" : to "compose"

and "evolve" a customised document.

Video display window



Gestrural Queries ?

What is the significance of
a particular "posture" or a
"movement" in a dance?

Can we "anchor" on a particular
aspect and "track" it (like
the feet of dancer)

to capture

and track hand movements

Voice overs for explanations

The Multi-Media Model ... 8

Authoring in MM ...

"Authoring" : to "compose"
and "evolve" a customised document.

Visitors to a cultural centre, museums, libraries
should be potential authors.

I.e. they prepare their customised documents
on their objects of study, put their own book marks
browse and finally generate a compact disc, Audio
or video document which they may carry with them.

They may carry the experts voice overs or their own

The Multimedia Model 9

COLLABORATIVE OPERATIONS (CSCW)

* Conferencing

- * Support for common drawing surface
- * Support for graphical annotations
- * Floor and member registration control

Typical conference tool kits are : LIZA MMCONF.

* Designers' teams

- * Support for common drawing surface
- * Support for graphical annotations
- * Multiple user interaction; shared visual space
(*May involve multiple media interaction*)

* Interprocess communication (Virtual reality)

An artificial environment is created to project what one may see, hear, feel (Flight simulators; 3 D)

THE INFORMATION KIOSK ... 1

*The IEEE Computer magazine introduced a new-column
and called it INFORMATION KIOSK*

Basic Objectives :

- * Internet services and tools*
- * Video conferencing tools and services*
- * Pricing information*
- * Standards Information*

What kind of information one is seeking

- * Company and product information*
- * Browsing system for rail, sea and air travel, places of interest*
- * College and University information, career planning services*
- * Browsing library, research material, conference announcements*

NETWORK SERVICES : GOPHER, WWW,

- * EVERY INDIVIDUAL MAY MAINTAIN A HOME PAGE !*

THE INFORMATION KIOSK ... 2

WWW : Originally promoted by CERN (Switzerland)

- * Global information service

- * A spurt of activity in providing a suitable interface to WWW

MOSAIC NETSCAPE BASAR (GMD)

WHAT DEFINES THE WWW

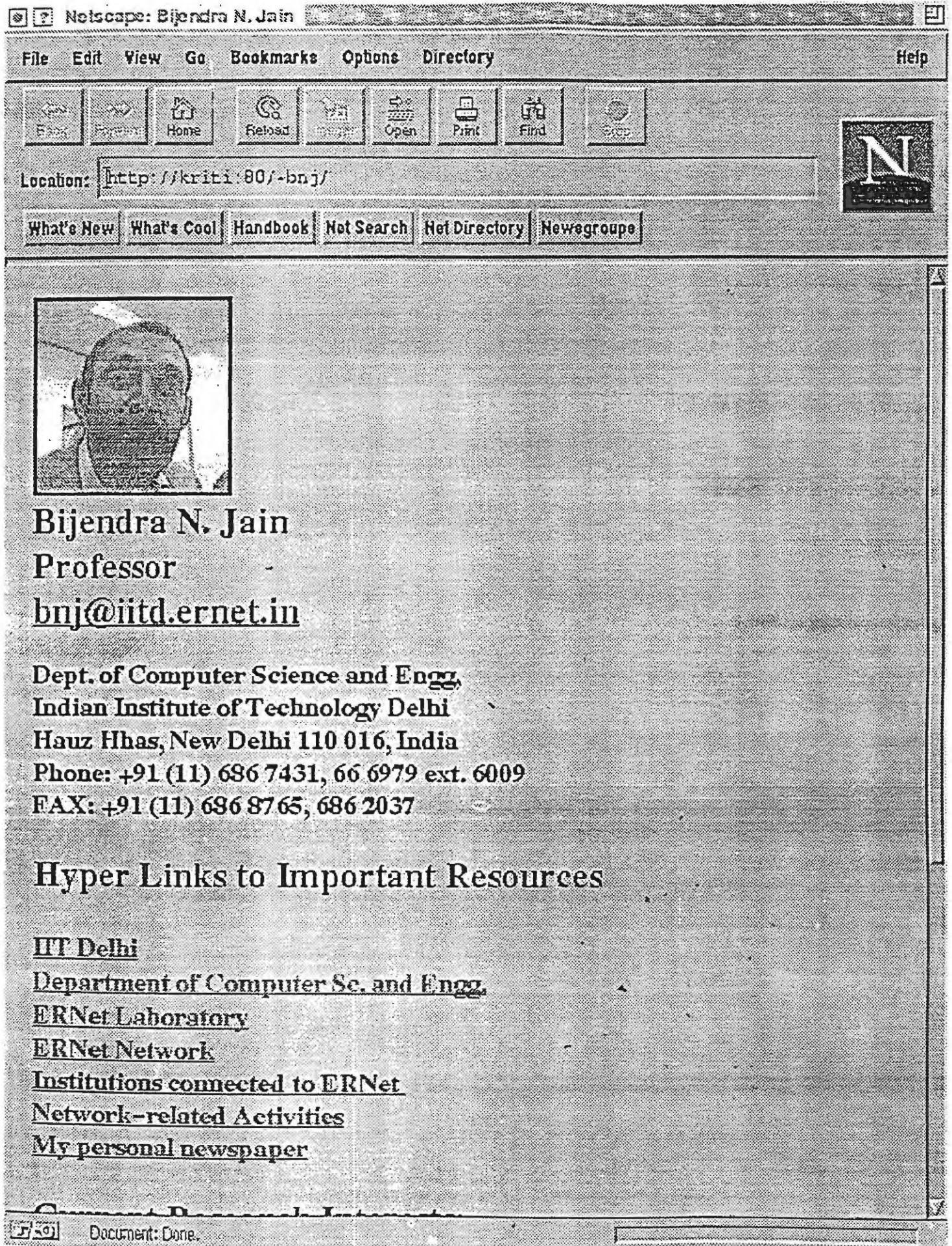
- * URI : UNIVERSAL RESOURCE IDENTIFIER (and locator URL)

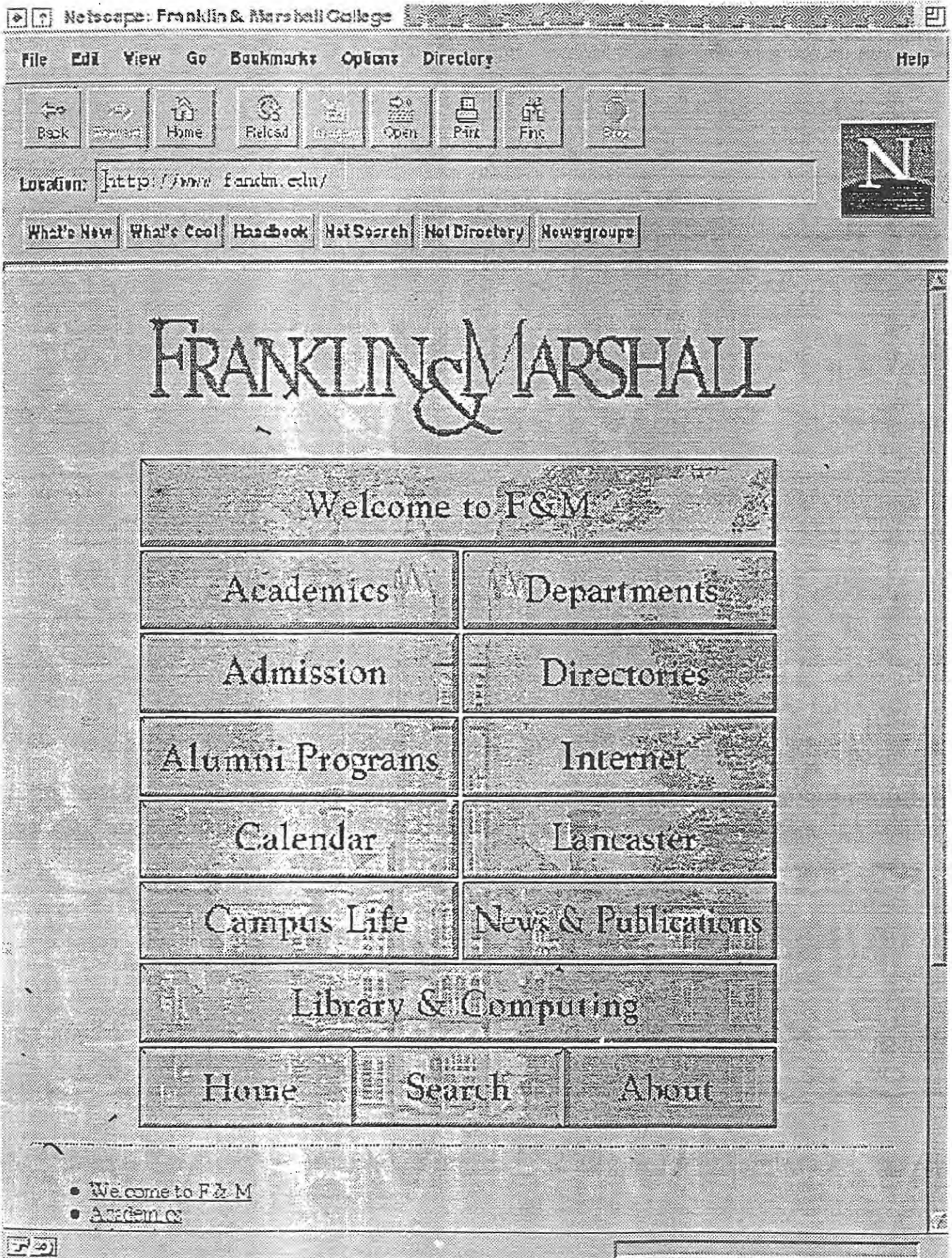
- * HTTP : A NW protocol used by native WWW server

Supports a HyperText Transfer protocol

- * HTML : A mark-up language; used to identify and transmit info.

such as text, images, menus, online help etc.





HUMAN COMPUTER INTERFACE

REFERENCES

1. Human Computer Interaction : Towards the year 2000

Ronald M Baecker; Jonathan Grudin William Buxton and Saul Greenberg

Morgan Kaufman 1995

2. IEEE Computer Magazine Multimedia issue (May 1995)

3. CACM Special issue Hypermedia (Feb. 1994)

Acknowledgements

Prof. BN Jain, Mr. Moorhty, Mr. Vadhera (IGIT A); Mr. Britto (CDOT)

DISKRETE SIMULATION -

F. Breiteneker, Technische Universität Wien

1. GRUNDLAGEN

Der Begriff "Simulation" ist an sich sehr vielschichtig, es ist daher notwendig, ihn zunächst einzugrenzen. Das Wort Simulation leitet sich aus dem lateinischen Zeitwort "simulare" ab, was in etwa "nachbilden", "nachahmen", "vortäuschen" bedeutet. Auch der VDI hat sich mit diesem Begriff befaßt und für ihn in der VDI-Richtlinie 3633 folgende "Definition" gegeben:

"SIMULATION ist die Nachbildung eines dynamischen Prozesses in einem Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind."

Sofort taucht natürlich die Frage auf, wozu man denn einen Prozeß, z.B. eine Fertigungsstraße, überhaupt simulieren soll. Erkenntnisse über einen Prozeß können ja auch am Prozeß direkt gesammelt werden.

Der Einsatz der Simulation lohnt sich, bzw. ist notwendig:

- * wenn Neuland beschritten wird (der Prozeß muß z.B. noch gar nicht existieren, der Bau eines Prototyps ist unmöglich oder zu teuer)
- * wenn die Grenzen analytischer Methoden erreicht sind (statische Betrachtungen reichen z.B. nicht mehr aus)
- * wenn das Experimentieren am realen Prozeß nicht möglich oder zu kostenintensiv ist (Stillstand einer Anlage!)
- * wenn das zeitliche Ablaufverhalten einer Anlage oder eines Organisationsablaufes untersucht werden soll
- * wenn zu komplexe Wirkungszusammenhänge die Vorstellungskraft überfordern

Allen diesen Betrachtungen gemeinsam ist der Wunsch, einen Prozeß zu verbessern, zu optimieren.

Bei den Prozessen kann man nun zweierlei Arten unterscheiden, wobei die Grenzen oft fließend sein können: bei kontinuierlichen Prozessen ändern sich die Systemgrößen stetig, während bei diskreten Prozessen diese Änderungen zu "diskreten" Zeitpunkten stattfinden. Beispiele für diskrete Prozesse sind:

- | | |
|-------------------------|---------------------------|
| * Fertigungssysteme | * Jobablauf (in Rechnern) |
| * Materialflußsysteme | * Aktenlauf (Büro) |
| * Werkstattsteuerung | * Auslastung |
| * Arbeitsstrukturierung | * Transportprozesse |

Eine Simulation kann nicht alle Probleme lösen. Generell aber können folgende Problemkreise mit Hilfe der Simulation analysiert und verbessert werden:

- * Dimensionierung der Produktmittel
- * Sichere Investitionen
- * Steuerung der Produktionsmittel
- * Intensive Schulungen
- * Effiziente Planung
- * Große Problemeinsicht

Die immer stärker in den Vordergrund rückende Alternative zu den üblichen statischen Planungen ist heutzutage die Simulation des Fertigungsprozesses auf einem Rechner, wobei gute PC's bzw. einfache Workstations durchaus ausreichen. Dazu ist zunächst der Prozeß "formal" zu beschreiben, wobei geeignete Größen zur Quantifizierung des Prozesses ausgewählt werden (Belegung von Maschinen, Inhalt einer Warteschlange (buffer), etc.)

Die wohl häufigsten Modelle in der Simulation diskreter Prozesse stochastische Modelle. Das hängt mit der Natur der Daten zusammen, denn z.B. werden Verarbeitungszeiten meistend streuen (normalverteilt, gleichverteilt, etc.), Werkstücke treffen "stochastisch" verteilt ein, die Ausschußquoten sind zufälliger Natur, , etc.

Man unterscheidet bei der Modellbeschreibung eines diskreten Prozesses prinzipiell drei Arten:

- * Ereignisorientierte Beschreibung
- * Aktivitätsorientierte Beschreibung
- * Prozeßorientierte Beschreibung

2. Modellbildung

Die Simulation eines Systems erfordert die Abbildung des Systems in einem Modell. Nachdem die wesentliche Komponente des Systems vom Analytiker ermittelt worden ist, muß er sein Wissen letztendlich in ein Computerprogramm transformieren. Allerdings ist es ein langer Weg bis zur Analyse des Systemverhaltens auf dem Bildschirm. Zunächst muß das Modell erstellt werden. Die Entwicklung eines Modells und die Implementation dieses Modells mit Hilfe einer Simulationssprache sind nicht voneinander unabhängig, da jede Simulationssprache in Bezug auf die Strukturierung eine bestimmte Sichtweise impliziert.

Komplexe Applikationen erfordern den Einsatz einer Repräsentationsform, um alle logischen und zeitabhängigen Interaktionen zwischen den Systemkomponenten in einer strukturierten Art und Weise beschreiben zu können. Man unterscheidet drei Ansätze zur Strukturierung eines Systems, die sich dann auch in der Verwendung einer Simulationssprache widerspiegeln:

- * reignisorientierter Ansatz
- * aktivitätsorientierter Ansatz
- * prozeßorientierter Ansatz

Diese Strukturierungskonzepte beeinflussen sowohl die Modellentwicklung als auch die Implementation des Modells. Wir betrachten das folgende Warteschlangensystem, bei dem ein kurzer Überblick über diese Konzepte gegeben wird. Hintereinander ankommende Kunden (CUSTOMERS) werden von einer einzigen Servicestelle (SERVICE) bedient. Kann ein Kunde nicht gleich bedient werden, so muß er in einer Warteschlange (QUEUE) warten. Die Bezeichnung Kunde/Servicestelle ist allgemein und steht stellvertretend für verschiedenartige Bedienungssysteme, bei denen man die Bezeichnungen Kunde/Servicestelle durch Patienten/Ambulatorium, Flugzeug/Landebahn, Werkstück/Bearbeitungsmaschine, Telefongespräche/Telefonzentrale oder Jobs/Rechenanlage ersetzen muß.

Die Struktur dieses einfachen Systems kann auf die folgenden Arten beschrieben werden:

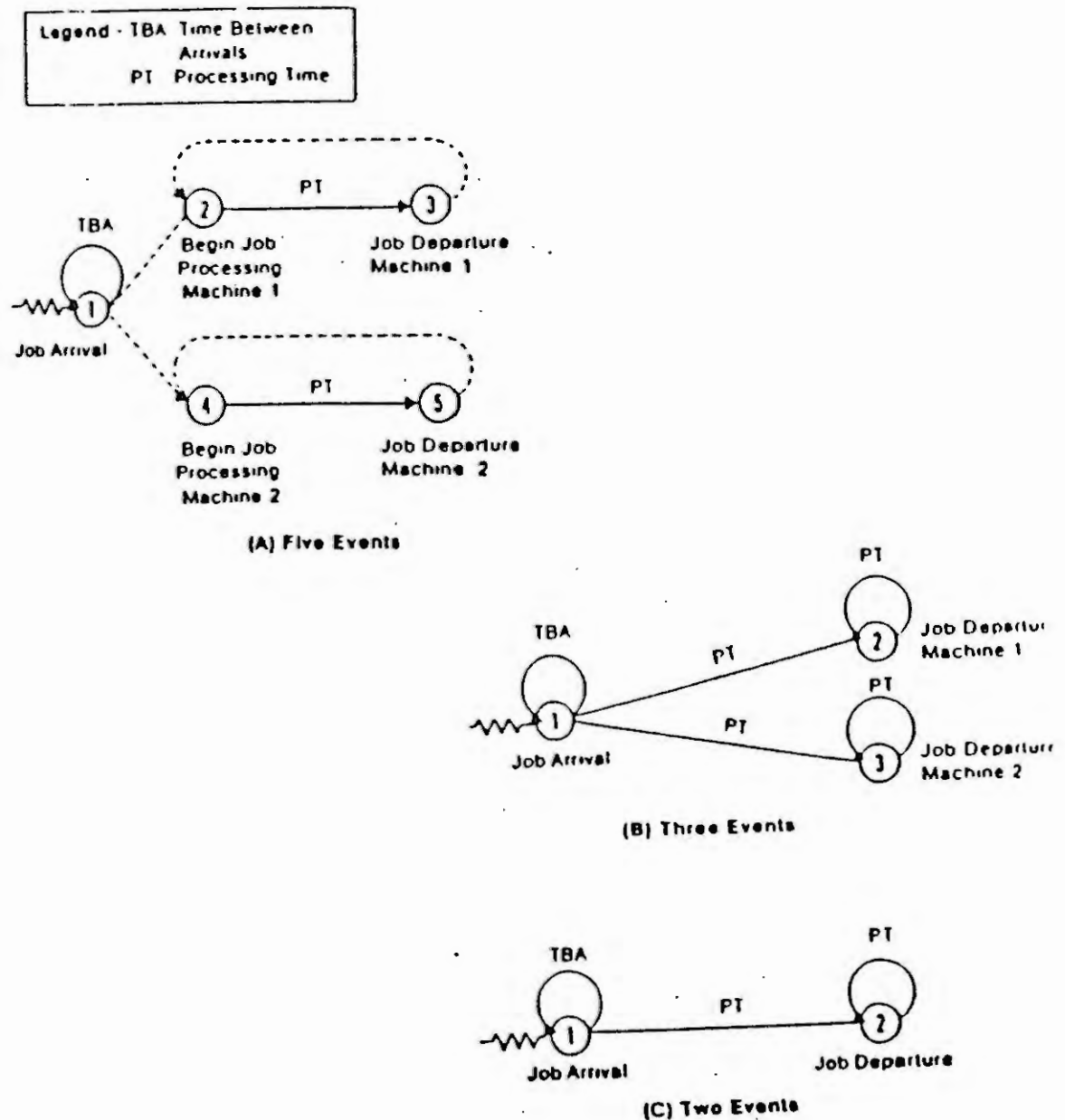
2.1 Ereignisorientierter Ansatz

In diesem Fall werden alle wesentlichen Ereignisse, die während der Untersuchung des Systems auftreten, definiert. In unserem Beispiel müssen zwei Ereignisse (ARRIVAL, DEPARTURE) modelliert werden.

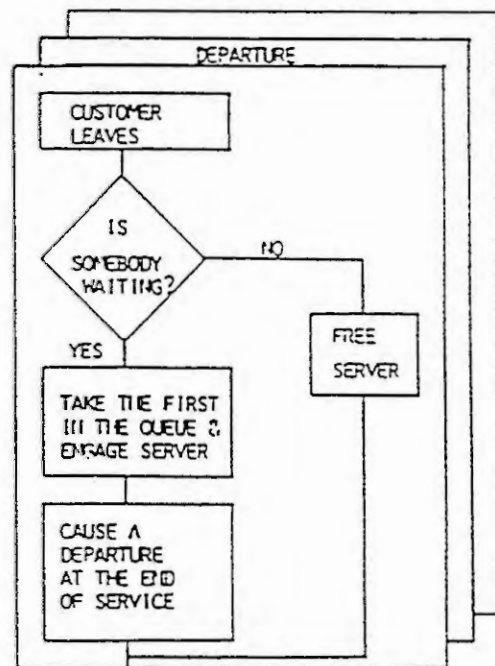
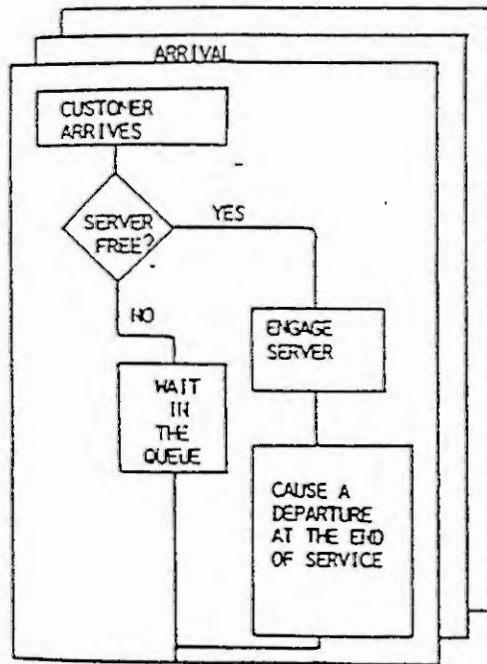
Das Ereignis "ARRIVAL" beinhaltet die Ankunft eines Kunden und seine Einordnung in eine Warteschlange, wenn die Bedienstelle nicht verfügbar ist. Das Ereignis "DEPARTURE" steht für das Ende der Bedienung und die Feststellung, ob in der Warteschlange jemand auf Bedienung wartet.

Ein Ereignis ist ein Zeitpunkt zu dem eine Änderung des Systemzustandes eintritt. Dabei hat der Analytiker zu entscheiden, ob ein Ereignis im Hinblick auf die Ziele der Simulation von Bedeutung ist. Angenommen die Abläufe in einem Spital sollen dargestellt werden. Die Ankunft eines Patienten im Spital und das Verlassen des Spitals sind Ereignisse, die abgebildet werden müssen, während etwa eine Änderung des Gesundheitszustandes für das Simulationsziel relevant ist, um als Ereignis Berücksichtigung zu finden. Die Entscheidung, ob ein Ereignis wichtig ist oder nicht, ist eine Frage des Zweckes der Studie.

Während Ereignisse in der Realität keine zeitkonsumierende Zustandsänderungen darstellt, entspricht jedem Ereignis im digitalen Computer der Aufruf einer Prozedur, welche natürlich Zeit konsumiert



EREIGNISORIENTIERTER ANSATZ



2.2. Aktivitätsorientierter Ansatz

Man kann auch alle Aktivitäten im System beschreiben, in dem man erklärt, unter welchen Umständen sie eintreten.

In unserem Beispiel tritt die Aktivität "ARRIVING" nur dann auf, wenn der nächste Kunde ankommen soll. Die Aktivität "SERVICE" kann nur dann durchgeführt werden, wenn mindestens ein Kunde darauf wartet und wenn die Bedienstelle frei ist. Die Aktivität "LEAVING" tritt auf, wenn das Ende einer Bedienung fällig ist. Man beachtet den Unterschied zu ereignisorientierten Systemen. Während ein Ereignis einen Zeitpunkt darstellt, wird eine Aktivität durch ein Zeitintervall beschrieben. Ereignisse haben die Dauer 0, während Aktivitäten eine positive Dauer aufweisen.

Eine Möglichkeit, gleichzeitige Prozesse in einem Simulationsmodell entsprechend dem aktivitätsorientierten Ansatz zu beschreiben, sind Simulationsgraphen. Das Konzept von Simulationsgraphen ist eine Erweiterung zu den Petrinetzen, die ursprünglich für die Modellierung des Informationsflusses eines Systems entwickelt worden sind.

Petrinetze sind ein effizientes Werkzeug für die Modellierung von Parallelitäten bei diskreten Systemen.

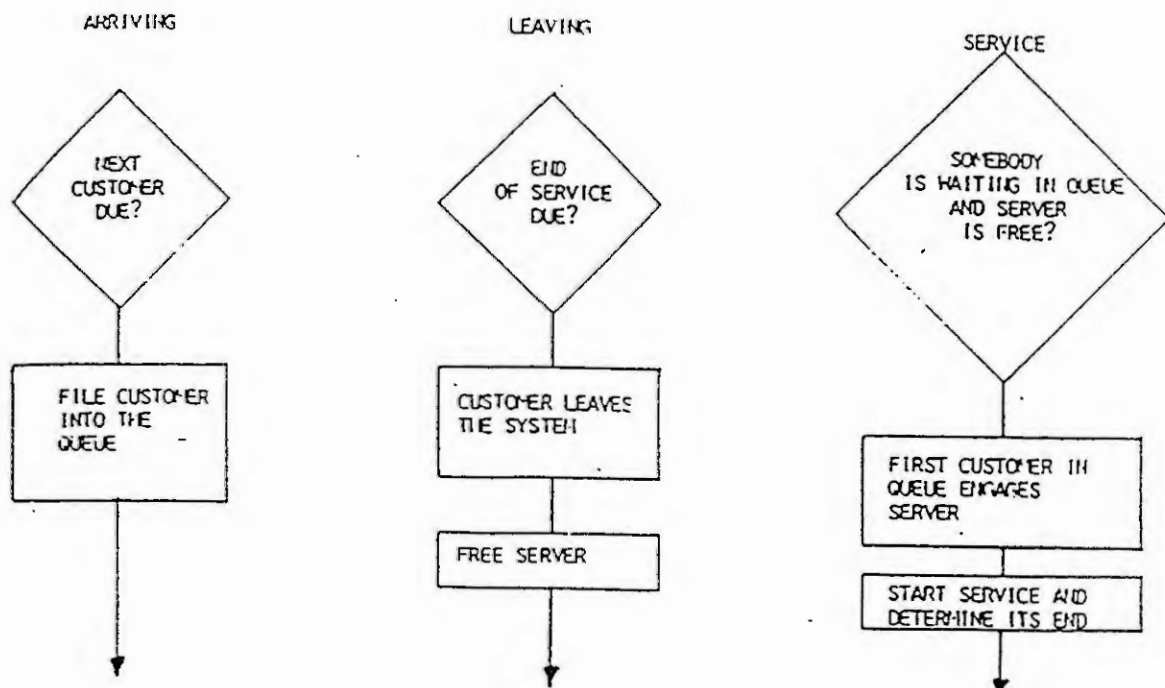
Petrinetze bestehen in erster Linie aus zwei Typen von Elementen:

Plätze ("places") und Übergänge ("transitions").

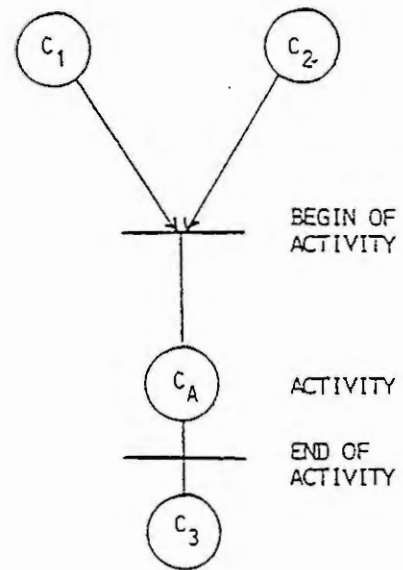
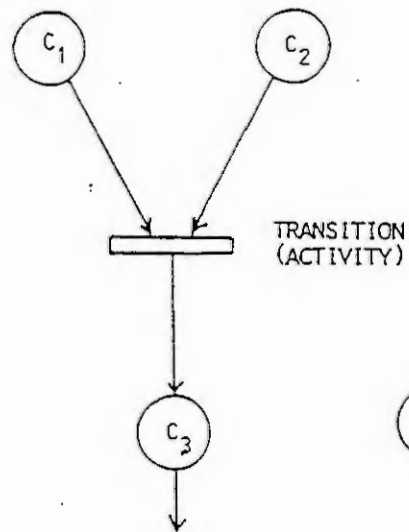
Die Plätze repräsentieren die Bedingung (im Graphen durch Kreise repräsentiert) unter denen Transitionen, das sind Aktivitäten (im Graphen durch Striche repräsentiert) oder Ereignisse (im Graphen durch Rechtecke repräsentiert) auftreten können

Die Menge der Plätze repräsentiert den Zustand des Systems. Simulationsgraphen sind markierte Petrinetze, in denen schwarze Punkte (sogenannte Tokens) die Dynamik des Modells wiedergeben, indem sie durch den Graphen fließen. Eine Bedingung ist erfüllt, wenn ein Token den entsprechenden Platz markiert.

Eine Transition findet nur dann statt, wenn alle Inputbedingungen erfüllt sind. In diesem Fall wird von jedem Inputplatz ein Token entfernt und in jedem Outputplatz der Transition wird ein Token plaziert. Diesen Vorgang nennt man "Feuern" der Transition



AKTIVITÄTSORIENTIERTE DIAGRAMME



FB - SIM

DISCRETE SIMULATION

2.3 Prozeßorientierter Ansatz

Für jeden Bausteintyp (Objekte wie z.B. Kunden) kann ein individueller Pfad durch das System beschrieben werden.

Das heißt, der Kunde erreicht die Bedienstelle, um dort bedient zu werden. Er wartet in der Warteschlange, um bedient zu werden. Nach der Bedienung verläßt er das System. Die Bedienstelle (ein passives Objekt) arbeitet immer dann, wenn ein Kunde auf das Service wartet.

Eine umfassendere Behandlung dieser Ansätze wird im nächsten Kapitel gegeben, aber schon dieses einfache Warteschlangensystem zeigt, auf welche Arten die Struktur und der dynamische Aspekt beschrieben werden können. Neben der Beantwortung der Frage, welcher Strukturierungsansatz für die Modellierung des Systems am besten geeignet ist, muß entschieden werden, in welcher Form die Struktur festgelegt wird. Drei Modellierungsmethoden scheinen praktikabel zu sein.

- . graphische Modellpräsentation
- . Modellspezifikations- und Dokumentationswerkzeuge
- . Computermodeillrepräsentation

Hier wird das Hauptaugenmerk auf die graphische Repräsentationsform gelegt, da diese den Anforderungen für die Entwicklung von klaren und verständlichen Modellen am besten gerecht werden.

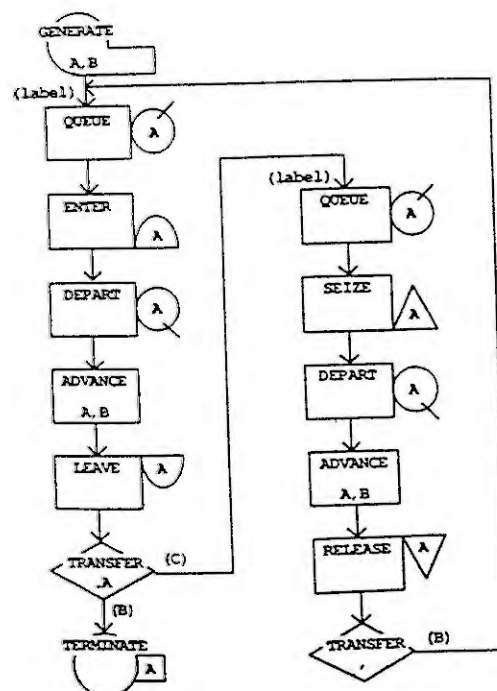
Bei der prozeßorientierten Modellweise ist der individuelle Fluß eines Entities durch das System zu beschreiben. Alle Änderungen des Systemzustandes, die mit einem Entity im Zusammenhang stehen, müssen berücksichtigt werden.

Für jeden Entitytyp (Kunde, Patient, Werkstücke, Autos,...) existiert eine Reihe von Ausprägungen, die generiert oder sortiert werden können. Z.B. Ankunft eines Patienten im System, Verlassen des Systems).

Jede Entityausprägung hat ihren eigenen Prozeß, eine Reihe von aufeinanderfolgenden Systemzustandsänderungen

Das gesamte System wird durch alle Entities, die sich gleichzeitig darin aufhalten, gemeinsam beschrieben

Beide Objektgruppen (Kunden und Bedienstelle) werden als Entities betrachtet und jedes von ihnen hat seinen eigenen Aktivitätszyklus. In manchen Fällen ist es allerdings effizienter, einen Objekttyp wie die Bedienstelle als Ressource zu modellieren. In diesem Fall existiert kein autonomer Teil für dieses Objekt und es wird durch das in Abb. 30 verwendete Symbol dargestellt.



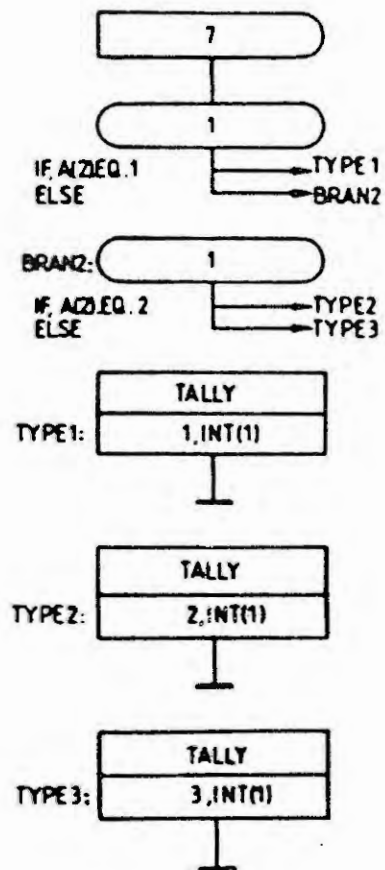
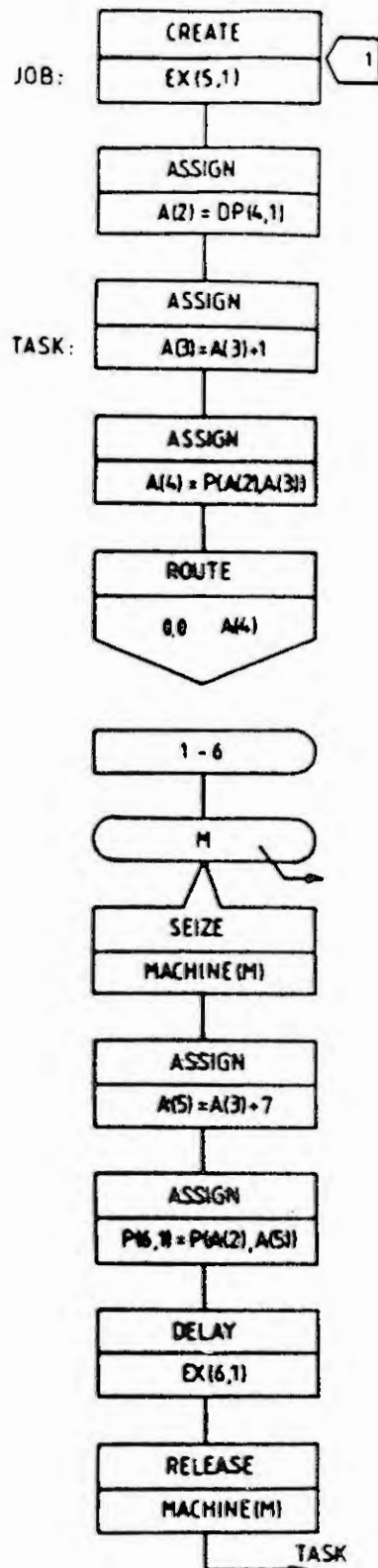


Abbildung 7: SIMAN-Blockdiagramm: *Job-Shop Scheduling*

2. Modellspezifikations- und -Dokumentationswerkzeuge

Graphische Repräsentationsmethoden sind sehr übersichtlich, haben aber den Nachteil, die verschiedenen Abstraktionsebenen nicht zu unterstützen. Eine verbale Modellformulierung erlaubt eine schrittweise Verfeinerung von Modellen erscheint aber für komplexe Anwendungen nicht so effizient.

Ein Modellspezifikations- und -dokumentationstool stellt einen Rahmen für die Modelldarstellung zur Verfügung, in den der Analytiker die wichtige Komponente des Systems eintragen kann. Auf diese Weise wird eine systematische und organisierte Dokumentation der signifikanten Aspekte des Modells möglich.

4. Daten

Bei Fertigungsprozessen ändern sich die Systemgrößen nur zu "Entscheidungszeitpunkten" (diskrete Zeitpunkte), die Modelle werden ebenso aufgebaut. Abhängig von der Festlegung dieser Zeitpunkte können nun die Modelle und Prozesse katalogisiert werden:

Zeitdauer zwischen Prozeß- bzw. Modellart	Entscheidungszeitpunkten
vorgegeben	deterministisch
berechnet	algorithmisch
zufällig	stochastisch

Die wohl häufigsten Modelle in der Simulation von Fertigungsabläufen, etc. sind stochastische Modelle. Das hängt mit der Natur der Daten zusammen, denn z.B. werden Verarbeitungszeiten meistend streuen (normalverteilt, gleichverteilt, etc.), Werkstücke treffen "stochastisch" verteilt ein, die Ausschußquoten sind Stochastisch, etc.

Die Datenbeschaffung für eine Simulation wird oft unterschätzt, dabei können die Ergebnisse der Simulation nur so gut sein wie die Eingangsdaten. Die Daten lassen sich in in zeit- und mengenorientierte Daten einteilen:

zeitorientierte Daten	mengenorientierte Daten
Taktzeit	Stückzahlen
Bearbeitungszeit	Pufferkapazität
Transportdauer	Transportmittellanzahl
Stördauer	Störungshäufigkeit

Eine schwierige Entscheidung ist, ob für diese Daten Mittelwerte ausreichen, oder ob Verteilungen angesetzt werden müssen. Im ersten Fall reichen dann deterministische bzw. algorithmische Modelle, der zweite Fall führt auf die stochastischen Modelle, die eher als realistisch anzusehen sind.

5. Experimente

Ziel des Experimentierens mit einem Modell ist es, günstige Werte für freie Systemparameter zu finden, also z.B. Ressourcenanzahl, etc. Experimentieren ist mehr als probieren, es bedeutet, sinnvolle Versuchsreihen mit der Variation einzelner Parameter aufzustellen, durchzuführen und Ergebnisse zu erzeugen. Bei der Planung von Experimenten sind folgende Punkte unbedingt zu beachten:

- * Abstimmung der Experimente mit dem Anwender
- * Nicht zu viele Parameter gleichzeitig ändern
- * Besser zu viel, als zu wenig Dokumentation
- * Änderungen des Modells erfordern Validierung

6. Ergebnisdarstellung

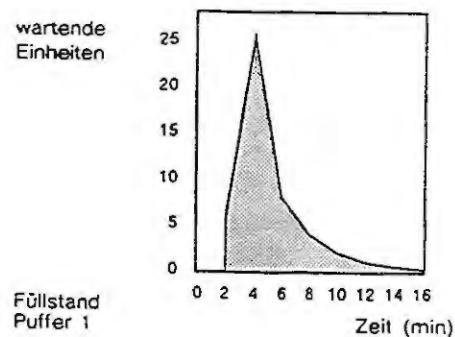
Die Ergebnisdarstellung und -Dokumentation ist wesentlich für die Interpretation der Simulation. Neben Liniendiagrammen spielen Balken- und Tortendiagrammen aufgrund der oft stochastischen Natur der Simulation eine wesentliche Rolle.

Diese Diagramme haben unterschiedlichen Anwendungsbereich:

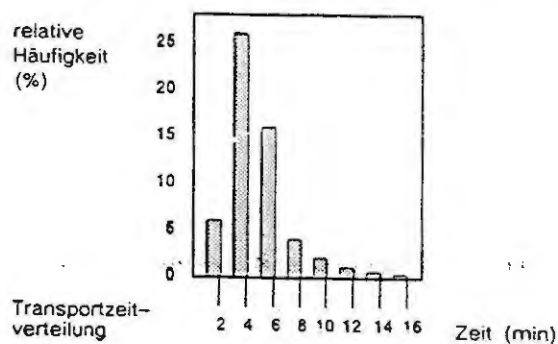
Liniendiagramme für zeitlich variierende Größen (z.B. Maschinenbelegung)

Balkendiagramme zur Darstellung von absoluten und relativen Häufigkeiten (z.B. Störungen, Wartezeiten)

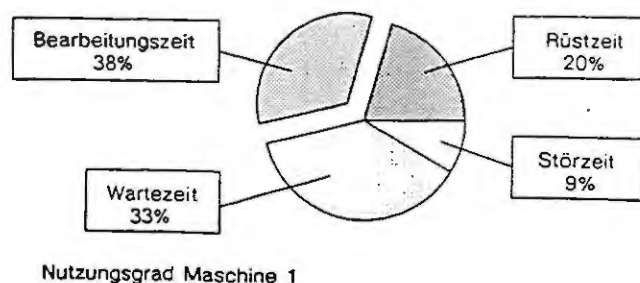
Kreisdiagramme zur Darstellung relativer Gewichtsanteile (z.B. Auslastung)



- Liniendiagramme für die Veranschaulichung von sich zeitlich ändernden Größen, wie z.B. des Füllgrades von Puffern



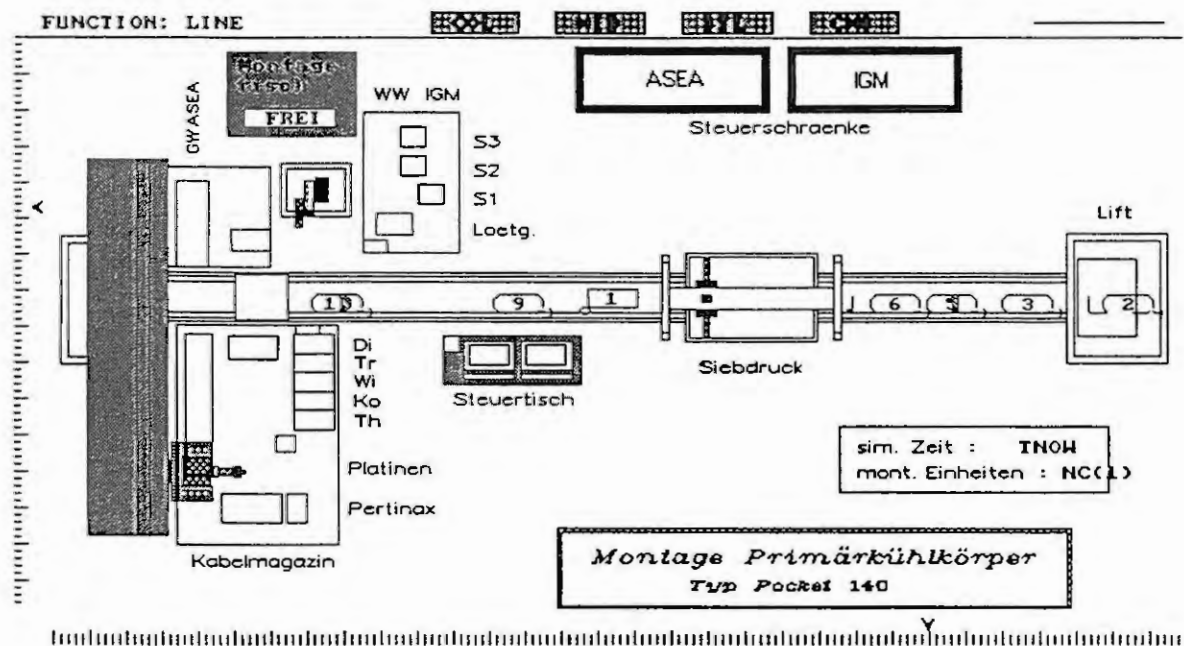
- Balkendiagramme zur Darstellung von absoluten und relativen Häufigkeiten, z.B. von Störungen, Blockierungen usw.

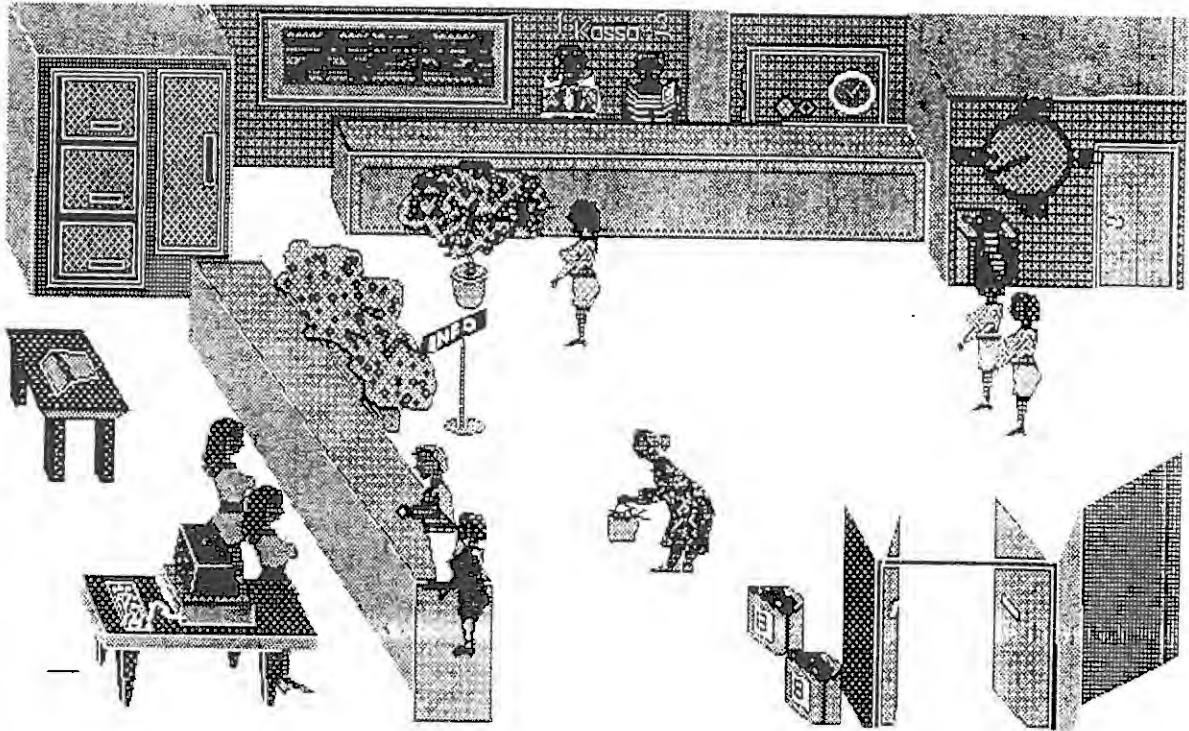


- Kreisdiagramme zur Visualisierung relativer Gewichtsanteile, z.B. für Nutzungsgrade, Auslastungen usw.

7. Animation

Mit der graphischen Leistungsfähigkeit der Rechner steigt die Beliebtheit einer weiteren Ergebnisrepräsentation, der Animation. Dabei wird der Prozeß graphisch am Bildschirm dargestellt, man kann seine zeitliche Veränderung am Schirm mitverfolgen (das Transportieren der Werkstücke, das Bearbeiten, etc. Allerdings wird die Animation oft überschätzt, denn die Qualität einer Animation sagt nichts über die Qualität des Modelles und der Experimente aus (Animation darf kein Videogame sein).







ARGESIM Report no. 5

Fuzzy Systems and Control

**COMETT - Course
Seminar Modellbildung und Simulation
EUROSIM'95 Seminar**

D. Murray-Smith, D.P.F. Möller, F. Breiteneker

**in ISBN ebook 978-3-901608-04-9 (3-901608-04-4)
DOI 10.11128/arep.4-5-6.ar5**

© 1995 ARGESIM

in ISBN ebook 978-3-901608-04-9 (3-901608-04-4)
DOI 10.11128/arep.4-5-6.ar5

ARGESIM Report No. 5

ARGE Simulation News (ARGESIM)
c/o Technical University of Vienna
Wiedner Hauptstr. 8-10
A-1040 Vienna, Austria
Tel: +43-1-58801 5386, 5374, 5484
Fax: +43-1-5874211
Email: argesim@simserv.tuwien.ac.at
WWW: <URL:<http://eurosim.tuwien.ac.at/>>

FOREWORD

Simulation may be defined as the process of developing a computer-based model of a system and of experimenting with this computer model. The purpose of simulation may be to gain a better understanding of the behavior of the system, or to locate specific problems, or to evaluate different strategies, or as an aid in designing a new process or system.. Automatic control systems provide one important area of application for simulation techniques.

This course is in two parts. The first deals with methods of process simulation for control systems applications, including the internal verification and external validation of plant models. The second part provides an introduction to the principles of fuzzy control and to fuzzy logic in system automation.

The introductory part of the course is general and addresses issues of plant modelling which are important in many different fields of application. Examples are presented, which illustrate the importance of accurate modelling and simulation in control system investigations for both conventional and fuzzy controller configurations. Two case studies are included, which provide further illustrations of the role of simulation techniques in control system applications, including issues of internal verification and external validation.

In the second half of the course the emphasis is on the application of fuzzy logic principles to automatic control problems. The first session provides an introduction to fuzzy logic concepts, while in the second the emphasis is on the application of fuzzy principles to control system design.

It is hoped that after the course a participant should be able to make decisions about the use of modelling and simulation methods for automatic control system applications, and to have an appreciation of the potential of fuzzy logic methods in control engineering and system automation. In particular, the course should serve to emphasize the importance of simulation techniques in the investigation of real control systems involving nonlinear plant characteristics or, as in the case of fuzzy systems, nonlinearities within the controller also. Analytical methods provide little real insight in these situations and simulation techniques are therefore of special importance in dealing with the complexities of practical control systems, whatever the specific area of application.

About ARGESIM

ARGE Simulation News (ARGESIM) is a non-profit working group providing the infra structure for the administration of **EUROSIM** activities and other activities in the area of modelling and simulation.

ARGESIM organizes and provides the infra structure for

- the production of the journal **EUROSIM Simulation News Europe**
- the comparison of simulation software (**EUROSIM Comparisons**)
- the organisation of seminars and courses on modelling and simulation
- **COMETT Courses on Simulation**
- "Seminare über Modellbildung und Simulation"
- development of simulation software, for instance: **mosis** - continuous parallel simulation, **D_SIM** - discrete simulation with Petri Nets, **GOMA** - optimization in ACSL
- running a WWW - server on **EUROSIM** activities and on activities of member societies of **EUROSIM**
- running a FTP-Server with software demos, for instance
 - * demos of continuous simulation software
 - * demos of discrete simulation software
 - * demos of engineering software tools
 - * full versions of tools developed within ARGESIM

At present ARGESIM consists mainly of staff members of the Dept. Simulation Technique and of the Computing Services of the Technical University Vienna.

In 1995 ARGESIM became also a publisher and started the series **ARGESIM Reports**. These reports will publish short monographs on new developments in modelling and simulation, course material for COMETT courses and other simulation courses, Proceedings for simulation conferences, summaries of the EUROSIM comparisons, etc.

Up to now the following reports have been published:

No.	Title	Authors / Editors	ISBN
# 1	Congress EUROSIM'95 - Late Paper Volume	F. Breitenecker, I. Husinsky	3-901608-01-X
# 2	Congress EUROSIM'95 - Session Software Products and Tools	F. Breitenecker, I. Husinsky	3-901608-01-X
# 3	EUROSIM'95 - Poster Book	F. Breitenecker, I. Husinsky	3-901608-01-X
# 4	Seminar Modellbildung und Simulation - Simulation in der Didaktik	F. Breitenecker, I. Husinsky, M. Salzmann	3-901608-04-4
# 5	Seminar Modellbildung und Simulation - COMETT - Course "Fuzzy Systems and Control"	D. Murray-Smith, D.P.F. Möller, F. Breitenecker	3-901608-04-4
# 6	Seminar Modellbildung und Simulation - COMETT - Course "Object-Oriented Discrete Simulation"	N. Kraus, F. Breitenecker	3-901608-04-4
# 7	EUROSIM Comparison 1 - Solutions and Results	F. Breitenecker, I. Husinsky	3-901608-07-9
# 8	EUROSIM Comparison 2 - Solutions and Results	F. Breitenecker, I. Husinsky	3-901608-07-9

For information contact:

ARGESIM, c/o Dept. Simulation Techniques,
attn. F. Breitenecker, Technical University Vienna
Wiedner Hauptstraße 8-10, A - 1040 Vienna
Tel. +43-1-58801-5374, -5386, -5484, Fax: +43-1-5874211
Email: argesim@simserv.tuwien.ac.at

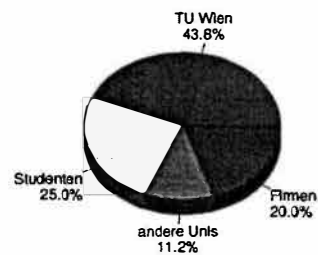
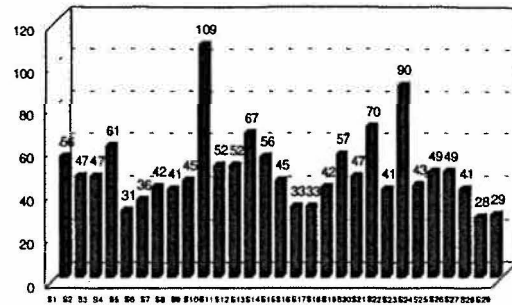
Seminare über Modellbildung und Simulation

Seit dem Frühjahr 1991 veranstaltet das EDV-Zentrum gemeinsam mit der Abteilung Regelungsmathematik und Simulationstechnik des Instituts für Technische Mathematik und der ARGE Simulation News (ARGESIM) Vortragsveranstaltungen zum Thema Modellbildung und Simulation (Simulationsseminare). Organisatoren sind I. Husinsky und F. Breitenacker. Das Ziel ist, verschiedene Simulationswerkzeuge vorzustellen, über ihre Einsatzmöglichkeiten zu informieren und Erfahrungen auszutauschen. Ferner werden bekannte Simulationsfachleute eingeladen, Grundsatzvorträge zum Thema Simulation zu halten. Im allgemeinen werden die Seminare teilweise von Firmen gesponsert oder über Simulationsprojekte mitfinanziert. Sie dauern einen halben oder einen Tag, es gibt schriftliche Unterlagen zu den Vorträgen und Softwareprodukten. Ein Buffet fördert die Kommunikation zwischen den Seminarteilnehmern in den Pausen.

Bis jetzt haben folgende Seminare stattgefunden:

S1	23. 4. 1991	ACSL
S2	4. 6. 1991	CTRL_C, XANALOG
S3	22. 10. 1991	SIMUL_R
S4	5. 5. 1992	ACSL
S5	6. 5. 1992	MicroSaint
S6	17. 6. 1992	Objektorientierte Modellbeschreibung und qualitative Simulation (F. Cellier, University of Arizona)
S7	1. 7. 1992	Diskrete Simulation und Analyse (D. Kelton, University of Minnesota)
S8	23. 10. 1992	GPSS/H (T. Schriber, University of Michigan)
S9	10. 12. 1992	SIMPLE
S10	2. 2. 1993	MATLAB und SIMULINK
S11	25. 3. 1993	Modellbildung mit Bondgraphen (D. Karnopp, University of California)
S12	24. 5. 1993	MicroSaint
S13	22. 6. 1993	ACSL
S14	21.10.1993	XANALOG, SIMNON
S15	22.10.1993	GPSS/H (T. Schriber, University of Michigan)
S16	11.11.1993	IDAS
S17	7.12.1993	SIMPLE++
S18	14.12.1993	Petrinetze, D_SIM (R. Hohmann, Magdeburg)
S19	4.2.1994	Modellbildung und Simulation in der Lehre
S20	14.3.1994	GPSS/H und Proof (T. Schriber, University of Michigan)
S21	13.4.1994	ACSL
S22	10.5.1994	SIMUL_R, Partielle Differentialgleichungen
S23	22. 11. 1994	MATLAB/SIMULINK
S24	14.12.1994	SIMPLE++
S25	31.1.1995	Parallele Simulation, mosis
S26	28.3.1995	ACSL
S27	29.3.1995	MicroSaint
S28	13.6.1995	COMETT II, Part one, Discrete Simulation
S29	28.6.1995	COMETT II, Part two, Simulation and Automatisierung

Teilnehmer(angemeldet)



Die Teilnehmer, etwa 30 bis 110 je Seminar, kommen zum Großteil von der TU, aber auch von anderen Universitäten und aus der Industrie. Bei den bisherigen Seminaren waren etwa 20% der Teilnehmer aus der Industrie.

Das Programm eines Seminars setzt sich im allgemeinen aus einem oder zwei Grundlagenvorträgen, mehreren Anwendervorträgen, Produktpräsentationen, Vorführungen am Rechner und Diskussionen zusammen.

Die Teilnehmer werden um eine Anmeldung gebeten, daher können die Unterlagen (Seminarberichte), die zu Beginn des Seminars verteilt werden, schon eine Teilnehmerliste enthalten. Ab Herbst 1995 erscheinen die Unterlagen als ARGESIM Report. Alle, die bereits an einem Seminar teilgenommen haben, werden automatisch zu den weiteren Seminaren eingeladen.

Information:

I. Husinsky, EDV-Zentrum, Technische Universität Wien, Wiedner Hauptstr. 8-10, A-1040 Wien, Tel: (0222) 58801 5484, Fax: (0222) 587 42 11, E-Mail: husinsky@edvz.tuwien.ac.at

Prof.Dr. F. Breitenacker, Abt. Regelungsmathematik u. Simulationstechnik, Inst. 114, Technische Universität Wien, Wiedner Hauptstr. 8-10, A-1040 Wien, Tel: (0222) 58801 5374, Fax: (0222) 587 42 11, E-Mail: fbreiten@email.tuwien.ac.at

TABLE OF CONTENTS

Foreword	iii
About ARGESIM	iv
Seminare "Modellbildung und Simulation"	v
Problem Organisation for Control System Simulation	1
Simple Examples of Control System Simulation Using Common Simulation Tools	21
Internal Verification and External Validation of Simulation Models for Control Systems Analysis and Design	49
SLIM - A Simple Continuous System Simulation Language	83
Was ist Fuzzy Logic	111
Fuzzy Logic in USA, Japan, Deutschland	119
Strukturelle Gliederung der Fuzzy Applikationen	133
Fuzzy Control	155

Fuzzy Control for Automatisations

Freitag, 8. September 1995,

Technische Universität Wien
Seminarraum des EDV-Zentrums

Programm

- 9⁰⁰ Begrüßung, Vorstellung des COMETT-Kurses**
F. Breiteneker
- 9¹⁵ Problem Organisation For Control System Simulation**
Simple Examples of Control System Simulation Using Common Simulation Tools
Internal Verification and External Validation of Simulation Models for Control
Systems Analysis and Design
Prof. Dr. D. Murray-Smith, University of Glasgow
- 10⁴⁵ Kaffeepause**
- 11⁰⁰ Case Study I - Plant Modelling for a Two-Tank Liquid Level Control System**
Case Study II - An Aircraft Automatic Landing System
Prof. Dr. D. Murray-Smith, University of Glasgow
- 12³⁰ Mittagspause**
- 13³⁰ Fuzzy Systems**
Prof. Dr. D.P.F. Möller, TU Clausthal-Zellerfeld
- 15⁰⁰ Kaffeepause**
- 15³⁰ Fuzzy Control**
Prof. Dr. D.P.F. Möller, TU Clausthal-Zellerfeld
- 17⁰⁰ Abschließende Diskussion**

Seminar über Modellbildung und Simulation, TU Wien, 8. 9. 1995, Teilnehmerliste

Dipl.Ing.	Astner	Klaus			Jochen-Rindt-Str. 22/4/2	A	- 1230 Wien astner@iiasa.ac.at
Prof. Dr.	Breitenecker	Felix	Technische Universität Wien	Abt. Simulationstechnik	Wiedner Hauptstr. 8-10	A	- 1040 Wien fbreiten@email.tuwien.ac.at
Dipl.Ing.	Cizl	Claudia	(TU Wien, Abt. Simulationstechnik)		Sibeliusstr. 9/3/4	A	- 1100 Wien
	Forsthuber	Edwin	(TU Wien, Abt. Simulationstechnik)		Ortsstr. 9	A	- 2362 Biedmannsdorf e9026231@fbma.tuwien.ac.at
Dipl.-Ing.	Gannadi-Khosh	Habib	TU Wien	Inst. f. Tragwerkslehre	Karlsplatz 13/254	A	- 1040 Wien
Dipl.Ing.	Geißler	Robert	TU Wien	Inst.f. Computertechnik	Gußhausstr. 25-29	A	- 1040 Wien robert@ict.tuwien.ac.at
Dr.	Hick	.	Österr. Forschungszentrum Seibersd			A	- 2444 Seibersdorf
	Klug	Markus	(TU Wien, Abt. Simulationstechnik)		Frauengasse 19/3	A	- 1170 Wien mklug@ws1.atv.tuwien.ac.at
Dipl.Ing.	Koroknai	Stefan	(TU Wien, Abt. Simulationstechnik)		Obermüllnerstr. 17/31	A	- 1020 Wien koroknai@ws1.atv.tuwien.ac.at
Dr.	Krösl	Peter	Ludwig Boltzmann Institut für	Experimentelle Traumatologie	Donaueschingenstr. 13	A	- 1200 Wien
	Lämmerhofer	Michael	(TU Wien, Abt. Simulationstechnik)		Mariahilferstr. 143/6	A	- 1150 Wien
	Mittermayr	Christian	TU Wien	Inst. f. Analytische Chemie	Getriedemarkt 9	A	- 1060 Wien
	Popp	Peter Hanns	(TU Wien, Abt. Simulationstechnik)		Weilburgstr. 10/5/1	A	- 2500 Baden
	Redlein	Alexander	TU Wien	Inst. f. Automation	Treitlstr. 3	A	- 1040 Wien
Dr.	Salzmann	Manfred	(TU Wien, Abt. Simulationstechnik)		Lorenz Müllerg. 1/138	A	- 1200 Wien msalz@ws1.atv.tuwien.ac.at
	Schäfer	Erich	Universität für Bodenkultur	IWGF, Abt. Siedlungswasserbau	Nußdorfer Lände 11	A	- 1190 Wien schaefer@iwgf-sig.boku.ac.at
Prof.Dr.	Troch	Inge	TU Wien	Abt. Simulationstechnik	Wiedner Hauptstr. 8-10	A	- 1040 Wien
	Valentin	Christoph	(TU Wien, Abt. Simulationstechnik)		Weinbergg. 93/18/1	A	- 1190 Wien
Dipl.Ing.	Weinmeier	Peter	TU Wien	Inst. f. El. Maschinen	Gußhausstr. 27-29	A	- 1040 Wien
Dr.	Weisz	Willy	TU Wien	EDV-Zentrum	Wiedner Hauptstr. 8-10	A	- 1040 Wien weisz@edvz.tuwien.ac.at
	Widerin	Ute	(TU Wien, Abt. Simulationstechnik)		Hofmühlg. 7A	A	- 1060 Wien

Problem Organisation For Control System Simulation

1. Introduction

Many methods exist for describing a mathematical model. Sets of ordinary differential equations and algebraic equations are clearly of central importance in lumped parameter dynamic models. In the case of linear models used for control system design there are other forms of description which may be more natural, including transfer functions, block diagrams, signal flow graphs and bond graphs. All of these can be used to show, in diagrammatic form, the mathematical operations to be performed and the order in which information must be processed.

2. Descriptions for continuous-variable models: reduced and state variable forms

Two of the most widely used forms of mathematical description for dynamic systems are the so-called **reduced form** and the **state variable form**. An example of a simple model expressed in reduced form is the equation

$$M\ddot{y} + R\dot{y} + Ky = f(t) \quad (1)$$

which is the differential equation commonly used to represent the mechanical system of Figure 1 involving the displacement, $y(t)$, of a mass M which is subjected to an external force $f(t)$ while suspended by means of a linear spring of stiffness K and damping element having viscous resistance R .

The corresponding description in state variable form involves two first order equations in place of the single second order equation. It takes the form

$$\dot{x}_1 = x_2 \quad (2)$$

$$\dot{x}_2 = -\frac{K}{M}x_1 - \frac{R}{M}x_2 + \frac{f(t)}{M} \quad (3)$$

where the new variable x_1 is the displacement, $y(t)$, and the new variable x_2 is the velocity, dy/dt . These are known as the **state variables** of the system. An n th order description in reduced form is equivalent to a set of n first order ordinary differential equations in state variable form. There is no unique set of state variables for a given model in reduced form.

Physical reasoning is often used in selecting quantities to be defined as state variables. In the system of Figure 1 it is clear that displacement and velocity have advantages over other possible sets of state variables. Displacement is a variable of particular interest in the model and is also likely to be a measured quantity in the real system. Velocity is also likely to be a measurable quantity.

The two-dimensional vector having components x_1 and x_2 is known as the state vector. The importance of the state vector is that if the initial state is defined and the system inputs

are known all future states are defined. The two equations defining the state-space model above can be rewritten as a single vector matrix equation

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{K}{M} & -\frac{R}{M} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{M} \end{bmatrix} f(t) \quad (4)$$

or, more concisely, as

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{b}u \quad (5)$$

where \mathbf{x} is the state vector, \mathbf{A} is a 2×2 square matrix and \mathbf{b} is a two-element column vector. This equation relates the rate of change of the state to the present state and the input. It is a form which is particularly convenient for simulation purposes since the numerical solution can then be obtained for each equation within the state-space model simply by a process of integration. A second order system, with two state variables, thus requires two integration operations in the corresponding simulation program; a sixth order model, with six state variables, would require six integrations.

In the case of a nonlinear model the state space model would have the form

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}(t), u(t)) \quad (6)$$

where \mathbf{f} now denotes the vector of derivative functions and $u(t)$ is the input.

3 Conversion from reduced form to state-variable form

If the model is given in the form

$$\frac{d^n y}{dt^n} = f\left[t, y, \frac{dy}{dt}, \dots, \frac{d^{n-1}y}{dt^{n-1}}, u(t)\right] \quad (7)$$

where $u(t)$ represents an input forcing function and f is a given linear or nonlinear function, it is always possible to convert the model to state space form by selecting the following as the state variables

$$x_1 = y$$

$$x_2 = \frac{dy}{dt}$$

$$\vdots$$

$$x_n = \frac{d^{n-1}y}{dt^{n-1}}$$

$$x_n = \frac{d^{n-1}y}{dt^{n-1}} \quad (8)$$

In the case of a mechanical system, this could involve selecting position, velocity, acceleration etc. as the set of state variables. The resulting state space description has the form

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = x_3$$

$$\cdot$$

$$\dot{x}_{n-1} = x_n$$

$$\dot{x}_n = f[t, x_1, x_2, \dots, x_n, u(t)] \quad (9)$$

Since the choice of state variables is never unique the state variables may be numbered in any order.

4 Transfer function descriptions

Linear lumped parameter models can also be written very conveniently as transfer functions, defined as the ratio of the Laplace transform of the model output variable to the Laplace transform of the input when all initial conditions are zero. Most introductory texts dealing with control systems engineering (e.g. [1], [2]) provide a detailed account of transfer function models and their derivation.

The assumption that all initial conditions are zero when using a transfer function type of model allows a given differential equation to be transformed into the Laplace domain simply by replacing d/dt by s , d^2/dt^2 by s^2 and so on for higher derivatives. Subsequent analysis is all carried out in terms of algebraic relationships which both simplifies the mathematical operations which have to be performed and can also provide additional physical insight. For example, the mass-spring-damper system defined in Equation (1) transforms to

$$Ms^2Y(s) + RsY(s) + KY(s) = F(s) \quad (10)$$

where $Y(s)$ is the Laplace transform of $y(t)$, $F(s)$ is the Laplace transform of $f(t)$ and all

initial conditions on $y(t)$ and its derivatives are zero. The quantity $Y(s)$ can then be manipulated algebraically to give

$$(Ms^2 + Rs + K) Y(s) = F(s) \quad (11)$$

and thus

$$\frac{Y(s)}{F(s)} = \frac{1}{Ms^2 + Rs + K} \quad (12)$$

In general a transfer function $G(s)$ may be written in the form

$$\frac{Y(s)}{U(s)} = G(s) = \frac{A(s)}{B(s)} \quad (13)$$

where $Y(s)$ is the Laplace transform of the output variable, $U(s)$ is the Laplace transform of the input and where $A(s)$ and $B(s)$ are polynomials in s .

Many properties of a transfer function depend upon the denominator $B(s)$. The roots of the **characteristic equation** $B(s)=0$ largely determine the form of the output when the transfer function is subjected to a given input. These roots are the **poles** of the transfer function. The number of poles is, of course, equal to the order of the system as discussed above in the context of state variable descriptions.

5. Block diagram and signal flow graph representations

Linear systems can be described using block diagrams or signal flow graphs. Block diagrams and signal flow graphs are important for simulation since these both provide a simple means of expressing the structure of a complex model. Many simulation packages are block diagram orientated and a thorough understanding of these diagrams is essential.

In the block diagram approach to the description of a system each element is described by a single block. The arrow entering the block represents the input variable and the arrow leaving the block represents the output variable. The block contains an expression, usually in transfer function form, which relates the output to the input. Signal flow diagrams incorporate exactly the same information as the block diagram. In this case input and output variables are represented by nodes and the line connecting two nodes is the equivalent of the block in the block diagram approach. A given system can be represented at many different levels using a block diagram or signal flow graph. For example Figure 2 shows valid block diagram and signal flow graph representations for the mass-spring-damper system of Equation (1). In these diagrams the only variables which appear are the input variable $F(s)$ and the output $Y(s)$. The diagrams describe only the relationship between the chosen input variable and the chosen output variable. Figure 3 shows another level of representation. Here the diagrams involve a number of blocks or signal flow graph elements in combination and the structure provides additional information about the system. It should, of course, be noted that the diagrams of Figures 2 and 3 are exactly equivalent and Figure 2 may be derived from Figure 3 using standard rules for block diagram or signal flow graph manipulation and reduction.

6. Block diagram and signal flow graph methods for transfer function simulation

Several methods exist for deriving a set of equations in state variable form from a given transfer function. The methods given here are based on block-diagrams or signal-flow graphs. more than one approach is presented because some methods, which can be entirely satisfactory with low order models, present problems of numerical robustness when applied to higher order problems.

6.1 The direct construction approach

Most transfer functions of practical importance can be manipulated into a general form involving a ratio of two polynomials in s and a gain factor. Consider the transfer function

$$\frac{Y(s)}{U(s)} = \frac{K (s^m + a_{m-1}s^{m-1} + \dots + a_2s^2 + a_1s + a_0)}{s^n + b_{n-1}s^{n-1} + \dots + b_2s^2 + b_1s + b_0} \quad (14)$$

where $n > m$ and K is a simple gain factor. This is in the required form and the restriction that the order of the denominator should be greater than that of the numerator is connected with conditions for physical realisability.

The only transfer functions which are of general practical importance for control systems applications but which cannot be manipulated into the form shown in Equation (14) involve pure delay elements involving factors $\exp(-sT)$ or distributed delay elements involving factors $\exp(-\sqrt{s}T)$. However, such cases can be handled without difficulty if the delay elements are simple multiplicative factors which can be treated as a separate block in cascade with a block described by a transfer function involving a ratio of polynomials in s .

Dividing all terms in the numerator and denominator of the right hand side by s^n gives

$$\frac{Y(s)}{U(s)} = \frac{K (s^{m-n} + a_{m-1}s^{m-1-n} + \dots + a_2s^{2-n} + a_1s^{1-n} + a_0s^{-n})}{1 + b_{n-1}s^{-1} + \dots + b_2s^{2-n} + b_1s^{1-n} + b_0s^{-n}} \quad (15)$$

It is now possible to rewrite the relationship between the transfer function output $Y(s)$ and the input $U(s)$ to involve an intermediate variable $E(s)$. This new variable need not have any obvious physical significance and is defined from the following equation

$$\frac{Y(s)}{U(s)} = \frac{Y(s)}{E(s)} \cdot \frac{E(s)}{U(s)} \quad (16)$$

The transfer function of Equation (15) may now be split into two parts as follows

$$\frac{Y(s)}{E(s)} = K (s^{m-n} + a_{m-1}s^{m-1-n} + \dots + a_2s^{2-n} + a_1s^{1-n} + a_0s^{-n}) \quad (17)$$

and

$$\frac{E(s)}{U(s)} = \frac{1}{1 + b_{n-1}s^{-1} + \dots + b_2s^{2-n} + b_1s^{1-n} + b_0s^{-n}} \quad (18)$$

Equation (18) may be re-arranged to give

$$E(s) = U(s) - (b_{n-1}s^{-1} + \dots + b_2s^{2-n} + b_1s^{1-n} + b_0s^{-n}) E(s)$$

which corresponds to a signal flow graph of the form shown in Figure 4. Here the new variable $E(s)$ is the input to a sequence of integrator elements, the output of each of which is fed back to the input through coefficient elements. The outputs of each of the integrators, taken in order from the left, are thus $s^{-1}E(s)$, $s^{-2}E(s)$, $s^{-3}E(s)$, ..., $s^{-n}E(s)$. However from Equation (17) the output $Y(s)$ is seen to be a weighted sum of $m+1$ quantities such as these and this gives a signal flow graph such as that shown in Figure 5.

Variables from each of the integrator blocks in Figure 5 may be assigned as state variables equations in state variable form may be written down from the signal flow graph or block diagram by inspection. If the output of the final integrator is x_1 , with the other state variables taken as the outputs of each of the remaining integrator elements. The set of state equations is then as follows

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = x_3$$

$$\begin{aligned} & \cdot \\ & \cdot \\ & \cdot \end{aligned}$$

$$\dot{x}_{n-1} = x_n$$

$$\dot{x}_n = -b_0x_1 - b_1x_2 - \dots - b_{n-1}x_n + u(t) \quad (19)$$

This set of equations is similar to Equations (9). A single algebraic equation relating the output y to the state variables must now be added. It is clear from the block diagram, that this equation is

$$y = K(a_0x_1 + a_1x_2 + \dots + a_{m-1}x_{n-m} + x_{n-m+1}) \quad (20)$$

Equations (19) and (20) together provide a simple way for implementing the given transfer function within a simulation. The only disadvantage of this direct construction approach is that the properties of the resulting state space model can be highly sensitive to small numerical inaccuracies in coefficient values, especially in the case of high-order models.

6.2 The parallel construction approach

The parallel programming approach is appropriate when the given transfer function has a denominator in factored form as shown below

$$\frac{Y(s)}{U(s)} = \frac{K(s^m + a_{m-1}s^{m-1} + \dots + a_2s^2 + a_1s + a_0)}{(s+\beta_1)(s+\beta_2)\dots(s+\beta_n)} \quad (21)$$

Using partial fractions it is then possible to express the right hand side of this equation as a sum of terms each of which involves a single pole. The resulting equation is

$$\frac{Y(s)}{U(s)} = \frac{\alpha_1}{s+\beta_1} + \frac{\alpha_2}{s+\beta_2} + \dots + \frac{\alpha_n}{s+\beta_n} \quad (22)$$

Each term on the right hand side of the equation has the same form and the corresponding signal flow graph involves a parallel structure as shown in Figure 6. The state variables are taken to be the outputs of each integrator. The state equations describing the signal flow graph are then of the form

$$\begin{aligned} \dot{x}_1 &= -\beta_1x_1 + u(t) \\ \dot{x}_2 &= -\beta_2x_2 + u(t) \\ &\vdots \\ \dot{x}_n &= -\beta_nx_n + u(t) \end{aligned} \quad (23)$$

An additional algebraic equation relates the output y to the n state variables. This has the form

$$y = \alpha_1x_1 + \alpha_2x_2 + \dots + \alpha_nx_n \quad (24)$$

A model with a parallel structure having first-order (or second-order) blocks is generally more robust in terms of numerical sensitivity to coefficient inaccuracies than the equivalent model in direct construction form. The higher the order of the original transfer function description the greater are the benefits of the parallel approach.

6.3 The iterative construction approach

If a transfer function has only real poles and zeros the numerator and denominator may both be factorised into products of first-order factors to give a description of the following form:

$$\frac{K(s+\alpha_1)(s+\alpha_2)\dots(s+\alpha_m)}{(s+\beta_1)(s+\beta_2)\dots(s+\beta_n)} \quad (25)$$

where K , α_i and β_j are all real constants. Grouping the factors into products of terms of the form

$$\frac{(s+\alpha_i)}{(s+\beta_j)} \quad (26)$$

it is possible to construct a sub-model block diagram or signal flow graph which represents a single factor of the complete transfer function. This sub-model diagram has the form shown in Figure 7. Since the order of the denominator is always equal to or greater than the order of the numerator of the transfer function for physically realisable systems. Hence, when the process of grouping these factors in pairs has been completed, there may be some additional terms involving denominators of the form $(s + \beta_k)$ which cannot be combined with any numerator factors. In such cases the additional factors are dealt with by means of the sub-model signal flow graph element in Figure 8. The form of the diagram for the complete transfer function for the special case where the numerator is of the same order as the denominator is shown in Figure 9. The state variables are the output variables of the integrator blocks and the resulting state equations are as follows:

$$\dot{x}_n = -\beta_n x_n + u \quad (27)$$

$$\dot{x}_{n-1} = -\beta_{n-1} x_{n-1} + \alpha_n x_n + \dot{x}_n = -\beta_{n-1} x_{n-1} + (\alpha_n - \beta_n) x_n + u \quad (28)$$

$$\begin{aligned} & \dots \dots \dots \\ & \dots \dots \dots \end{aligned}$$

$$\dot{x}_{n-j} = -\beta_{n-j} x_{n-j} + (\alpha_{n-j+1} - \beta_{n-j+1}) x_{n-j+1} + \dots + (\alpha_n - \beta_n) x_n + u \quad (29)$$

$$\begin{aligned} & \dots \dots \dots \\ & \dots \dots \dots \end{aligned}$$

$$\dot{x}_1 = -\beta_1 x_1 + (\alpha_2 - \beta_2) x_2 + \dots + (\alpha_n - \beta_n) x_n + u \quad (30)$$

As with the parallel construction approach the iterative method is often superior to the direct construction method in terms of coefficient sensitivity. A cascaded arrangement of first or second order transfer functions tends to be more robust to coefficient errors than the

equivalent structure involving multiple feedback loops.

6.4 An example of block diagram construction from a transfer function

Consider the following transfer function:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{s^2 + 9s + 20}{s^3 + 6s^2 + 11s + 6} \quad (31)$$

Direct construction method

First of all the numerator and denominator polynomials on the right hand side of the equation should be divided by the highest power of s to give

$$G(s) = \frac{Y(s)}{U(s)} = \frac{s^{-1} + 9s^{-2} + 20s^{-3}}{1 + 6s^{-1} + 11s^{-2} + 6s^{-3}} \quad (32)$$

Introducing a dummy variable $E(s)$ gives

$$\frac{Y(s)}{E(s)} = s^{-1} + 9s^{-2} + 20s^{-3} \quad (33)$$

and

$$\frac{E(s)}{U(s)} = \frac{1}{1 + 6s^{-1} + 11s^{-2} + 6s^{-3}} \quad (34)$$

That is

$$E(s) = U(s) - (6s^{-1}E(s) + 11s^{-2}E(s) + 6s^{-3}E(s)) \quad (35)$$

and

$$Y(s) = s^{-1}E(s) + 9s^{-2}E(s) + 20s^{-3}E(s) \quad (36)$$

Equations (35) and (36) may be expressed in block diagram form by a model involving three integrators connected in cascade as shown in Figure 10. Taking the outputs of integrator blocks as state variables allows the following set of simultaneous first order differential equations to be established:

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = x_3$$

$$\dot{x}_3 = -6x_1 - 11x_2 - 6x_3$$

or, in matrix notation,

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6 & -11 & -6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u \quad (37)$$

The output equation is

$$y = 20x_1 + 9x_2 + x_3$$

or

$$y = [20 \ 9 \ 1] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad (38)$$

Parallel construction method

In this case the first step involves factorising the denominator of the given transfer function to give:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{s^2 + 9s + 20}{s^3 + 6s^2 + 11s + 6} = \frac{s^2 + 9s + 20}{(s+1)(s+2)(s+3)} \quad (39)$$

Expressing the right hand side of this equation in partial fraction form gives:

$$\frac{Y(s)}{U(s)} = \frac{6}{s+1} - \frac{6}{s+2} + \frac{1}{s+3} \quad (40)$$

and this may be represented as a block diagram with the parallel structure shown in Figure

9. Again the outputs of the integrator blocks are taken as the state variables to give the following set of first-order differential equations

$$\dot{x}_1 = -x_1 + u; \quad \dot{x}_2 = -2x_2 + u; \quad \dot{x}_3 = -3x_3 + u$$

and an algebraic equation relating the output, $y(t)$, to the three state variables $x_1(t)$, $x_2(t)$ and $x_3(t)$ which has the form

$$y = -6x_1 - 6x_2 - x_3$$

In matrix form the state and output equations are thus

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} u \quad (41)$$

and

$$y = \begin{bmatrix} 6 & 6 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad (42)$$

Iterative construction method

There are two zeros and three poles in the given transfer function. Hence the two zeros can be associated with two of the poles in the standard way for the iterative approach and one pole must be treated separately, as shown in the transfer function below

$$\frac{Y(s)}{U(s)} = \frac{s^2 + 9s + 20}{s^3 + 6s^2 + 11s + 6} = \left(\frac{s+5}{s+1} \right) \left(\frac{s+4}{s+2} \right) \left(\frac{1}{s+3} \right) \quad (43)$$

The resulting block diagram thus has two stages involving elements of the type shown in Figure 7 and one stage of the type shown in Figure 8. Figure 12 shows the complete block diagram. Taking outputs of integrator blocks as state variables gives a set of first order differential equations

$$\begin{aligned} \dot{x}_1 &= -3x_1 + 4x_2 + \dot{x}_2 \\ &= -3x_1 + 2x_2 + 4x_3 + u \end{aligned} \quad (44)$$

$$\dot{x}_2 = -2x_2 + 5x_3 + \dot{x}_3$$

$$= -2x_2 - 4x_3 + u \quad (45)$$

$$\dot{x}_3 = -x_3 + u \quad (46)$$

and an algebraic output equation

$$y = x_1 \quad (47)$$

7. Modelling of distributed parameter elements

7.1 Introduction

Distributed parameter models arise in the study of systems in which quantities are transmitted from one point to another, as in the case of electrical signals in a transmission line, flow of fluid in a pipe, phenomena associated with the conduction of heat etc.. The "signals" in such systems are distributed in space as well as time and the mathematical models are described by partial differential equations.

In situations in which the system reduces essentially to a single input variable and a single output variable of interest at one point in space, as often applies in control systems applications, it is possible to derive a transfer function type of description which involves a distributed parameter. Such distributed parameter transfer function models may then be used in the same way as conventional transfer functions for lumped parameter models. The two types of distributed parameter element considered in this section are the pure time delay and the distributed time delay.

7.2 Pure Time Delay

A pure time delay (known also as a transportation lag, or "dead time") arises in systems in which quantities are transmitted at finite velocity from one point to another. In the case of a pure delay this transmission does not involve any frequency dependent change of signal amplitude.

A system involving a pure time delay (due perhaps to physical phenomena such as heat flow but where losses can be neglected), may be described approximately by a partial differential equation of the form

$$\frac{\partial z}{\partial t} = v \frac{\partial z}{\partial x} \quad (48)$$

where $z(x,t)$ is the state of a point at position x and time t . The parameter v is the velocity of transmission.

Applying the Laplace transformation to this equation gives, assuming zero initial conditions, a new equation as follows

$$sZ(x, s) = v \frac{\partial Z(x, s)}{\partial x} \quad (49)$$

where

$$Z(x, s) = \int_0^{\infty} \exp(-st) z(x, t) dt \quad (50)$$

The solution of this equation is

$$Z(x, s) = A \exp\left(-s \frac{x}{v}\right) \quad (51)$$

where A is an arbitrary constant.

If $U(s)$ is the Laplace transform of the input of the system at point x_1 and $Y(s)$ is the transform of the output of the element at point x_2 then it follows that

$$U(s) = Z(x_1, s) = A \exp\left(-s \frac{x_1}{v}\right) \quad (52)$$

and

$$Y(s) = Z(x_2, s) = A \exp\left(-s \frac{x_2}{v}\right) \quad (53)$$

Thus the transfer function relating the output transform $Y(s)$ to the input transform $U(s)$ is

$$\frac{Y(s)}{U(s)} = \exp\left(-s \frac{x_2 - x_1}{v}\right) \quad (54)$$

That is

$$\frac{Y(s)}{U(s)} = \exp(-sT) \quad (55)$$

where T is the transmission time from point x_1 to point x_2 . The parameter T is thus the duration of the pure time delay.

7.3 Distributed time delay

A distributed time delay arises where quantities are transmitted from one point to another with a finite velocity but with an attenuation which varies with frequency. examples include the conduction of heat and the transmission of electrical signals in a medium which is not loss free. The partial differential equation describing this situation has the form

$$\frac{\partial z}{\partial t} = a \frac{\partial^2 z}{\partial x^2} \quad (56)$$

where $z(x,t)$ represents the value of the quantity concerned at point x and time t . The parameter a is a constant and could represent, for example, thermal conductivity. Applying the unilateral Laplace transform for zero initial conditions gives

$$s Z(x, s) = a \frac{\partial^2 Z(x, s)}{\partial x^2} \quad (57)$$

where

$$Z(x, s) = \int_0^{\infty} \exp(-st) z(x, t) dt \quad (58)$$

The solution of this equation has the form

$$Z(x, s) = A \exp\left(-\sqrt{\frac{s}{a}} x\right) + B \exp\left(\sqrt{\frac{s}{a}} x\right) \quad (59)$$

It is now assumed that at infinite distance the value of the variable $z(x,t)$ is always zero. This allows one of the terms in the above equation to be eliminated, giving

$$Z(x, s) = A \exp\left(-\sqrt{\frac{s}{a}} x\right) \quad (60)$$

for $x \geq 0$.

If $U(s)$ is the Laplace transform of the input to the system being modelled (at point x_1) and $Y(s)$ is the Laplace transform of the output of the system (at point x_2) then it follows that

$$U(s) = Z(x_1, s) = A \exp\left(-\sqrt{\frac{s}{a}} x_1\right) \quad (61)$$

Similarly

$$Y(s) = Z(x_2, s) = A \exp\left(-\sqrt{\frac{s}{a}} x_2\right) \quad (62)$$

Hence the transfer function relating $Y(s)$ to $U(s)$ has the form

$$\frac{Y(s)}{U(s)} = \exp\left(-\sqrt{\frac{s}{a}} (x_2 - x_1)\right) \quad (63)$$

Thus

$$\frac{Y(s)}{U(s)} = \exp\left(-\sqrt{s \frac{(x_2 - x_1)^2}{a}}\right) = \exp(-\sqrt{sT}) \quad (64)$$

where the parameter T is an equivalent time.

It should be noted that elimination of the term involving the positive exponent in this derivation has physical significance. It is equivalent to rejecting the possibility of reflected waves in the system under consideration. The use of the transfer function resulting from this analysis is therefore restricted to cases in which reflected waves are not expected.

7.4 Simulation models involving pure and distributed delay elements

Most simulation languages incorporate facilities for the representation of pure delays. Pure delays also present few difficulties in a simulation which is developed using a general purpose high level language, provided the delay does not itself vary with time. Cases involving variable delays present additional problems [3]. An alternative approach, and more approximate representation for a pure delay, is based upon the properties of Padé approximations. These approximations, based upon truncated series for the exponential function, were widely used for the representation of pure delays in analog simulations and it is equally possible to use this approach in a digital simulation.

More complex problems involving distributed parameter elements can be approached in a very general way using finite differences or finite element methods or other numerical techniques for the solution of partial differential equations. Detailed treatments of these specialised topics can be found in appropriate textbooks and a review of some of the problems of simulation of such systems may be found in the book by Spriet and Vansteenkiste [4]. Simulations involving distributed parameter elements can be numerically intensive and for time-critical applications distributed parameter problems are often reduced to quite simple lumped-parameter approximations. One example of this type can be found in the work of Bryce et al. [5] which is concerned with the real-time simulation of a water pipeline as part of an investigation of water-turbine governing systems.

References

- [1] Palm, W.J., "Control Systems Engineering", John Wiley & Sons, New York, U.S.A., 1986.
- [2] Golten, J. and Verwer, A., "Control System Design and Simulation", McGraw-Hill, London, U.K., 1991.
- [3] Doebelin, E.O. "System Modelling and Response", pp. 193-201, J. Wiley & Sons, New York, U.S.A., 1980.
- [4] Spriet, J.A. and Vansteenkiste, G.C. "Computer-aided Modelling and Simulation", Academic Press, London, U.K., 1982.
- [5] Bryce, G.W., Foord, T.R., Murray-Smith, D.J. and Agnew, P.W. 'Hybrid simulation of water-turbine governors', Simulation Councils Proceedings, Vol. 6, Part 1, pp. 35-44, 1976.

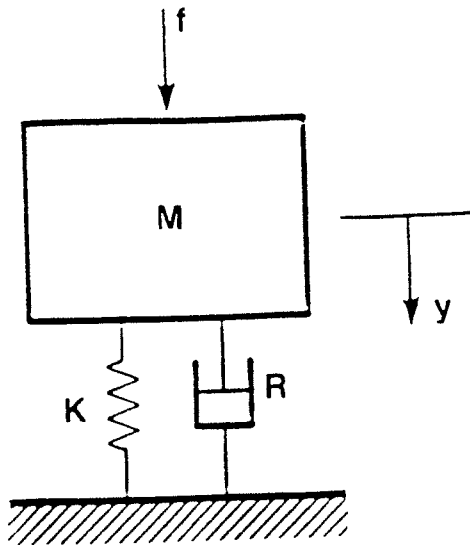


Figure 1: Mechanical system involving mass, spring and viscous damping elements.

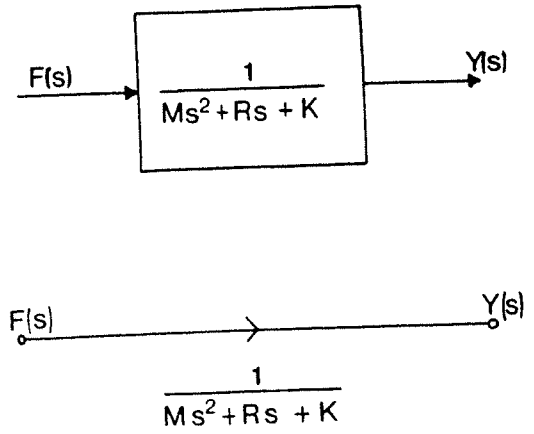


Figure 2: Transfer function based block diagram and signal flow graph for system of Figure 1.

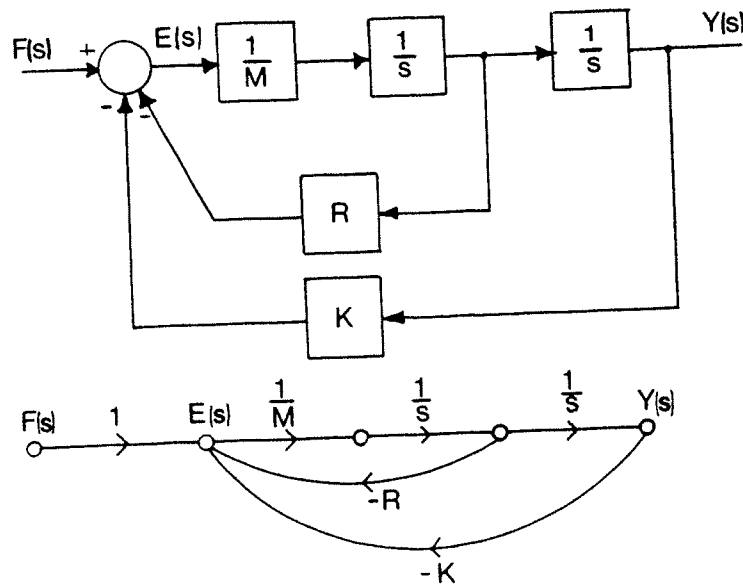


Figure 3: Detailed block diagram and signal flow graph for system of Figure 1.

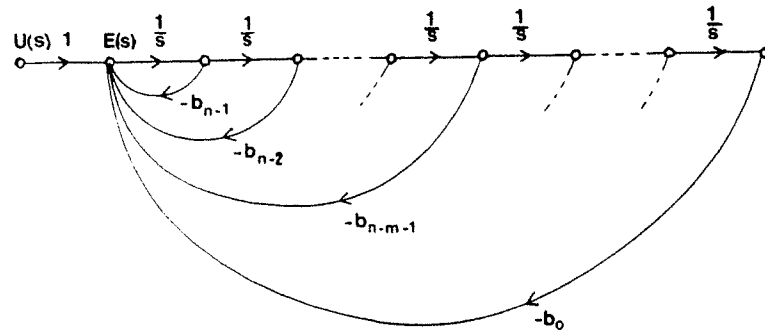


Figure 4: Direct construction signal flow graph for denominator terms.

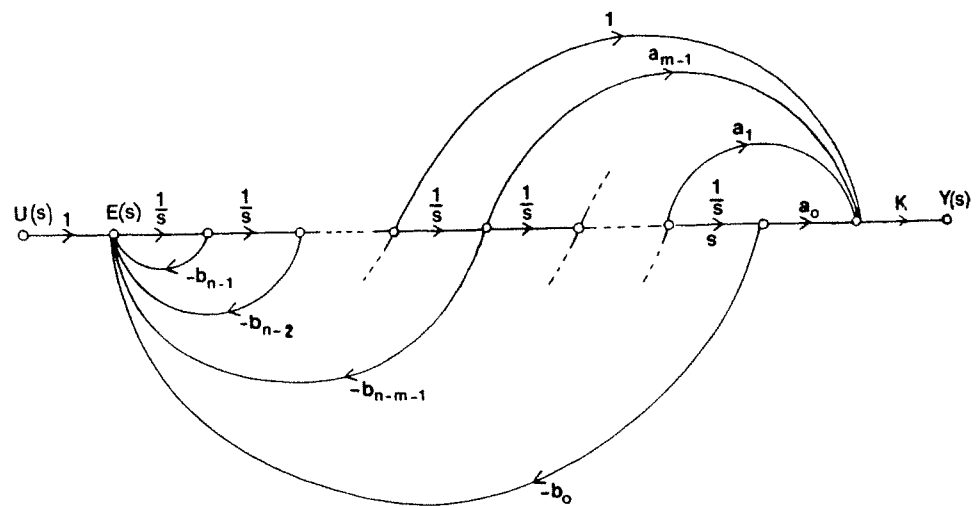


Figure 5: Complete signal flow graph for direct construction representation.

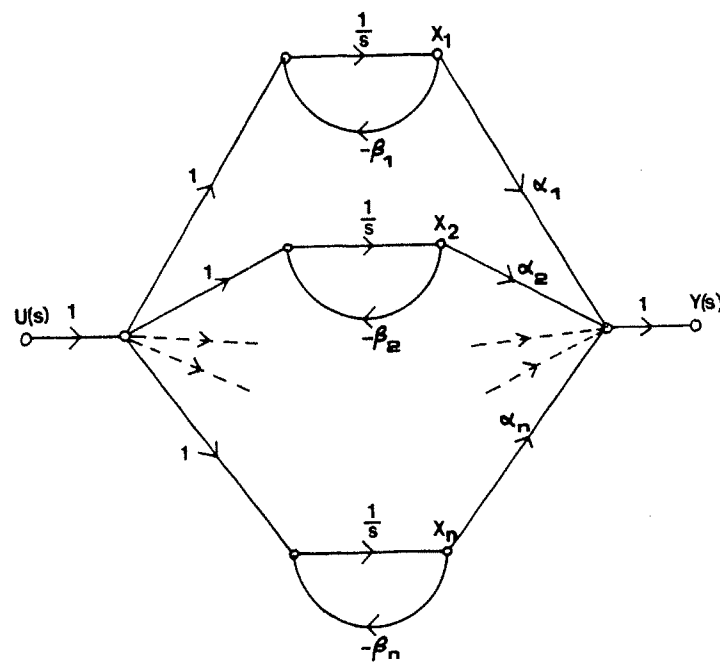


Figure 6: Signal flow graph illustrating parallel construction method.

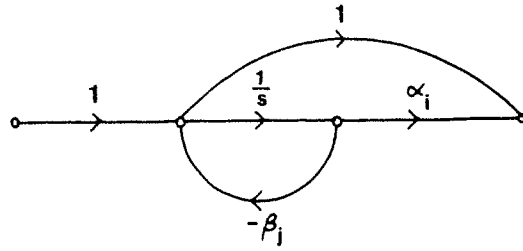


Figure 7: Signal flow graph for single pole-zero pair.

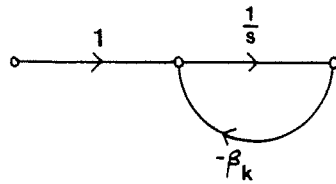


Figure 8: Signal flow graph for single pole.

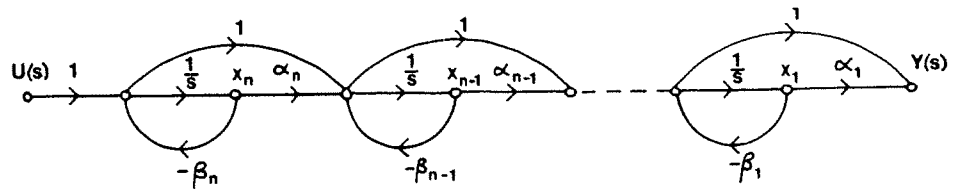


Figure 9: Signal flow graph illustrating iterative construction method.

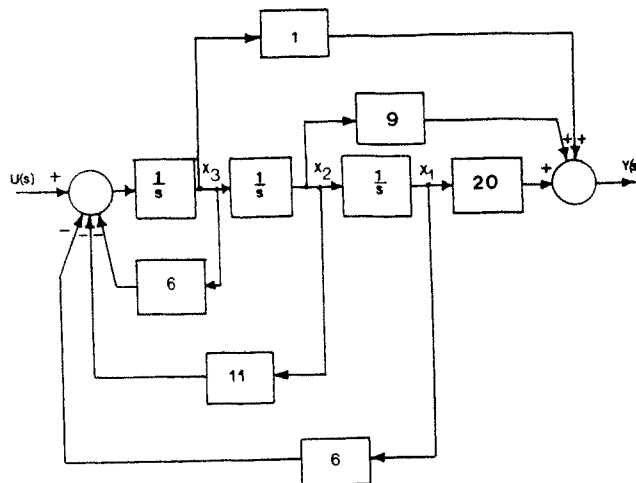


Figure 10: Block diagram for example system by direct construction method.

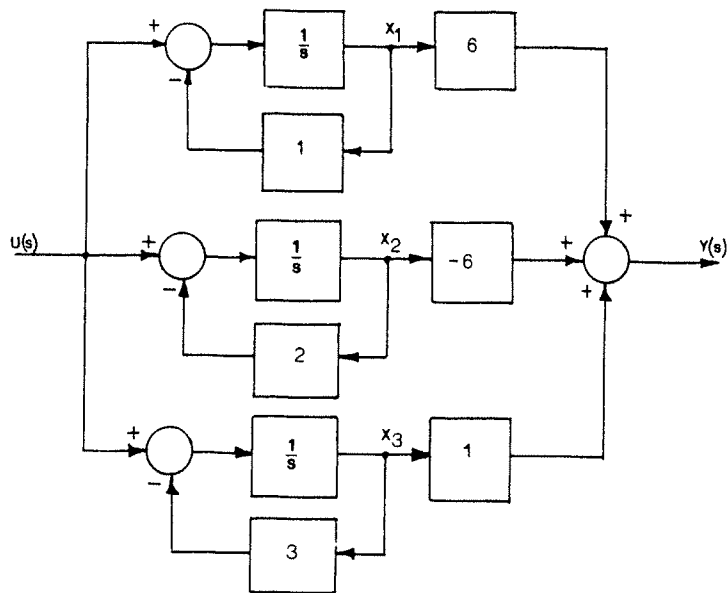


Figure 11: Block diagram for example system by parallel construction method.

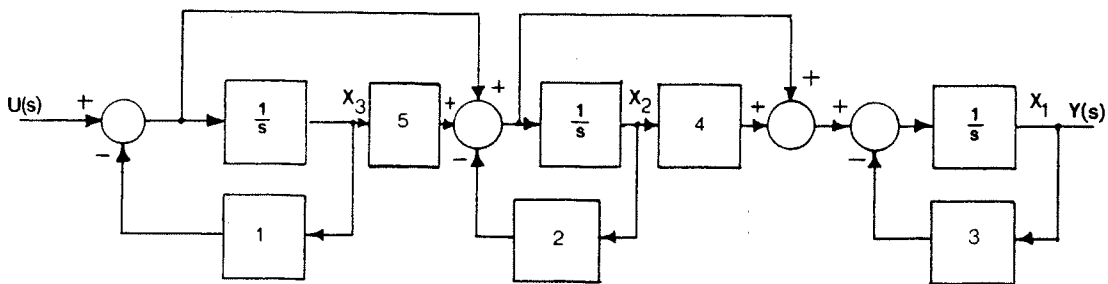


Figure 12: Block diagram for example system by iterative construction method.

Simple Examples of Control System Simulation Using Common Simulation Tools

1. Introduction

This section of the course provides examples of a number of different types of simulation problem involving control system applications. The models upon which the simulations are based are relatively simple and have been chosen to illustrate features of some typical simulation tools which are currently available. Further details of the SLIM simulation language may be found in the Users' Manual and in an associated textbook [1] which also provides some additional linear and nonlinear simulation examples. The underlying principles which can be applied both in simulation studies of systems developed using classical methods of control system design and systems designed using fuzzy control principles. Particular simulation techniques have been selected but this choice is not intended to suggest that other tools could not have provided an equally appropriate solution but strengths and weaknesses of some methods of approach are discussed. In each case suggestions are made of further investigations which could be made with these simple models.

2. A control system simulation: speed control of a water turbine

Control systems investigations form one of the most common application areas for continuous system simulation tools. This example is concerned with a simulation of a speed control system for a water turbine used for the generation of electricity. Figure 1 is a block diagram of a highly simplified description of a system of this kind. The turbine, which is of the impulse type, is represented by a linear transfer function which also incorporates dynamics of the pipeline system. Control of the turbine is accomplished through feedback of a signal proportional to turbine shaft speed and comparison with the desired (reference) speed signal. Any difference between the desired and actual speed provides an error signal which is processed by the controller block to provide the signal applied to the turbine actuator input to change the water flow in such a way that the error is continuously minimised. The controller may be implemented either in analogue (continuous) or digital form. The model is linear throughout, apart from backlash in the mechanical linkages associated with the turbine inlet actuator. The purpose of the model is to allow the performance of the model of the closed loop system to be investigated using both continuous and digital controllers. This is a useful illustration of continuous and discrete simulation techniques being used together to approach an important class of practical engineering problem.

A number of controller transfer functions can be considered for this type of application. Detailed consideration of possible forms of controller are not of primary interest in the present context but any readers interested in this aspect, or in the underlying model, may find further details elsewhere (e.g.[1-2]). For the present purposes the controller considered is of the "temporary droop" type implemented in continuous form. This type of controller has a transfer function of the form:-

$$C(s) = \frac{1 + sT_x}{\sigma + (\sigma + \mu) T_x s + T_x T_y s^2} \quad (1)$$

In this controller transfer function only the parameters σ and μ may be regarded as adjustable quantities. The other parameters (T_x and T_y) are fixed quantities and are not available for controller tuning.

Appropriate parameter values for the plant and controller are as follows:

$T_s = 7.0$ sec. (inertial time constant)

$T_w = 1.1$ sec. (water time constant)

$T_s = 0.2$ sec. (actuator servo time constant)

All model variables, such as turbine speed are expressed as normalised (per unit) quantities.

Figure 2 shows the DYNAMIC segment of a SLIM program (TURB.SLI) for the simulation model and the full source may be found on the diskette. Parameter values for the nominal set of conditions are provided in the program listing. The simulation experiment in this case involves investigation of the response of the control system to a step change of the reference speed. It can be seen from Figure 3 that the response is stable but oscillatory in nature for the controller parameters values used. A program TURB2.SLI, which allows input from a data file, is also included in the Appendix to this section of the notes. This version of the simulation program, together with the appropriate data files in the same format as the test file TURB2.IN, provides a convenient means of experimenting with controller parameter values.

One interesting extension to this simulation program involves the inclusion of mechanical backlash between the servomotor and the turbine inlet. This is an important feature of real mechanical systems of this kind and is known to have a destabilising effect on the overall control system. Backlash is a double-valued form of nonlinearity which has a steady-state input-output characteristic of the form shown in Figure 3. It generally arises because of 'slack' in mechanical linkages and gears. With backlash present movement of the input in one direction produces a proportional movement of the output, but any reversal of the direction of the input will cause the output to stop before following the input again.

The representation of backlash and other similar double-valued nonlinearities which involve a form of "memory", such as hysteresis, can be difficult. One approach is to use principles first established for the modelling of backlash elements using analogue computers [3]. This involves the use of an integrator with feedback to provide the memory element. Figure 4 shows a block diagram for a representation of backlash by this type of method. Essentially the system works by comparing the integrator output, which is the output variable for the backlash element, with the input. The integrator input is controlled through comparator elements and logic blocks and when the input reverses direction (at the extremes of travel) the integrator input becomes zero. The integrator input remains zero until the input variable has changed by an amount equal to the width of the backlash element. It is then switched back to the output of the summing element. This causes the integrator output to remain fixed in value for a period of time and this corresponds to the flat top and bottom sections on the diagram showing the input-output characteristics for the backlash. Although presented here in block-diagram form the implementation of a backlash model of this kind is quite simple using the equation-oriented methods. Figure 5 shows the DYNAMIC segment of a SLIM program TURB2.SLI which is a version of the turbine speed control system model with backlash incorporated. This program and the corresponding data file TURB2.IN are listed in the Appendix.

The oscillation observed on the response shown in Figure 2 is influenced by the backlash parameter a_b as well as by the parameters of the controller. A well damped step response in the absence of backlash ($a_b=0.0$) can become a maintained oscillation if the backlash is increased sufficiently (Figure 6).

Figure 7 is an ACSL program listing for this example. Note that a special statement BCKLSH is available in ACSL to represent backlash effects and use has been made of this facility. Figure 8 shows results obtained from this ACSL program. In modern block-oriented tools such as XANALOG and SIMULINK special facilities are also available for the simulation of nonlinear elements such as backlash. Figure 9 shows an XANALOG block diagram for this problem.

Further investigations which could be carried out using this model and the SLIM simulation programs include determination of the minimum value of the backlash parameter b which gives rise to a maintained oscillation in terms of turbine speed. This "limit cycling" type of behaviour is clearly an undesirable situation in a speed control system and one that should be avoided in practice. For those with an appropriate level of understanding of control systems analysis techniques the simulation result for the critical value of the backlash parameter may be compared with predictions from theory, based upon describing function analysis. It is interesting to consider the reasons for any differences between the simulation result and the value predicted by theory. It should be remembered that the describing function approach involves some important simplifying assumptions and approximations; the significance of some of these can be investigated easily using the simulation.

3. Simulation of a simple digital control system

Figure 10 is a block diagram of a simple closed-loop digital control system. The difference between the reference signal and the plant output forms an error which is sampled by the digital-to-analogue converter. The digital processor carries out some form of numerical operation on the error samples and provides an actuating signal to the plant input through the digital-to-analogue converter. Figure 11 shows part of the corresponding SLIM program listing for the simplest possible situation in which the control computer simply samples the error signal and outputs the sampled error values periodically as input to the plant. The complete program file, named DIGCON.SLI, and the necessary input data file DIGCON.IN, can be found on the diskette. Examination of the DIGCON.SLI shows that, because SLIM does not have special facilities for mixed continuous and discrete system simulation, the sampling period is defined as a multiple of the communication interval parameter and the facilities of the DYNAMIC segment and DERIVATIVE section are used to emulate the discrete action of the controller. The sampled variable is held constant in a zero-order hold type of action by a loop within the DYNAMIC segment. Any discrete calculations representing the action of a control algorithm within the control computer must also be performed within the DYNAMIC segment. Values of variables of interest in the simulation may be written to the output file in the usual way, at times set by the communication interval.

Figure 12 shows results obtained for three different sampling periods. It can be seen that the system shows an increasing tendency to oscillate as the sampling period is increased and eventually becomes unstable. Sampled data theory (see, for example [14]) predicts that for this system instability occurs when the sampling period is greater than 0.549 sec.. This is consistent with the results presented in Figure 12 and detailed investigations using the

simulation program can confirm the theoretical result more precisely.

The DIGCON.SLI program can be modified easily to allow for more complex controller action. For example, if one wanted to simulate the system with a controller which implemented a difference equation of the form

$$O(kT) = O((k-1)T) + I(kT) - 0.5I((k-1)T) \quad (2)$$

the changes to the simulation program would all be made in the initial part of the DYNAMIC segment. Figure 13 shows the relevant part of the listing and the complete program is listed in the Appendix as the file DIGCON1.SLI. Note how the discrete input and output variables are stored for one sample period and updated. The transfer function of the controller of Equation (2), expressed in terms of z transforms, is as follows:

$$\frac{O(z)}{I(z)} = \frac{1 - 0.5z^{-1}}{1 - z^{-1}} = \frac{z - 0.5}{z - 1} \quad (3)$$

In the special case when the sample period is 0.347 sec. the controller should, from sampled-data theory, act as a "dead beat" compensator [4]. In such a situation the plant output should exhibit zero steady state error and should rise to its final value, in response to a step change of system reference input, in one sampling period. Figure 14 shows results from the simulation program which are consistent with theory for this special case.

Some other equation-oriented simulation languages, such as ACSL, include special facilities which can be very useful for the simulation of digital control systems. In ACSL, for example, DISCRETE sections representing the difference equations or z-transfer function of a digital controller may be inserted within the DYNAMIC segment. Such DISCRETE sections are thus similar to DERIVATIVE sections but communicate with the continuous parts of the simulation at regular predetermined times. Figures 15 and 16 show XANALOG and SIMULINK block diagrams for this digital control problem and illustrate some more of the specialised blocks and icons available with these simulation tools.

This example offers any reader interested in automatic control systems many opportunities for experimentation. It is clear from the listing of the SLIM program DIGCON1.SLI that, with some minor changes to the DYNAMIC segment, it would be very easy to replace the dead beat compensator by some other form of controller. Similarly any other form of plant transfer function could be used in place of the one given in Figure 10, with only some simple changes to the DERIVATIVE section of the program being necessary.

References

- [1] Murray-Smith, D.J. "Continuous System Simulation", Chapman and Hall, London, 1995.
- [2] Bryce, G.W., Foord, T.R., Murray-Smith, D.J. and Agnew, P., 'Hybrid simulation of water turbine governors', Simulation Council Proceedings, Vol.6, Part 1, pp. 35-44, 1976.
- [3] Ricci, F.J. "Analog-Logic Computer Programming and Simulation", Spartan Books 1972.
- [4] Leigh, J.R., "Applied Digital Control", Prentice-Hall Intl., Englewood Cliffs, N.J., U.S.A., 1984.

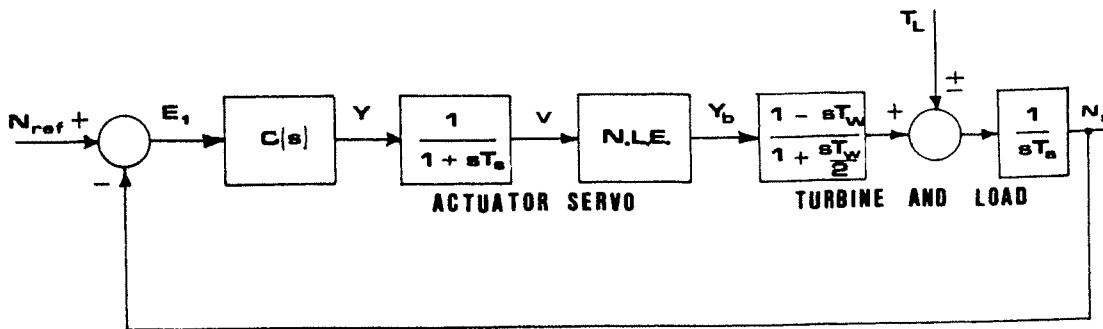


Figure 1: Block diagram of closed-loop system for automatic control of speed of water turbine connected to an electrical generator.

```

C
C *****Start of Dynamic Segment*****
C
C   DYNAMIC
C
C *****Start of Derivative Segment*****
C
C   DERIVATIVE
C     DT1=(V-T1)*(2.0/TW)
C     T1=INTEG(DT1,T10)
C     T0=T1-TW*DT1
C     TA=T0-TL
C     DNS=TA/TIA
C     ANS=INTEG(DNS,ANS0)
C     E1=REF-ANS
C     DDY1=(E1-SIG*Y1-((SIG+AMU)*TX+TY)*DY1)/(TX*TY)
C     DY1=INTEG(DDY1,DDY10)
C     Y1=INTEG(DY1,Y10)
C     Y=Y1+TX*DY1
C     DV=(Y-V)/TS
C     V=INTEG(DV,V0)
C   DERIVATIVE END
C
C *****End of Derivative Section*****
C
C   Values of t, ref, tl and ans for current communication
C   interval output to results file
C
C     TYPE T,REF,TL,ANS
C
C   Test for end of simulation run
C
C     IF(T-TMAX)10,10,12
10   DYNAMIC END
C
C *****End of Dynamic Segment*****
C
C *****Terminal Segment*****
12   STOP
    END

```

Figure 2: Part of SLIM program TURB.SLI for simulation of the turbine speed control system with a temporary-droop type of governor.

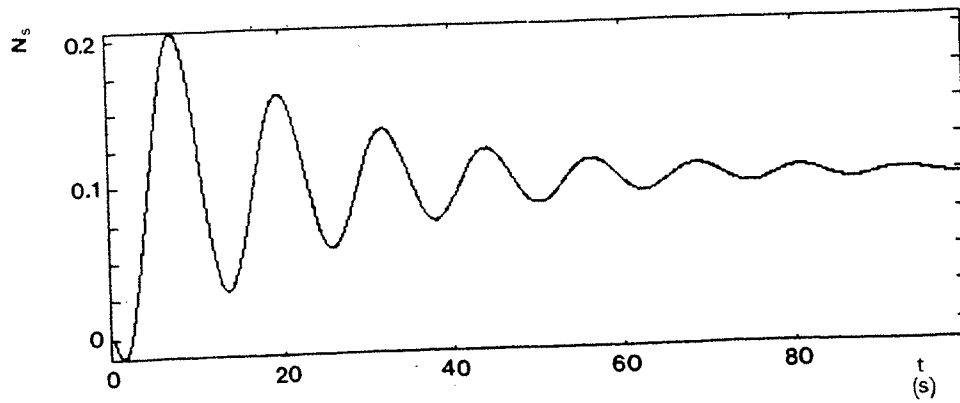


Figure 3: A typical simulated response of the speed control system to a step change in reference speed.

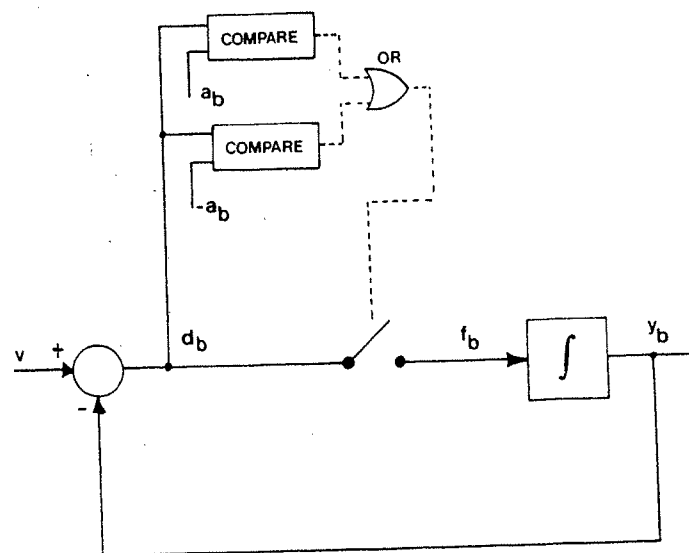


Figure 4: Block diagram illustrating one technique for representation of backlash element.

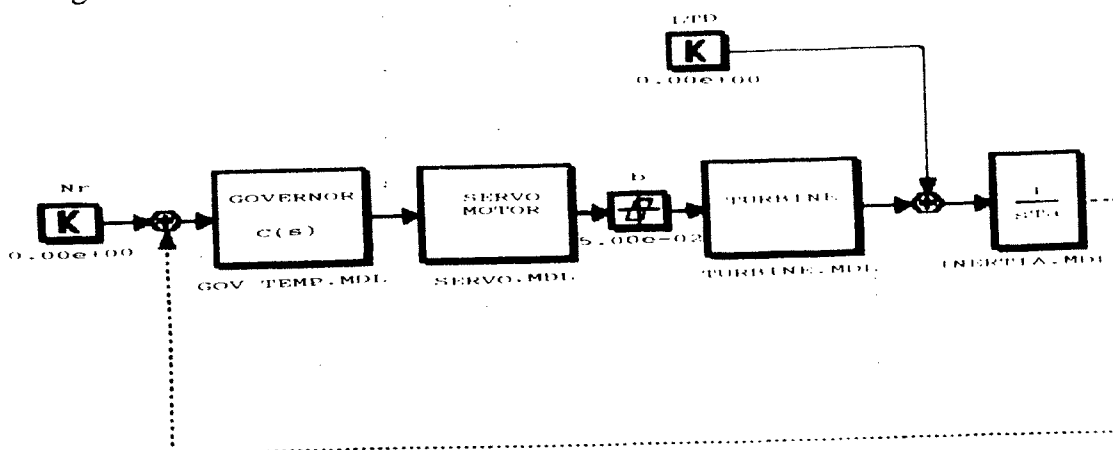


Figure 9: XANALOG block diagram of speed control problem.

*****Start of Dynamic Segment*****

DYNAMIC

*****Start of Derivative Segment*****

DERIVATIVE

Section for representation of backlash element

```
IF(AB-0.001)40,45,45
DB=V-YB
IF(DB+AB)60,50,50
FB=DB
GOTO 100
IF(DB-AB)80,70,70
FB=DB
GOTO 100
FB=0.0
YB=INTEG(10.0*FB,YB0)
GOTO 110
YB=V
```

End of section for backlash

```
DT1=(YB-T1)*(2.0/TW)
T1=INTEG(DT1,T10)
T0=T1-TW*DT1
TA=T0-TL
DNS=TA/TIA
ANS=INTEG(DNS,ANS0)
E1=REF-ANS
DDY1=(E1-SIG*Y1-((SIG+AMU)*TX+TY)*DY1)/(TX*TY)
DY1=INTEG(DDY1,DDY10)
Y1=INTEG(DY1,Y10)
Y=Y1+TX*DY1
DV=(Y-V)/TS
V=INTEG(DV,V0)
DERIVATIVE END
```

*****End of Derivative Section*****

Values of t, ref, tl, ans and yb for current communication interval output to results file

```
TYPE T,REF,TL,ANS,YB
```

Test for end of simulation run

```
IF(T-TMAX)10,10,12
DYNAMIC END
```

*****End of Dynamic Segment*****

*****Terminal Segment*****

```
STOP
END
```

Figure 5: Part of SLIM program TURB2.SLI for simulation of the turbine speed control system with a temporary-droop type of governor and backlash between the servomotor and the turbine inlet valve.

PROGRAM turb2.csl

"Simulation of governed hydro-turbine system with"
 "a temporary-droop governor and with backlash between"
 "servo-motor and turbine inlet actuator"

INITIAL

"Data for plant and governor transfer functions"

CONSTANT tw=1.1, ts=0.2, tia=7.0

CONSTANT ab=0.06 \$ "backlash parameter"

CONSTANT tx=16.0, ty=0.3, sig=0.03, mu=0.25

"Data for reference input and disturbance input"

CONSTANT ref=0.1, tl=0.0

"Data for experiment duration"

CONSTANT tend=99.9

ALGORITHM IALG=4 \$ "Runge-Kutta second order"

CINTERVAL CINT=1.0

"Initial conditions"

tl=0.0

v=0.0

y1=0.0

dyl=0.0

yb0=0.0

END \$ "of INITIAL"

DYNAMIC

DERIVATIVE

"Turbine"

dtl=(yb-tl)*(2.0/tw)

tl=INTEG(dtl,0.0)

t0=tl-tw*dtl

"Load"

ta=t0+tl

dns=ta/tia

ns=INTEG(dns,0.0)

"Governor"

e1=ref-ns

ddy1=(e1-sig*y1-((sig+mu)*tx+ty)*dyl)/(tx*ty)

dyl=INTEG(ddy1,0.0)

y1=INTEG(dyl,0.0)

y=y1+tx*dyl

"Servo motor"

dv=(y-v)/ts

v=INTEG(dv,0.0)

yb=BCKLSH(yb0,ab,v)

END \$ "of DERIVATIVE"

TERMT (t.GE.tend)

END \$ "of DYNAMIC"

TERMINAL

END \$ "of TERMINAL"

END \$ "of PROGRAM"

Figure 7: ACSL Program listing for the speed control problem.

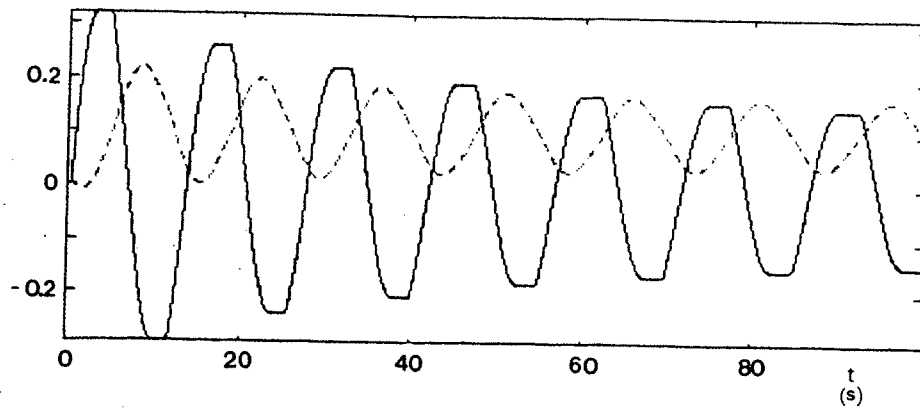


Figure 6: Response of the simulated speed control system with sufficient backlash to cause limit cycle oscillations. The continuous line is the control valve position while the dashed line is the output speed.

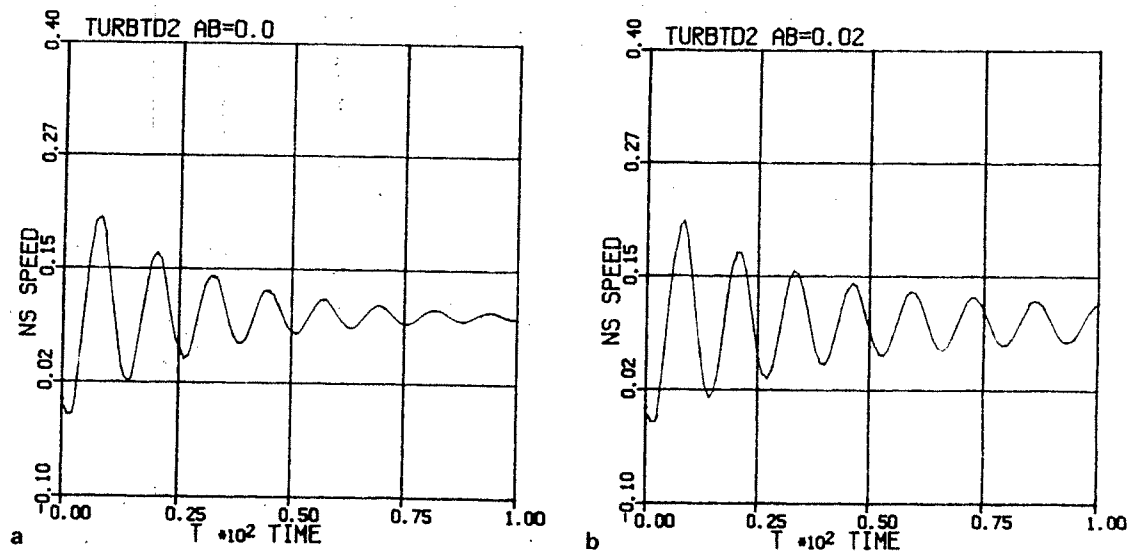


Figure 8: Results from ACSL simulation, a) with no backlash and b) with backlash of 0.02

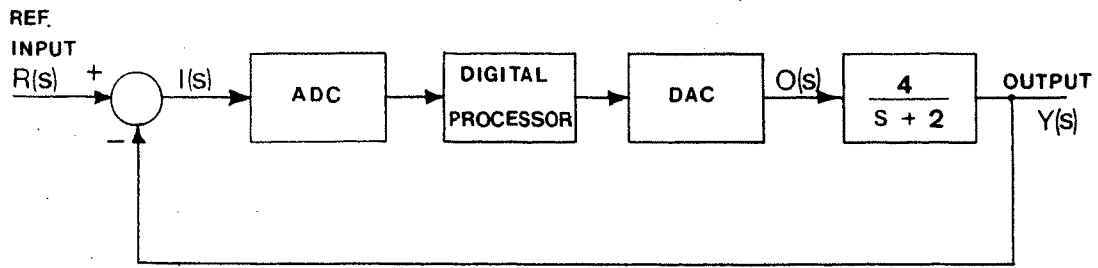


Figure 10: Block diagram of simple digital control system

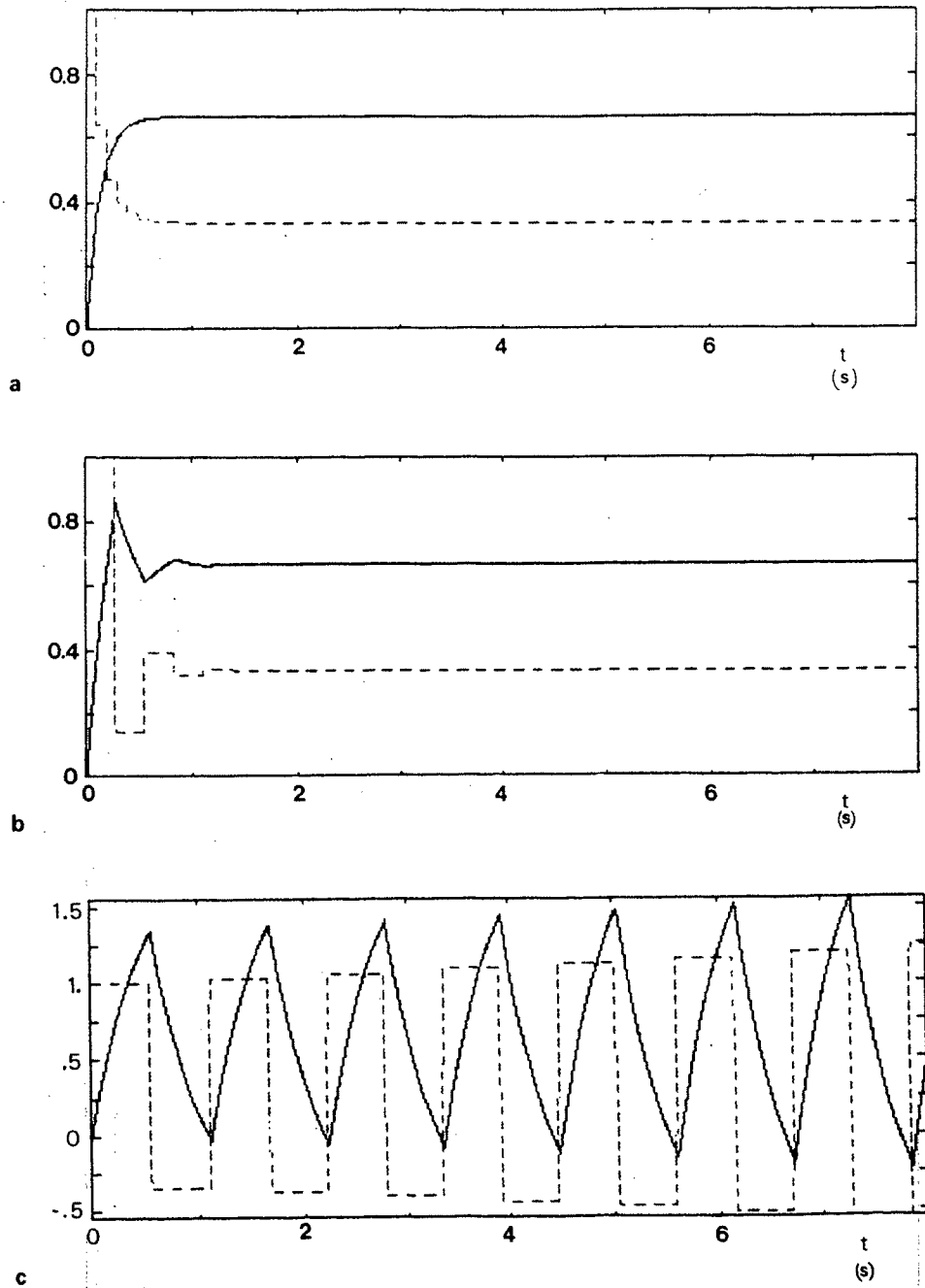


Figure 12: Responses {output (continuous line) and error (dashed line)} of digital control system for 3 values of controller sampling interval ($a=0.1s$, $b=0.28s$, $c=0.56s$).

```

DYNAMIC
  I=I+1
  IF(I)15,25,15
  IF(I-MULT)20,25,25
  E=R-Y
  AIN=E
  OUT=OUT1+AIN-0.5*AIN1
  OUT1=OUT
  AIN1=AIN
  I=0
  TYPE T,V,E,Y

```

*****Start of Derivative Section*****

```

DERIVATIVE
  DERIV=-(Y-AK*OUT)/TAW
  Y=INTEG(DERIV,0.0)
DERIVATIVE END

```

*****End of Derivative Section*****

Test for end of simulation run

```

IF(T-TMAX)10,10,12
DYNAMIC END

```

*****End of Dynamic Segment*****

*****Terminal Segment*****

```

STOP
END

```

Figure 13: Part of SLIM program DIGCON1.SLI for simulation of digital control system.

```

C *****Start of Dynamic Segment*****
C
C      DYNAMIC
C      I=I+1
C      IF(I)15,25,15
15      IF(I-MULT)20,25,25
25      AIN=R-Y
C      OUT=AIN
C      I=0
20      TYPE T,R,AIN,Y
C
C *****Start of Derivative Section*****
C
C      DERIVATIVE
C      DERIV=-(Y-AK*OUT)/TAW
C      Y=INTEG(DERIV,0.0)
C      DERIVATIVE END
C
C *****End of Derivative Section*****
C
C      Test for end of simulation run
C
C      IF(T-TMAX)10,10,12
10      DYNAMIC END
C
C *****End of Dynamic Segment*****
C
C *****Terminal Segment*****
C
12      STOP
C      END

```

Figure 11: Part of SLIM program DIGCON.SLI for simulation of digital control system.

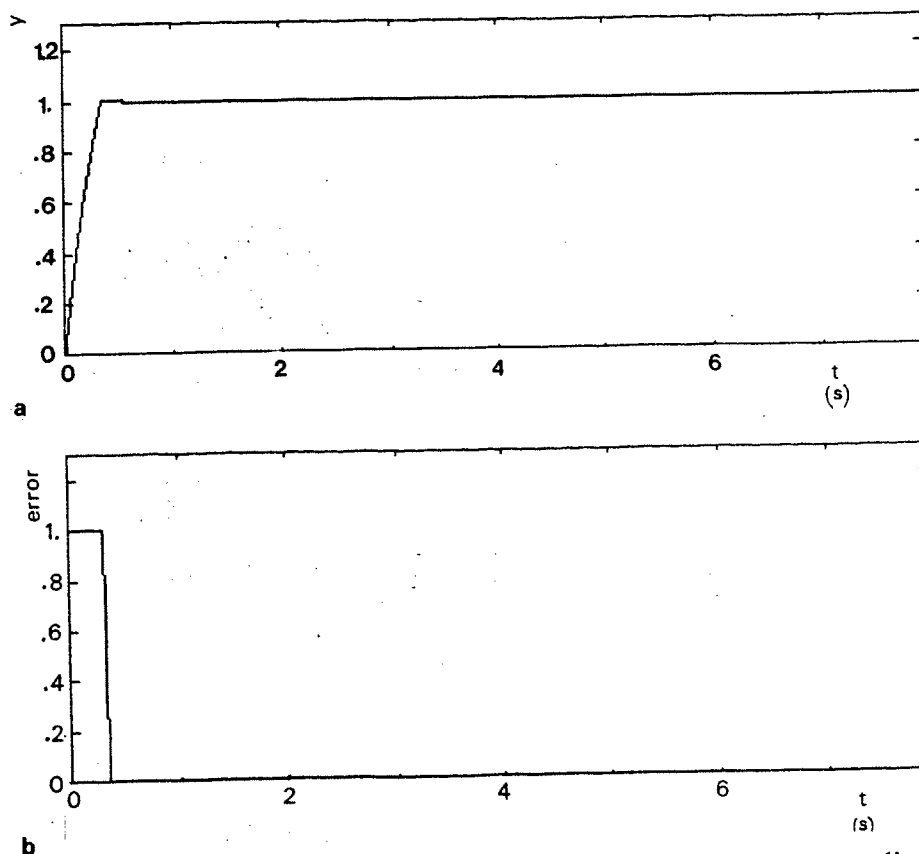


Figure 14: Responses from system with dead-beat control applied for step change of reference

```
DYNAMIC
  I=I+1
  IF(I)15,25,15
  IF(I-MULT)20,25,25
  E=R-Y
  AIN=E
  OUT=OUT1+AIN-0.5*AIN1
  OUT1=OUT
  AIN1=AIN
  I=0
  TYPE T,V,E,Y
```

*****Start of Derivative Section*****

```
DERIVATIVE
  DERIV=-(Y-AK*OUT)/TAW
  Y=INTEG(DERIV,0.0)
DERIVATIVE END
```

*****End of Derivative Section*****

Test for end of simulation run

```
IF(T-TMAX)10,10,12
DYNAMIC END
```

*****End of Dynamic Segment*****

*****Terminal Segment*****

```
STOP
END
```

Figure 13: Part of SLIM program DIGCON1.SLI for simulation of digital control system.

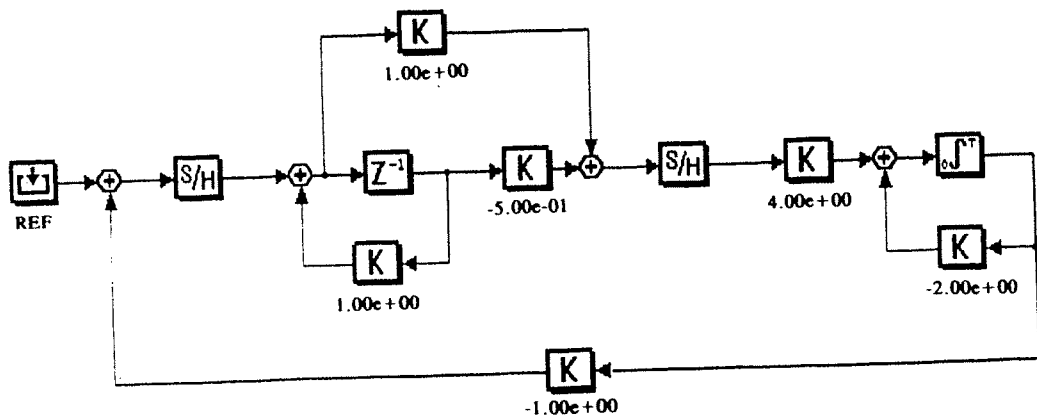


Figure 15: XANALOG block diagram for digital control system.

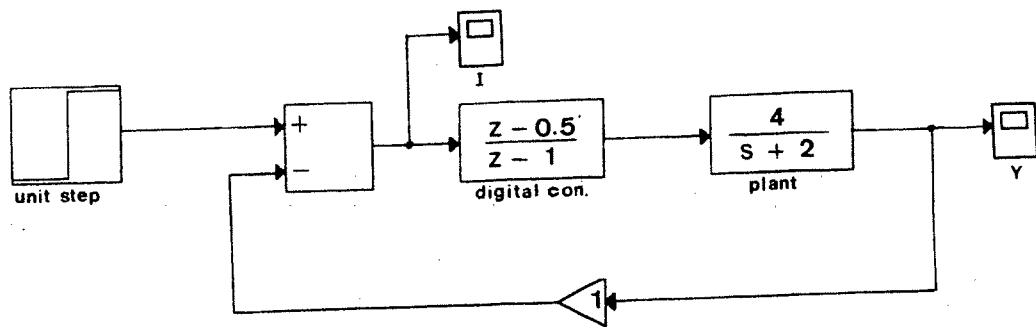
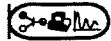


Figure 16: SIMULINK diagram for digital control system.

APPENDIX

Listings of SLIM Programs for Examples.



```

C
C Simulation of hydro-turbine speed control system with temporary
C droop type of governor. No backlash included in turbine model.
C
C *****Initial Segment*****
C
C First three lines of the program establish the number of
C channels, the number of samples and the time interval
C for outputs to the results file
      NCH=4
      NSA=400
      SAM=0.25
C
C Parameter values for coefficients in the system equations
C
      TW=1.1
      TS=0.2
      TIA=7.0
      TX=16.0
      TY=0.3
      SIG=0.03
      AMU=0.25
C
C Data for reference and disturbance inputs
C
      REF=0.1
      TL=0.0
C
C Set communication interval and values for integration
C parameters
C
      CINTERVAL(SAM)
      MINTERVAL(1.0E-6)
      MERROR(1.0E-6)
      XERROR(1.0E-6)
C
C Set initial conditions
C
      T=0.0
      T10=0.0
      V0=0.0
      Y10=0.0
      DY10=0.0
      ANSO=0.0
C
C Define the maximum value of the independent variable (time t)
C
      TMAX=100.0
C
C Information about file organisation written to the results
C file
C
      TYPE NCH
      TYPE NSA
      TYPE SAM
C
C Information about initial values of variables written to
C results file
C
      TYPE T,REF,TL,ANSO

```

```

C
C *****Start of Dynamic Segment*****
C
C   DYNAMIC
C
C *****Start of Derivative Segment*****
C
C   DERIVATIVE
C     DT1=(V-T1)*(2.0/TW)
C     T1=INTEG(DT1,T10)
C     T0=T1-TW*DT1
C     TA=T0-TL
C     DNS=TA/TIA
C     ANS=INTEG(DNS,ANS0)
C     E1=REF-ANS
C     DDY1=(E1-SIG*Y1-((SIG+AMU)*TX+TY)*DY1)/(TX*TY)
C     DY1=INTEG(DDY1,DDY10)
C     Y1=INTEG(DY1,Y10)
C     Y=Y1+TX*DY1
C     DV=(Y-V)/TS
C     V=INTEG(DV,V0)
C   DERIVATIVE END
C
C *****End of Derivative Section*****
C
C   Values of t, ref, t1 and ans for current communication
C   interval output to results file
C
C     TYPE T,REF,TL,ANS
C
C   Test for end of simulation run
C
C     IF(T-TMAX)10,10,12
10   DYNAMIC END
C
C *****End of Dynamic Segment*****
C
C *****Terminal Segment*****
12   STOP
C     END

```

```

C
C Simulation of hydro-turbine speed control system with temporary
C droop type of governor. No backlash included in turbine model.
C Typical data from file TURB1.IN.
C
C *****Initial Segment*****
C
C First three lines of the program establish the number of
C channels, the number of samples and the time interval
C for outputs to the results file
C
      NCH=4
      NSA=400
      SAM=0.25
C
C Parameter values for coefficients in the system equations
C
      ACCEPT TW
      ACCEPT TS
      ACCEPT TIA
      ACCEPT TX
      ACCEPT TY
      ACCEPT SIG
      ACCEPT AMU
C
C Data for reference and disturbance inputs
C
      ACCEPT REF
      ACCEPT TL
C
C Set communication interval and values for integration
C parameters
C
      CINTERVAL(SAM)
      MINTERVAL(1.0E-6)
      MERROR(1.0E-6)
      XERROR(1.0E-6)
C
C Set initial conditions
C
      T=0.0
      T10=0.0
      V0=0.0
      Y10=0.0
      DY10=0.0
      ANS0=0.0
C
C Define the maximum value of the independent variable (time t)
C
      TMAX=100.0
C
C Information about file organisation written to the results
C file
C
      TYPE NCH
      TYPE NSA
      TYPE SAM
C
C Information about initial values of variables written to
C results file

```

```

C      TYPE T,REF,TL,ANS0
C
C      *****Start of Dynamic Segment*****
C
C      DYNAMIC
C
C      *****Start of Derivative Segment*****
C
C      DERIVATIVE
C      DT1=(V-T1)*(2.0/TW)
C      T1=INTEG(DT1,T10)
C      T0=T1-TW*DT1
C      TA=T0-TL
C      DNS=TA/TIA
C      ANS=INTEG(DNS,ANS0)
C      E1=REF-ANS
C      DDY1=(E1-SIG*Y1-((SIG+AMU)*TX+TY)*DY1)/(TX*TY)
C      DY1=INTEG(DDY1,DDY10)
C      Y1=INTEG(DY1,Y10)
C      Y=Y1+TX*DY1
C      DV=(Y-V)/TS
C      V=INTEG(DV,V0)
C      DERIVATIVE END
C
C      *****End of Derivative Section*****
C
C      Values of t, ref, tl and ans for current communication
C      interval output to results file
C
C      TYPE T,REF,TL,ANS
C
C      Test for end of simulation run
C
C      IF(T-TMAX)10,10,12
10      DYNAMIC END
C
C      *****End of Dynamic Segment*****
C
C      *****Terminal Segment*****
12      STOP
      END

```

```

C
C Simulation of hydro-turbine speed control system with
C temporary droop type of governor. Backlash included
C in turbine model. Input data from file TURB2.IN.
C
C *****Initial Segment*****
C
C First three lines of the program establish the number of
C channels, the number of samples and the time interval
C for outputs to the results file
C     NCH=5
C     NSA=400
C     SAM=0.25
C
C Input parameter values for coefficients in the system equations
C (from data file)
C
C     ACCEPT TW,TS,TIA
C     ACCEPT AB
C     ACCEPT TX,TY
C     ACCEPT SIG,AMU
C
C Input data for reference and disturbance inputs
C
C     ACCEPT REF,TL
C
C Set communication interval and values for integration
C parameters
C
C     CINTERVAL(SAM)
C     MINTERVAL(1.0E-5)
C     MERROR(1.0E-4)
C     XERROR(1.0E-4)
C
C Set initial conditions
C
C     T=0.0
C     T10=0.0
C     V0=0.0
C     Y10=0.0
C     DY10=0.0
C     YB0=0.0
C     ANS0=0.0
C
C Define the maximum value of the independent variable (time t)
C
C     TMAX=100.0
C
C Information about file organisation written to the results
C file
C
C     TYPE NCH
C     TYPE NSA
C     TYPE SAM
C
C Information about initial values of variables written to
C results file
C
C     TYPE T,REF,TL,ANS0,YB0
C

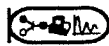
```

COMETT - COURSE "FUZZY SYSTEMS AND CONTROL"

```

C *****Start of Dynamic Segment*****
C
C      DYNAMIC
C
C *****Start of Derivative Segment*****
C
C      DERIVATIVE
C
C      Section for representation of backlash element
C
C          IF(AB-0.001)40,45,45
45      DB=V-YB
C          IF(DB+AB)60,50,50
60      FB=DB
C          GOTO 100
C          IF(DB-AB)80,70,70
70      FB=DB
C          GOTO 100
80      FB=0.0
100     YB=INTEG(10.0*FB,YB0)
C          GOTO 110
40      YB=V
C
C      End of section for backlash
C
110     DT1=(YB-T1)*(2.0/TW)
C          T1=INTEG(DT1,T10)
C          T0=T1-TW*DT1
C          TA=T0-TL
C          DNS=TA/TIA
C          ANS=INTEG(DNS,ANS0)
C          E1=REF-ANS
C          DDY1=(E1-SIG*Y1-((SIG+AMU)*TX+TY)*DY1)/(TX*TY)
C          DY1=INTEG(DDY1,DDY10)
C          Y1=INTEG(DY1,Y10)
C          Y=Y1+TX*DY1
C          DV=(Y-V)/TS
C          V=INTEG(DV,V0)
C      DERIVATIVE END
C
C *****End of Derivative Section*****
C
C      Values of t, ref, tl, ans and yb for current communication
C      interval output to results file
C
C          TYPE T,REF,TL,ANS,YB
C
C      Test for end of simulation run
C
C          IF(T-TMAX)10,10,12
10      DYNAMIC END
C
C *****End of Dynamic Segment*****
C
C *****Terminal Segment*****
12      STOP
C          END

```



1.1
0.2
7.0
16.0
0.3
0.03
0.25
0.1
0.0

Data file TURB2.IN

1.1
0.2
7.0
0.02
16.0
0.3
0.03
0.25
0.1
0.0

Simulation of a simple digital control system.
Input data from file DIGCON.IN.

*****Initial Segment*****

First three lines of the program establish the number
of channels, the number of samples and the time interval
for outputs to the results file

NCH=4
NSA=400
ACCEPT SAM

Input parameter value for coefficient in the system equations
(from data file)

ACCEPT AK
ACCEPT TAW

Read in value of reference

ACCEPT R

Set initial value of the independent variable (time t)

T=0.0
AIN=0.0
I=-1
Y=0.0
OUT=0.0

Define the maximum value of the independent variable (time t)

ACCEPT TMAX

Set interval for output of variables to output file and
read in integer value representing the multiple
of this interval for sampling interval for digital control.
Also set values for integration parameters

CINTERVAL(SAM)
ACCEPT MULT
MINTERVAL(1.0E-6)
MERROR(1.0E-4)
XERROR(1.0E-4)

Information about file organisation written to the results
file

TYPE NCH
TYPE NSA
TYPE SAM

*****Start of Dynamic Segment*****

DYNAMIC
I=I+1
IF(I)15,25,15

```

15  IF(I-MULT)20,25,25
25  AIN=R-Y
    OUT=AIN
    I=0
20  TYPE T,R,AIN,Y
C
C *****Start of Derivative Section*****
C
    DERIVATIVE
        DERIV=-(Y-AK*OUT)/TAW
        Y=INTEG(DERIV,0.0)
    DERIVATIVE END
C
C *****End of Derivative Section*****
C
C  Test for end of simulation run
C
    IF(T-TMAX)10,10,12
10  DYNAMIC END
C
C *****End of Dynamic Segment*****
C
C *****Terminal Segment*****
C
12  STOP
    END

```

Listing of SLIM Program DIGCON1.SLI

COMETT - COURSE "FUZZY SYSTEMS AND CONTROL"

```
C
C Simulation of a simple digital control system with "dead-
C beat" compensation.
C Input data from file DIGCON1.IN.
C
C *****Initial Segment*****
C
C First three lines of the program establish the number
C of channels, the number of samples and the time interval
C for outputs to the results file
C
      NCH=4
      NSA=230
      ACCEPT SAM
C
C Input parameter value for coefficient in the system equations
C (from data file)
C
      ACCEPT AK
      ACCEPT TAW
C
C Read in value of reference
C
      ACCEPT R
C
C Set initial value of the independent variable (time t)
C
      T=0.0
      E=R
      I=-1
      OUT=0.0
      AIN1=0.0
      OUT1=0.0
      Y=0.0
      V=1.3
C
C Define the maximum value of the independent variable (time t)
C
      ACCEPT TMAX
C
C Set interval for output of variables to ouput file and
C read in integer value representing the multiple
C of this interval for sampling interval for digital control.
C Also set values for integration parameters
C
      CINTERVAL(SAM)
      ACCEPT MULT
      MINTERVAL(1.0E-6)
      MERROR(1.0E-4)
      XERROR(1.0E-4)
C
C Information about file organisation written to the results
C file
C
      TYPE NCH
      TYPE NSA
      TYPE SAM
C
C *****Start of Dynamic Segment*****
```

```

C
DYNAMIC
  I=I+1
  IF(I)15,25,15
15  IF(I-MULT)20,25,25
25  E=R-Y
    AIN=E
    OUT=OUT1+AIN-0.5*AIN1
    OUT1=OUT
    AIN1=AIN
    I=0
20  TYPE T,V,E,Y
C
C *****Start of Derivative Section*****
C
  DERIVATIVE
    DERIV=-(Y-AK*OUT)/TAW
    Y=INTEG(DERIV,0.0)
  DERIVATIVE END
C
C *****End of Derivative Section*****
C
C Test for end of simulation run
C
  IF(T-TMAX)10,10,12
10  DYNAMIC END
C
C *****End of Dynamic Segment*****
C
C *****Terminal Segment*****
C
12  STOP.
    END

```

Data file DIGCON.IN

COMETT - COURSE "FUZZY SYSTEMS AND CONTROL"

0.02
2.0
0.5
1.0
8.0
14

Data file DIGCON1.IN

0.0347
2.0
0.5
1.0
8.0
10

Internal Verification and External Validation of Simulation Models for Control Systems Analysis and Design

Topics considered include the following:

a) Terminology and definitions concerning simulation model verification validation.

Useful sources of information include a set of guidelines published in 1979 by the Technical Committee on Model Credibility of the Society for Computer Simulation [1] and various textbooks (e.g. [2-4]). An important distinction may be made between *internal verification* and *external validation* [2,5].

b) Internal Verification

The processes of internal verification involve checks for:

i) Internal consistency

and ii) Algorithmic validity

Appropriate methods involve *static checks* and *dynamic checks*.

c) External Validation

Criteria for external validation involve assessment of accuracy and suitability of a model for the intended application. They may include [6]:

Theoretical validity (consistent with accepted theories)

Empirical validity (adequate agreement with real system)

Pragmatic validity (satisfies application's requirements)

Heuristic validity (basis for explanation of system)

Empirical validation involves comparisons between behaviour of the model and behaviour of the real system. This can involve a number of methods including:

(i) Methods based on comparisons of response data (e.g. [4],[7])

(ii) Methods based on system identification methods (e.g. [7-12])

(iii) Methods based on sensitivity analysis (e.g. [4],[7])

(iv) Methods based on inverse models (e.g. [10])

Of the above four approaches to empirical validation methods (i) and (ii) are the most widely

used at present.

d) Robustness issues in external validation

This is especially important when system identification techniques are used as part of the external validation process. One must have confidence in the accuracy and reliability of the tools being used. These issues are explored in detail elsewhere (e.g. [13])

e) Possible outcomes of the external validation process

At least three possible outcomes arise. These are:

1. The measured data sets from the real system cannot be explained by any model structure and parameter set considered. The model structure and assumptions must be reviewed.
2. One or more models gives a satisfactory match to system response data but the uncertainty level for some parameters is high. The model may not be of much predictive value.
3. Satisfactory agreement is obtained with experimental test results and model parameter values are plausible. The model may be used for the intended application until new evidence falsifies the model in some way.

f) Documentation of the validation process

Model documentation should include the following:

1. A clear statement of the purpose of the model.
2. Descriptions of the model in conceptual and mathematical terms, including basic assumptions.
3. A statement concerning the range of conditions for which the model has been tested and the level of agreement throughout that range.
4. A description of all tests used for internal verification and external validation with relevant results.

References

- [1] S.C.S. Technical Committee on Model Credibility, 'Terminology for model credibility', Simulation, Vol. 32, pp. 103-4, 1979.
- [2] Shannon, R.E., "System Simulation. The Art and the Science", Prentice-Hall, Englewood Cliffs, N.J., U.S.A., 1975.
- [3] Spriet, J.A. and Vansteenkiste, G.C. "Computer-Aided Modelling and Simulation", Academic Press, London, U.K., 1982.
- [4] Murray-Smith, D.J. "Continuous System Simulation", Chapman & Hall, London, 1995.

- [5] Murray-Smith, D.J., 'A review of methods for the validation of continuous system simulation models', in Nock, K.G. (editor), "Proceedings 1990 UKSC Conference on Computer Simulation", pp. 108-111, United Kingdom Simulation Council, Burgess Hill, 1990.
- [6] Murray-Smith, D.J. and Carson, E.R., 'The modelling process in respiratory medicine', in Cramp, D.G. and Carson, E.R. (editors), "The Respiratory System", pp. 296-333, Croom Helm, London, 1988.
- [7] Murray-Smith, D.J. Advances in simulation model validation: theory, software and applications, in Proceedings EUROSIM'95 Congress, Wien, September 1995.
- [8] Unbehauen, H. and Rao, G.P., "Identification of Continuous Systems", North-Holland, Amsterdam, The Netherlands, 1987.
- [9] Beck, J.V. and Arnold, K.J., "Parameter Estimation in Engineering and Science", John Wiley & Sons, New York, U.S.A., 1977.
- [10] Bradley, R., Padfield, G.D., Murray-Smith, D.J. and Thomson, D.G., 'Validation of Helicopter Mathematical Models', Transactions of Institute of Measurement and Control, Vol. 12, pp. 186-196, 1990.
- [11] AGARD Advisory Report No. 280, "Rotorcraft System Identification", AGARD, Neuilly sur Seine, France, 1991.
- [12] Sinha, N.K. and Kusztá, B., "Modeling and Identification of Dynamic Systems", Van Nostrand Reinhold Company, New York, U.S.A., 1983.
- [13] Murray-Smith, D.J., 'Modelling and robustness issues in rotorcraft system identification', AGARD Lecture Series No. 178, "Rotorcraft System Identification", AGARD, Neuilly sur Seine, France, 1991.

INTERNAL VERIFICATION

Assessment of consistency and accuracy of a simulation model compared with the underlying mathematical model in terms of :-

- a) logical, mathematical and conceptual features.**
- b) algorithmic correctness (e.g. appropriate numerical methods for intended application).**

EXTERNAL VALIDATION

Assessment of the mathematical model and its suitability for the intended application.

Must distinguish between:-

a) Functional Validation

and b) Physical or Theoretical Validation

EXTERNAL VALIDATION AS PART OF MODEL DEVELOPMENT PROCESS:

components → sub-models → complete model

Both functional and physical/theoretical validation processes involve elements of empirical validation

i.e. assessment of level of agreement between model variables and corresponding data from the real system.

EMPIRICAL VALIDATION

Complexity of instrumentation

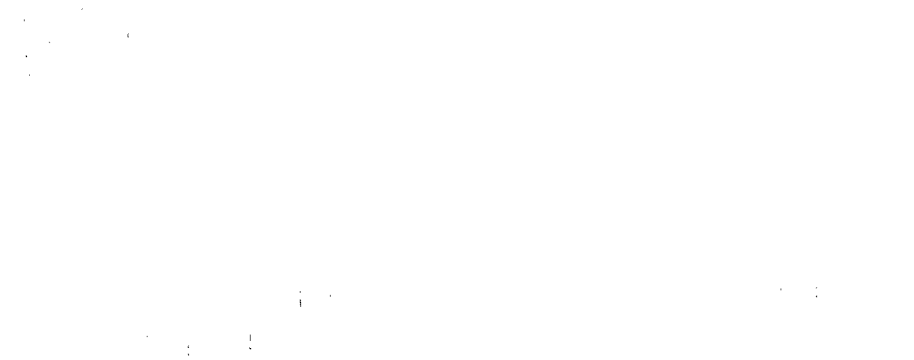
$$\propto \frac{1}{\text{amount of a priori knowledge}}$$

METHODS BASED ON COMPARISONS OF RESPONSE DATA

- a) Overlaying of plots for model variables and corresponding system variables.

METHODS BASED ON COMPARISONS OF RESPONSE DATA

- b) Plots of differences between model and system responses.



THEIL'S INEQUALITY COEFFICIENT (TIC)

$$\text{TIC} = \frac{\sqrt{\sum_{i=1}^n (y_i - z_i)^2}}{\sqrt{\sum_{i=1}^n y_i^2} + \sqrt{\sum_{i=1}^n z_i^2}}$$

$$0 \leq \text{TIC} \leq 1$$

TIC ≤ 0 : time series almost identical

TIC ≤ 1 : time series differ significantly

FITNESS FUNCTION APPROACH

e.g.
$$f(y) = \frac{1}{1 + e(y)}$$

where
$$e(y) = \frac{1}{n} \sum_{i=1}^n (y_i - z_i)^2$$

$$0 \leq f(y) \leq 1$$

$f(y) \leq 1$: time series almost identical

$f(y) \leq 0$: time series differ significantly

METHODS BASED ON COMPARISONS OF RESPONSE DATA

c) Involving numerical measures of goodness-of-fit

$$\text{e.g. } J = \sum_{i=1}^n (y_i - z_i)^T w_i (y_i - z_i)$$

y_i = measured response at sample i

z_i = model response at sample i

w_i = weighting function

METHODS BASED ON COMPARISONS OF RESPONSE DATA

d) STATISTICAL METHODS

e.g. Histograms
Frequency distributions } of complete data set

Fitting of stochastic models

METHODS BASED ON COMPARISONS OF RESPONSE DATA

MODEL DISTORTION APPROACH

(Butterfield and Thomas)

**Assessment in terms of PARAMETER
DISTORTION needed to ensure that model achieves
exact fit.**

**Compare parameter changes with uncertainty levels
associated with the nominal parameter set.**

METHODS BASED ON SYSTEM IDENTIFICATION TECHNIQUES

- a) **Identifiability analysis**
- b) **Estimation of parameters in identifiable models**
- c) **Cross-validation**

METHODS BASED ON SENSITIVITY ANALYSIS (Tomović et al.)

- a) Can help establish dependence of each part of response on each of the parameters of the model.**
- b) Can use to investigate whether parameter changes alone sufficient to provide required match of model and system.**
- c) Can suggest possible presence of fundamental structural errors in the model.**

METHODS BASED ON INVERSE MODELS

Steps:

- a) **Create inverse model**
- b) **Drive inverse model using measured response data from the real system.**
- c) **Compare input time series predicted by inverse model with input time series recorded from the real system.**

Advantages

Avoids effects of small offset and bias components at system input.

Differences may be more apparent at the system input.

MODEL DOCUMENTATION

Should include:-

- a) Full details of purpose of model and the intended application.**
- b) Assumptions within the model.**
- c) Simplifications and approximations introduced.**
- d) Details of tests applied for internal verification.**
- e) Details of external validation processes.**

EXTERNAL VALIDATION viewed as a form of MODEL CALIBRATION PROCESS:-

**Establishing useful range of model expressed in
terms of**

a) amplitude

b) frequency

**i.e. range over which variables of model match
corresponding quantities of real system to a
stated level of accuracy.**

Case Study I - Plant Modelling for a Two-Tank Liquid Level Control System

1. Introduction

For a vessel holding a mass of fluid M , the rate of change of mass in the container must equal the total mass inflow rate (Q_i) minus the total mass outflow rate (Q_o). That is

$$\frac{dM}{dt} = Q_i - Q_o \quad (1)$$

The mass of fluid, M , is related to the volume of fluid in the vessel, V , by the equation

$$M = \rho V \quad (2)$$

where ρ is the fluid density. For an incompressible fluid ρ is constant and thus

$$\dot{M} = \rho \dot{V} \quad (3)$$

Figure 1 shows a vessel of rectangular cross section, with surface area A , it is possible to relate the mass of liquid, M , to the liquid height, H , through an equation

$$M = \rho AH \quad (4)$$

The hydrostatic pressure at the base of the vessel is then

$$P = \rho gH \quad (5)$$

where g is the gravitational constant. For the system of Figure 1, if the pressure at the surface of the liquid and at the outlet are the same and equal to P_a (say, atmospheric pressure) the pressure difference between the tank base and the outlet is given by $(P_a + P) - P_a$ which is simply P . The output flow rate Q_o is dependent on P and, for the case of laminar flow, is conventionally described by an equation of the form

$$Q_o = \frac{P}{R} \quad (6)$$

where R is the fluid resistance.

Assume now that $Q_i(t)$ is known and that we want to predict the system behaviour in terms of the liquid height $H(t)$. From Equations (1), (4), (5) and (6) we may write

$$\rho A \frac{dh}{dt} = Q_i - \frac{\rho g H}{R} \quad (7)$$

so that

$$A \frac{dH}{dt} = \frac{Q_i}{\rho} - \frac{gH}{R} \quad (8)$$

Note that Q_i is the mass flow rate and thus Q_i/ρ is the volume flow rate, (Q_{vi}), so that

$$A \frac{dH}{dt} = Q_{vi} - \frac{gH}{R} \quad (9)$$

Note that the relationship describing the flow at the outlet of the vessel is not always that shown in Equation (6) and the form of expression which is appropriate depends upon the nature of the outlet. For example, if the outlet is simply a hole in the side wall of the tank a condition known as **orifice flow** occurs. In this case, if the size of the orifice is small, and the pressure variation over the orifice area is thus negligible compared with the average orifice pressure, it can be shown (from the principle of conservation of energy) that the mass flow rate through the orifice is given by

$$Q_o = C_d a_o \sqrt{\frac{2P}{\rho}} = C_d a_o \sqrt{2gH} \quad (10)$$

where a_o is the orifice area and C_d is the discharge coefficient. If, on the other hand, the outlet is through a pipe with turbulent flow, the appropriate relationship is

$$Q_o = \sqrt{\frac{P}{R_T}} \quad (11)$$

where R_T is a constant. Practical hydraulic components, such as valves, can be described by Equation (6) for small pressure drops but have to be described by Equation (11) in many cases due to turbulence at typical operating conditions.

2. Modelling of a pair of interconnected tanks

When a hydraulic system incorporates more than one liquid storage vessel the principle of conservation of mass (Equation (1)) may be applied to each element in turn. However, there is coupling between the vessels and the nature of this coupling depends upon the precise configuration of the vessels and upon the operating conditions. The interconnected tanks being modelled in this chapter are bench-top systems intended for use in teaching the principles of automatic control engineering [1].

Figure 2 is a schematic diagram of the system being considered. It consists of a

container of volume 6 litres having a centre partition which divides the container into two separate tanks. Coupling between the tanks is provided by a number of holes of various diameters near the base of the partition and the extent of the coupling may be adjusted through the insertion of plugs into one or more of these holes. The system is equipped with a drain tap, under manual control, and the flow rate from one of the tanks can be adjusted through this. The other tank has an inflow provided by a variable speed pump which is electrically driven. Both tanks are equipped with sensors which measure the pressure at the base of each tank and thus provide an electrical output voltage proportional to the liquid level.

2.1 A nonlinear mathematical model

Following the approach used in Section 1 the equation describing tank 1 in Figure 2 has the form

$$A_1 \frac{dH_1}{dt} = Q_{vi} - Q_{v1} \quad (12)$$

where H_1 is the height of liquid in tank 1, Q_{vi} is the input volume flow rate and Q_{v1} is the volume flow rate from tank 1 to tank 2 and A_1 is the cross-sectional area. Similarly for tank 2 we can write

$$A_2 \frac{dH_2}{dt} = Q_{v1} - Q_{vo} \quad (13)$$

where H_2 is the height of liquid in tank 2 and Q_{vo} is the flow rate of liquid out of tank 2. Considering the holes connecting the two tanks and the drain tap all as simple orifices allows the flow rates to be related to the liquid heights by the following two equations

$$Q_{v1} = C_{d1} a_1 \sqrt{2g(H_1 - H_2)} \quad (14)$$

and

$$Q_{vo} = C_{d2} a_2 \sqrt{2g(H_2 - H_3)} \quad (15)$$

where a_1 is the cross sectional area of the orifice between the two tanks, a_2 is the cross-sectional area of the orifice representing the drain tap, H_3 is the height of the drain tap above the base of the tank and g is the gravitational constant.

2.2 Linearisation of the model

For control system design studies it is appropriate to consider a linearised model in which the model variables represent small variations about steady state values. Thus the input flow variable is q_{vi} representing small variations about a steady flow rate Q_{vi} . Similarly the other variables represent small variations about steady values q_{v1} in Q_{v1} , q_{vo} in Q_{vo} , h_1 in H_1 and h_2 in H_2 . In the steady state

$$Q_{vi} = Q_{v1} = Q_{vo} \quad (16)$$

$$q_{vi} - q_{vi} = A_1 \frac{dh_1}{dt} \quad (17)$$

$$q_{vi} - q_{vo} = A_2 \frac{dh_2}{dt} \quad (18)$$

From Equation (14) it is clear that Q_{vi} is a function of both H_1 and H_2 . Hence the small variation in flow, q_{vi} , must depend on the steady levels H_1 and H_2 about which the system is operating. In general, one may therefore write

$$q_{vi} = \frac{\partial Q_{vi}}{\partial H_1} h_1 + \frac{\partial Q_{vi}}{\partial H_2} h_2 \quad (19)$$

Differentiating Equation (14) partially with respect to H_1 and H_2 in turn gives

$$q_{vi} = \frac{C_{d1} a_1 \sqrt{2g}}{2\sqrt{H_1 - H_2}} (h_1 - h_2) \quad (20)$$

Similarly

$$q_{vo} = \frac{\partial Q_{vo}}{\partial H_2} h_2 = \frac{C_{d2} a_2 \sqrt{2g}}{2\sqrt{H_2 - H_3}} h_2 \quad (21)$$

Substituting for q_{vi} and q_{vo} in Equations (17) and (18) gives

$$A_1 \frac{dh_1}{dt} = q_{vi} - \alpha_1 (h_1 - h_2) \quad (22)$$

$$A_2 \frac{dh_2}{dt} = \alpha_1 (h_1 - h_2) - \alpha_2 h_2 \quad (23)$$

where

$$\alpha_1 = \frac{C_{d1} a_1 \sqrt{2g}}{2\sqrt{H_1 - H_2}} \quad (24)$$

and

$$\alpha_2 = \frac{C_{d2} a_2 \sqrt{2g}}{2\sqrt{H_2 - H_3}} \quad (25)$$

Reorganisation of Equations (22) and (23) gives a second order state-space model as follows

$$\begin{bmatrix} \dot{h}_1 \\ \dot{h}_2 \end{bmatrix} = \begin{bmatrix} -\frac{\alpha_1}{A_1} & \frac{\alpha_1}{A_1} \\ \frac{\alpha_1}{A_2} & -\frac{(\alpha_1 + \alpha_2)}{A_2} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} + \begin{bmatrix} \frac{1}{A_1} \\ 0 \end{bmatrix} q_{vi} \quad (26)$$

Taking Laplace transforms it is possible to obtain the transfer function descriptions relating the depth h_1 and the depth h_2 to the input flow rate q_{vi} . These are as follows:

$$h_2(s) = \frac{\frac{1}{\alpha_2}}{1 + \frac{(A_1\alpha_1 + A_2\alpha_1 + A_2\alpha_2)}{\alpha_1\alpha_2}s + \frac{A_1A_2}{\alpha_1\alpha_2}s^2} q_{vi}(s) \quad (27)$$

and

$$h_1(s) = \frac{\frac{(\alpha_1 + \alpha_2)}{\alpha_1\alpha_2} \left(1 + s \frac{A_2}{\alpha_1 + \alpha_2} \right)}{1 + \frac{(A_1\alpha_1 + A_2\alpha_1 + A_2\alpha_2)}{\alpha_1\alpha_2}s + \frac{A_1A_2}{\alpha_1\alpha_2}s^2} q_{vi}(s) \quad (28)$$

These both involve a pair of simple real poles and the characteristic equation in both cases is

$$(1 + sT_1)(1 + sT_2) = 1 + \frac{(A_1\alpha_1 + A_2\alpha_1 + A_2\alpha_2)}{\alpha_1\alpha_2}s + \frac{A_1A_2}{\alpha_1\alpha_2}s^2 = 0 \quad (29)$$

or

$$(1 + sT_1)(1 + sT_2) = 0 \quad (30)$$

where

$$T_1T_2 = \frac{A_1A_2}{\alpha_1\alpha_2} \quad (31)$$

and

$$T_1 + T_2 = \frac{A_1\alpha_1 + A_2\alpha_1 + A_2\alpha_2}{\alpha_1\alpha_2} \quad (32)$$

3. Programs for simulation of the nonlinear coupled tank system

A complete source program for simulation of the two-tanks system, using the SLIM simulation language and based on Equations (12) to (15), is included as a .SLI file (TANKS.SLI) in the Appendix to these notes. Nominal parameter values for this nonlinear model, corresponding to a real laboratory-scale coupled tank system, are as shown in Table 1. A data file (TANKS.IN) is provided for this nominal set of parameters. Figure 3 shows an XANALOG block diagram for this simulation model.

Some results based on the simulation program TANKS.SLI and the given parameter set are shown in Figure 4. In all cases the inflow was zero and the drain tap on the second tank was fully open.

3.1 Internal verification of the simulation program

The first stage of internal verification is concerned with checking that the structure of the simulation program is consistent with the mathematical model. This involves working backwards from the statements in the program, especially those within the derivative section, to ensure that when translated back to the form of differential equations they are the same as those of the original model. Checks should also be made of the parameter values used in the program or in the parameter input file to ensure that they correspond exactly to the parameter set of the model itself.

The second stage of internal verification is concerned with numerical accuracy. In the case of fixed step integration methods comparisons can be made of results obtained with a number of different sizes of integration step length and with different integration techniques. This provides the user with some understanding of the sensitivity of results to the step length and of the overall suitability of the numerical methods chosen. In the case of variable step integration algorithms tests can be carried out to compare results with different settings of the relative and absolute error limits and with different values of the minimum integration step to be allowed. In both cases some comparisons can be made using a number of different values of the communication interval to ensure that interesting events in the simulation model are not being hidden from the user simply because of an inappropriate choice of this parameter which determines the interval between output samples.

3.2 External validation of the simulation model

Statements about model validity must always be made in the context of an intended application. In the case of the coupled tank system the computer simulation is to be used as a basis for the design of an automatic control system for regulation of level in one of the tanks. There is particular interest therefore in the accuracy of the model in predicting steady state conditions and in predicting the form of small transients about any given steady operating point. Such comparisons are very easily made in the case of small-scale laboratory equipment of this kind and agreement between steady-state measurements and steady-state model predictions is generally quite good for most parts of the operating range. Table 2 shows some typical results obtained from measurements on the real system and corresponding tests on the simulation model. Differences between the steady-state liquid levels in the simulation model and in the real system, for a given value of input flow rate, are significant and vary slightly with operating point. Figures 5 shows discrepancies between the model and

system for a test involving a large change in operating conditions. Response data for small perturbation step tests carried out about one chosen operating point give time constants which are broadly in agreement with values determined from the linearised model in the form of Equations (27) and (28).

The discrepancies in the model exposed by the steady state tests and the large perturbation responses are believed to be due mainly to the limitations of Equations (14) and (15) in describing the relationships between output flow and the liquid level in each tank. These equations apply to an ideal simple orifice and the actual physical effects at the tank outlets do not agree exactly with this simplified model.

With closed-loop control added to the real system and to the simulation model the agreement can be shown to be significantly closer. This is important since the equipment is intended to be used for investigations of closed-loop control. In simulation studies involving control system design applications there is always particular interest in the overall robustness of the control system and the effect which modelling errors and uncertainties may have on the performance of the closed-loop system. Although control systems are normally designed using linearised models, simulation studies carried out on a proposed closed-loop system using a nonlinear model of the plant can often be highly illuminating. Such an investigation may reveal problems with the proposed design which would otherwise only come to light during the commissioning and testing of the real system .

4 Discussion

This case study provides an illustration of a relatively simple nonlinear system which can be modelled in a classical way using physical laws and principles. The simulation model is easily implemented using either equation-oriented or block-oriented tools. The relatively simple nature of the system and the variables which are accessible for measurement in the real hardware make this an interesting but straightforward system for the application of external validation methods.

Possibilities for using the simulation program TANKS.SLI as a basis for further investigations may be found elsewhere [2].

References

- [1] Wellstead, P.E., "Coupled Tanks Apparatus: Manual", TecQuipment International Ltd, Long Eaton, Nottingham, NG10 2AN, U.K., 1981.
- [2] Murray-Smith, D.J., "Continuous System Simulation", Chapman & Hall, London, 1995.

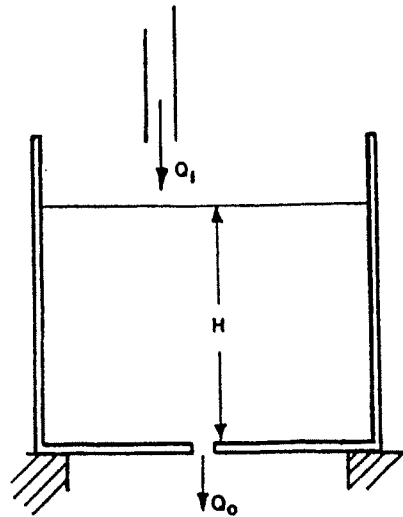


Figure 1: Simple tank of rectangular cross-section.

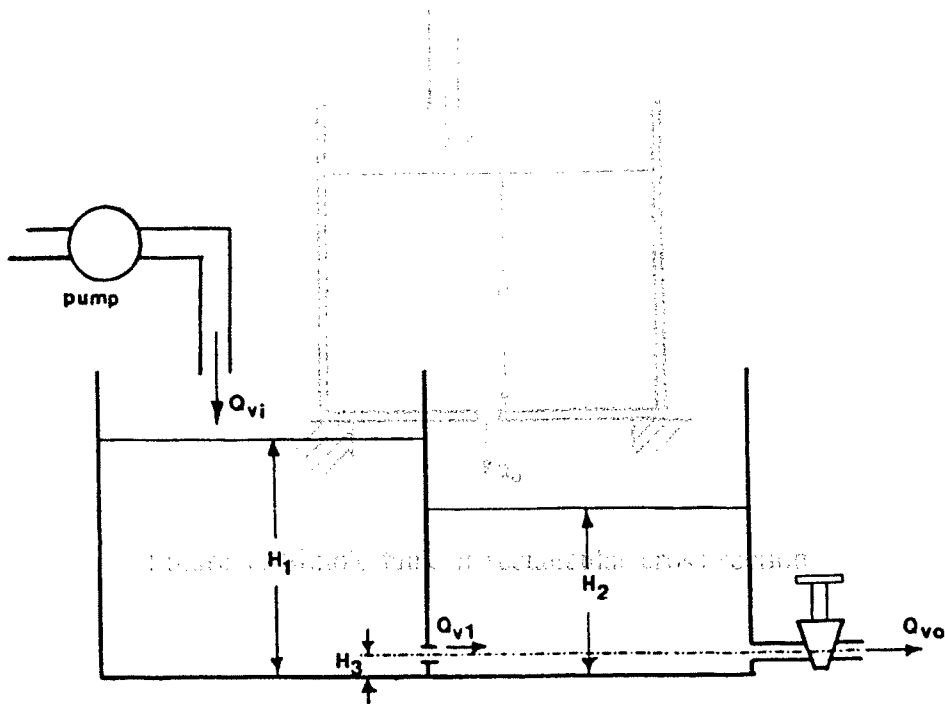
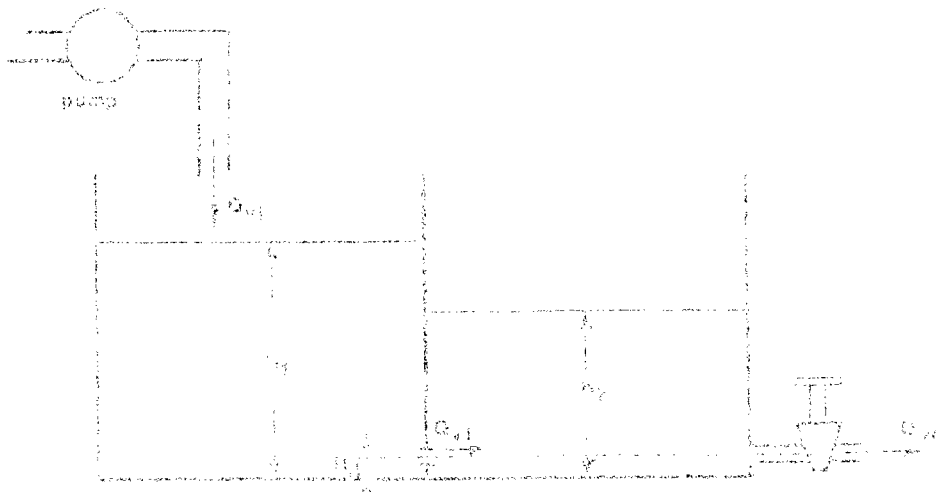


Figure 2: Pair of inter-connected tanks.



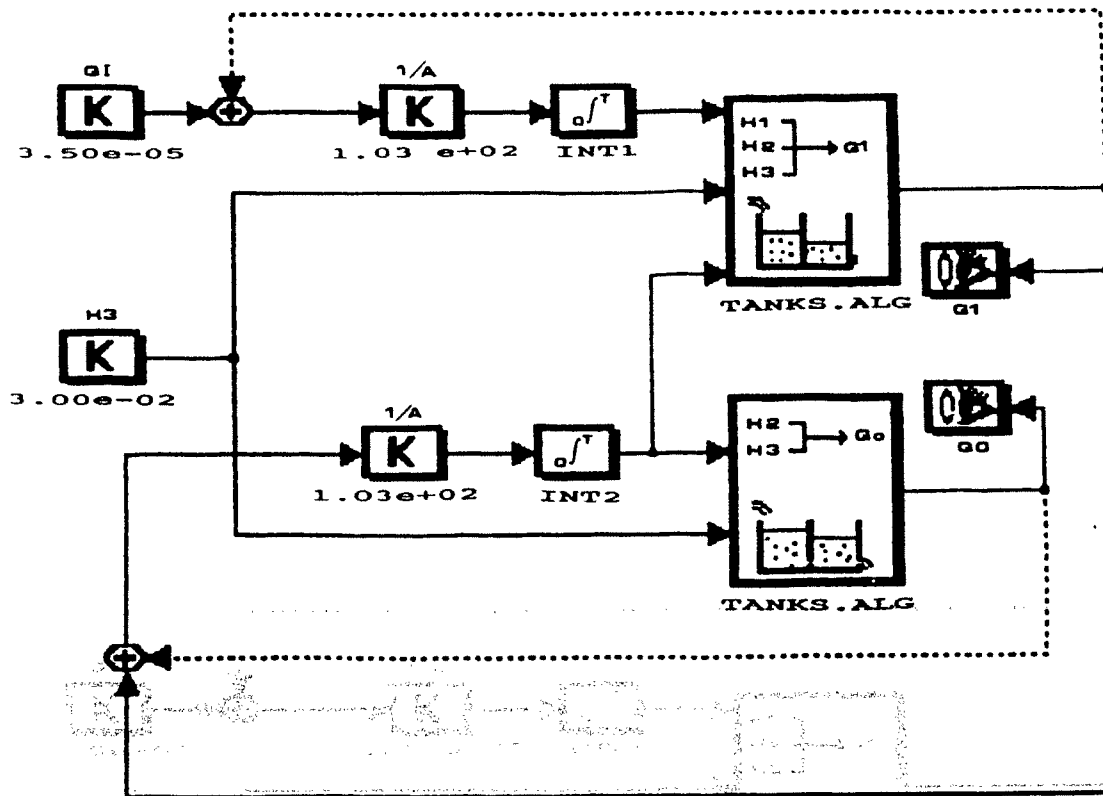


Figure 3: XANALOG block diagram for two-tank system simulation.

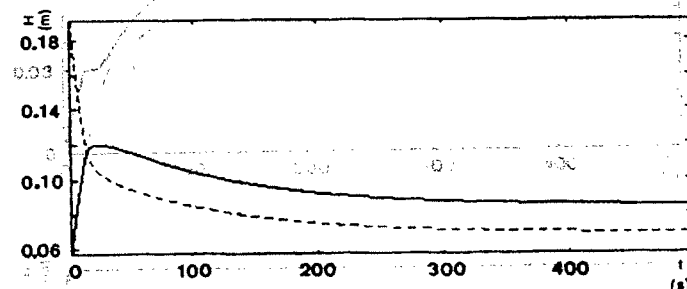
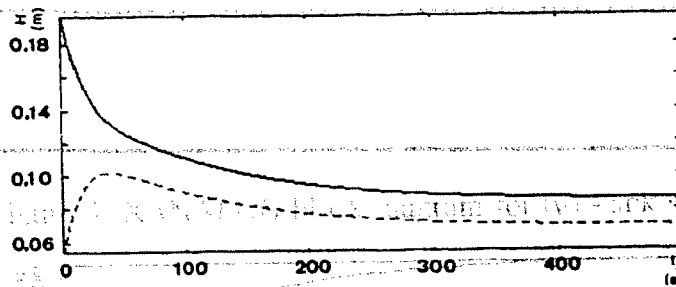
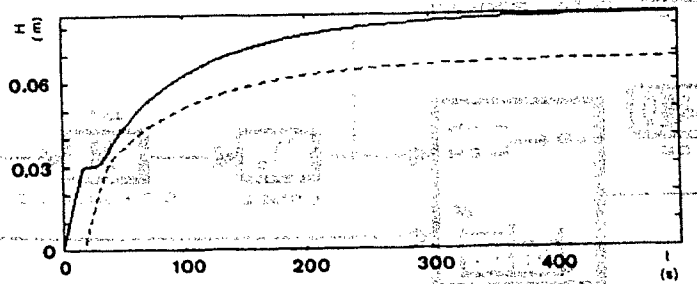


Figure 4: Simulation results for three sets of initial conditions.

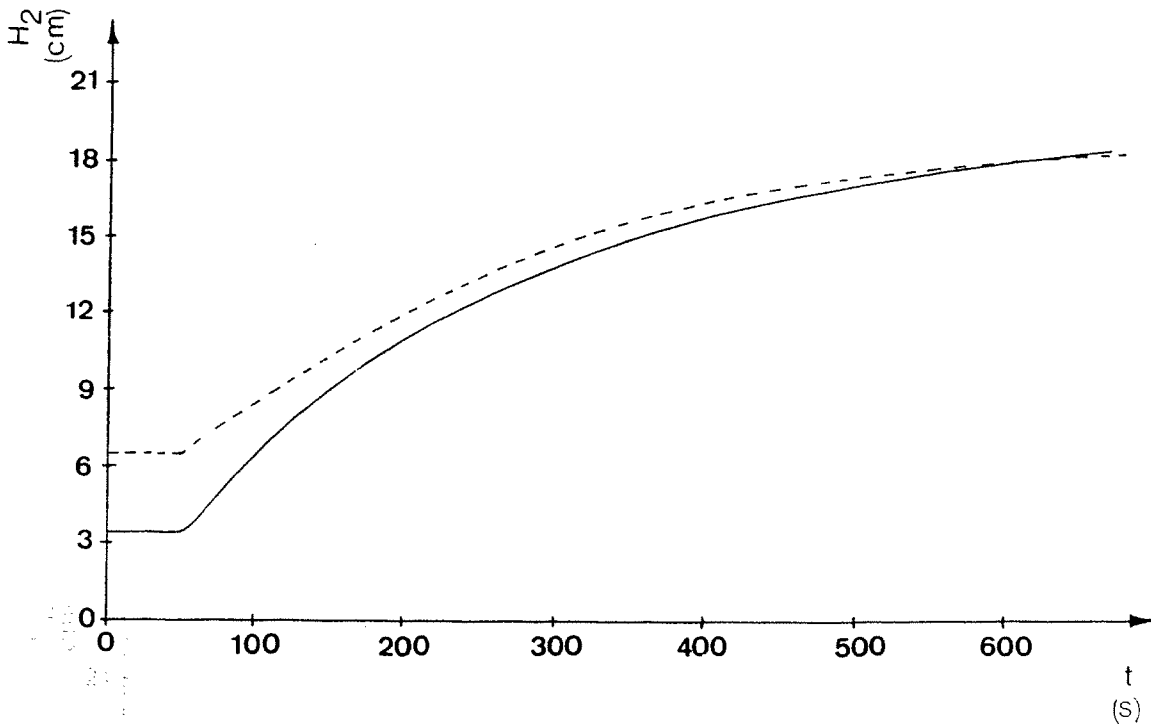


Figure 5: Simulation results for level in second tank, with measured response from the real system.

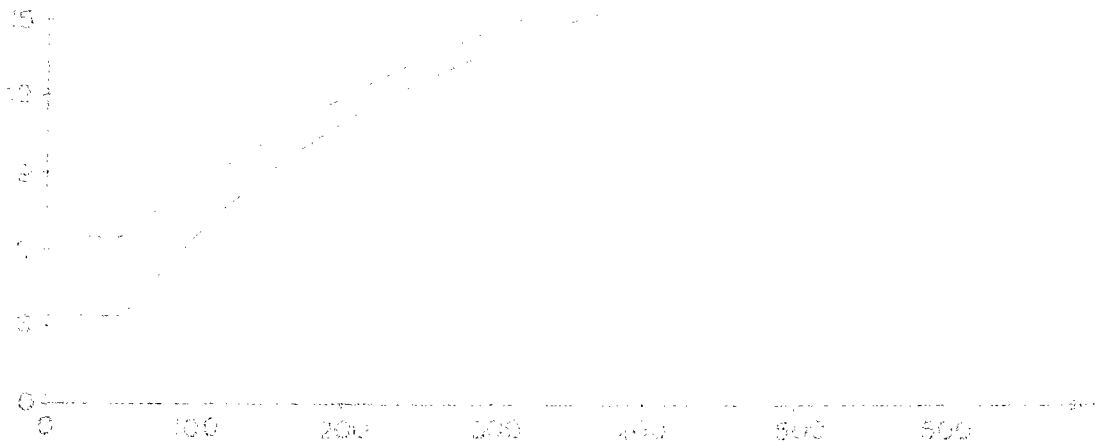


Figure 6: Simulation results for level in first tank, with measured response from the real system.

PARAMETER	SYMBOL	VALUE
Cross-sectional area, tank no. 1	A_1	0.0097 m^2
Cross-sectional area, tank no. 2	A_2	0.0097 m^2
Orifice area, between tanks	a_1	0.00003956 m^2
Orifice area, outlet from tank no. 2	a_2	0.0000385 m^2
Coefficient of discharge, inter-tank orifice	C_{d1}	0.75
Coefficient of discharge, outlet orifice from tank no. 2	C_{d2}	0.5
Gravitational constant	g	9.81 ms^{-2}
Pump calibration constant	G_p	$0.0000072 \text{ m}^3\text{s}^{-1}\text{V}^{-1}$
Depth sensor calibration constant	G_d	33.33 Vm^{-1}
Height of outlet above base of tank	H_3	0.03 m

Table 1: Parameter values for the coupled-tank model

Pump Q_1 (calibration constant) ($\text{cm}^3\text{min}^{-1}$)	H_1 measured (cm)	H_1 model (cm)	H_2 measured (cm)	H_2 model (cm)
1000	5.0	8.4	3.4	6.8
1500	13.1	15.2	9.7	11.6
2000	25.0	24.7	18.9	18.3

Table 2: System and model variables under steady-state conditions for three operating points

APPENDIX

Listing of SLIM program for Case Study

APPENDIX

Listing of SLIM program for Case Study

Listing of program TANKS.SLI

COMETT - COURSE "FUZZY SYSTEMS AND CONTROL"

Simulation of the coupled tanks apparatus under open-loop conditions. A typical input data set is available in the file TANKS.IN.

NCH=5
NSA=500
SAM=1.0
TMAX=500.0

Information about output file organisation

TYPE NCH
TYPE NSA
TYPE SAM

Input parameter values for model

ACCEPT QI,H10,H20
CD1=0.75
CD2=0.5
A1=3.956E-5
A2=3.85E-5
G=9.81
H3=0.03
A=9.7E-3

Calculation of equilibrium levels

H2SS=H3+(1.0/(2.0*G))*((QI/(CD2*A2))*((QI/(CD2*A2))))
H1SS=H2SS+(1.0/(2.0*G))*((QI/(CD1*A1))*((QI/(CD1*A1))))
H1=H10
H2=H20
T=0.0
CINTERVAL(SAM)

Values for integration parameters

MININTERVAL(1.0E-6)
MERROR(1.0E-4)
XERROR(1.0E-4)

End of initial segment, start of DYNAMIC segment

DYNAMIC

DERIVATIVE

Calculation IF(H1-H3)4,2,2

IF(H2-H3)3,5,5

Q1=0.0
GOTO 7

Q1=-CD1*A1*SQRT(2.0*G*(H2-H1))

GOTO 10

IF(H1-H2)5,6,6

Q1=CD1*A1*SQRT(2.0*G*(H1-H2))

GOTO 10

Values for integration parameters IF(H2-H3)7,7,8

Q0=0.0

GOTO 20

Q0=CD2*A2*SQRT(2.0*G*(H2-H3))

GOTO 20

End of initial segment, start of DYNAMIC segment

DYNAMIC

DERIVATIVE

IF(H1-H3)4,2,2

IF(H2-H3)3,5,5

Q1=0.0

```

20      CONTINUE
        DH1=(1.0/A)*(QI-Q1)
        DH2=(1.0/A)*(Q1-QO)
        H1=INTEG(DH1,H10)
        H2=INTEG(DH2,H20)
        DERIVATIVE END
        TYPE T, H1SS,H2SS, H1,H2
        IF(T-TMAX)50,50,60
50      DYNAMIC END
60      STOP
        END

```

Data file TANKS.IN

```

33.32E-6 DYNAMIC END
0.05     STOP
0.2      STOP

```

Data file TANKS.IN

```

33.32E-6
0.05
0.2

```

SLIM - A Simple Continuous System Simulation Language

USER'S GUIDE

D.J. Murray-Smith
Department of Electronics and Electrical Engineering
University of Glasgow
Glasgow G12 8QQ
Scotland
United Kingdom

Version 1.0

July 1994

(as distributed on diskette with the book "Continuous System Simulation",
by D.J. Murray-Smith, published by Chapman and Hall, London, 1994)

1. INTRODUCTION

The name SLIM stands for "A Simulation Language for Introductory Modelling". It involves a subset of the CSSL Standard established by the Sci Technical Committee in 1967. SLIM has been developed specifically as a language through which those new to continuous system simulation can be introduced to the principles of the subject.

The SLIM software included on the diskette which accompanies the book "Continuous System Simulation" by D.J. Murray-Smith is the most basic version of the SLIM family. Only one integration method is available (a variable-step method) and features such as function tables, pure delays, the use of arrays, the use of multiple DERIVATIVE sections, sub-model capabilities, sorting etc. are not provided. Such features may be found in many commercially-supported simulation languages. SLIM provides only the most basic simulation facilities but it is a very easily-used simulation tool and, with a little ingenuity, can be applied to a remarkably wide range of problems, as shown by the examples of Chapter 8 in the book and by the case studies in later Chapters.

Detailed practical information concerning the installation of the SLIM software and on the preparation and running of SLIM programs may be found in Section 4 of this Guide. This information is specific to the version of SLIM included on the diskette accompanying the book. Other versions of SLIM have been developed which incorporate additional facilities and are suitable for use with other operating systems, such as VMS and UNIX. A version has also been produced for use on a Parsytec parallel processor system under the PARIX system.

The SLIM software consists of a language processor which has been written in FORTRAN 77 which is capable of translating and interpreting programs written using the special SLIM instruction set. This instruction set involves a small subset of FORTRAN together with a number of the special CSSL instructions needed for simulation applications. SLIM, like many other examples of application-specific software systems, is based upon the use of a syntax-driven parser with a separate executor routine. The language processor first parses the user's program and detects any statements which are invalid. Once the validity of the complete program has been established, the software produces intermediate code which is not seen by the user but which can be interpreted by the executor routine.

2. THE SYNTAX OF THE SLIM LANGUAGE

SLIM follows the principles of the 1967 CSSL standard and the syntax has much in common with ACSL and other widely used languages. There are, however, no explicit INITIAL and TERMINAL statements and the structure of the program is defined only through DYNAMIC, DERIVATIVE, DERIVATIVE END and DYNAMIC END statements.

It should be noted that SLIM follows some of the conventions of FORTRAN and, in accordance with the standard for that language, all program statements in SLIM must begin at, or to the right of, the seventh column of a program line. The first five columns are used only for comments or labels. Comments are indicated by the character 'C' in the first column of the line and are

ignored completely in the translation process. Labels must be numeric symbols and may be entered in any or all of the first five columns. The sixth column must always contain a space unless the line in question is a comment line. Upper and lower case characters must be used throughout, except for comments which may involve upper or lower case characters, as desired. Blank lines should not be included in a SLIM program and tab facilities should not be used to insert spaces.

An additional link with FORTRAN is the fact that the two types of variable defined within SLIM, integer variables and real variables, are distinguished according to the letter of the alphabet with which the variable name starts. All integer variable names must begin with the letters I, J, K, L, M and N whereas real variable names must start with any other letter of the alphabet. Real variables and constants may have magnitudes in the range $1.18E-38$ to $40E+38$ while integer variables and constants must have magnitudes in the range 0 to 32768.

Variable names may involve any combination of letters and numerals (after the first character which must of course be a letter) but can only have six characters in total.

Examples of valid real variables are:

```
VALUE
VALUE1
GAIN
ACC2A
```

Examples of valid integer variables are:

```
INVAL5
MPOINT
NFLAG
KP
```

Real constants are exemplified by:

```
1.0
3.142
4.8E-7
6E4
```

and integer constants by:

```
1
567
22141
```

Statements in SLIM may be divided conveniently into general-purpose statements which provide the normal facilities for numerical programming and which have counterparts in most high-level programming languages, and those statements which are of special significance for simulation.

It should be noted that in the SLIM language expressions are evaluated according to the normal operator precedence rules. The order of evaluation is therefore as follows:

```
** raising to a power
*,/ multiplication or division
+,- addition or subtraction
```

It should be noted that SLIM statements must not extend beyond one line of code since there is no facility in the language for any form of continuation line. Complex expressions may therefore have to be split initially into a number of smaller expressions and combined at a later stage in the program.

i) GENERAL-PURPOSE STATEMENTS

a) INTEGER AND REAL ASSIGNMENTS

This type of statement takes the general form:

	Real variable = Real expression
or	Integer variable = Integer expression

Examples of this type of assignment are:

```
VELOCITY=(7.5-DISTCE)/4.982
IVALUE=8+7*K
```

Note that mixed expressions involving combinations of integers with real variables or constants are not permitted and if detected will be rejected as a syntax error. The functions FLOAT and IFIX may be used for data type conversion to overcome this limitation.

Real intrinsic functions available within SLIM which may be included within Real Assignment statements are as follows:

ABS, ALOG, ATAN, COS, EXP, FLOAT, SIN, SQRT and RND.

All these functions except RND are quite standard and have the same role as the correspondingly named functions in FORTRAN. All except FLOAT involve real arguments and give real results. FLOAT requires an integer argument and produces a real result.

The function RND generates uniformly distributed random numbers having values which lie between 0.0 and 1.0. The seed value used for random number generation is determined by the argument used in calling the function. Ten different seed values are available. All values of the argument which are less than 2.0 lead to the use of one particular seed, values equal to or greater than 2.0 but less than 3.0 lead to use of a second predetermined seed value and so on for all other similar intervals in the range 3.0 to 10.0.. All argument values of 10.0 or greater lead to the use of the tenth seed value.

The only integer intrinsic function in SLIM is IFIX. It can be used in the same way as the corresponding function in FORTRAN. It requires a real argument and returns an integer result.

b) INPUT STATEMENT

Input of data in SLIM is from a file named by the user. The form of the Input statement is as follows:

ACCEPT variable list

The variable list contains variable names which can be of either real or integer type. An example of this type of SLIM assignment is:

ACCEPT VALUE, CONST, GAIN

OUTPUT STATEMENT

Output is always written to a named file on disk but may also appear in tabular form on screen if desired. The form of the Output statement is:

TYPE variable list

e.g. TYPE VALUE,CONST,GAIN
TYPE J,IVAL

The variable list within an Output statement must only involve variables of one type. A mixture of integer and real variables in a single variable list is not permitted in an Output statement. The first few items in the output file provide information about the conditions selected within the simulation run and are written to the file automatically. These include three integers providing system programming information which are not of direct interest to the user followed by the values of the communication interval, the absolute and relative error parameters used in the program, and the minimum interval.

The next items have to be included in the output file but require the inclusion of TYPE statements in the SLIM source program for the simulation. They involve, in order, the number of channels in the output (say NCH), the number of samples used in the simulation run (say NSA) and the communication interval (say SAM). Every SLIM program must therefore include, at the end of the initial segment, three lines such as:

TYPE NCH
TYPE NSA
TYPE SAM

where NCH is the integer constant representing the number of output variables (including the independent variable where appropriate), NSA is the integer constant representing the number of samples over the complete time history of each variable and SAM is the real constant representing the communication interval. It is, of course, important that these three quantities be written to the output file in this particular order and that they should be the first items output from the simulation program. It should be noted that in all cases the quantities NSA and SAM must be chosen to be consistent with the interval of the independent variable (usually time) over which the results are required. If, for example the independent variable has an initial value $T=0.0$ and a final value $T=TMAX$ in the simulation program the quantities NSA and SAM must be such that their product is equal to TMAX.

Immediately after this it is normal to output the initial values of the independent variable and of the system variables of interest. Four TYPE statements therefore normally appear within the initial segment of any SLIM simulation program for which an output file is required.

d) ARITHMETIC IF STATEMENT

The syntax of an Arithmetic IF statement is as follows:

IF(arithmetic expression) Label 1, Label 2, Label 3

When this type of statement is executed the arithmetic expression is evaluated and control passes to Label 1 if the result is negative, to Label 2 if the result is zero and to Label 3 if the result is positive. Examples of such a statement are as follows:

```
IF(VELOCITY-THRESH)175,90,90
IF(IVAL-K)10,20,30
```

In the first case the expression (VELOCITY-THRESH) is evaluated and control is passed to the statements in lines labelled 175 or 90 depending on sign of the result. In the second case the integer result from the evaluation of the expression (IVAL-K) determines whether control passes to the statements with labels 10, 20 or 30. It should be noted that in the case of real expressions the condition corresponding to the result being zero is not of practical value due to problems of round-off and other inaccuracies in real arithmetic.

e) GOTO STATEMENT

This statement causes the normal sequential process of program statement execution to be interrupted. Control is passed unconditionally to the line which has the label contained in the GOTO statement. For example, the following section of program:

```

X=1.0
40  TYPE X
    X=(X+10.0)/2.0
    GOTO 40
```

would produce an endless loop which, if executed, would produce an output sequence as follows:

```

5.50
7.75
8.875
9.4375
etc.
```

f) CONTINUE STATEMENT

A CONTINUE statement is a dummy execution statement and simply causes control to be passed to the next statement in the program.

g) STOP STATEMENT

A STOP statement causes execution to cease. This is usually the last executable statement in a SLIM program. Like all other executable statements it can be given a label.

h) END STATEMENT

The purpose of an END statement is to indicate the end of the program. It must therefore always be the last statement and since it is not executable it cannot be given a label.

ii) SPECIAL-PURPOSE SIMULATION STATEMENTS

a) DYNAMIC STATEMENT

The DYNAMIC statement is used to indicate the start of the dynamic segment of the simulation program. In SLIM only one dynamic segment is allowed within a program. On entering the dynamic segment the processor assigns initial values to the integration variables which are evaluated within the derivative section. The dynamic segment contains the derivative section and the segment is therefore in two parts, one above the derivative section and the other below it.

D) DERIVATIVE STATEMENT

The DERIVATIVE statement indicates the beginning of the derivative section which lies entirely within the dynamic segment. The differential equations and other equations which define the dynamic simulation model are normally contained within this section.

E) DERIVATIVE END STATEMENT

The DERIVATIVE END statement defines the end of the derivative section. It indicates that all the integral assignments have been evaluated and the numerical integration routine is called. On returning from the integration routine the problem variables are all updated. Once control has been passed into the derivative section it will not be passed beyond the DERIVATIVE END statement until a new communication interval is about to start. If the start of a new communication interval has not been reached control is passed back to the start of the derivative section and the integration process is repeated with the updated variables. This continues until, at the appropriate time determined by the communication interval, control is transferred from the derivative section to the second part of the dynamic segment but without re-assignment of initial values.

F) DYNAMIC END STATEMENT

The DYNAMIC END statement signifies the end of the dynamic segment. In order to enter the region of the SLIM program lying beyond the DYNAMIC END statement control must be transferred explicitly by means of a GOTO or arithmetic IF statement.

G) INTEGRAL ASSIGNMENT STATEMENT

Each first order differential equation gives rise to an Integral Assignment statement. This type of statement has the form:

Real variable=INTEG(real expression, initial value)

where initial value=real variable or real constant

An example of a statement of this kind could have the following form:

X1=INTEG(V*W,X1INIT)

This indicates that the real expression $V*W$ is to be evaluated and from this the real variable $X1$ is to be calculated by integration with respect to the independent variable (normally time). The initial condition for this integration operation is provided by the real variable $X1INIT$ which must have been defined at an earlier stage in the program (usually in the initial segment).

H) CINTERVAL STATEMENT

The CINTERVAL statement sets the communication interval which defines the increment of the independent variable at which control is passed from the derivative section to the dynamic segment. The format of this statement is as follows:

CINTERVAL(unsigned real constant or real variable)

For example the statement

CINTERVAL(0.2)

will cause control to be passed from the derivative section to the dynamic segment every 0.2 units of the independent variable (e.g. every 0.2 sec.). The default communication interval is 1.0 and this value is used in any program in which no CINTERVAL is defined explicitly.

It should be noted, incidentally, that the default initial value of the independent variable in a SLIM simulation is zero. Other initial values are possible but must be set by the user using an assignment statement such as:

T=1.6

g) MINTERVAL STATEMENT

This statement defines the smallest step size allowed for a variable step length method. If this lower limit is reached an error message is displayed on the screen and control of the program is returned to the user. The MINTERVAL statement has the general form:

MINTERVAL(unsigned real constant or real variable)

e.g. MINTERVAL(1.0E-5)

The default is MINTERVAL(1.0E-4).

h) XERROR STATEMENT

The XERROR statement is used to set the maximum absolute error that can be accepted in the results of the variable step length numerical integration. The value is defined on a basis of the error per integration step. The variable step length routine used within SLIM either satisfies the requirement defined in the XERROR statement or indicates to the user that the step length has reached the minimum. The form of the statement is:

XERROR(unsigned real constant or real variable)

A typical example is :

XERROR(1.0E-5)

The default value corresponds to XERROR(1.0E-4).

i) MERROR STATEMENT

This statement is similar to XERROR but sets the maximum relative error (per integration step). Again the requirement is satisfied automatically unless the user is otherwise informed through an error message. The format of the MERROR statement is similar to CINTERVAL, MINTERVAL and XERROR and a typical example is:

MERROR(1.0E-4)

which would set the value to the default.

3. OUTPUT FACILITIES AND GRAPHICS

Graphical output may be obtained from the SLIM program by means of a post-processing program which uses the results file from SLIM as input. This

rogram, called SLIMPLOT, requests the name of the results file, together with information about the channels of the output to be plotted. Both time history plots and x-y plots can be produced by SLIMPLOT.

The first few items in the output files produced by the SLIM program provide information about the simulation run and about the internal organisation of the output file, as specified above. This information, which must appear in the standard format described in Section 2 (c) above, provides the post-processing program with essential information which allows it to interpret the simulation data which follows.

4. INSTRUCTIONS FOR INSTALLATION AND USE OF SLIM

The software included on this diskette includes the SLIM.EXE program and the post-processing program SLIMPLOT.EXE for the display of results in graphical form. Example programs discussed in Chapters 6, 8, 10, 11 and 12 of "Continuous System Simulation" by D.J. Murray-Smith may also be found on the diskette, together with the necessary input data files.

4.1 Hardware requirements

The SLIM and SLIMPLOT programs run on any IBM-compatible personal computer running the Microsoft DOS (MS-DOS) operating system. A computer with at least an 80286 or 80386 microprocessor is preferred, although not essential. It is also recommended that the computer has the 80287 or 80387 floating-point co-processor which will significantly reduce the time required to obtain results.

A hard disk is desirable and the computer must have CGA, EGA or VGA display capabilities. The computer should contain a minimum of 512k bytes of RAM memory. A standard ASCII text editor is necessary.

4.2 Software installation

This section of the User's Guide provides practical information about the steps to be followed in installing the SLIM software on your computer. The README.DOC file on the diskette also provides details of procedures for installation of the SLIM and SLIMPLOT software, including a list of the names of all the files which are provided and any updated features.

SLIM and SLIMPLOT will run successfully from a floppy disk drive but most users prefer to mount the software on their hard disk. To do this you should create a new directory (normally named SLIM) on the hard disk. All files on the distribution diskette should be transferred to this new directory. To create such a directory use the normal DOS command:

```
mkdir slim
```

If you wish you could then create separate path-accessible sub-directories for user files. However, for simplicity, it is assumed in the remainder of these notes that all SLIM files are in the SLIM directory, that the default hard disk drive is c: and that floppy disk drive is a:.

To transfer the files from the SLIM diskette to the hard disk place the diskette in the a: drive and copy all the distribution files into the c:\SLIM directory as follows:

```
copy a: *.* c:\slim
```

It is important to make back-up copies of all of the files supplied on the distribution diskette prior to attempting to make use of the software. The original diskette should be stored safely.

4.3 Starting and exiting SLIM

Make the SLIM directory the current directory by entering the DOS command `cd\slim`. When the DOS prompt (in this case `C:\SLIM>`) appears type:

`slim`

and the SLIM program should start to execute. It first requests the name of the SLIM program file to be run. From this point the sequence of operations follows the pattern of the example of Section 6.6.4 of the book. Any error messages relating to errors in translation or occurring at run time should be self explanatory. Syntax errors detected at the translation stage should provide a reference to the line in which the error was found but will not pinpoint the precise nature of the error. It is necessary for the user to examine the relevant section of code to determine why the translation has failed. Comment lines are ignored in the line numbering system for SLIM error messages.

It may be useful to use the `VDPOLS.SLI` program file from the example in Section 6.6.4 of the book to ensure that you are using SLIM correctly and that the results of Chapter 6 can be reproduced on your own computer.

At the end of a simulation run the user is provided with an opportunity to carry out a further run with the same program through a query of the form:

DO YOU WANT ANOTHER RUN? (Y/N)

Responding N returns the user to DOS and the DOS prompt appears.

4.4 Useful hints for using SLIM

In preparing simulation programs using the SLIM language features described in Chapter 6 and in this User's Guide it is important to take note of the following points:

- a) A SLIM program must follow the conventions explained in Chapter 6 and Section 2 of this Guide. These conventions are similar in many respects to the conventions for FORTRAN 4 or FORTRAN 77 programming.
- b) It should be noted that executable code must start in column 7 of each line of program. Columns 1 to 5 are reserved for labels or for use in comment lines. Comments must start with an upper case C in column 1.
- c) In creating a SLIM program using a general-purpose text editor, use of the TAB key should be avoided. The space bar should be used to move the cursor to the required column (e.g. column 7).
- d) No blank lines should be included in a SLIM program. The presence of a blank line in a SLIM source program causes problems at the translation stage and may cause the computer to 'hang'.
- e) Line numbers quoted in connection with error messages relate to lines involving SLIM language statements only. Lines involving comments are not given line numbers.
- f) Continuation lines are not permitted in SLIM programs and care must be taken to avoid lengthy statements which involve more than 70 characters

or spaces. Attempts to use longer statements in SLIM program may cause the computer to 'hang' at the translation stage. Long statements must be split into two or more statements, each occupying a separate line of code.

- g) The distinction between integer and real quantities is important and involves the normal naming conventions of FORTRAN. Real and integer quantities cannot be used in direct combination within a single statement.
- h) All executable statements in a SLIM program must involve only upper case letters. Comments may involve upper case and lower case letters but must include an upper case C in column 1.
- i) When a syntax error is detected at the program translation stage a number of messages may be displayed which appear to relate to syntax errors in a number of different lines. If there is no apparent reason for an error in one particular line check the statements in the other lines for which syntax errors have been reported. Clearing the error reported in one line may eliminate the apparently meaningless errors reported in other lines.

.5 Starting and exiting SLIMPLOT

SLIMPLOT is extremely easy to use. It is entered using the following command:

```
slimplot
```

The program then prompts the user for the file name of the SLIM output file which is to be plotted. This is followed by a request to the user to select the type of output file (y versus time or an x-y plot), the number of variables (in the case of an y-t plot) and the column numbers of the variables.

The column numbers referred to in the SLIMPLOT prompts are the column numbers arising from use of the TYPE statement to generate the output data from the SLIM program. For example, if the output is obtained from a statement of the form:

```
TYPE T,X1,X2,X3
```

there will be four columns in the output file. Column 1 involves values of T (the independent variable) and columns 2 to 4 involve the corresponding values of the dependent variables X1, X2 and X3. If one wanted to plot X1 and X3 the columns to be entered in response to the prompt from the SLIMPLOT program would be 2 and 4. These column numbers may be entered as:

```
2,4
```

or

```
2 4
```

Text can be inserted by the user to provide a main (top) title and up to two titles which appear below the graph. Both upper and lower case characters may be used in these titles.

When graphical output has been displayed on the screen and the user wishes to move on to a further stage of processing pressing the ENTER key clears the graphical output from the screen and causes a new menu to appear. This has one option which allows the user to exit from the SLIMPLOT program and others which allow further plots to be produced.

4.6 Useful hints for using SLIMPLOT

The following points should be noted:

- a) Problems will be encountered if the number of time steps in a SLIM output file is greater than the maximum allowed by SLIMPLOT. If the maximum is exceeded an error message is displayed during the running of the SLIMPLOT program. The SLIM program should then be re-run with an increased communication interval (CINTERVAL) or a smaller range for the independent variable. For problems where the independent variable is time this means either increasing the time step for the plot or reducing the length of the simulation run.
- b) Hard copy output from SLIMPLOT is most easily obtained using the DOS GRAPHICS command and the use of the Print Screen key. An alternative, if an appropriate printer is available, is to make use of the printer or plotter facilities in the list of output devices which is displayed during the running of SLIMPLOT.

5. EXAMPLES

Chapters 6, 8, 10, 11 and 12 of "Continuous System Simulation" include examples of SLIM programs. SLIM program files and data files for these examples are provided on this diskette. The program files all involve file names with a .SLI extension, while the input data files all involve files with a .IN extension. Filenames are the same as those used in the relevant chapters. It should be possible in each case to reproduce results presented in tabular or graphical form in the book.

6. COMPARISONS OF SLIM WITH WIDELY-USED SIMULATION LANGUAGES

SLIM is intended to provide beginners with an introduction to the principles of continuous system simulation and has been designed as an inexpensive but reasonably versatile teaching tool. It is not intended to compete directly with commercially-supported languages. SLIM is most suitable for relatively small simulation problems. It does conform to many of the conventions of the CSSL specification and provides a convenient, efficient and easily-used tool. Having gained some initial experience with SLIM the user should be able to move on without difficulty to make use of other more powerful languages, such as ACSL.

The maximum number of state variables which can be accommodated within a SLIM simulation program using the version of the SLIM software which accompanies the book "Continuous System Simulation" is 40. This is not seen as imposing any serious restriction on the average beginner. All of the examples and case studies considered involve less than ten state variables. The maximum number of statements allowed within a SLIM program is 250. Once again typical applications involve much smaller programs. Some other simulation languages, such as ACSL, do not impose limitations of this kind.

While the restrictions in terms of program size are not thought to represent limits which are likely to be approached by the beginner, it is important to understand that languages such as CSSL IV, ACSL and DESIRE provide many features which are not available within SLIM. Some of these restrict the type of system which can be modelled using SLIM and others limit the options available to the user at run time. The sub-set of facilities incorporated in SLIM has been chosen to allow the beginner to gain experience rapidly, without having to deal with the complexities associated with features in more comprehensive languages. A careful examination of the user manual for each specific language allows the potential user to establish the precise differences in every case.

The absence of facilities to model a pure time delay (transport delay) is one important example of a feature, available in many simulation tools, which has not been included within the version of SLIM provided with "Continuous System Simulation". The lack of any facility to represent a function in tabular form is another example of a potentially useful feature which has not been included. Sorting facilities and facilities for sub-models are also not provided within SLIM. Simulation problems which require the use of facilities such as transport delays and function tables cannot therefore be tackled using SLIM, unless the problem can be reformulated in some way to allow it to be solved within the limitations of the language. Some other languages, such as ACSL, provide a much more extensive set of facilities and therefore provide much more flexibility at the model definition stage.

In comparison with ACSL and other similar languages SLIM has limited capabilities in terms of run-time options. There is no real separation of the model definition and the experiment in SLIM. Changes of parameters require editing of the simulation program itself or changes of an associated data file. However, flexibility is available in terms of output display facilities in SLIM because of the fact that the graphical output options involve a separate post-processing program.

SLIM can be extremely effective when learning about the principles of simulation and when used for the solution of problems for which its facilities are appropriate. It is important, however, to understand its limitations and to have an appreciation of the additional facilities of the software products which are available commercially.

. DISCLAIMER

All parties must rely on their own skill and judgement when making use of the SLIM and SLIMPLOT software. Neither the author nor the publishers assume any liability to anyone for any loss or damage caused by any error or omission in the work, whether such error or omission is the result of negligence or any other cause. Any and all such liability is disclaimed.

. ACKNOWLEDGEMENTS

The features included in this version of the SLIM language are, as explained in Section 1 of this User's Guide, a subset of the features considered desirable for equation-oriented continuous system simulation languages, as recommended in 1967 by the relevant Simulation Councils inc. technical committee. In considering the features to incorporate in this simulation language, which has been designed specifically for teaching applications, a number of existing simulation tools were reviewed. One existing language which has had a particularly strong influence on SLIM is the MIMESIS language which was developed by Ian Ricketts at the University of Dundee and as described in his Ph.D. dissertation in 1977 (Ricketts, I.W. "MIMESIS - A Continuous System Simulation Language", Ph.D. Thesis, University of Dundee, 1977). SLIM and MIMESIS are both based on the principles of syntax-driven parsers as described by R.L. Gauthier and R.D. Ponto in their text entitled "Designing Systems Programs" (Gauthier, R.L. and Ponto, R.D. "Designing Systems Programs", Prentice-Hall, New Jersey, 1970). Many features of the instruction set of SLIM are similar to those of MIMESIS, although the internal structure of the SLIM program and the detailed coding are different. Experience gained in using the MIMESIS software has been of particular value in the development of SLIM and the author is very grateful to Dr. Ricketts for making MIMESIS available.

The author also wishes to record his thanks to present and former students in the Department of Electronics and Electrical Engineering at the University

of Glasgow who have assisted in the development of the SLIM software. A number of student projects have been associated with the application of SLIM to practical engineering problems and these have provided valuable information about the range of problems which can be tackled conveniently using this software. Very helpful feedback has also been provided freely by students who have used SLIM in connection with courses in computer simulation methods, in control engineering and in avionic systems at undergraduate and post-graduate level.

The software has undergone much revision and modification during the course of its development. Work on SLIM began in 1991 during a period of sabbatical leave which was spent at the Technical University of Vienna. The author is very grateful to Professor Dr. Felix Breiteneker of the Department of Simulation Techniques at the Technical University of Vienna for his assistance and support in making that visit possible and to the University of Glasgow for providing the necessary leave of absence. Without the opportunity provided by this period of uninterrupted work it is unlikely that SLIM would have been developed.

The graphical routines included in the SLIMPLOT program are part of the NAG Graphics Library. This product contains a collection of over 100 carefully designed routines which provides FORTRAN and C programmers with a versatile means of producing a graphical representation of numerical and statistical results. It is used worldwide in commerce, financial modelling, industry academic research and many other areas, and is available for a wide spectrum of machines. Interfaces are also supplied to the most commonly used plotting packages. NAG also provide a wide range of other products such as mathematical and statistical routines in FORTRAN, C, Pascal and Ada; FORTRAN 90 compilers and tools; symbolic solvers; data visualization software; linear programming software and a gateway generator which automatically generates MATLAB gateway. For further information, a quotation or details of your local distributor please contact:

Sales Department
NAG Ltd.
Wilkinson House
Jordan Hill Road
Oxford OX2 9BX
United Kingdom.

SLIM - A Simple Continuous System Simulation Language

USER'S GUIDE

D.J. Murray-Smith
Department of Electronics and Electrical Engineering
University of Glasgow
Glasgow G12 8QQ
Scotland
United Kingdom

Version 1.0

July 1994

(as distributed on diskette with the book "Continuous System Simulation",
by D.J. Murray-Smith, published by Chapman and Hall, London, 1994)

1. INTRODUCTION

The name SLIM stands for "A Simulation Language for Introductory Modelling". It involves a subset of the CSSL Standard established by the Sci Technical Committee in 1967. SLIM has been developed specifically as a language through which those new to continuous system simulation can be introduced to the principles of the subject.

The SLIM software included on the diskette which accompanies the book "Continuous System Simulation" by D.J. Murray-Smith is the most basic version of the SLIM family. Only one integration method is available (a variable-step method) and features such as function tables, pure delays, the use of arrays, the use of multiple DERIVATIVE sections, sub-model capabilities, sorting etc. are not provided. Such features may be found in many commercially-supported simulation languages. SLIM provides only the most basic simulation facilities but it is a very easily-used simulation tool and, with a little ingenuity, can be applied to a remarkably wide range of problems, as shown by the examples of chapter 8 in the book and by the case studies in later Chapters.

Detailed practical information concerning the installation of the SLIM software and on the preparation and running of SLIM programs may be found in Section 4 of this Guide. This information is specific to the version of SLIM included on the diskette accompanying the book. Other versions of SLIM have been developed which incorporate additional facilities and are suitable for use with other operating systems, such as VMS and UNIX. A version has also been produced for use on a Parsytec parallel processor system under the PARIX system.

The SLIM software consists of a language processor which has been written in FORTRAN 77 which is capable of translating and interpreting programs written using the special SLIM instruction set. This instruction set involves a small subset of FORTRAN together with a number of the special CSSL instructions needed for simulation applications. SLIM, like many other examples of application-specific software systems, is based upon the use of a syntax-driven parser with a separate executor routine. The language processor first parses the user's program and detects any statements which are invalid. Once the validity of the complete program has been established, the software produces intermediate code which is not seen by the user but which can be interpreted by the executor routine.

2. THE SYNTAX OF THE SLIM LANGUAGE

SLIM follows the principles of the 1967 CSSL standard and the syntax has much in common with ACSL and other widely used languages. There are, however, no explicit INITIAL and TERMINAL statements and the structure of the program is defined only through DYNAMIC, DERIVATIVE, DERIVATIVE END and DYNAMIC END statements.

It should be noted that SLIM follows some of the conventions of FORTRAN and, in accordance with the standard for that language, all program statements in SLIM must begin at, or to the right of, the seventh column of a program line. The first five columns are used only for comments or labels. Comments are indicated by the character 'C' in the first column of the line and are

nored completely in the translation process. Labels must be numeric symbols and may be entered in any or all of the first five columns. The sixth column must always contain a space unless the line in question is a comment line. Upper case characters must be used throughout, except for comments which may involve upper or lower case characters, as desired. Blank lines should not be included in a SLIM program and tab facilities should not be used to insert spaces.

An additional link with FORTRAN is the fact that the two types of variable defined within SLIM, integer variables and real variables, are distinguished according to the letter of the alphabet with which the variable name starts. All integer variable names must begin with the letters I, J, K, L, M and N whereas real variable names must start with any other letter of the alphabet. Real variables and constants may have magnitudes in the range $1.18E-38$ to $40E+38$ while integer variables and constants must have magnitudes in the range 0 to 32768.

Variable names may involve any combination of letters and numerals (after the first character which must of course be a letter) but can only have six characters in total.

Examples of valid real variables are:

```
VALUE
VALUE1
GAIN
ACC2A
```

Examples of valid integer variables are:

```
INVAL5
MPOINT
NFLAG
KP
```

Real constants are exemplified by:

```
1.0
3.142
4.8E-7
6E4
```

and integer constants by:

```
1
567
22141
```

Statements in SLIM may be divided conveniently into general-purpose statements which provide the normal facilities for numerical programming and which have counterparts in most high-level programming languages, and those statements which are of special significance for simulation.

It should be noted that in the SLIM language expressions are evaluated according to the normal operator precedence rules. The order of evaluation is therefore as follows:

```
** raising to a power
*,/ multiplication or division
+,- addition or subtraction
```

It should be noted that SLIM statements must not extend beyond one line of code since there is no facility in the language for any form of continuation line. Complex expressions may therefore have to be split initially into a number of smaller expressions and combined at a later stage in the program.

i) GENERAL-PURPOSE STATEMENTS

a) INTEGER AND REAL ASSIGNMENTS

This type of statement takes the general form:

	Real variable = Real expression
or	Integer variable = Integer expression

Examples of this type of assignment are:

```
VELOCITY=(7.5-DISTCE)/4.982
IVALUE=8+7*K
```

Note that mixed expressions involving combinations of integers with real variables or constants are not permitted and if detected will be rejected as a syntax error. The functions FLOAT and IFIX may be used for data type conversion to overcome this limitation.

Real intrinsic functions available within SLIM which may be included within Real Assignment statements are as follows:

ABS, ALOG, ATAN, COS, EXP, FLOAT, SIN, SQRT and RND.

All these functions except RND are quite standard and have the same role as the correspondingly named functions in FORTRAN. All except FLOAT involve real arguments and give real results. FLOAT requires an integer argument and produces a real result.

The function RND generates uniformly distributed random numbers having values which lie between 0.0 and 1.0. The seed value used for random number generation is determined by the argument used in calling the function. Ten different seed values are available. All values of the argument which are less than 2.0 lead to the use of one particular seed, values equal to or greater than 2.0 but less than 3.0 lead to use of a second predetermined seed value and so on for all other similar intervals in the range 3.0 to 10.0.. All argument values of 10.0 or greater lead to the use of the tenth seed value.

The only integer intrinsic function in SLIM is IFIX. It can be used in the same way as the corresponding function in FORTRAN. It requires a real argument and returns an integer result.

b) INPUT STATEMENT

Input of data in SLIM is from a file named by the user. The form of the Input statement is as follows:

ACCEPT variable list

The variable list contains variable names which can be of either real or integer type. An example of this type of SLIM assignment is:

ACCEPT VALUE, CONST, GAIN

)OUTPUT STATEMENT

Output is always written to a named file on disk but may also appear in tabular form on screen if desired. The form of the Output statement is:

TYPE variable list

.g. TYPE VALUE,CONST,GAIN
TYPE J,IVAL

The variable list within an Output statement must only involve variables of one type. A mixture of integer and real variables in a single variable list is not permitted in an Output statement. The first few items in the output file provide information about the conditions selected within the simulation run and are written to the file automatically. These include three integers providing system programming information which are not of direct interest to the user followed by the values of the communication interval, the absolute and relative error parameters used in the program, and the minimum interval.

The next items have to be included in the output file but require the inclusion of TYPE statements in the SLIM source program for the simulation. They involve, in order, the number of channels in the output (say NCH), the number of samples used in the simulation run (say NSA) and the communication interval (say SAM). Every SLIM program must therefore include, at the end of the initial segment, three lines such as:

TYPE NCH
TYPE NSA
TYPE SAM

where NCH is the integer constant representing the number of output variables (including the independent variable where appropriate), NSA is the integer constant representing the number of samples over the complete time history of each variable and SAM is the real constant representing the communication interval. It is, of course, important that these three quantities be written to the output file in this particular order and that they should be the first items output from the simulation program. It should be noted that in all cases the quantities NSA and SAM must be chosen to be consistent with the interval of the independent variable (usually time) over which the results are required. If, for example the independent variable has an initial value $T=0.0$ and a final value $T=TMAX$ in the simulation program the quantities NSA and SAM must be such that their product is equal to TMAX.

Immediately after this it is normal to output the initial values of the independent variable and of the system variables of interest. Four TYPE statements therefore normally appear within the initial segment of any SLIM simulation program for which an output file is required.

d) ARITHMETIC IF STATEMENT

The syntax of an Arithmetic IF statement is as follows:

IF(arithmetic expression) Label 1, Label 2, Label 3

When this type of statement is executed the arithmetic expression is evaluated and control passes to Label 1 if the result is negative, to Label 2 if the result is zero and to Label 3 if the result is positive. Examples of such a statement are as follows:

```
IF(VELOCITY-THRESH)175.90,90
IF(IVAL-K)10,20,30
```

In the first case the expression (VELOCITY-THRESH) is evaluated and control is passed to the statements in lines labelled 175 or 90 depending on sign of the result. In the second case the integer result from the evaluation of the expression (IVAL-K) determines whether control passes to the statements with labels 10, 20 or 30. It should be noted that in the case of real expressions the condition corresponding to the result being zero is not of practical value due to problems of round-off and other inaccuracies in real arithmetic

e) GOTO STATEMENT

This statement causes the normal sequential process of program statement execution to be interrupted. Control is passed unconditionally to the line which has the label contained in the GOTO statement. For example, the following section of program:

```

X=1.0
40  TYPE X
    X=(X+10.0)/2.0
    GOTO 40
```

would produce an endless loop which, if executed, would produce an output sequence as follows:

```

5.50
7.75
8.875
9.4375
etc.
```

f) CONTINUE STATEMENT

A CONTINUE statement is a dummy execution statement and simply causes control to be passed to the next statement in the program.

g) STOP STATEMENT

A STOP statement causes execution to cease. This is usually the last executable statement in a SLIM program. Like all other executable statements it can be given a label.

h) END STATEMENT

The purpose of an END statement is to indicate the end of the program. It must therefore always be the last statement and since it is not executable it cannot be given a label.

ii) SPECIAL-PURPOSE SIMULATION STATEMENTS

a) DYNAMIC STATEMENT

The DYNAMIC statement is used to indicate the start of the dynamic segment of the simulation program. In SLIM only one dynamic segment is allowed within a program. On entering the dynamic segment the processor assigns initial values to the integration variables which are evaluated within the derivative section. The dynamic segment contains the derivative section and the segment is therefore in two parts, one above the derivative section and the other below it.

c) DERIVATIVE STATEMENT

The DERIVATIVE statement indicates the beginning of the derivative section which lies entirely within the dynamic segment. The differential equations and other equations which define the dynamic simulation model are normally contained within this section.

d) DERIVATIVE END STATEMENT

The DERIVATIVE END statement defines the end of the derivative section. It indicates that all the integral assignments have been evaluated and the numerical integration routine is called. On returning from the integration routine the problem variables are all updated. Once control has been passed into the derivative section it will not be passed beyond the DERIVATIVE END statement until a new communication interval is about to start. If the start of a new communication interval has not been reached control is passed back to the start of the derivative section and the integration process is repeated with the updated variables. This continues until, at the appropriate time determined by the communication interval, control is transferred from the derivative section to the second part of the dynamic segment but without re-assignment of initial values.

e) DYNAMIC END STATEMENT

The DYNAMIC END statement signifies the end of the dynamic segment. In order to enter the region of the SLIM program lying beyond the DYNAMIC END statement control must be transferred explicitly by means of a GOTO or Arithmetic IF statement.

f) INTEGRAL ASSIGNMENT STATEMENT

Each first order differential equation gives rise to an Integral Assignment statement. This type of statement has the form:

Real variable=INTEG(real expression, initial value)

where initial value=real variable or real constant

An example of a statement of this kind could have the following form:

X1=INTEG(V*W,X1INIT)

This indicates that the real expression V*W is to be evaluated and from this the real variable X1 is to be calculated by integration with respect to the independent variable (normally time). The initial condition for this integration operation is provided by the real variable X1INIT which must have been defined at an earlier stage in the program (usually in the initial segment).

g) CINTERVAL STATEMENT

The CINTERVAL statement sets the communication interval which defines the increment of the independent variable at which control is passed from the derivative section to the dynamic segment. The format of this statement is as follows:

CINTERVAL(unsigned real constant or real variable)

For example the statement

CINTERVAL(0.2)

will cause control to be passed from the derivative section to the dynamic segment every 0.2 units of the independent variable (e.g. every 0.2 sec.). The default communication interval is 1.0 and this value is used in any program in which no CINTERVAL is defined explicitly.

It should be noted, incidentally, that the default initial value of the independent variable in a SLIM simulation is zero. Other initial values are possible but must be set by the user using an assignment statement such as:

T=1.6

g) MININTERVAL STATEMENT

This statement defines the smallest step size allowed for a variable step length method. If this lower limit is reached an error message is displayed on the screen and control of the program is returned to the user. The MININTERVAL statement has the general form:

MININTERVAL(unsigned real constant or real variable)

e.g. MININTERVAL(1.0E-5)

The default is MININTERVAL(1.0E-4).

h) XERROR STATEMENT

The XERROR statement is used to set the maximum absolute error that can be accepted in the results of the variable step length numerical integration. The value is defined on a basis of the error per integration step. The variable step length routine used within SLIM either satisfies the requirement defined in the XERROR statement or indicates to the user that the step length has reached the minimum. The form of the statement is:

XERROR(unsigned real constant or real variable)

A typical example is :

XERROR(1.0E-5)

The default value corresponds to XERROR(1.0E-4).

i) MERROR STATEMENT

This statement is similar to XERROR but sets the maximum relative error (per integration step). Again the requirement is satisfied automatically unless the user is otherwise informed through an error message. The format of the MERROR statement is similar to CINTERVAL, MININTERVAL and XERROR and a typical example is:

MERROR(1.0E-4)

which would set the value to the default.

3. OUTPUT FACILITIES AND GRAPHICS

Graphical output may be obtained from the SLIM program by means of a post-processing program which uses the results file from SLIM as input. This

rogram, called SLIMPLOT, requests the name of the results file, together with information about the channels of the output to be plotted. Both time history plots and x-y plots can be produced by SLIMPLOT.

The first few items in the output files produced by the SLIM program provide information about the simulation run and about the internal organisation of the output file, as specified above. This information, which must appear in the standard format described in Section 2 (c) above, provides the post-processing program with essential information which allows it to interpret the simulation data which follows.

1. INSTRUCTIONS FOR INSTALLATION AND USE OF SLIM

The software included on this diskette includes the SLIM.EXE program and the post-processing program SLIMPLOT.EXE for the display of results in graphical form. Example programs discussed in Chapters 6, 8, 10, 11 and 12 of "Continuous System Simulation" by D.J. Murray-Smith may also be found on the diskette, together with the necessary input data files.

1.1 Hardware requirements

The SLIM and SLIMPLOT programs run on any IBM-compatible personal computer running the Microsoft DOS (MS-DOS) operating system. A computer with at least an 80286 or 80386 microprocessor is preferred, although not essential. It is also recommended that the computer has the 80287 or 80387 floating-point co-processor which will significantly reduce the time required to obtain results.

A hard disk is desirable and the computer must have CGA, EGA or VGA display capabilities. The computer should contain a minimum of 512k bytes of RAM memory. A standard ASCII text editor is necessary.

1.2 Software installation

This section of the User's Guide provides practical information about the steps to be followed in installing the SLIM software on your computer. The README.DOC file on the diskette also provides details of procedures for installation of the SLIM and SLIMPLOT software, including a list of the names of all the files which are provided and any updated features.

SLIM and SLIMPLOT will run successfully from a floppy disk drive but most users prefer to mount the software on their hard disk. To do this you should create a new directory (normally named SLIM) on the hard disk. All files on the distribution diskette should be transferred to this new directory. To create such a directory use the normal DOS command:

```
mkdir slim
```

If you wish you could then create separate path-accessible sub-directories for user files. However, for simplicity, it is assumed in the remainder of these notes that all SLIM files are in the SLIM directory, that the default hard disk drive is c: and that floppy disk drive is a:.

To transfer the files from the SLIM diskette to the hard disk place the diskette in the a: drive and copy all the distribution files into the c:\SLIM directory as follows:

```
copy a: *.* c:\slim
```

It is important to make back-up copies of all of the files supplied on the distribution diskette prior to attempting to make use of the software. The original diskette should be stored safely.

4.3 Starting and exiting SLIM

Make the SLIM directory the current directory by entering the DOS command `cd\slim`. When the DOS prompt (in this case `C:\SLIM>`) appears type:

`slim`

and the SLIM program should start to execute. It first requests the name of the SLIM program file to be run. From this point the sequence of operations follows the pattern of the example of Section 6.6.4 of the book. Any error messages relating to errors in translation or occurring at run time should be self explanatory. Syntax errors detected at the translation stage should provide a reference to the line in which the error was found but will not pinpoint the precise nature of the error. It is necessary for the user to examine the relevant section of code to determine why the translation has failed. Comment lines are ignored in the line numbering system for SLIM error messages.

It may be useful to use the `VDPOLS.SLI` program file from the example in Section 6.6.4 of the book to ensure that you are using SLIM correctly and that the results of Chapter 6 can be reproduced on your own computer.

At the end of a simulation run the user is provided with an opportunity to carry out a further run with the same program through a query of the form:

DO YOU WANT ANOTHER RUN? (Y/N)

Responding N returns the user to DOS and the DOS prompt appears.

4.4 Useful hints for using SLIM

In preparing simulation programs using the SLIM language features described in Chapter 6 and in this User's Guide it is important to take note of the following points:

- a) A SLIM program must follow the conventions explained in Chapter 6 and Section 2 of this Guide. These conventions are similar in many respects to the conventions for FORTRAN 4 or FORTRAN 77 programming.
- b) It should be noted that executable code must start in column 7 of each line of program. Columns 1 to 5 are reserved for labels or for use in comment lines. Comments must start with an upper case C in column 1.
- c) In creating a SLIM program using a general-purpose text editor, use of the TAB key should be avoided. The space bar should be used to move the cursor to the required column (e.g. column 7).
- d) No blank lines should be included in a SLIM program. The presence of a blank line in a SLIM source program causes problems at the translation stage and may cause the computer to 'hang'.
- e) Line numbers quoted in connection with error messages relate to lines involving SLIM language statements only. Lines involving comments are not given line numbers.
- f) Continuation lines are not permitted in SLIM programs and care must be taken to avoid lengthy statements which involve more than 70 character

or spaces. Attempts to use longer statements in SLIM program may cause the computer to 'hang' at the translation stage. Long statements must be split into two or more statements, each occupying a separate line of code.

- g) The distinction between integer and real quantities is important and involves the normal naming conventions of FORTRAN. Real and integer quantities cannot be used in direct combination within a single statement.
- h) All executable statements in a SLIM program must involve only upper case letters. Comments may involve upper case and lower case letters but must include an upper case C in column 1.
- i) When a syntax error is detected at the program translation stage a number of messages may be displayed which appear to relate to syntax errors in a number of different lines. If there is no apparent reason for an error in one particular line check the statements in the other lines for which syntax errors have been reported. Clearing the error reported in one line may eliminate the apparently meaningless errors reported in other lines.

4.5 Starting and exiting SLIMPLOT

SLIMPLOT is extremely easy to use. It is entered using the following command:

```
slimplot
```

The program then prompts the user for the file name of the SLIM output file which is to be plotted. This is followed by a request to the user to select the type of output file (y versus time or an x-y plot), the number of variables (in the case of an y-t plot) and the column numbers of the variables.

The column numbers referred to in the SLIMPLOT prompts are the column numbers arising from use of the TYPE statement to generate the output data from the SLIM program. For example, if the output is obtained from a statement of the form:

```
TYPE T,X1,X2,X3
```

there will be four columns in the output file. Column 1 involves values of T (the independent variable) and columns 2 to 4 involve the corresponding values of the dependent variables X1, X2 and X3. If one wanted to plot X1 and X3 the columns to be entered in response to the prompt from the SLIMPLOT program would be 2 and 4. These column numbers may be entered as:

```
2,4
```

or

```
2 4
```

Text can be inserted by the user to provide a main (top) title and up to two titles which appear below the graph. Both upper and lower case characters may be used in these titles.

When graphical output has been displayed on the screen and the user wishes to move on to a further stage of processing pressing the ENTER key clears the graphical output from the screen and causes a new menu to appear. This has one option which allows the user to exit from the SLIMPLOT program and others which allow further plots to be produced.

4.6 Useful hints for using SLIMPLOT

The following points should be noted:

- a) Problems will be encountered if the number of time steps in a SLIM output file is greater than the maximum allowed by SLIMPLOT. If the maximum is exceeded an error message is displayed during the running of the SLIMPLOT program. The SLIM program should then be re-run with an increased communication interval (CINTERVAL) or a smaller range for the independent variable. For problems where the independent variable is time this means either increasing the time step for the plot or reducing the length of the simulation run.
- b) Hard copy output from SLIMPLOT is most easily obtained using the DOS GRAPHICS command and the use of the Print Screen key. An alternative, if an appropriate printer is available, is to make use of the printer or plotter facilities in the list of output devices which is displayed during the running of SLIMPLOT.

5. EXAMPLES

Chapters 6, 8, 10, 11 and 12 of "Continuous System Simulation" include examples of SLIM programs. SLIM program files and data files for these examples are provided on this diskette. The program files all involve file names with a .SLI extension, while the input data files all involve files with a .IN extension. Filenames are the same as those used in the relevant chapters. It should be possible in each case to reproduce results presented in tabular or graphical form in the book.

6. COMPARISONS OF SLIM WITH WIDELY-USED SIMULATION LANGUAGES

SLIM is intended to provide beginners with an introduction the principles of continuous system simulation and has been designed as an inexpensive but reasonably versatile teaching tool. It is not intended to compete directly with commercially-supported languages. SLIM is most suitable for relatively small simulation problems. It does conform to many of the conventions of the CSSL specification and provides a convenient, efficient and easily-used tool. Having gained some initial experience with SLIM the user should be able to move on without difficulty to make use of other more powerful languages, such as ACSL.

The maximum number of state variables which can be accommodated within a SLIM simulation program using the version of the SLIM software which accompanies the book "Continuous System Simulation" is 40. This is not seen as imposing any serious restriction on the average beginner. All of the examples and case studies considered involve less than ten state variables. The maximum number of statements allowed within a SLIM program is 250. Once again typical applications involve much smaller programs. Some other simulation languages, such as ACSL, do not impose limitations of this kind.

While the restrictions in terms of program size are not thought to represent limits which are likely to be approached by the beginner, it is important to understand that languages such as CSSL IV, ACSL and DESIRE provide many features which are not available within SLIM. Some of these restrict the type of system which can be modelled using SLIM and others limit the options available to the user at run time. The sub-set of facilities incorporated in SLIM has been chosen to allow the beginner to gain experience rapidly, without having to deal with the complexities associated with features in more comprehensive languages. A careful examination of the user manual for each specific language allows the potential user to establish the precise differences in every case.

The absence of facilities to model a pure time delay (transport delay) is one important example of a feature, available in many simulation tools, which has not been included within the version of SLIM provided with "Continuous System Simulation". The lack of any facility to represent a function in tabular form is another example of a potentially useful feature which has not been included. Sorting facilities and facilities for sub-models are also not provided within SLIM. Simulation problems which require the use of facilities such as transport delays and function tables cannot therefore be tackled using SLIM, unless the problem can be reformulated in some way to allow it to be solved within the limitations of the language. Some other languages, such as ACSL, provide a much more extensive set of facilities and therefore provide much more flexibility at the model definition stage.

In comparison with ACSL and other similar languages SLIM has limited capabilities in terms of run-time options. There is no real separation of the model definition and the experiment in SLIM. Changes of parameters require editing of the simulation program itself or changes of an associated data file. However, flexibility is available in terms of output display facilities in SLIM because of the fact that the graphical output options involve a separate post-processing program.

SLIM can be extremely effective when learning about the principles of simulation and when used for the solution of problems for which its facilities are appropriate. It is important, however, to understand its limitations and to have an appreciation of the additional facilities of the software products which are available commercially.

. DISCLAIMER

All parties must rely on their own skill and judgement when making use of the SLIM and SLIMLOT software. Neither the author nor the publishers assume any liability to anyone for any loss or damage caused by any error or omission in the work, whether such error or omission is the result of negligence or any other cause. Any and all such liability is disclaimed.

. ACKNOWLEDGEMENTS

The features included in this version of the SLIM language are, as explained in Section 1 of this User's Guide, a subset of the features considered desirable for equation-oriented continuous system simulation languages, as recommended in 1967 by the relevant Simulation Councils inc. technical committee. In considering the features to incorporate in this simulation language, which has been designed specifically for teaching applications, a number of existing simulation tools were reviewed. One existing language which has had a particularly strong influence on SLIM is the MIMESIS language which was developed by Ian Ricketts at the University of Dundee and was described in his Ph.D. dissertation in 1977 (Ricketts, I.W. "MIMESIS - A Continuous System Simulation Language", Ph.D. Thesis, University of Dundee, 1977). SLIM and MIMESIS are both based on the principles of syntax-driven parsers as described by R.L. Gauthier and R.D. Ponto in their text entitled "Designing Systems Programs" (Gauthier, R.L. and Ponto, R.D. "Designing Systems Programs", Prentice-Hall, New Jersey, 1970). Many features of the instruction set of SLIM are similar to those of MIMESIS, although the internal structure of the SLIM program and the detailed coding are different. Experience gained in using the MIMESIS software has been of particular value in the development of SLIM and the author is very grateful to Dr. Ricketts for making MIMESIS available.

The author also wishes to record his thanks to present and former students in the Department of Electronics and Electrical Engineering at the University

of Glasgow who have assisted in the development of the SLIM software. A number of student projects have been associated with the application of SLIM to practical engineering problems and these have provided valuable information about the range of problems which can be tackled conveniently using this software. Very helpful feedback has also been provided freely by students who have used SLIM in connection with courses in computer simulation methods, in control engineering and in avionic systems at undergraduate and post-graduate level.

The software has undergone much revision and modification during the course of its development. Work on SLIM began in 1991 during a period of sabbatical leave which was spent at the Technical University of Vienna. The author is very grateful to Professor Dr. Felix Breitenecker of the Department of Simulation Techniques at the Technical University of Vienna for his assistance and support in making that visit possible and to the University of Glasgow for providing the necessary leave of absence. Without the opportunity provided by this period of uninterrupted work it is unlikely that SLIM would have been developed.

The graphical routines included in the SLIMPLOT program are part of the NAG Graphics Library. This product contains a collection of over 100 carefully designed routines which provides FORTRAN and C programmers with a versatile means of producing a graphical representation of numerical and statistical results. It is used worldwide in commerce, financial modelling, industry academic research and many other areas, and is available for a wide spectrum of machines. Interfaces are also supplied to the most commonly used plotting packages. NAG also provide a wide range of other products such as mathematical and statistical routines in FORTRAN, C, Pascal and Ada; FORTRAN 90 compilers and tools; symbolic solvers; data visualization software; linear programming software and a gateway generator which automatically generates MATLAB gateway. For further information, a quotation or details of your local distributor please contact:

Sales Department
NAG Ltd.
Wilkinson House
Jordan Hill Road
Oxford OX2 9BX
United Kingdom.

1. Was ist Fuzzy-Logik?

FUZZY-LOGIK

Methodik des sog. unscharfen
Schließens

Subjektive Unbestimmtheit von
Begriffen

Mathematisch fundierte Methode;
erlaubt graduierbare oder vage
Prädikate

So far as the laws of mathematic
refer to reality they are not cer-tain;
And so far as they are certain
they do not refer to reality
(A. Einstein)

1. Beispiel:

Tatsache: Sokrates war ein Mensch

Wissen: Menschen sind sterblich

Schlußfolgerung: Sokrates ist ge-
storben

2. Beispiel:

Tatsache: Die Tomate hat eine leicht rötliche Farbe

Wissen: Eine rote Tomate ist reif

Schlußfolgerung: Die Tomate ist nicht ganz reif

Klassische Logik versus Fuzzy Logik

Scharfe Menge charakterisiert durch zweiwertige Funktion

$$f_A(x) \rightarrow \{0,1\}$$

Unscharfe Mengen charakterisiert durch Zugehörigkeitsfunktion

$$\mu_A(X)$$

Die Funktion $f_A(x)$ wird als zweiwertig bezeichnet, da sie entweder den Wert 0 oder 1 annimmt, je nachdem ob das Element x zur Menge A gehört oder nicht. Die Menge $\{0,1\}$ ist die Wertemenge von A .

Ist als Wertemenge das gesamte Intervall $[0,1]$ zugelassen, geht die scharfe Menge A über in die unscharfe Menge A .

Eine unscharfe Menge A wird durch eine verallgemeinerte charakteristische Funktion μ_A gekennzeichnet: $X \rightarrow [0,1]$, die **Zugehörigkeitsfunktion** von A genannt wird und über einen (von Fall zu Fall geeignet festzulegenden) Grundbereich X definiert ist.

Je dichter der Wert $\mu_A(X)$ zu dem Wert 1 tendiert, desto mehr entspricht x der Charakteristik der Menge A .

DEFINITION:

Sei X eine unscharfe Menge.

Eine unscharfe Menge (fuzzy set) A über X wird charakterisiert durch eine Zugehörigkeitsfunktion $\mu_A(X)$, die jedem Element aus X eine reelle Zahl aus dem Intervall $[0,1]$ zuordnet.

Der Wert von μ_A an der Stelle X wird *Zugehörigkeitsgrad* von X zur Menge A bezeichnet.

3. Beispiel:

Sie wollen sich ein Cello kaufen. Die von Ihnen favorisierten Instrumente liegen zwischen DM 3.500,00 und 8.000,00.

GRUNDMENGE:

$$X = \{3500, 4500, 5600, 7200, 8000, 8500\}$$

UNSCHARFE MENGE:

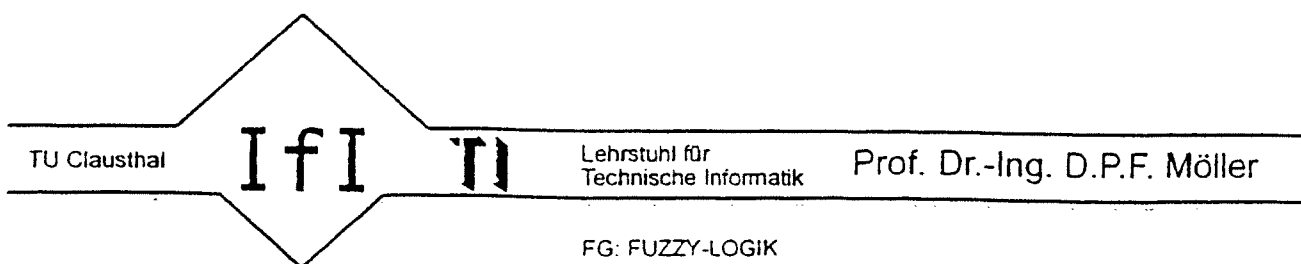
$$A = \{(3500, 0.3), (4500, 0.6), (5600, 0.9), (7200, 0.4), (8000, 0.3), (8500, 0.1)\}$$

FUZZY-LOGIK

ist ein expandierender Markt!

1988 begann in Japan die Verbreitung des industriellen Einsatzes einer neuartigen Logik, die nicht mehr auf den absoluten Aussagen der monokontextuellen Logik „wahr“ und „falsch“ bzw. „ja“ und „nein“ beruhte. Für diesen neuen Logiktyp wurde der Begriff FUZZY-LOGIK eingeführt, der am treffendsten mit „unscharfe Logik“ übersetzt werden kann.

00000000



FUZZY-LOGIC Marktanteile

- Es ist zu erwarten, daß 1995 für mehr als 2,5 Mrd DM Geräte im Bereich der Konsumelektronik verkauft werden, z. B. Audio, Video, Haushalt, etc., mit einem Anteil von 2 - 15 DM an Bauteilen der Mikroelektronik
- Es ist zu erwarten, daß 1995 der Gesamtmarkt von Geräten und Systemen mit FUZZY-LOGIK-Bausteinen eine Größenordnung von ca. 6 Mrd DM haben wird.
- Im Jahr 2000 wird der Weltmarktanteil für „FUZZY-LOGIK-HALBLEITER“ auf ca. 15 Mrd DM geschätzt

Der Erfolg der FUZZY-LOGIC erklärt sich aus den vielen Vorteilen, die diese Logik mit sich bringt. Die mit FUZZY-LOGIC realisierten Geräte sind beispielsweise wesentlich

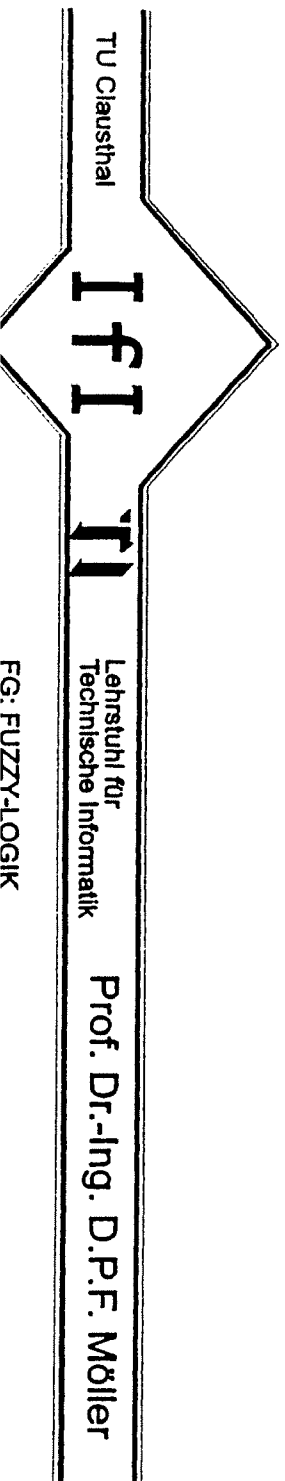
- benutzerfreundlicher als Geräte, die auf den bisherigen klassischen Verfahren basieren
- robuster und erfordern weniger Entwicklungszeit, bevor sie in den Markt eingeführt werden könnten, so daß eine Just-in-Time-Entwicklung möglich ist
- weniger Regeln als vergleichbare Expertensysteme z. B. bei Diagnoseanwendungen (Performancevorteil)

00000000

FUZZY-LOGIC basierte Regelung

- FUZZY-LOGIC wird z. Z. in hochentwickelten Systemen eingesetzt und repräsentiert bereits heute den nächsten Schritt der Entwicklung des sogenannten „Embedded Control“
- Hochentwickelte Regler sind das erste Ziel für State-of-the-Art-Controller die auf FUZZY-LOGIC basieren
- Bereits für 1994 wird von einer Kostenreduzierung für FUZZY-LOGIC-Bauteile erwartet, die es erlauben, auch das Segment für die Routine-Regelung einzubeziehen, sogenanntes LOW-END-CONTROLLER-Segment
- Insgesamt kann der FUZZY-LOGIC-Anteil im Controller-Segment bereits heute auf ca. 40 % abgeschätzt werden

2. Fuzzy-Logik in USA, J, D!



TU Clausthal

IfI TI

Lehrstuhl für
Technische Informatik

Prof. Dr.-Ing. D.P.F. Möller

USA

Realisierte Projekte:

Simulation der automatischen Ankopplung einer Station im Weltall mit FUZZY CONTROL bei der NASA

Roboteranwendungen maschinelles Sehen mittels FUZZY LOGIK bei der NASA

Zugangskontrolle mittels FUZZY LOGIK im Los Alamos National Laboratory

Reaktorsteuerung mittels FUZZY CONTROL beim MIT

TU Clausthal

IfI

Lehrstuhl für
Technische Informatik

Prof. Dr.-Ing. D.P.F. Möller

FG: FUZZY-LOGIK

Forscherguppen die sich mit der FUZZY LOGIK beschäftigen

University of Alabama, Birmingham

AT&T Bell Laboratories, Whippany

University of California, Berkeley (L.A.Zadeh)

University of Cincinnati

Florida State University, Tallhassee

City University of New York

Iona College, New Rochelle

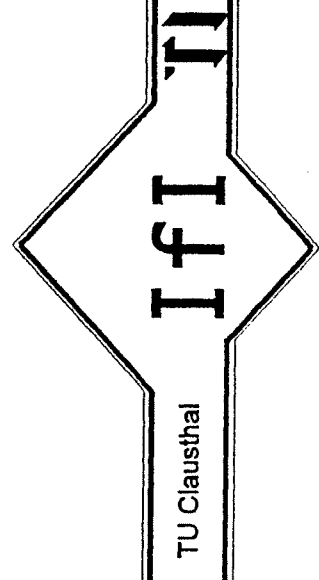
State University of New York, Binghamton

North carolina State University

University of Saskatchewan

University of Southern California (B.Kosko)

Wayne State University, Detroit



- Expertensysteme für *fuzzy pattern recognition* und *fuzzy scene analysis*,
- eine *Fuzzy-Experten-Shell* für allgemeine Anwendungen,
- unscharfe intelligente Differentialgleichungslöser,
- *Fuzzy Controller* für industrielle Prozesse,
- ein *Fuzzy-Expertensystem*, um die Produktivität bei der Software-Entwicklung zu steigern,
- unscharf vernetzte Datenbasen für "intelligent tutors" – Anwendungen,
- unscharfe Expertensysteme für Super-Computer-Anwendungen,
- theoretische Untersuchungen für unscharfe Schlußfolgerungs-Mechanismen,
- Studien über unscharfe Relationen zur Rettung verlorengangener Informationen.

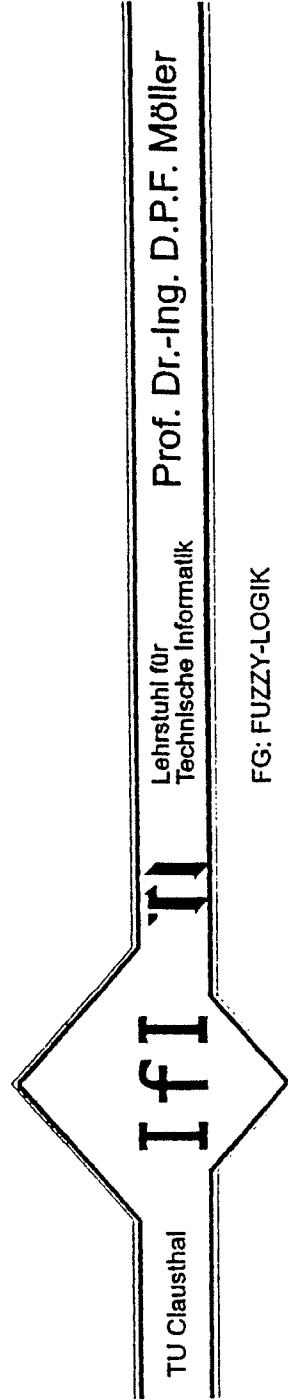
Die Aufgeschlossenenheit der Japaner gegenüber Fuzzy-Logik im Vergleich zu Deutschland (Europa) kann durch folgende Zitate begründet werden:

Prof. Bart Kosko, University of Southern California: "Fuzzyness begins where Western logic ends".

Dr. Nobori Wakami, Seniorforscher bei Matsushita: "The concept of fuzzyness is like soy sauce and sushi — a perfect match".

Prof. L.A. Zadeh: "LIFE will be the world's premier center of fuzzy logic development".

Prof. M. Sugeno, Tokyo Institute of Technology: "In expert systems here in Japan we have about 2000 projects, but only two percent or about 40 were successful. In fuzzy logic we have already had 80 successful projects out of 100. About 90 percent of those are in control engineering and the others are ordinary expert and decision support systems."



JAPAN

Steuerungs- und Regelungssysteme

Sinterofen, Zementdrehofen, Wärmeaustauscher, Druckkessel, Atomkraftwerke, automatische Kurssteuerung, Kraftfahrzeuge, Zugverkehr, Wasseraufbereitungsanlagen, Trinkwasserversorgungssysteme, Glasschmelzer, elektrische Haushaltsgeräte, Gasabsperrventile, Selbstanlasser

Künstliche Intelligenz

Mustererkennung, semantische Analyse natürlicher Sprachen, Kanji-Erkennung, Experten-Systeme, Erkennen landwirtschaftlicher Produkte, Müllsammelautomat, Pflegeroboter (Domestic Robot)

Konsultationssysteme

Häusliche Beratungssysteme, CAI, CAD, Entscheidung des günstigsten Seeweges, verschiedene Optima dynamischer Pläne (Durchlauf, Produktion, Kontrolle), Unterstützung der Betriebsleitung bei der Entscheidungsfindung, Sicherheitsbestimmung eines Gebäudes, Ausrüstungsd Diagnose, Fehlersuche, Suchgeräte zum Auffinden fehlerhafter Metalle, Qualitätstestlegungen für Nutzholz, Gesundheitspflegesystem in einer Gesellschaft mit überwiegend älteren Menschen, Hilfestellung

TU Clausthal

I f i

Lehrstuhl für
Technische Informatik

Prof. Dr.-Ing. D.P.F. Möller

FG: FUZZY-LOGIK

Medizinische Systeme

Zahnmedizinische Diagnose, Diagnose und Behandlung von Herzproblemen, Diagnose von Leberkrankheiten, medizinische Bildübertragung, Pankreas-Behandlung, Blutübertragungsberatung, Optimale Probeentnahme, Bestrahlungsmessung, Untersuchung medizinischer Behandlungsmethoden

Modelle menschlichen Verhaltens

Soziale Modelle

Analyse des in der Öffentlichkeit herrschenden Bewußtseins, Systeme zur Gesundheitspflege, ...

.

.

.

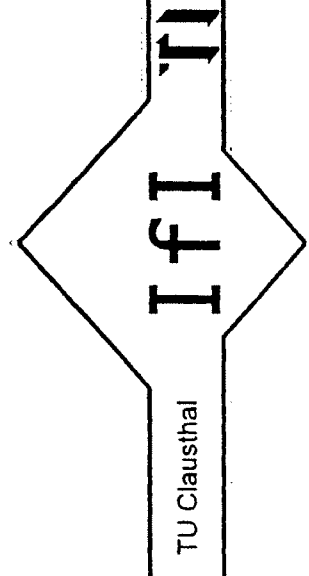
Datenbankauswertungen

Semantik

.

.

.



1. Laboratorium: Fuzzy Control

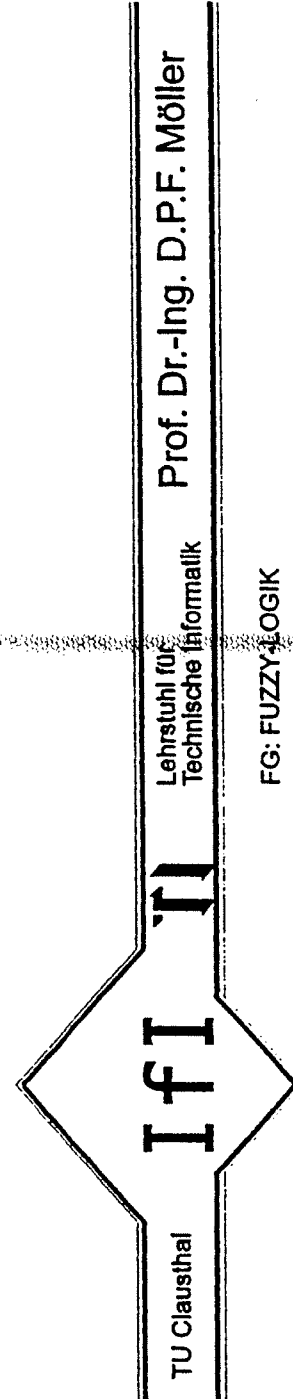
Unter der Leitung von Dr. Takeshi Yamanaka von der Omron Tateishi Electronics Co. befaßt sich dieses Laboratorium mit den folgenden Themen:

- Studium der Fuzzy Control–Theorie, mit dem Schwerpunkt auf der Untersuchung der Stabilität von Fuzzy Control–Systemen.
- Entwicklung und Aufbau von Hilfsmitteln zur Unterstützung des Entwurfs von Fuzzy Control–Systemen (Benutzeroberflächen, Entwicklungssysteme, o.ä.)
- Entwurf, Aufbau sowie Auswertung von konkreten Anwendungsbereichen für die Fuzzy Logic in Form eines "unscharfen" Roboters und eines "unscharfen" Kraftwerk–Kontroll-Systems im Labormaßstab.

2. Laboratorium: Fuzzy Intellectual Information Processing

Die Arbeitsgebiete des zweiten Laboratoriums unter der Führung von Dr. Tomohiro Takagi von der Matsushita Electric Industrial Co., Ltd. gliedern sich wie folgt:

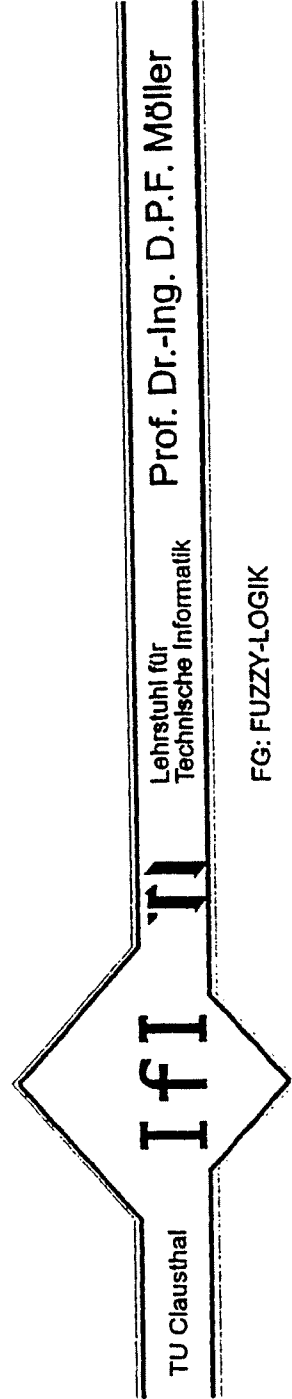
- Bildererkennung,
- Systeme zur Unterstützung von Entscheidungsfindungsprozessen,
- Fuzzy Expertensysteme.



3. Laboratorium: Fuzzy Computer

Dr. Seiji Yasonobu von der Firma Hitachi, Ltd. beschäftigt sich mit seinen Mitarbeitern im wesentlichen mit folgenden Aufgaben:

- Fuzzy Computer–Architektur,
- Fuzzy Software,
- Fuzzy Hardware.



DEUTSCHLAND

Aachen

Lehrstuhl für Operations-Research (H.J.Zimmermann)
Erforschung von Operatoren und linguistischen Modifikatoren,
mathematische Ableitung von Operatoren, Entscheidungsfindung,
Entscheidungsunterstützungssysteme, Fuzzy Set Theory, Fuzzy
Control

MIT (Management Intelligenter Technologien)

ELITE (European Laboratory for Intelligent Techniques Engineering)

Braunschweig

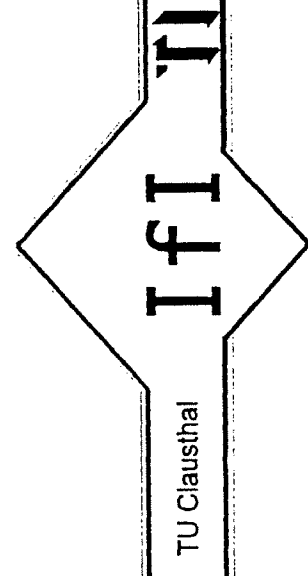
Fuzzy Arbeitsgruppe für Fuzzy Information (R.Kruse)

Chemnitz

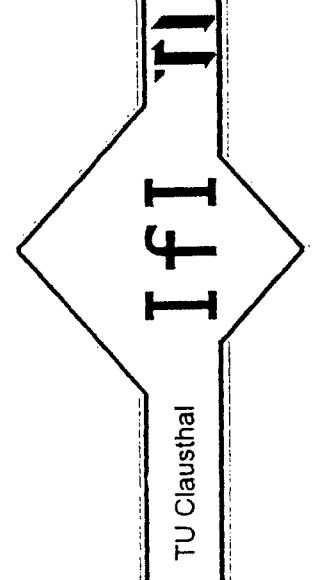
An der FhG IUW und der TU Fuzzy Klassifikation (U.Priber)

Clausthal-Zellerfeld

Lehrstuhl für Technische Informatik (D.P.F.Möller)
Prozeßautomatisierung, Medizintechnik, Datenbanken



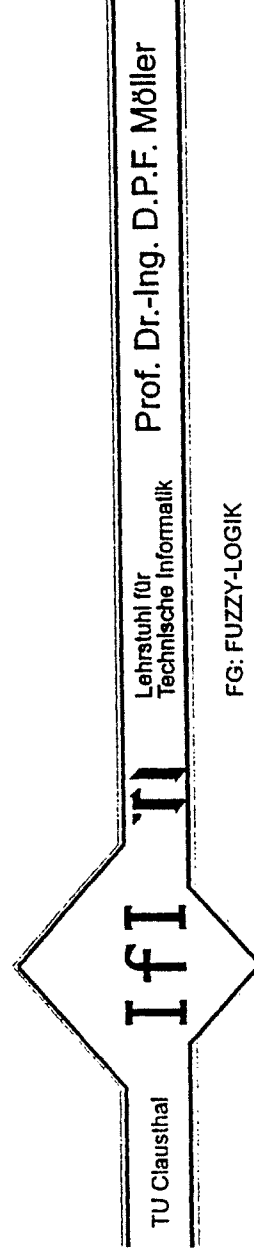
- Dortmund
Lehrstuhl für Elektrische Steuerung und Regelung (H.Kiendl)
Fuzzy Control, Stabilitätsuntersuchungen, automatische Regelengenerierung
- Lehrstuhl Informatik I (B.Reusch)
Zielorientierte Entscheidungshilfe, Schaltungsentwurf mittels Fuzzy Logik, Expertensysteme
- FDZ (Fuzzy DEMonstrationszentrum Dortmund)
- Erlangen, München, Passau
Neuronale Netze und Fuzzy Logik (FORWISS), Fuzzy Mini Messen, Special Interest Groups (P.Protzel)
- Leipzig
Fuzzy Set Theory, Fuzzifizierung von Petri Netzen, Datenanalyse mit Fuzzy-Methoden (S.Gottwald)
- München
Siemens AG, ZFE Fuzzy Task Force Gruppe (D.Reinfrank)
Prüfung der Fuzzy Logik für die Belange des Siemens Konzerns
- dto. Daimler-Benz Forschungszentrum Berlin (W.Lohnert)



Prof. Dr.-Ing. D.P.F. Möller

FG: FUZZY-LOGIK

- versität Braunschweig,
- *Fuzzy Topology*, von U. Hoele, Bergische Universität Wuppertal,
- *Theory of Fuzzy Measures* von S. Weber, Fakultät für Mathematik der Universität Mainz,
- *Evidential Conditional Reasoning*, von M. Spies, IBM Scientific Center, Heidelberg,
- *Psychological Aspects of Fuzzy Reasoning*, von A. Zimmer und H. Koerudle, Institut für Psychologie der Universität von Regensburg,
- Unscharfe Modellbildung technischer Prozesse von H.P. Lipp, TH Chemnitz,
- Unscharfe Mengen von S. Gottwald, Univ. Leipzig

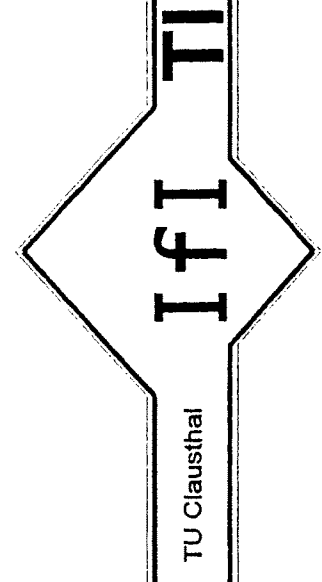


3. Strukturelle Gliederung der Fuzzy - Applikationen

TU Clausthal	IfI	Lehrstuhl für Technische Informatik	Prof. Dr.-Ing. D.P.F. Möller
TI			
FG: FUZZY-LOGIK			

1991	450	225	150
1995	1500	680	750
2000	4500	5250	2250

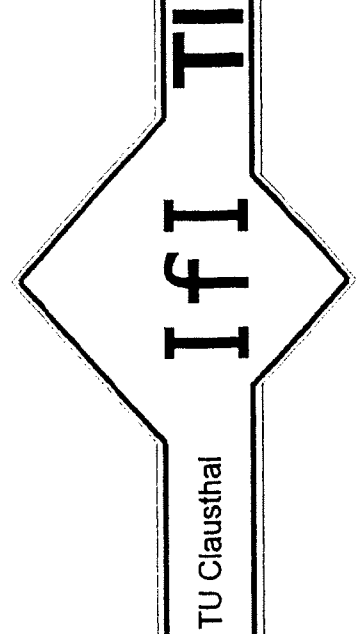
Fuzzy-Märkte (in Mio. US \$)



Robotik	6	8
Entscheidungsunterstützung	6	8
Engineering	6	7
Prozeßregelung	6	6
Spracherkennung	6	4
Risikoanalyse	6	3
Natürliche Sprachinterpret.	6	3
Textinterpretation	6	3
Medizin	5	8
FuE	5	1
Finanzen u. Ökonomie	5	1
Soziologie	4	9
Luft- u. Raumfahrt	4	8
Energie	4	4
Mathematik	4	2
Transport	4	1
Geologie	3	6
Chemie	2	4

Strukturelle Gliederung der Fuzzy-
Applikationen bezüglich ihrer kom-
merziellen Relevanz (10=dominant,
1=uninteressant)

Quelle Frost&Sullivan 1992



Prof. Dr.-Ing. D.P.F. Möller

FG: FUZZY-LOGIK

Strukturelle Gliederung der Fuzzy-Applikationen:

Umsetzungsfördernde Kriterien für Fuzzy-Control:

- einfacher Zugang zu meß- und regeltechnischen Problemstellungen auf der Basis menschlicher Anschauungsweise,
- Implementierung von Erfahrungswissen,
- potentiell Reduktion der Entwicklungszeit,
- Möglichkeit neuer oder verbesserter Funktionalität,
- Breite der Anwendungsfelder.

Indikatoren zur Identifikation sinnvoller Anwendungsfelder:

- fehlende mathematische Modelle,
- nichtlineare Problemstellungen,
- eingeschränkte Sensorik,
- totzeitbehaftete Systeme.

Umsetzungshemmende Defizite:

- Fehlen allgemeiner Entwurfs- und Optimierungsstrategien,
- Fehlen von Kriterien zur Gütebeurteilung und zum Stabilitätsnachweis,
- Fehlen von Normungs- und Zulassungskriterien.

strukturelle Gliederung der Fuzzy-Applikationen:

(ausgewählte Anwendungsgebiete)

Automobilsektor
Erfahrungstechnik
Steuerungstechnik
Maschinenbau
Mediotechnik
Fotoapparate
Haushaltsgeräte
Mustererkennung

Entwicklungstendenzen:

Intelligente Sensorik und Aktorik
Anbindung an Standard-Schnittstellen
Integration in Werkzeugumgebung
Integration in Datenbanken

Automobilsektor Im Jahre 1990, angeregt durch die Veröffentlichungen von Nissan, Subaru und Honda, beschäftigten sich alle deutschen Automobilfirmen und deren Zulieferer mit den Problemkreisen von Antiblockiersystemen, Antischlupfregelung, automatischen Schaltgetrieben, Klimaanlage sowie mit Abstandsmeßsystemen in Verbindung mit automatischem Kolonnenfahren. Auch die aktive Fahrwerksdämpfung wurde untersucht. Es gab Versuche, wie die Stabilität der Straßenlage durch ein Fuzzy-Logik-System verbessert werden kann. Eine interessante Anwendung stellt die sogenannte "Kollisions-Vermeidung" dar, die von verschiedenen Herstellern mit unterschiedlichem Erfolg untersucht wurde.

Besondere Beachtung fand das von Mazda in Japan vorgestellte automatische Viergang-Schaltgetriebe, das auch die für eine Automatik problematischen Bereiche "Bergauf-Fahren", "Bergab-Fahren" hervorragend beherrschte. Siemens berichtete anlässlich des 1. Berliner FAN-Treffens von den Erfolgen in einem umgebauten Serienfahrzeug. Das Fuzzy-System wählt zwischen den beiden Kennlinienfeldern "wirtschaftlicher Fahrer" und "sportlicher Fahrer" aus, und ist insbesondere am Berg durch das im Vergleich zur konventionellen Automatik wesentlich effizienter schaltende Getriebe deutlich überlegen.

Im Bereich der Last- und Nutzfahrzeuge wurde insbesondere die Problematik der beliebig variierbaren Gewichtsverteilung in Schaufeln, Anhängern oder Transportmedien untersucht. Potentielle Anwendungen sind z.B. das Transportieren von Mülltonnen (die unterschiedlich gefüllt sind) oder das Transportieren von Baggerschaufeln (ähnliche Anwendung).

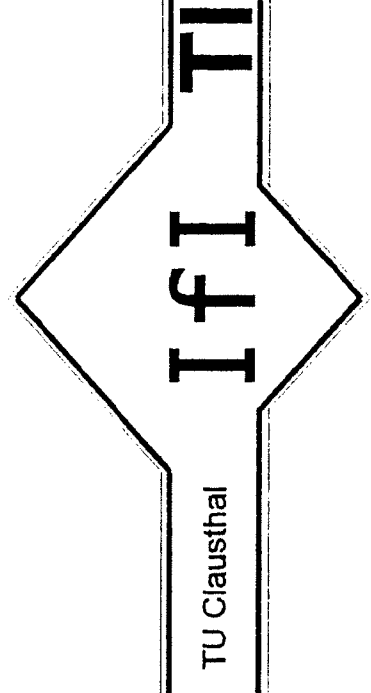
Die Implementierungen reichen hier von der reinen Softwarelösung auf den traditionellen Meßsystemen in Prototyp-Fahrzeugen bis zu bislang nur "angedachten" Implementationen auf Mikrocontroller-Ebene. Daneben gibt es Studien zu Kombinationen von Fuzzy-Logik und Neuronalen Netze, z.B. auf dem Gebiet der automatische Anfahrhilfe, des führerlosen Fahrzeugs (Volkswagen) oder für das Kolonnenfahren innerhalb des Projektes Prometheus (Opel). Hier wurden sowohl der Tempomat als auch der Abstandsregler mittels Fuzzy-Logik sehr effizient gelöst. Dabei handelt es sich hier noch um einen Versuch, vom Einsatz in der Serie sind diese Regler noch weit entfernt. Dies ist nicht zuletzt auch darauf zurückzuführen, daß die heute realisierte Abstandsmessung mit Laser noch nicht kostengünstig für die Serie zu produzieren ist.

Die Automobil-Firmen und ihre Zulieferer sind sowieso mit der Bekanntgabe ihrer Ergebnisse recht zurückhaltend, und da die Zeit bis zum Einführen eines neuen Automobilmodells relativ lange ist, werden wohl auch in der nächsten Zeit noch keine weitreichenden Ergebnisse bekannt werden.

Verfahrenstechnik in der chemischen Industrie Im chemischen Anlagenbau gibt es eine Vielzahl von Prozessen, die durch einen bewährten Anlagenfahrer bedient werden, und die mit konventionellen Expertensystemen nicht oder nur sehr schwer zu beherrschen sind. Z.B. tritt in chemischen Anwendungen eine sogenannte exotherme Reaktion auf, die mit konventionellen Reglern bedient, aber in diesem Falle nicht oder kaum noch beherrschbar ist. In diesem Falle tritt der Fuzzy-Regler parallel zum konventionell aufgebauten Regler auf und korrigiert bzw. führt den konventionellen Regler wieder in den stabilen Bereich.

Wenn man sich das Prinzip des Brennofens vor Augen führt (dieses Verfahren wurden bereits vor ca. 10 Jahren in Dänemark mit Erfolg eingesetzt), so gibt es eine Vielzahl von ähnlichen Anwendungen, wie die Ergebnisse der Firmen BASF, AKZO, BÖHRINGER und BAYER zeigen. Der Stand in diesen Unternehmen ist heute so, daß z.T. Pilotanwendungen vorliegen, über deren Vorteile auf vielen Tagungen bereits gesprochen wird. In einem solchen Pilotprojekt versucht die EC-Erdölchemie GmbH, Köln in Kooperation mit MIT (Management Intelligenter Technologien GmbH) die Entkockung der Öfen mit Hilfe von Fuzzy Datenanalyse Methoden zu automatisieren. Darüber hinaus soll das Energiemanagement des Crackers mit diesen Methoden verbessert werden.

Aber noch sind nicht alle Unternehmen bereits auf einem für die allgemeine Anwendung wünschenswerten Wissensstand. Das weite Aufgabenfeld der "wissensgeführten" Anlagentechnik läßt in Zukunft eine breite Anwendung der Fuzzy-Logik erwarten.



Prof. Dr.-Ing. D.P.F. Möller

FG: FUZZY-LOGIK

Hersteller von Steuerungen Daß die japanische Firma OMRON die Fuzzy-Logik bereits vor Jahren forciert hat, um ihre eigenen Produkte zu verbessern, darf als bekannt vorausgesetzt werden. In Deutschland war die Interkama 1992 für einige Firmen der Start in die Fuzzy-logik. Während Siemens bereits seit ca. 2 Jahren mit Togai InfraLogic mit einem Fuzzy-Hersteller intensiv zusammenarbeitet und auf der Interkama für ihre Systeme Teleperm-M fertige Fuzzy-Systeme mit bereits realisierten Anwendungen präsentierte, zeigten die Firmen Hartmann & Braun und Klöckner-Möller in Zusammenarbeit mit Herstellern von Systemen die Integration von existierenden Entwicklungsoberflächen (TILShell bei Hartmann & Braun, fuzzyTECH bei Klöckner-Möller) in ihre Systeme. Daneben sind weitere Projekte vorgesehen. Der Vorteil für diese Art der Integration liegt darin, daß der Kundenkreis der Steuerungshersteller eine erweiterte Möglichkeit hat, komplexe Systemproblematiken zu programmieren.

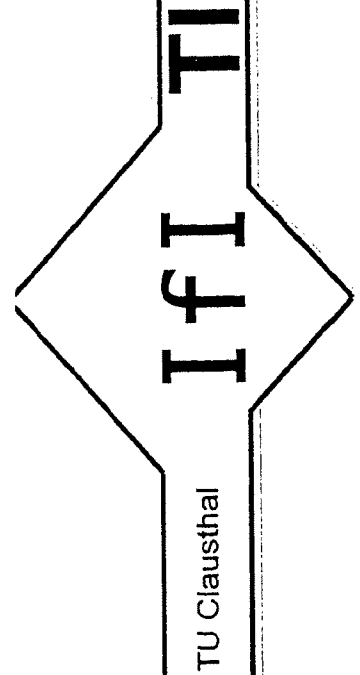
Eine weitere auf diesem Gebiet aktive Firma ist ABB in Mannheim, die die Fuzzy-Logik ihren Kunden als neues Leistungsmerkmal anbietet. Zusammen mit der Universität Zittau wurde ein Fuzzy-Controller mit wissensbasiertem analytischen Regler (WAR) entwickelt, der die Vorteile des konventionellen Reglers nutzt und gleichzeitig dessen Nachteile umgeht, indem statt der "Fuzzy-Sets" verschiedene Stellgrößen analytisch ermittelt werden.

Siemens sieht die Fuzzy-Technologie als eine der wichtigsten Kerntechnologien für den Konzern an. In einer von Dr. Reinfank im Zentralbereich ZFE geleiteten Arbeitsgruppe von 20 Personen wird Applikationsunterstützung für alle Unternehmensbereiche angeboten. Die Gruppe Automatisierungstechnik in Erlangen hat beispielsweise eine Steuerung von Zellstoffkochern in Portugal entwickelt, und zwar nicht als reine Fuzzy-Logik-Steuerung, sondern als Erweiterung konventioneller Systeme auf der Basis der Prozeßautomatisierungssysteme Teleperm-M. In Planung ist die Realisierung für die Serie Simatic S5. Siemens hat überdies mit der Fuzzy-Steuerung SIFLOC ein eigenes Produkt als Projektierungswerkzeug für Labor und reale Prozesse entwickelt.

Es zeichnet sich ab, daß die Hersteller industrieller Steuerungen Fuzzy-Komponenten in ihre Produkte integrieren und kompakte Fuzzy-Steuerungen einsetzen werden. Von den SPS-Hersteller wird oftmals bemängelt, daß die Anwender zwar die reine Steuerung mit den SPS-Geräten erledigen, aber komplexe Regelungen mit anderen Hilfsmitteln realisieren. Da oft auf die Erfahrung von praxiserprobten Regelungstechnikern zurückgegriffen wird, was sich natürlich im Preis für komplexe Anlagen auswirkt, reizt natürlich die Fuzzy-Logik die SPS-Hersteller, ihren Anwendern die Möglichkeit zu verkaufen, die Regelung durch vor Ort vorhandene Techniker einstellen zu lassen, da ja für die Fuzzy-Logik-basierten Systeme kein Systemwissen, sondern nur Anwendungswissen notwendig ist.

eine Reihe von Anwendungen in diesem Bereich realisiert:

1. Automatische Belichtungssteuerung, die insbesondere den kritischen Bereich mit Gegenlicht, Streulicht usw. abdeckt.
2. Autofokus: Fokussieren auf bewegte Objekte, insbesondere sich schnell aus dem Zentrum entfernende Objekte.
3. Kompensation von Verwackelungen: Diese elektronische Korrektur war am Anfang wohl die interessanteste Fuzzv-Anwendung. Es existieren bei den vorhandenen Systemen Methoden, den Bildinhalt elektronisch zu vergrößern, um einen Bereich für die elektronische Korrektur zu haben. Leider ergibt sich hier eine deutliche Verschlechterung der Bildqualität, da die Auflösung entsprechend dem gewählten Ausschnitt verschlechtert wird. Damit waren auch insgesamt die Testergebnisse für den elektronischen Verwackelungsschutz nicht zufriedenstellend ("man schaltet die Verwackelungsautomatik am besten ab"). Inzwischen sind allerdings gut funktionierende Verfahren bekannt, die die Verwackelung mit einem mechanischen Spiegel durch elektronische Korrektur kompensieren. Eine Verschlechterung der Bildqualität ergibt sich dadurch nicht.
4. Farbkorrektur: Die Fuzzy-Logik wird als Hilfsmittel benutzt, um eine optimale Umstellung der Farben vorzunehmen, beispielsweise wenn von einer Kunstlichtumgebung in eine Tageslichtszene gefilmt wird. Jeder Videoamateur kennt die Problematik des Blau- oder Rotstichs bei falsch gewählter Farbtemperatur.



Lehrstuhl für
Technische Informatik

Prof. Dr.-Ing. D.P.F. Möller

FG: FUZZY-LOGIK

Maschinenbau Der Maschinenbau-Markt, in Deutschland einer der wichtigsten Märkte, hat bislang noch nicht die Vorteile der Fuzzy-Logik verstanden, hauptsächlich deswegen, weil die konventionelle Regelungstechnik eine Technik darstellt, die etabliert ist und als beherrschbar gilt. Deshalb werden zunächst die Fuzzy-Systeme daraufhin untersucht, ob sie die konventionellen Systeme ersetzen können. Da die Fuzzy-Technik als Ergänzung und Erweiterung für die konventionelle Regelungstechnik zu sehen ist, insbesondere da, wo man es mit strukturstabilen Systemen zu tun hat, oder wo kein vernünftiges Modell existiert, ist dies zunächst ein unglücklicher, eher sogar falscher Ansatz.

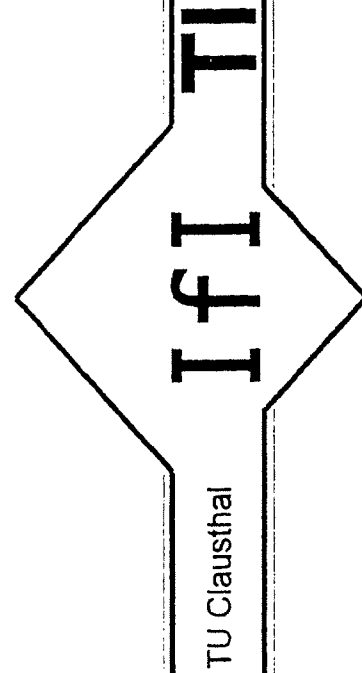
Der zweite Grund, warum die Maschinenbauer noch nicht von den Vorteilen der Fuzzy-Logik profitieren können, ist darin zu sehen, daß die Maschinenbauer traditionell ihre Elektronik zukaufen, und zwar weniger in Spezial-Lösungen, sondern bevorzugt durch die SPS-Anbieter. Die notwendigen Programmierung geschieht auf der SPS-Seite, und hierfür gibt es derzeit noch kaum Fuzzy-Logik-Unterstützung. Die einzige uns bislang bekannte Ausnahme ist die Teleperm-M von Siemens (Anwendungsbereich chemischer Anlagenbau). Eine Portierung der FPL (Fuzzy Programming Language) von Togai auf die Programmiersprache STEP5 der Siemens S5 wurde angekündigt.

Da aber die Fuzzy-Logik in den Steuerungsgeräten mehr und mehr berücksichtigt wird, dürfte hier bald eine Verbesserung der Situation erfolgen.

Haushaltsgeräte In Japan gibt es den intelligenten Reiskocher, der aus der Reis- und Wassermenge sowie der vom Bediener gewünschten Körnigkeit des Reises die optimale Garzeit einstellt - via Fuzzy-Logik mit einem kundenspezifischen Schaltkreis.

Bei Staubsaugern paßt die Fuzzy-Logik die Saugleistung dem jeweiligen Verschmutzungsgrad des zu saugenden Objekts an. Das zu saugende Objekt (Teppich, Fliesen, Polstermöbel u.a.) wird über spezielle Sensorik erkannt. Ein Modell wurde mit etwa 40.000 verschiedenen Teppichen und Untergründen trainiert, um daraus automatisch die optimale Einstellung der Saugdüsen, des Motors etc. zu erhalten. Ein einfacher Fotowiderstand mit Infrarotdiode ermittelt den augenblicklichen Staubgrad der angesaugten Luft. Damit wird automatisch die Motorleistung verringert oder erhöht.

Mit Blick auf die Verkaufserfolge in Japan werden die Hersteller von Konsumprodukten voraussichtlich auf der nächsten Domotechnika (Februar 1993) ihre Produkte auf dem Markt zu plazieren versuchen: dazu gehören Mikrowellen-Geräte, Geschirrspüler, Waschmaschinen, Staubsauger, Elektronik-Herde und ähnliche Objekte. Da in diese Produkten der Kostenanteil der Elektronik als äußerst niedrig anzusetzen ist, wird die Implementation entweder als reines ASIC realisiert oder auf 4-Bit-Mikrocontrollern basieren.



TU Clausthal

IfI TI

Lehrstuhl für
Technische Informatik

Prof. Dr.-Ing. D.P.F. Möller

FG: FUZZY-LOGIK

Mustererkennung mit Fuzzy-Logik Zur Untersuchung mechanischer Systeme wird die akustische Analyse als Stand der heutigen Technik eingesetzt. Während Erfassung und Darstellung der Daten im allgemeinen heute kein Problem mehr ist, erfordert die Interpretation und Auswertung der Messungen meist ein hohes Maß an Expertenwissen und Erfahrung, um Fehlaussagen zu vermeiden.

Bei der Firma IDS wurde ein Analysesystem für akustische Signale im Audiofrequenzbereich entwickelt. Während ein digitaler Signalprozessor das Spektrum des Meßsignals berechnet, wird die Klassifikation durch einen Fuzzy-Prozessor übernommen.



IfI TI

TU Clausthal

Lehrstuhl für
Technische Informatik

Prof. Dr.-Ing. D.P.F. Möller

FG: FUZZY-LOGIK

Intelligente Sensorik und Aktorik ~~Die Sensorik an sich stellt oftmals eine große Herausforderung dar.~~ Die meisten Sensoren ermitteln eigentlich nicht den exakt vorliegenden Wert, sondern einen anderen, oftmals verfälschten Wert. Durch Interpretation der Meßergebnisse, häufig in Verbindung mit einer Korrelation anderer Daten, versucht man eine

Schätzung vorzunehmen. Ein gutes Beispiel hierfür sind die chemischen Sensoren, die die Konzentration eines bestimmten Stoffes erkennen und meßtechnisch erfassen sollen.

Der heute oft beschrittene Weg ist meist recht aufwendig, indem man die analogen Meßdaten verstärkt, mit einem präzisen Analog-Digitalwandler in binäre Form umwandelt, um sie anschließend mit einer mathematischen Funktion über einen Rechner zu verarbeiten, wie etwa im Beispiel der schnellen Fourier-Transformation, oder mittels statistischer Verfahren. Das Ergebnis der Berechnungen wird schließlich einem Komparator zugeführt, der wiederum nur feststellt, ob das Meßergebnis innerhalb einer geforderten Toleranz liegt.

Die Integration von Fuzzy-Systemen zur Auswertung von Meßdaten in Sensoren oder zur vereinfachten Ansteuerung in Aktoren kann zu verringerten Datenübertragungsraten führen und damit die Systemkosten senken.

Anbindung an Standard-Schnittstellen Die heute vielfach gelieferte Schnittstelle in Form eines C-Quellprogrammes erfordert die richtige Parameterübergabe an das Fuzzy-Subsystem und ist für den Software-Entwickler recht transparent. Wir sind oftmals mit potentiellen Anwendern ins Gespräch gekommen, die eine Anbindung an ihr Visual Basic, ihr Kalkulationsprogramm EXCEL oder Einbindung in ihr Prozeßdatenerfassungssystem forderten.

Es gibt viele Schnittstellen in der heutigen Technik, die sich immer weiter entwickeln, oder durch neue ergänzt bzw. ersetzt werden. Man denke z.B. an die Schnittstelle IEC-Bus, die diversen Feldbus-Systeme oder aber auch an die Steckkartensysteme wie VME-Bus, ISA-Bus und SBUS. Für den Fuzzy-unerfahrenen Anwender ist es umso einfacher, diese Technologie mit einzubinden, je besser ihm die Werkzeughersteller mit der Unterstützung seiner Schnittstelle helfen. Natürlich machen nicht alle Schnittstellen in dieser Form Sinn, aber es lassen sich von der Anwenderseite diverse Schnittstellen als Schwerpunkte erkennen.

Integration in Werkzeugumgebung Der Entwickler eines Fuzzy-Systems wird vor der eigentlichen Realisierung sein Modell simulieren. Nach der Validierung seiner Lösung möchte er möglichst sofort ohne große Umwege die bislang erzielten Ergebnisse in die Praxis umsetzen. Bei einigen Werkzeugen steht der Anwender nun vor der Aufgabe, die entsprechenden Interface-Routinen selbst zu programmieren, da der Werkzeughersteller die Schnittstelle zu seiner Anwendung nicht standardmäßig implementiert hat. Findet er allerdings hier eine verbreitete Schnittstelle vor, weil er sich vielleicht bei seinem System für eine standardisierte Lösung entschieden hat, so ist die Wahrscheinlichkeit deutlich höher, vom Werkzeughersteller auch eine geeignete Schnittstelle erhalten zu können.

Integration in Datenbanken Unscharfe Datenbankabfragen können deren Verwendung in vielen Situationen vereinfachen. Man stelle sich einen reisewilligen Kunden vor, der eine Urlaubsunterkunft in Strandnähe, nicht zu weit vom Einkaufszentrum entfernt sucht.

IfI

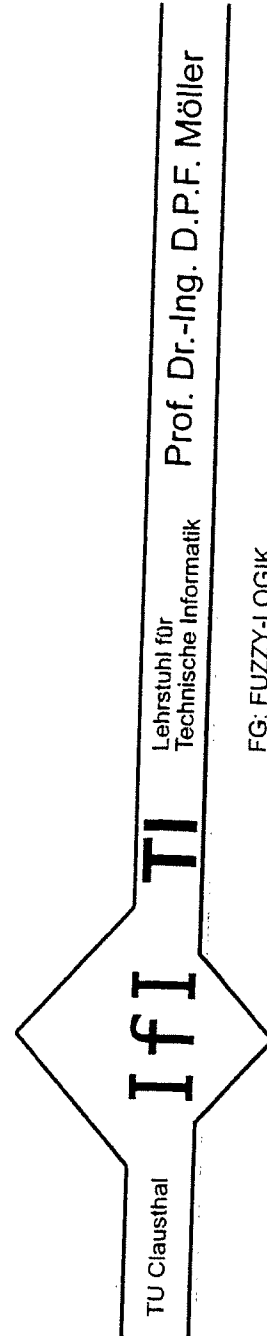
TI

Lehrstuhl für
Technische Informatik

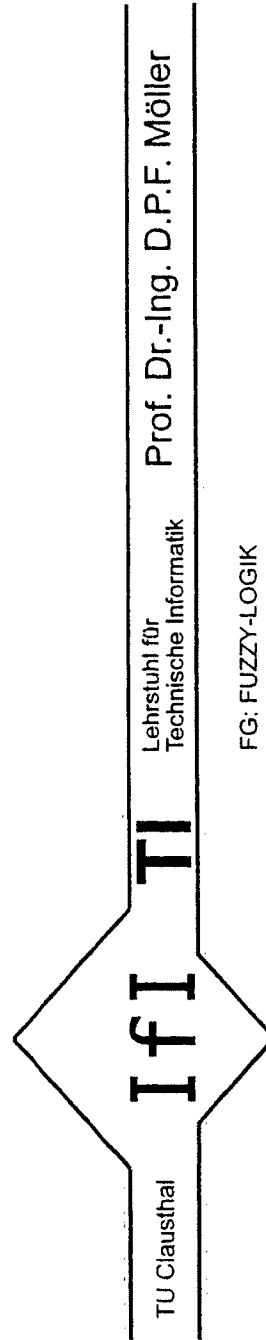
Prof. Dr.-Ing. D.P.F. Möller

FG FUZZY-LOGIK

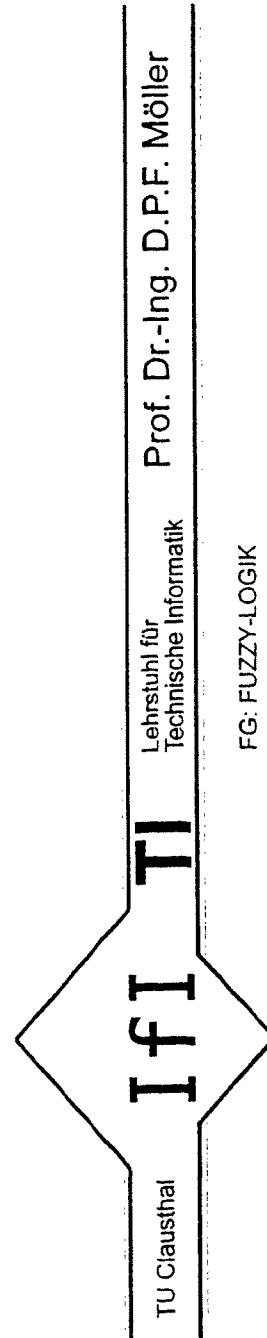
Im Bereich der Regelungstechnik wurde schon in den 70'ger Jahren ein Zementdrehofen in Dänemark mit FUZZY Logik gesteuert. In den 80'ger Jahren wurde die Technik durch die japanischen Ingenieure umgesetzt in die FUZZY Waschmaschine, den FUZZY Camcorder, den FUZZY Staubsauger. Frühzeitig wurde FUZZY Logik im Bereich des Fahrzeugbaus eingesetzt, wie dies u.a. durch eine Getriebesteuerung dokumentiert ist. Bekannt wurde auch die Steuerung der U-Bahn in Sendai, bei der durch die FUZZY Logik ein weiches Anfahren wie auch ein weiches Bremsen mit einer einhergehenden Energieersparnis realisiert wurde. Omron als ein Unternehmen mit intensiver FUZZY Logik Forschung integrierte diese Technik in seine SPS Steuerungen. In Deutschland finden sich die ersten Applikationen im Bereich der Hydrauliksteuerung beim KFZ, zur Stabilisierung von Schiffen, einer Enteisungsanlage von Flugzeugen, einer Kransteuerung u.v.m. Es läßt sich absehen, daß durch die FUZZY Logik in dem Segment der Regelungs- und Steuerungstechnik vorhandene Problemlösungen optimiert und zukünftige schneller zum Erfolg führen wird. Hierbei sind adaptive Parameteroptimierungen ein erster Einsatz z.B. bei der Anfahrcharakteristik von Autoklaven.



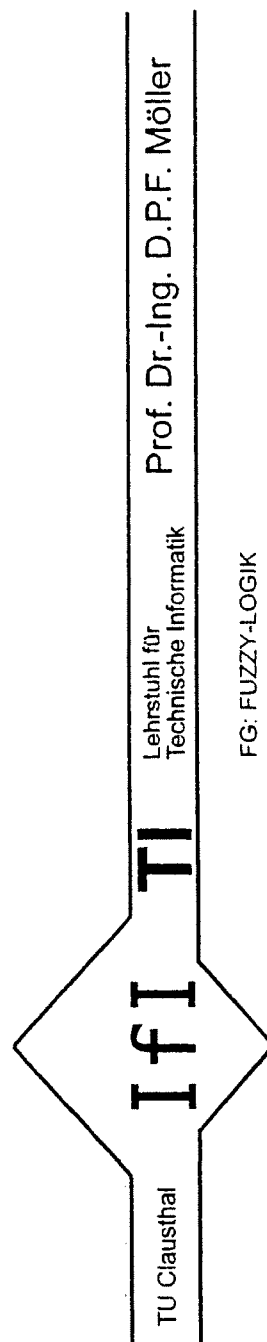
In der Meßdatenverarbeitung wird sich die FUZZY Logik auch durch den Einsatz von hybriden Hardwarelösungen durchsetzen. Unscharfen Meßdaten werden über einen FUZZY Logik Ansatz mit anderen Daten in Bezug gesetzt und liefern bei verteilten Prozeßstrukturen dem Zentralrechner nur die relevanten Meßwerte. Bei automatischer Kalibrierung in temperatursensiblen Bereichen mit hochgradig nichtlinearen Abhängigkeiten ist so eine Vorverarbeitung ohne Belastung des Hauptrechners zu realisieren.



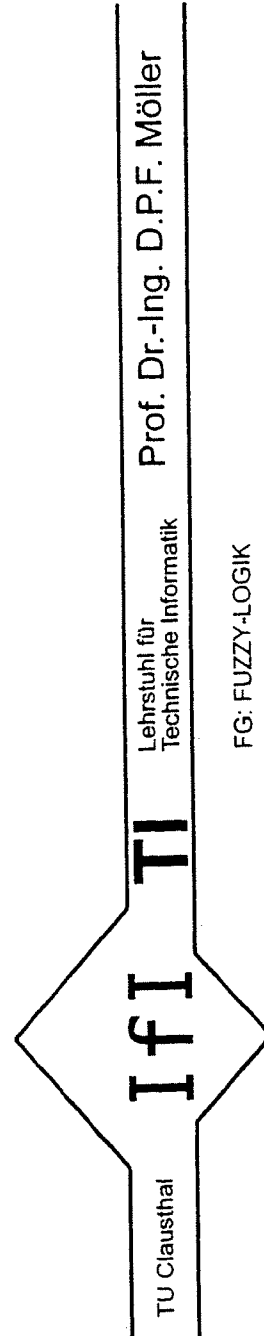
Im Bereich der Prozeßsteuerung findet die FUZZY Logik durch ihre Kombination mit den Petri Netzen Zugang zur Steuerung von Fertigungsprozessen wie sie z.B. in der Nahrungsmittelindustrie und in der chemischen Industrie zu finden sind. Unschärfen zu bewertende Größen wie sie z.B. ein Speichereintrag oder eine optimale Chargengröße darstellen, können in eine automatische Prozeßsteuerung integriert werden. Hierbei kommt durch die unscharfe Bewertung das Fachwissen des Anlagenfahrers zum Tragen, welches klassisch nicht eingebunden werden konnte. Durch geeignete Simulation läßt sich nachweisen, daß ähnlich zu der Prozeßsteuerung in der Verkehrstechnik durch die FUZZY Logik eine Harmonisierung von Verkehrsströmen herbeiführen läßt. Im Hinblick auf die Umweltbelastungen sowie auch auf die Verkehrssicherheit sind dadurch wirtschaftliche Verbesserungen erzielbar.



Im Bereich der Klassifikation sind durch die unscharfe Mustererkennung Applikationen z.B. im Bereich der Bildverarbeitung, der Erkennung von Rohrleckagen, der Qualitätssicherung bei Elektromotoren zu finden. Hierbei werden auftretende Signifikanzen einer unscharfen Bewertung unterworfen. Diese Bewertungen werden durch ein das Fachwissen repräsentierendes Regelwerk zueinander in Beziehung gesetzt. Die resultierende Größe ist dann ein Größemaß für den untersuchten Gegenstand. So kann durch eine FUZZY Logik gestützte Bewertung der R-, G- und B- Signale einer Farbkamera eine Farbzurordnung von Gegenständen erfolgen, bei denen eine konstante Farbgebung nicht gewährleistet ist. Mit einem ähnlichen Prinzip ist es möglich, die unterschiedlichen Schadstoffzusammensetzungen von Bodenproben durch eine Spektralbewertung zu bestimmen. In der Qualitätskontrolle bietet die FUZZY Logik die Möglichkeit, das Spektrum von GUT und SCHLECHT durch den Zwischenbereich zu erweitern, wobei bei geeigneter Struktur des Regelwerkes auch Tendenzen z. B. von Maschinenverschleiß frühzeitig erkannt werden.

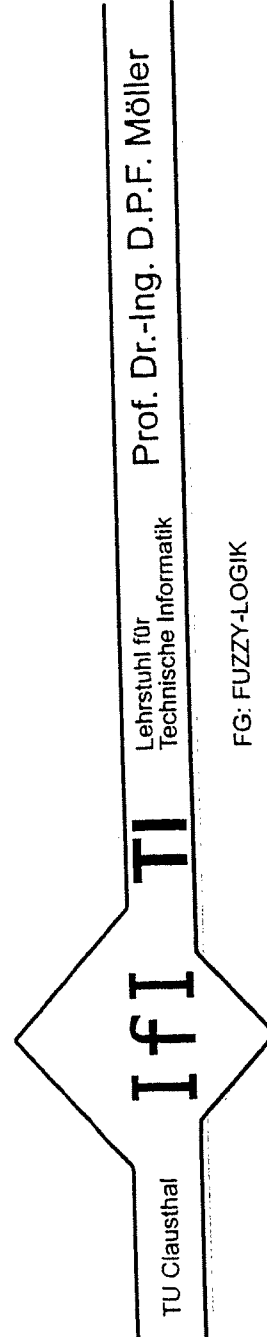


Im Bereich der Expertensysteme, wie es z. B. ein System zur Prognose des Dollarkursverhaltens ist, erweitert die FUZZY Logik die zur Zeit auf Neuronalen Netzen basierende Expertensoftware. In Kombination mit den Neuronalen Netzen wie auch mit genetischen Algorithmen liefern die Systeme zufriedenstellende Ergebnisse. Erst durch die FUZZY Logik war es möglich, spezielles, nicht in Zahlen darstellbares Expertenwissen, wie es u.a. auch ein Mediziner besitzt, mit in ein Computerprogramm zu integrieren.



Durch die offensichtliche Affinität zu den Neuronalen Netzen gibt es Entwicklungen, die eine Verbindung beider Techniken derart anstreben, daß Vorwissen über einen zu lernenden Vorgang durch die FUZZY Logik in das Neuronale Netz integriert wird. Durch diese Vorbelegung des Netzes läßt sich die Trainingszeit erheblich reduzieren. Weiterhin ist es möglich, sogenannte lokale Minima durch einen FUZZY Ansatz zu umgehen, um so eine Qualitätsverbesserung des Netzes und des Erkennens zu ermöglichen.

Mit Hilfe von Neuronalen Netzen lassen sich auf der anderen Seite FUZZY Logik Ansätze optimieren. Hierbei wird ein erster FUZZY Logik Ansatz realisiert. Mit den im realen Prozeß erhaltenen Datensätzen ist es möglich, durch ein Neuronales Netz eine neue, optimierte Regelbasis zu generieren. Diese Art des Vorgehens bieten sich bei der Umsetzung von FUZZY Logik in HARD REAL TIME Systemen an, wo Zeitrestriktionen zu beachten sind.



Bedarfsanalyse/Anwendungsgebiete

- Messen, Steuern, Regeln

Unschärfe ist in dynamischen Systemen darin begründet, daß exakte mathematische Beschreibungen entweder gar nicht existieren oder für das Anwendungsgebiet zu kompliziert sind.

Mittels Fuzzy-Logik kann diese Unschärfe modelliert werden. Die Konzepte der Fuzzy-Logik werden im MSR-Anwendungsgebiet angewendet, indem aktuell vorliegende Fakten mit einer Wissensbasis verglichen und daraus gültige Schlüsse gezogen werden. Die Wissensbasis wird in Form von WENN...DANN...-Regeln notiert. Diese gründen sich auf die Fähigkeiten und die Erfahrung des Anwenders, manchmal auch auf einfache Dimensionierungsregeln. Ihre Formulierung ist an die natürliche Sprache angelehnt.

- Entscheidungsfindung

Für eine Vielzahl von Entscheidungsproblemen liegen keine befriedigenden analytisch-numerischen Lösungen vor, da die Quantifizierung der zu berücksichtigenden Kriterien unangemessen erscheint. Mittels Fuzzy-Logik können Ziele und Randbedingungen, wie sie bei Entscheidungsproblemen vorliegen, qualitativ formuliert werden. Hierfür sind Hilfsmittel entwickelt worden, mit den Zusammenhänge zwischen Zielen, Aktionen und Wirkungen qualitativ beschrieben werden können, und die den Entscheider bei der Evaluierung möglicher Alternativen unterstützen.

-Bildverarbeitung/Mustererkennung

Anforderungen steigen hinsichtlich Leistungsfähigkeit und Genauigkeit, werden z.Z. jedoch nicht vollständig erfüllt und sind auch in naher Zukunft durch die klassischen Verfahren und Entwicklungen nicht erfüllbar. So werden z.B. klassische Kompressionsalgorithmen der Notwendigkeit einer qualitativen Auswertung der Bildinformationen nicht gerecht. Fuzzy-Logik kann in diesem Anwendungsgebiet hilfreich angewandt werden, indem sie mehrfach vorkommende Subbilder zu erkennen versucht, oder durch Vergrößerungen, die durch das menschliche Auge adaptiv ausgeglichen werden, diese Datenmengen reduzieren. Fuzzy-Logik kann somit auch in der Erkennung von Abstufungen der Helligkeit oder der Strukturierung eingesetzt werden. Rein quantitative Methoden besitzen diese Möglichkeiten in der Regel nicht.

- Datenanalyse/Klassifikation

Bei der Rechnerunterstützten Überwachung und Diagnose muß in der Regel eine Vielzahl von Meßwerten (Daten) beobachtet (erfaßt) und für Klassifikations- und Steuerungsaufgaben bewertet werden. Dabei sind verschiedene Merkmale zu beobachten, deren Ausprägungen in Klassen eingeteilt werden. Bei der Beobachtung bzw. der späteren Analyse von Meßwerten werden zumeist Toleranzbereiche für die einzelnen Merkmale angegeben, wobei aber keine eindeutige Klassenzuordnung erreicht wird. Durch qualitative Formulierung der Toleranzbereiche mittels Fuzzy-Logik (unscharfe Mengen) kann das Anwendungsproblem gelöst werden.

- Optimierung

Optimierung ist ein breites Anwendungsgebiet, z.B. Optimierung von Produktionsprozessen.

Klassische Optimierungsverfahren orientieren sich an der numerischen Bestimmung von Extremwerten einer im Regelfall analytisch beschriebenen Funktion bei gleichzeitiger Einhaltung eines Gütekriteriums (Fehlerfunktional). In der praktischen Anwendung sind qualitative Charakterisierungen wie z.B. geringe Kosten oder hohe Auslastung häufig ausreichend und analytische oder quantitative Aussagen über die Optimierungskriterien häufig nicht möglich. Die Fuzzy-Logik erlaubt es hier die Anwendungsproblemstellungen einfacher zu beschreiben und vorhandene "Toleranzräume" auszu-schöpfen.

- Produktionsplanung

In der Produktionsplanung gibt es eine Vielzahl von Einzelproblemen, die je nach Anwendungsgebiet des Produktionsprozesses zu lösen sind, wie z.B. bei der Auftragsvergabe im Falle der Fertigungsleittechnik oder das operative Produktionsmanagement für unterschiedliche Zeithorizonte und Entscheidungsebenen.

In Abhängigkeit des Anwendungsgebietes kann Fuzzy-Logik, unter Einbindung von Expertenwissen, z.B. in Form von WENN...DANN...-Regeln oder wissensbasierten Komponenten die Lösungsfindung erleichtern. Auch sind Fuzzy-Hybridlösungen unter Einbezug klassischer Verfahren, wie z.B. Petri-Netze bei stark strukturierten Prozessen einsetzbar.

3.1 Fuzzy Control

TU Clausthal

IFT

Lehrstuhl für
Technische Informatik

Prof. Dr.-Ing. D.P.F. Möller

3. FUZZY-REGELUNGSSYSTEME (FUZZY CONTROL)

Die Regelungstechnik befaßt sich mit Methoden, Ausgangsgrößen und Zustände eines dynamischen Systems auf vorgegebenen Werten zu halten oder einer Solltrajektorie nachzuführen. Daher wird bei "klassischen" Regelungssystemen für deren Synthese die Übertragungsfunktion zwischen den Ein- und Ausgängen des Reglers **quantitativ** mit Hilfe mathematischer Gleichungen beschrieben. Die Idee, Fuzzy-Logik zur **qualitativen** Beschreibung von Regelungssystemen zu verwenden, wurde zuerst von Zadeh formuliert, die Umsetzung geht auf Mamdani zurück. Mitte der 70er Jahre untersuchte er die Anwendbarkeit unscharfer Verfahren zur Automatisierung komplexer, nichtlinearer Prozesse mit großen Totzeiten, die von erfahrenen Anlagenbetreibern gesteuert wurden, sich aber einer (quantitativen) mathematischen Beschreibung entzogen. Eine der ersten erfolgreichen Realisierungen einer Fuzzy-Logik-Regelung war die Automatisierung eines Zementbrennprozesses mit einem Fuzzy-Regler vom Mamdani-Typ. Seither hat das Gebiet der Fuzzy-Regelungen vielfältige Anwendungen aufzuweisen. Bevor die spezifischen Methoden der Fuzzy-Regelung dargestellt werden, sind die regelungstechnischen Grundlagen in kurzen Zügen zu skizzieren.

3.1 Grundlegende Begriffe und Definitionen der Regelungstechnik

Ein *Regelkreis* besteht, wie aus Bild 3.1 ersichtlich, aus der Reihenschaltung eines *Reglers* und einer *Strecke* in einer *Regelschleife*. Die zu regelnden Systemgrößen werden als *Regelgrößen* bezeichnet, die Sollwertvorgaben als *Führungsgrößen*. Der momentane Wert der Führungsgröße und die aktuelle Regelgröße bilden die *Regeldifferenz*. Der Regler generiert einen *Stelleingriff*, der über Aktoren auf die Regelstrecke einwirkt.

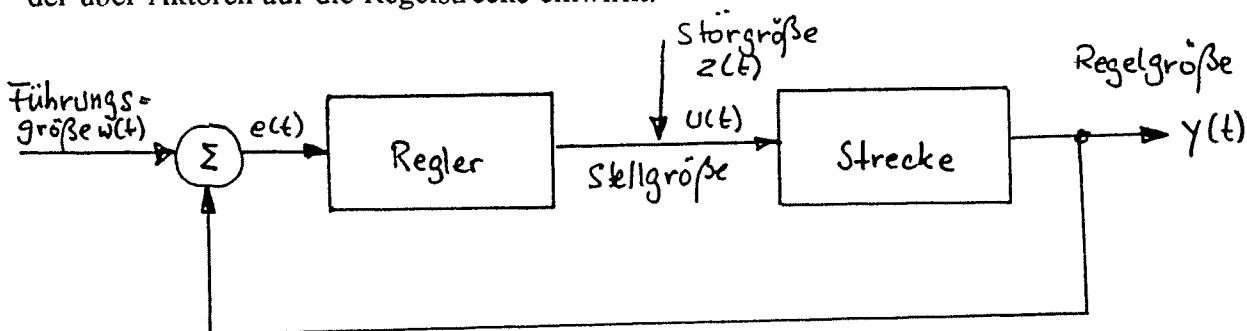


Bild 3.1 Regelkreis

Bei einer *Festwertregelung* muß die Regelgröße möglichst genau auf einen vorgegebenen Führungswert gehalten werden. Dabei müssen Störungen (die prinzipiell an jedem beliebigen Punkt angreifen können) ausgeglichen werden. Sprungförmige Änderungen der Sollwertvorgabe sollen schnell und ohne starke Oszillationen (Einschwingverhalten) ausgeregelt werden.

Bei einer *Folgeregelung* (Servoregelung) wird die Regelgröße dem Verlauf der Führungsgröße kontinuierlich nachgeführt, wobei der Entwurf auf ein geringes Schwingverhalten abzielt.

3.2 Beschreibungsformen linearer Regelungssysteme

3.2.1 Grundlagen

In der Theorie linearer Regelungssysteme kann man auf eine Fülle bewährter Verfahren zur Synthese stabiler und robuster Regelungen zurückgreifen. Voraussetzung für die Anwendung dieser Methoden ist die Verfügbarkeit eines linearen mathematischen Modelles der Regelungsstrecke, dessen dynamisches Verhalten hinreichend genau durch eine lineare Differentialgleichung (DGL) beschrieben wird.

Setzt man die **Linearität** als hervorragendes Systemmerkmal voraus, da muß bei dynamischen Regelungssystemen das **Superpositionsprinzip** gelten, welches besagt, daß die Reaktion eines linearen Systems auf eine Summe von Eingangssignalen gleich der Summe der Reaktionen auf die Einzelsignale ist. Daraus folgt:

Eingangsgröße $u_1(t)$ führt zur Ausgangsgröße $y_1(t)$

Eingangsgröße $u_2(t)$ führt zur Ausgangsgröße $y_2(t)$

und damit:

$u(t) = u_1(t) + u_2(t)$ führt zur Ausgangsgröße $y(t) = y_1(t) + y_2(t)$.

Darüber hinaus muß das **Verstärkungsprinzip** erfüllt sein:

Eingangsgröße $u_1(t)$ führt zur Ausgangsgröße $y_1(t)$

und damit:

Eingangsgröße $u(t) = c_1(t)$ führt zur Ausgangsgröße $y(t) = c_1(t)$.

Superpositionsprinzip und Verstärkungsprinzip lassen sich bei linearen Systemen zusammenfassen!

Beschreibungsformen für dynamische Regelungssysteme sind DGLn im Zeitbereich z.B. der Form:

$$y^{(n)} + a_{n-1}y^{(n-1)} + \dots + a_1\dot{y} + a_0y = b_m u^{(m)} + b_{m-1}u^{(m-1)} + \dots + b_1\dot{u} + b_0u$$

mit $m < n$, den reellen Koeffizienten a_i , b_i und $y^{(i)}$ als i -te Ableitung von $y(t)$ nach der Zeit. Die höchste auftretende Ableitung (hier n) wird **Ordnung** der DGL genannt.

Eine weitere Beschreibungsform ist die Modellierung dynamischer linearer Systeme im Laplace- bzw. Frequenzbereich. Dabei geht die lineare DGL durch Laplacetransformation über in die zugehörige **Übertragungsfunktion**:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{b_m s^m + b_{m-1} s^{m-1} + \dots + b_1 s + b_0}{s^n + a_{n-1} s^{n-1} + \dots + a_1 s + a_0}$$

Bislang wurden Beschreibungsformen für dynamische Systeme betrachtet, die daß Eingangs-Ausgangsverhalten des Systems beschreiben. Mit diesen Verfahren kann nur der Verlauf der Ausgangsgröße bei einer bestimmten Eingangsgröße ermittelt werden, jedoch nicht der das im inneren des System ablaufende dynamische Verhalten, welches durch die Zustandsgrößen des Systems beschrieben wird. Hierzu müssen **Zustandsraummodelle** angewandt werden, welche Systemen von DGLn 1. Ordnung entsprechen. Für lineare Systeme haben sie die Form:

$$\dot{\underline{X}} = \underline{A}\underline{X} + \underline{B}\underline{U}$$

$$y = \underline{c}^T \underline{X}$$

3.2.2 Klassische Reglertypen

In der regelungstechnischen Praxis haben sich drei Kategorien bewährt:

- PID-Regler (und Unterklassen)
- Kennlinien- bzw. Kennfeldregler
- Zustandsregler

3.2.2.1 PID-Regler

Der PID-Regler besteht aus einem P-Regler, dessen Übertragungsverhalten durch das Reglerfunktional

$$u = K_p e$$

beschrieben wird, d.h. die Stellgröße u ergibt sich aus der Regelabweichung e durch Multiplikation mit der Reglerverstärkung K_p (Proportional-Regler), wobei der Parameter K_p der einzige Freiheitsgrad beim Entwurf ist, sowie einem Integral-Anteil, was zum PI-Regler führt mit dem Reglerfunktional

$$u = K_p (e + 1/T_N \int e dt)$$

wobei T_N die Nachstellzeit ist, die eine Gewichtung des Integralterms erlaubt. Dieser sorgt dafür, daß sich auch geringe Regelabweichungen mit zunehmender Zeit immer stärker in der Stellgröße auswirken und die bleibende Regelabweichung verschwindet.

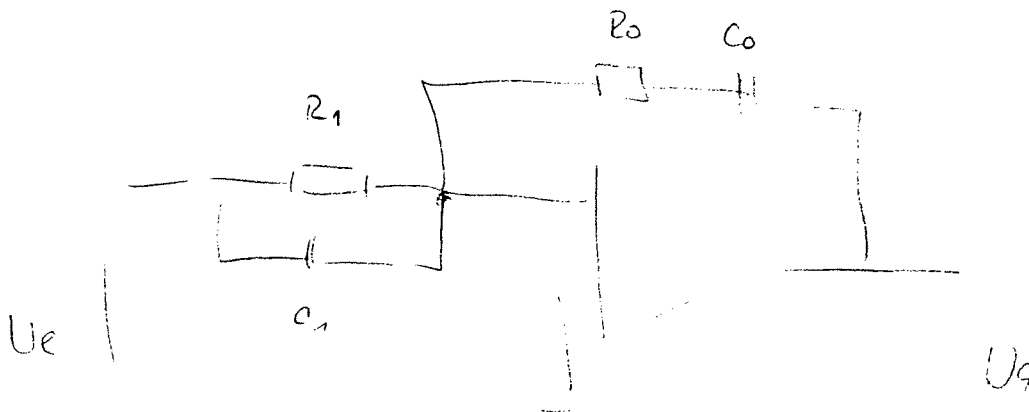
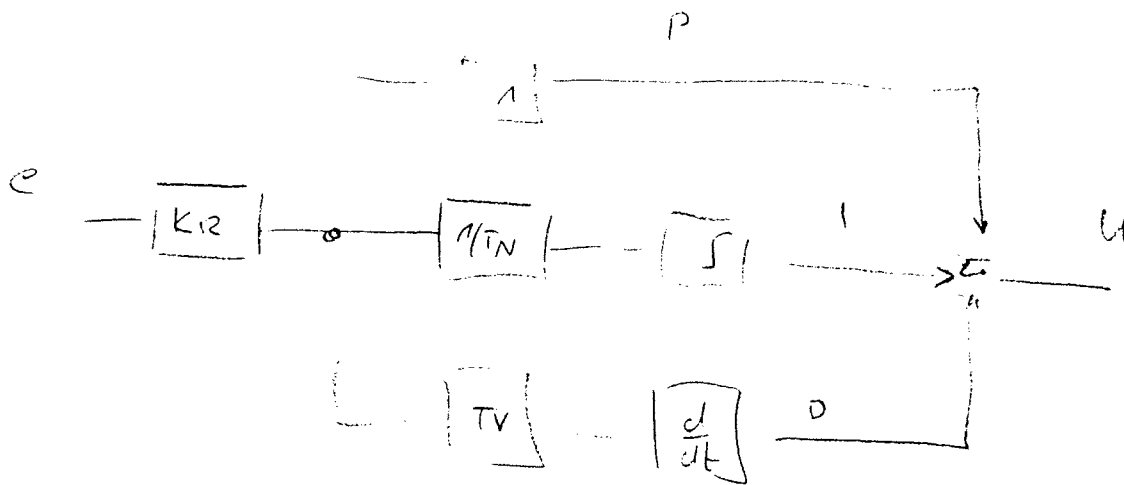
Um die Schnelligkeit des Reglers zu verbessern wird ein D-Term zugefügt und man hat den PID-Regler in der Form:

$$u = K_p (e + 1/T_N \int e dt + T_v \dot{e})$$

wobei T_V die Vorhaltzeit des D-Anteils ist. Dabei bewirkt der D-Anteil, daß sich plötzliche Änderungen der Führungsgröße, die einen unmittelbaren Einfluß auf die Regelabweichung haben, vom Regler schnell erkannt und verarbeitet werden können.

Somit kann zum PID-Regler allgemeingültig festgehalten werden:

- Der P-Anteil sorgt für allgemein günstiges Regelverhalten,
- der I-Anteil sorgt für stationäre Genauigkeit,
- der D-Anteil sorgt für schnelle Ausregelung.



$$U_g = - \left(\frac{R_0}{R_1} + \frac{C_1}{C_0} \right) U_e - \frac{1}{C_0 R_1} \int U_e dt - C_1 R_0 \cdot U_e'$$

3.3 Grundstruktur der Fuzzy-Regelungssysteme

Das Blockschaltbild in Bild 3.2 zeigt an einem Beispiel die prinzipielle Struktur eines Reglers der auf **Fuzzy-Control** basiert. Der besseren Verständlichkeit halber wird ein single input/single output fuzzy-controller (SISO) gewählt - andere Fuzzy-Regler wie z.B. MIMO, SIMO, MISO, arbeiten nach dem gleichen Prinzip-. Es gibt nur **eine Stellgröße**, also nur eine vom Regler vorzugebende Eingangsgröße des Prozesses. Die Aktionen des Reglers hängen nur von einer einzigen Eingangsgröße ab, der **Regelabweichung e** .

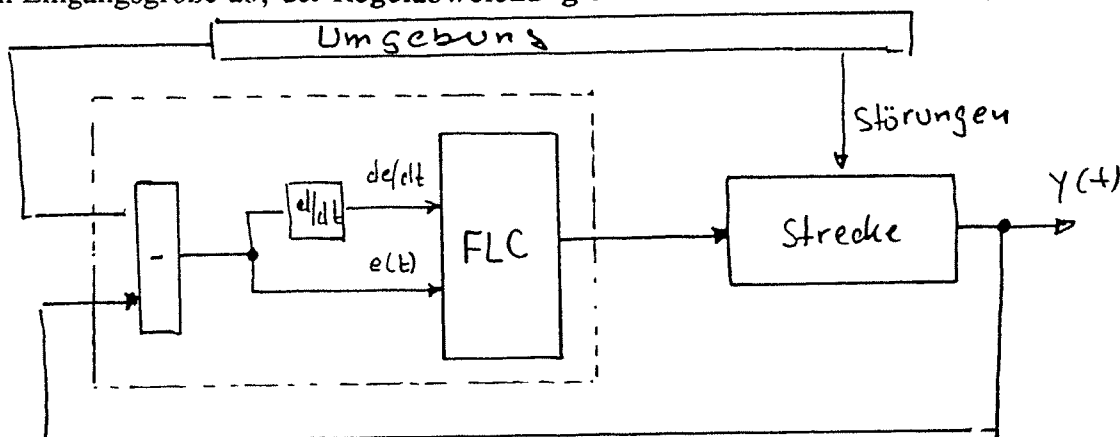


Bild 3.2. Regelung durch fuzzy controller mit e und de/dt als Eingangsgrößen

Der fuzzy controller berechnet seine Ausgangsgröße(n) aus seinen Eingangsgröße(n) anhand linguistischer Regeln. Jede linguistische Regel verknüpft:

- eine Prämisse, formuliert durch eine linguistische Aussage oder eine logische Verknüpfung mehrerer linguistischer Aussagen über die Eingangsgrößen des Reglers,
- mit einer Konklusion, ausgedrückt durch eine linguistische Aussage über die Ausgangsgröße (n) des Reglers.

Ein Fuzzy-Regler kann eine **linguistische Regel** nur dann nutzen, wenn die Aussagen in Prämisse und Konklusion folgender Beschränkung genügen: *Jede Aussage gilt dem momentanen Wert einer Größe - und nicht ihrem zeitlichen Verlauf*. Ein Fuzzy-Regler berechnet grundsätzlich den aktuellen Wert seiner Ausgangsgröße aus den aktuellen Werten seiner Eingangsgrößen arbeitet in Bezug auf seine Eingangsgrößen stets als reiner Proportionalregler.

Um dem Regler eine zeitliche Dynamik zu geben, kann man in den **linguistischen Regeln** die Ausgangsgröße vom Zeitverhalten der Eingangsgröße(n) abhängig machen. Dann ist der Regler so auszulegen, daß der Fuzzy-Regler eine Kenngröße des Zeitverhaltens dieser Eingangsgröße(n) als zusätzliche Eingangsgröße(n) erhält. Die zeitliche Ableitung der Regelabweichung sie wird mit de/dt bezeichnet, wird außerhalb des Fuzzy-Reglers gebildet, denn nur dann kann der Fuzzy-Regler eine linguistische Regel auswerten, in deren Prämisse eine Aussage über de/dt gemacht wird. Damit kann davon ausgegangen werden, daß die gewünschte zeitliche Dynamik erreicht wird. Für den Fuzzy-Regler sind e und de/dt damit zwei unabhängige Eingangsgrößen X und Y . Die Einbettung des Fuzzy-Reglers in seine Umgebung zeigt Bild 3..... Der Regler besteht aus dem eigentlichen Fuzzy-Regler und einem Differenzierer. Ein analoges Vorgehen ist erforderlich, wenn die Regelung einen integralen Anteil haben soll.

3.3.1 Aufbau und Wirkungsweise des Fuzzy Controller

Nachfolgend wird der Aufbau und die Wirkungsweise eines Reglers auf Fuzzy-Logik-Basis, der als Fuzzy Logic Controller FLC bezeichnet wird, beschrieben werden. Bild 3.3 zeigt das Blockschaltbild eines FLC mit zwei Eingangs- und einer Ausgangsgröße.

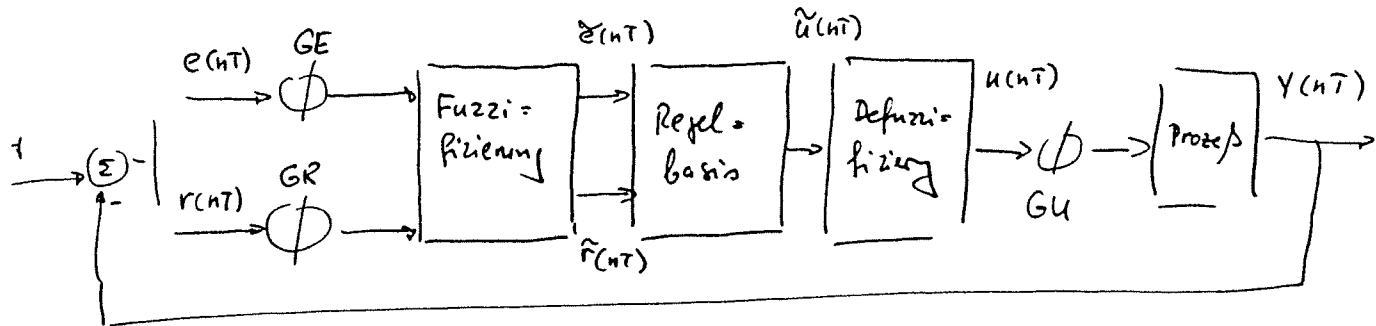


Bild 3.3 Fuzzy Logic Controller FLC

Der FLC besteht im wesentlichen aus drei Komponenten:

- der Fuzzifizierung, d.h. der Einteilung der Ein- und Ausgangsgrößen in unscharfe Mengen
- der Regelbasis mit Inferenzmechanismus, die neben den Verknüpfungsregeln zwischen Ein- und Ausgangsgrößen auch die Entscheidungslogik zur Bestimmung des unscharfen Ausgangswertes beinhalten
- der Defuzzifizierung, d.h. der Umwandlung der unscharfen Mengen für die Ausgangsgröße in einen diese unscharfen Mengen repräsentierenden diskreten Ausgangswert

Aus Bild 3.3 lassen sich somit die folgenden Beziehungen ablesen:

$$e(nT) = y(nT) - \text{Sollwert}$$

$$\tilde{e}(nT) = F[GE \cdot e(nT)]$$

$$r(nT) = [e(nT) - e(nT-T)]/T$$

$$\tilde{r}(nT) = F[GR + r(nT)]$$

$$u(nT) = GU + DF[\tilde{u}(nT)]$$

dabei ist n eine positive natürliche Zahl und T kennzeichnet die Abtastschritte. Die Werte $e(nT)$, $r(nT)$, $y(nT)$ und $u(nT)$ beschreiben den Fehler (error), die Änderung des Fehlers (rate), den Prozeßausgang (y) sowie den Fuzzy-Regler-Ausgang (u), der gleichzeitig den Prozeßeingang darstellt.

Der Fehler zum Zeitpunkt $(nT-T)$ ist gegeben durch den Term $e(nT-T)$.

GE (gain of error) und GR (gain of rate) bezeichnen die Skalierungsfaktoren für den Fehler bzw. die Änderung des Fehlers und GU (gain of controller output) den Skalierungsfaktor für die Ausgangsgröße des Fuzzy-Reglers.

Mit Hilfe diese Skalierungsfaktoren werden die physikalischen Größen in das Intervall von 0 bis 255 abgebildet, was einer 8 Bit Zahl entspricht-

F[] beschreibt die sog. Fuzzifizierung, d.h. die Einteilung der Eingangsgrößen und der Ausgangsgröße in unscharfe Klassen. Der diskrete Wert des Regleausgangs $u(nT)$ zum Zeitpunkt nT , wird durch die Defuzzifizierung DF[] der unscharfen Mengen der Ausgangsgröße berechnet.

Die Umsetzung dieser drei Komponenten hängt von der Realisierungsform des Fuzzy-Reglers ab. Einerseits können die Schritte Fuzzifizierung und Regelbais/Inferenzmechanismus durch Diskretisierung der linguistischen Variablen in Form einer (i.a. mehrdimensionalen) Relationsmatrix realisiert werden.

Durch Einbeziehung der Defuzzifizierung kann der Regler dann in eine mehrdimensionale Tabelle (eine sog. Look up-Table) überführt werden, aus welcher sich der (scharfe) Stellgrößenwert für eine bestimmte Kombination von (scharfen) Eingangsgrößen unmittelbar oder durch Interpolation von Zwischenwerten entnehmen läßt, ohne daß dazu noch irgendwelche Rechenschritte erforderlich sind. Diese **Off-Line** Realisierungsform ist für spezielle Hardwarestrukturen von besonderem Interesse.

Für die Regelbasis spielt die Wahl der Ein- und Ausgangsgrößen eine besondere Rolle. so daß die Eingangsgrößen dem jeweiligen Anwendungsfall angepaßt werden müssen. Neben der Differenz zwischen Soll- und Istwert (Fehler/error) kann z.B. auch anstelle der Ableitung dieses Fehlers die Ableitung des Istwertes als zweite Eingangsgröße verwendet werden.

Die Komponenten eines Fuzzy-Reglers mit zwei Eingangs- und einer Ausgangsgröße können somit wie folgt zusammengefaßt werden:

- den Skalierungsfaktoren GE, GR und GU für die Ein- bzw. Ausgangsgrößen
- ein Fuzzifizierungsalgorithmus für die skalierten Ein- und Ausgangsgrößen
- die Regelbasis (Fuzzy Control Rules)
- der Fuzzy-Logik zur Berechnung der Inferenz
- der Defuzzifikationsmethode, um aus der unscharfen Menge der Ausgangsgröße einen diskreten Ausgangswert zu berechnen.

Ein großer Vorteil des FLC ist begründet in dem Tatbestand, daß innerhalb dieses Reglers auch Nichtlinearitäten berücksichtigt werden könne, was mit herkömmlichen Reglern nur mit entsprechendem Aufwand möglich ist. Die Implementierungsmöglichkeiten für Nichtlinearitäten sind beim Fuzzy-Regler die folgenden:

- beim verwendeten Fuzzifizierungs-Algorithmus
- dem Fuzzy-Regelwerk
- der Art der Fuzzy-Logik, die für die Berechnung der Inferenz verwendet wird (decision making logic)
- der Defuzzifizierungsmethode, die zur Bestimmung der diskreten Ausgangsgröße herangezogen wird.

Aus der Darstellung wird ersichtlich, daß beim Fuzzy-Regler im Vergleich zu einem konventionellen PID-Regler weitaus mehr Parameter für die Variation zur Verfügung stehen, was zum einen den Vorteil einer guten Anpassungsmöglichkeit des Reglers an spezielle Anforderungen beinhaltet, zum anderen jedoch die Gefahr mit sich bringt, daß bei der Reglerauslegung nur eine suboptimale Einstellung erreicht wird. Aus dieser kurzen Darstellung wird deutlich, daß auf dem Gebiet der Fuzzy-Logik nach wie vor ein Bedarf an theoretischer Arbeit vorhanden ist um z.B. Kriterien zur Reglerauslegung bzw. Stabilitätsbetrachtungen entsprechend den konventionellen Reglern durchführen zu können.

Eine weitere Realisierungsvariante für FLC besteht darin, die Stellgröße für die aktuelle Kombination von Eingangsgrößen jeweils **On-Line** zu berechnen. Dazu geht man wie folgt vor:

- Ermittlung des Erfüllungsgrades für die einzelnen Prämissen (WENN-Teile) der Regel
- Verknüpfung der einzelnen Erfüllungsgrade über den UND- bzw. ODER-Operator (z.B. MIN- bzw. MAX-Operator)
- Regeln mit einem Erfüllungsgrad $E > 0$ sind aktiv.
- Ermittlung der zugehörigen Stellgrößen-Fuzzy-Mengen für alle aktiven Regeln
- Ermittlung der resultierenden Stellgrößen-Fuzzy-Mengen durch Überlagerung aller Stellgrößen-Fuzzy-Sets,
- Ermittlung der scharfen Stellgrößen durch Defuzzifizierung.

Diese allgemeine Vorgehensweise kann verdeutlicht werden an folgendem Beispiel:

Für FLC mit zwei Eingangsgrößen $e_1 = 0,25$ und $e_2 = -0,3$ soll die Stellgröße bestimmt werden. Es sei angenommen, das nur zwei Regeln aktiv sind und zwar:

R_1 : WENN $e_1 = \text{PK}$ UND $e_2 = \text{NU}$ DANN $u = \text{PK}$

R_2 : WENN $e_1 = \text{N}$ UND $e_2 = \text{NK}$ DANN $U = \text{N}$

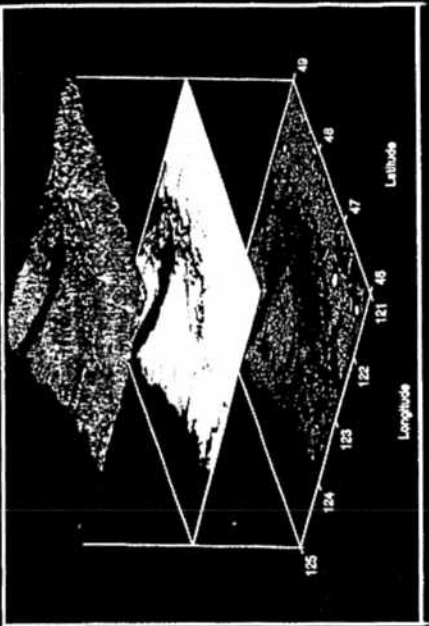
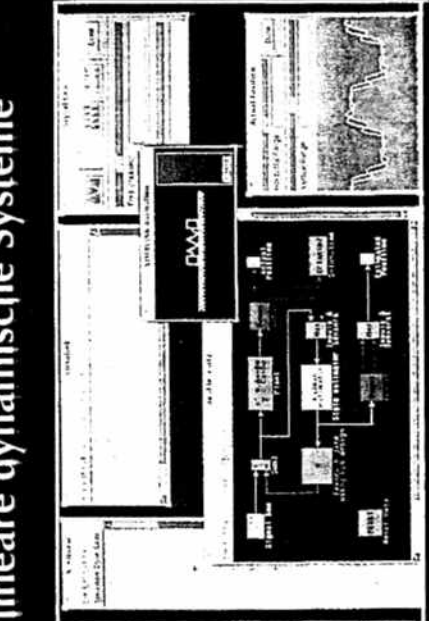
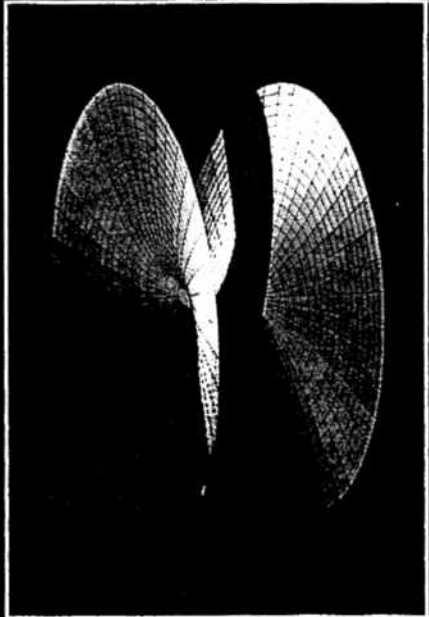
mit PK = positiv klein
 NU = null
 NK = negativ klein

Anhand des nachfolgenden Bildes soll die Vorgehensweise schrittweise erläutert werden:

Die Software für math.-techn. Berechnungen

In MATLAB integriertes Simulationssystem für nicht-lineare dynamische Systeme

Anwendungsspezifische Ergänzungen für MATLAB



MATLAB® für Ingenieure und Naturwissenschaftler. Einfach anzuwenden. Ersetzt aufwendige Eigenprogrammierung.

Anwendungsgebiete:

- Gleichungsdefinition, Matrizenarithmetik
- Grafische Darstellung, 2D+3D
- Gleichungsbasierte Simulation nichtlinear. Systeme
- Auswertung von Versuchsdaten, Visualisierung, Animation, Algorithmen-Entwicklung
- Formelauswertung, Statistik
- Eigenwertrechnung, Polynomarithmetik

Eigenschaften:

- Interaktive Anwendung, einfache Syntax
- PCs, Workstations und Mainframes
- Eigene Funktionen mit Fortran und/oder C
- Speichern und Wiederverwend. benutzereig. Funktionen
- Lesen und Schreiben beliebiger Dateiformate
- MATLAB ab **DM 1.600,-**, TOOLBOXEN ab **DM 990,-**

SIMULINK™ für die grafische blockbildbasierte Modellierung, Analyse und Simulation.

Modellierung:

- Lineare, nichtlineare, kontinuierliche und diskrete Modellteile in einem Modell
- Blockorientierte grafische Eingabe, aufbauend auf MS-Windows (PC), X/Motif (Unix-Workstation) oder Macintosh Windowing
- Teilmodelle, Zahl der Hierarchie-Ebenen praktisch unbegrenzt, viele Standardblöcke verfügbar
- Eigene Blöcke in MATLAB-, C- oder Fortran-Code
- Speicherung in lesbarem MATLAB-Code

Systemuntersuchung:

- Bestimmung des eingeschwungenen Zustands
- Linearisierung nichtlinearer Modelle
- Parameteroptimierung, Reglerentwurf, Signalanalyse mit MATLAB-Toolboxen
- Generierung von C-Quellcode: C Code Generator

TOOLBOXEN (TB) zur Ergänzung von MATLAB und SIMULINK mit leistungsfähigen, fachspezifischen Zusatzfunktionen.

Signalverarbeitung: Signal Processing TB

Regelungstechnik und Systemidentifikation:

Control System TB, Nonlinear Control Design TB, Robust Control TB, u-Analysis and Synthesis TB, System Identification TB, State Space Identification TB

Simulation mechanischer Systeme: MECHMACS (Ergänzung zu SIMULINK)

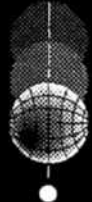
Meßdatenerfassung, -verarbeitung, Steuerung,

Regelung in Echtzeit:

ECHTZEIT-ERWEITERUNG (Ergänzung zu MATLAB und SIMULINK)

Universell einsetzbar: Optimization TB, Neural Network TB, Chemometrics TB, Spline TB, Statistics TB, Image Processing TB, Symbolic Math TB

Software mit Zukunft



scientific computers

D-52064 Aachen, Franzstraße 107, Tel. 0241/260 41, Fax 0241/449 83
Geschäftsstelle München:
D-85744 Unterföhring, Firkenweg 7, Tel. 089/99 59 01-0, Fax 089/99 59 01-11



ARGESIM Report no. 6

COMETT - Course "Object-Oriented Discrete Simulation"

Seminar Modellbildung und Simulation EUROSIM'95 Seminar

N. Kraus, F. Breiteneker

**in ISBN ebook 978-3-901608-04-9 (3-901608-04-4)
DOI 10.11128/arep.4-5-6.ar6**

© 1995 ARGESIM

ISBN 3-901608-06-0

ARGESIM Report No. 6

in ISBN ebook 978-3-901608-04-9 (3-901608-04-4)

DOI 10.11128/arep.4-5-6.ar6

ARGE Simulation News (ARGESIM)

c/o Technical University of Vienna

Wiedner Hauptstr. 8-10

A-1040 Vienna, Austria

Tel: +43-1-58801 5386, 5374, 5484

Fax: +43-1-5874211

Email: argesim@simserv.tuwien.ac.at

WWW: <URL:<http://eurosim.tuwien.ac.at/>>

FOREWORD

At present there is a big need for automatisisation of production processes, of manufacturing processes, of control systems, etc. For automatisisation at each level different hardware and software tools are offered. It turns out that simulation is the most general tool offering not only certain software but also methodology and know-how for automatisisation

Simulation is the process of designing a computerised model of a system and experimenting with this model in order to understand better the behaviour of the system, or to locate specific problems, or to evaluate different strategies for the operation of the system, or to plan a new process.

This course offers an introduction for engineers, scientists, and interested people from other fields to modern methods of computer simulation of and for automatisisation.

Due to the nature of the investigated processes this course deals on the one side with discrete simulation and optimisation in discrete simulation - in order to handle the planning and scheduling problems in e.g. production systems.

On the other hand this course gives from the technical point of view insight into simulation tools for controlling and optimising machines like robots, in order to increase the efficiency of stations, robots, AGVs, etc. Especially the modern technique of fuzzy control and simulation will be considered.

The course consists of three independent units:

- Introduction into Discrete Simulation
- Simulation and Automatisisation with Object-Oriented Tools (this report)
- Fuzzy Control for Automatisisation

The first unit "Introduction into Discrete Simulation" is a basic unit introducing into the concepts of discrete simulation, and into applications for automatisisation. Also mathematical background (statistic) will be sketched and modern methods for optimisation (genetic algorithms).

This unit adresses newcomers in simulation and automatisisation. In this unit the classical simulation tool GPSS/H and the PROOF animation system will be used.

The second unit "Simulation and Automatisation with Object-Oriented Tools" addresses people from application areas giving an overview about modelling,

simulating and optimising discrete processes (manufacturing processes, etc.) by means of modern powerful object-oriented software tools.

Application and case studies presented in this unit work with the modern simulator SIMPLE++, which allows to formulate optimisation and automatisation strategies by means of methods. As this unit is based on applications, only little or no previous knowledge is necessary.

The third unit "Fuzzy Control for Automatisation" deals with the technical aspects of automatisation and addresses (control) engineers or people from related areas. After a short introduction into basic principles of (optimal) control for automatisation (of machines, robots, etc.) principles and applications of the new tools *Fuzzy Control* are considered.

Specific software tools and general ones (MATLAB) will be discussed. Although this unit deals with technical aspects and seems to be independent from the world of discrete processes, it completes the first and the second unit from the view of automatisation.

After the course a participant should be able to make decisions about the use of modelling and simulation methods in the area of automatisation, to work with simulation tools for automatisation purposes, and to decide on an efficient use of simulation for a specific automatisation

All three units of the COMETT Course are available as ARGESIM Reports.

We wish to thank AESOP for making available the material for the second unit of this course.

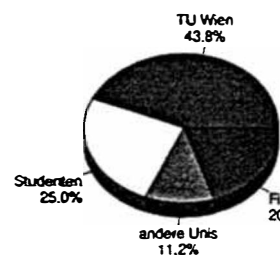
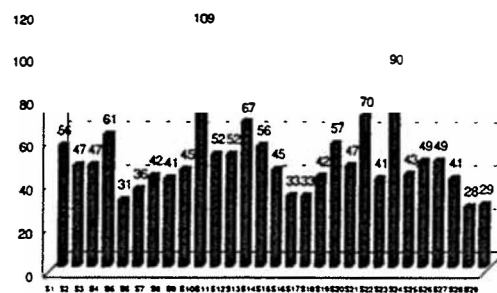
Seminare über Modellbildung und Simulation

Seit dem Frühjahr 1991 veranstaltet das EDV-Zentrum gemeinsam mit der Abteilung Regelungsmathematik und Simulationstechnik des Instituts für Technische Mathematik und der ARGE Simulation News (ARGESIM) Vortragsveranstaltungen zum Thema Modellbildung und Simulation (Simulationsseminare). Organisatoren sind I. Husinsky und F. Breitenacker. Das Ziel ist, verschiedene Simulationswerkzeuge vorzustellen, über ihre Einsatzmöglichkeiten zu informieren und Erfahrungen auszutauschen. Ferner werden bekannte Simulationsfachleute eingeladen, Grundsatzvorträge zum Thema Simulation zu halten. Im allgemeinen werden die Seminare teilweise von Firmen gesponsert oder über Simulationsprojekte mitfinanziert. Sie dauern einen halben oder einen Tag, es gibt schriftliche Unterlagen zu den Vorträgen und Softwareprodukten. Ein Buf-fet fördert die Kommunikation zwischen den Seminarteilnehmern in den Pausen.

Bis jetzt haben folgende Seminare stattgefunden:

S1	23. 4. 1991	ACSL
S2	4. 6. 1991	CTRL C XANALOG
S3	22. 10. 1991	SIMUL R
S4	5. 5. 1992	ACSL
S5	6. 5. 1992	MicroSaint
S6	17. 6. 1992	Objektorientierte Modellbeschreibung und qualitative Simulation (F. Cellier University of Arizona)
S7	1. 7. 1992	Diskrete Simulation und Analyse (D. Kelton, University of Minnesota)
S8	23. 10. 1992	GPSS/H (T. Schriber, University of Michigan)
S9	10. 12. 1992	SIMPLE
S10	2. 2. 1993	MATLAB und SIMULINK
S11	25. 3. 1993	Modellbildung mit Bondgraphen (D. Kamopp, University of California)
S12	24. 5. 1993	MicroSaint
S13	22. 6. 1993	ACSL
S14	21.10.1993	XANALOG SIMNON
S15	22.10.1993	GPSS/H (T. Schriber, University of Michigan)
S16	11.11.1993	IDAS
S17	7.12.1993	SIMPLE++
S18	14.12.1993	Petrinetze, D_SIM (R. Hohmann, Magdeburg)
S19	4.2.1994	Modellbildung und Simulation in der Lehre
S20	14.3.1994	GPSS/H und Proof (T. Schriber, University of Michigan)
S21	13.4.1994	ACSL
S22	10.5.1994	SIMUL_R, Partielle Differentialgleichungen
S23	22. 11. 1994	MATLAB/SIMULINK
S24	14.12.1994	SIMPLE++
S25	31.1.1995	Parallele Simulation, mosis
S26	28.3.1995	ACSL
S27	29.3.1995	MicroSaint
S28	13.6.1995	COMETT II, Part one, Discrete Simulation
S29	28.6.1995	COMETT II, Part two, Simulation and Automatisierung

Teilnehmer(angemeldet)



Die Teilnehmer, etwa 30 bis 110 je Seminar, kommen zum Großteil von der TU, aber auch von anderen Universitäten und aus der Industrie. Bei den bisherigen Seminaren waren etwa 20% der Teilnehmer aus der Industrie.

Das Programm eines Seminars setzt sich im allgemeinen aus einem oder zwei Grundlagenvorträgen, mehreren Anwendervorträgen, Produktpräsentationen, Vorführungen am Rechner und Diskussionen zusammen.

Die Teilnehmer werden um eine Anmeldung gebeten, daher können die Unterlagen (Seminarberichte), die zu Beginn des Seminars verteilt werden, schon eine Teilnehmerliste enthalten. Ab Herbst 1995 erscheinen die Unterlagen als ARGESIM Report. Alle, die bereits an einem Seminar teilgenommen haben, werden automatisch zu den weiteren Seminaren eingeladen.

Information:

I. Husinsky, EDV-Zentrum, Technische Universität Wien, Wiedner Hauptstr. 8-10, A-1040 Wien,
Tel: (0222) 58801 5484, Fax: (0222) 587 42 11,
E-Mail: husinsky@edvz.tuwien.ac.at

Prof.Dr. F. Breitenacker, Abt. Regelungsmathematik u. Simulationstechnik, Inst. 114, Technische Universität Wien, Wiedner Hauptstr. 8-10, A-1040 Wien,
Tel: (0222) 58801 5374, Fax: (0222) 587 42 11,
E-Mail: fbreiten@email.tuwien.ac.at

About ARGESIM

ARGE Simulation News (ARGESIM) is a non-profit working group providing the infra structure for the administration of **EUROSIM** activities and other activities in the area of modelling and simulation.

ARGESIM organizes and provides the infra structure for

- the production of the journal **EUROSIM Simulation News Europe**
- the comparison of simulation software (**EUROSIM Comparisons**)
- the organisation of seminars and courses on modelling and simulation
- **COMETT Courses on Simulation**
- "Seminare über Modellbildung und Simulation"
- development of simulation software, for instance: mosis - continuous parallel simulation, D_SIM - discrete simulation with Petri Nets, GOMA - optimization in ACSL
- running a WWW - server on **EUROSIM** activities and on activities of member societies of **EUROSIM**
- running a FTP-Server with software demos, for instance
 - * demos of continuous simulation software
 - * demos of discrete simulation software
 - * demos of engineering software tools
 - * full versions of tools developed within ARGESIM

At present ARGESIM consists mainly of staff members of the Dept. Simulation Technique and of the Computing Services of the Technical University Vienna.

In 1995 ARGESIM became also a publisher and started the series **ARGESIM Reports**. These reports will publish short monographs on new developments in modelling and simulation, course material for COMETT courses and other simulation courses, Proceedings for simulation conferences, summaries of the **EUROSIM** comparisons, etc.

Up to now the following reports have been published:

No.	Title	Authors / Editors	ISBN
# 1	Congress EUROSIM'95 - Late Paper Volume	F. Breitenecker, I. Husinsky	3-901608-01-X
# 2	Congress EUROSIM'95 - Session Software Products and Tools	F. Breitenecker, I. Husinsky	3-901608-01-X
# 3	EUROSIM'95 - Poster Book	F. Breitenecker, I. Husinsky	3-901608-01-X
# 4	Seminar Modellbildung und Simulation - Simulation in der Didaktik	F. Breitenecker, I. Husinsky, M. Salzmann	3-901608-04-4
# 5	Seminar Modellbildung und Simulation - COMETT - Course "Fuzzy Systems and Control"	D. Murray-Smith, D.P.F. Möller, F. Breitenecker	3-901608-04-4
# 6	Seminar Modellbildung und Simulation - COMETT - Course "Object-Oriented Discrete Simulation"	N. Kraus, F. Breitenecker	3-901608-04-4
# 7	EUROSIM Comparison 1 - Solutions and Results	F. Breitenecker, I. Husinsky	3-901608-07-9
# 8	EUROSIM Comparison 2 - Solutions and Results	F. Breitenecker, I. Husinsky	3-901608-07-9

For information contact: ARGESIM, c/o Dept. Simulation Techniques,
attn. F. Breitenecker, Technical University Vienna
Wiedner Hauptstraße 8-10, A - 1040 Vienna
Tel. +43-1-58801-5374, -5386, -5484, Fax: +43-1-5874211
Email: argesim@simserv.tuwien.ac.at

TABLE OF CONTENTS

Foreword	iii
Seminare über Modellbildung und Simulation	v
About ARGESIM	vi
SIMPLE++ Overview	1
Profit from PROFIT	34
SIMPLE++ Tutorial	44

The New Class of Simulation Software

The logo for SIMPLE++ is a dark rectangular box with a textured, grainy background. The text "SIMPLE++" is centered within the box in a bold, sans-serif font.

- The standard software for object-oriented, graphical and integrated modeling, simulation and animation.



**AESOP GmbH, Königstraße 82
70173 Stuttgart, Germany
Tel: +49-711-16 359 / 0
Fax: +49-711-16 359 / 99**

AESOP Partner in Austria :



**Unsel + Partner, Lerchenfelderstr. 44/9
A-1080 Vienna, Austria
Tel: +43-1-40 30 371
Fax: +43 -1-40 30 371 / 90**

SIMPLE++ Overview

- 1. Abstract**
- 2. SIMPLE++ the product**
 - 2.1 Design philosophy**
 - 2.2 Innovative edge**
 - 2.3 Description**
 - 2.4 Specifications**
 - 2.5 Optional products**
 - 2.6 History**
- 3. SIMPLE++ - the range of application and some examples**
- 4. AESOP - the vendor of SIMPLE++**
- 5. Unseld + Partner. AESOP Partner in Austria**

1. Abstract

In the past, simulation software lacked user friendliness and functionality and therefore, the market penetration is still very low despite of the outstanding profitability of using simulation. SIMPLE++ has been developed to overcome these deficits in an innovative manner and its success in the market place shows that SIMPLE++ has broken down former barriers.

AESOP developed SIMPLE++ together with the Fraunhofer Institute for Production and Automation (IPA) and is based on more than 20 years of experience in developing and applying simulation software. SIMPLE++ was introduced to the market place in February 1992. Since its introduction, was installed over 200 times in the German speaking market in the first two years and became one of the market leaders. Since April 1993 the full English version of SIMPLE++ has been shipped world-wide.

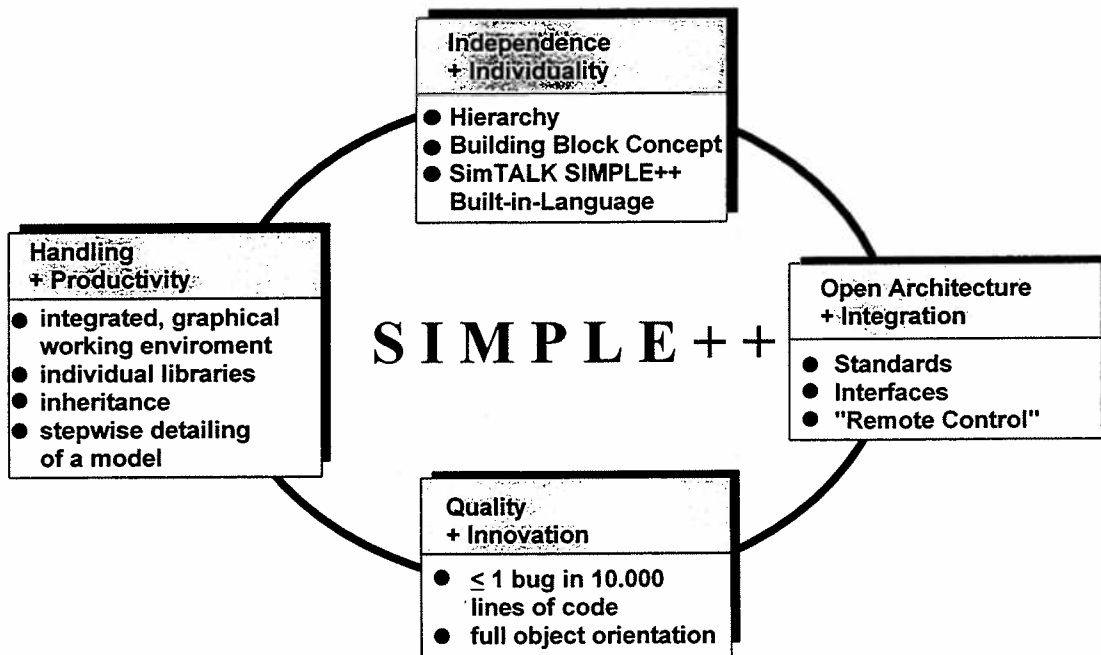
SIMPLE++ stands for **S**IMulation in **P**roduction, **L**ogistics and **E**ngineering design and its implementation in C++. SIMPLE++ is the standard software for object oriented, graphical and integrated modelling, simulation and animation. SIMPLE++ is fully object oriented: the graphical user-interface, system architecture and implementation all conform to the demands of object orientation. Very different and complex systems and business processes can now be modelled accurately and true to life. Subsequently, SIMPLE++ is used in a wide range of applications in most segments of the economy. With SIMPLE++, you can simulate entire enterprises, technical systems and business processes. There is now no need to address each of these areas separately using different and incompatible systems.

A major strength of SIMPLE++ is the significant increase in productivity when building, changing and maintaining models. The most powerful features of building block, list and language concepts are provided in a single, powerful, integrated simulation environment. Features of object-orientation like inheritance, hierarchy, reusable objects or even models as part of models etc. make SIMPLE++ extremely efficient. These and more features are described in Section 2 and the intuitive, graphical and object oriented user interface of SIMPLE++ provides functionality and user-friendliness that overcomes former barriers.

2. SIMPLE++ the product

2.1 Design philosophy

Well engineered software should incorporate proven design fundamentals. SIMPLE++ has been developed according to the requirements in functionality shown below. Next to each are listed the key points of the implementation for which further description appears later in this chapter.



Graphical and intuitive model handling to increase **productivity** is a key requirement for the widespread adoption of simulation throughout the economy. In SIMPLE++ this is achieved with a unique integrated and incremental working environment, individual libraries (application templates), inheritance and a graphical object interface.

Users should be **independent**, and so are free to make an infinite range of tailored Application Objects which serve as templates for efficiently creating models. **Individuality** ensures that the simulation model has a lifelike animation and functionality. This functionality is facilitated by the use of hierarchy, the SIMPLE++ built-in language SimTALK and other features.

Quality and Innovation must be embedded by design. With SIMPLE++ this is achieved by the full object orientation and also by the use of industry standards.

Open Architecture and Integration requires the use of standards and the capability of exchanging data in real-time and communicating directly with other systems.

2.2 Innovative edge

SIMPLE++ is innovative in three key areas: Software technology, product features and its range of application.

SIMPLE++ is the software standard for integrated, graphical and object-oriented modelling, simulation and animation. SIMPLE++ is fully object oriented: Graphical user interface, system architecture and implementation all comply to the demands of object orientation. Very different and complex systems and enterprise processes can now be modelled accurately and true to life. The significant increase of productivity in building, changing and maintaining models is a major strength of SIMPLE++.

Object orientation is a modern and productive software technology. Technology per se has no value unless it has beneficial consequences for product development, the application and the user. The use of **object orientation** in developing SIMPLE++ has the following benefits:

- **High development speed and higher quality simultaneously.**
Our experience with object orientation shows that our productivity is increased by factors ranging from 3 to 20. That means that the development team of SIMPLE++ is significantly more productive compared to teams using traditional techniques. The volume of the software code is reduced by several factors versus software with the same functionality but traditional technology. This also leads to a reduction of software bugs.
- **Longer product life cycle and better protection of your investment.**
Software using traditional design and implementation methods can quickly reach a point where complexity is no longer controllable. In SIMPLE++ complexity is limited only by design.
- **Change management and enhancements in SIMPLE++** are easily effected by adding new objects. These communicate with existing objects by defined interfaces. Thus market requirements can be implemented quickly with a minimum of effort.

These are aspects of a software which cannot be seen easily by looking only at the functionality. Nevertheless, functionality has high priority. **In addition to the innovatory concept of the total product, SIMPLE++ offers some remarkable features:**

- a) Integrated, graphical environment
- b) Hierarchy
- c) Inheritance
- d) Object Concept
- e) Changeability and maintenance
- f) Integration and openness

These features are described in more details below:

a) Integrated, graphical environment

When working with traditional simulation software you have first to build your complete model. Next you are able to run simulation and finally, the simulation file is used to depict the process (animation). You cannot change the model during simulation/animation despite this stage being the one at which modelling bugs can most easily be identified. This means that you are forced by the system to work in a procedural manner.

The integrated, graphical environment in SIMPLE++ means that all functions are available at any time and all information about the model is graphically represented. You can simulate and animate parts of the model during modelling. You can even execute, test and debug controls without running simulations. This significantly increases modelling productivity and user acceptance.

b) Hierarchy

Hierarchy is needed to build lifelike and structured models which resemble the real world. Entire enterprises, complex distribution centres or even total national railway networks can be modelled realistically and in any accuracy. Managers and engineers can understand and examine the model at the appropriate hierarchy level without loosing the overall view and increases the efficiency of communication

Hierarchy is created and reduced dynamically by nesting or deleting objects. This means that models can be detailed or simplified during the planning process at any time without having had to consider it in advance. The user defines, without any constraints, the structure and the modularisation in SIMPLE++. Thus you create hierarchy dynamically without pre-planning.

c) Inheritance

Inheritance is a very important and productive feature of SIMPLE++. It allows the user to generate and update models very quickly and securely. **In SIMPLE++ inheritance can be controlled by the user up to the parameter level.** Definitions and changes at one point will be automatically installed at all other related points. Alternative models which you need for optimisation are effectively created and updated. Maximum efficiency and minimum faults are the result. Using inheritance with SIMPLE++ gives a productivity increase of a factor of 10 or more

Inherited instances of models and classes are not copies - a copy does not maintain a relation to its origin. In SIMPLE++ the new instance (child object) stays in a user controlled relationship to the class (parent object) and any aspect of this relationship can be updated at any time.

To illustrate the efficiency and security of using inheritance consider the change from traditional production to a just-in-time production, when the entrance buffer of dozens of machines and resources might become obsolete. In a SIMPLE++ model the entrance buffer is deleted in the parent unit and all child units are updated immediately and automatically.

d) Object Concept

The Object Concept of SIMPLE++ is unique. **Any application specific object can be graphically and interactively created from generic Basic Objects.** In this way specific libraries (templates) containing Basic and Application Objects are constructed. You can use and update the Application Objects as required and according your needs. As a result, you will always work productively and with well structured models.

e) Changeability and Maintenance

The ease and speed of building, modifying and maintaining models is very important. Since simulation often runs in parallel to the planning of systems, the necessary information and data required for modelling is frequently unavailable during initial stages. Having built a first draft of a model, you are invariably faced with having to make adaptations and changes. **Prototyping and the incremental detailing and adaptation to the final model are well supported by SIMPLE++.** Using traditional software, it's often the case that even a minor change demands a completely new implementation or at least, a great deal of effort and time consumption. This problem was frequently responsible for the lack of acceptance and justification of simulation projects in the past.

f) Integration and openness

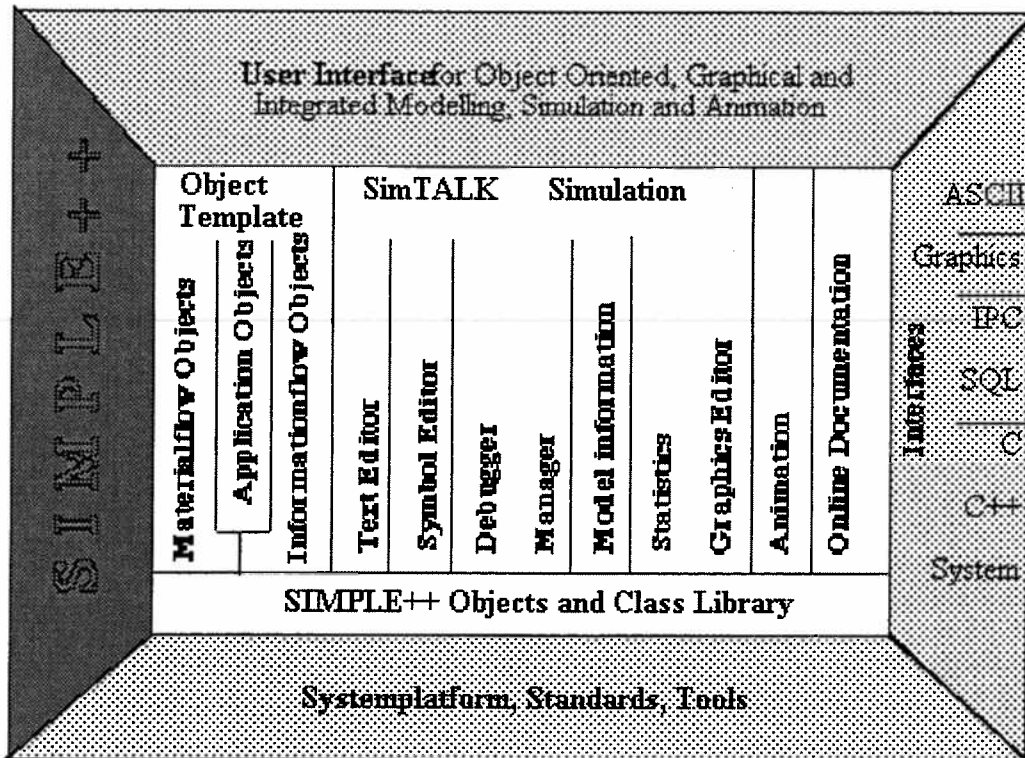
Standards and interfaces of SIMPLE++ make sure that integration and openness become a reality. **The following interfaces are available:**

- ASCII Files
- SQL Database
- Inter-Process-Communication
- Graphics
- C and C++
- Operating System

The real-time data exchange during simulation and the full external control of SIMPLE++ by other programs are important features for integrated solutions.

The full object oriented software technology, the remarkable features described above along with those described beneath '2.4 Specifications', summarise the functionality of SIMPLE++ which enables such a wide spectrum of application which are listed in section 3. Together, these remove the limitations of previously available software and characterise the innovative edge of SIMPLE++.

2.3 Description



The above picture shows the **architecture of SIMPLE++**. The elements of the architecture are described below.

The strength of SIMPLE++ lies in the ease and speed of building, modifying and maintaining models. High flexibility in modelling stems from the material and information flow being modelled independently. The required connections are implemented as a sensor-actor-concept. At the entrance and exit of an active material flow object is a sensor which activates a control where the desired actions are defined.

SIMPLE++ is software designed for use in technical application fields. Common **standards of technical application software** are used and supported. For example, SIMPLE++ is implemented in the language C++ under UNIX and X-Windows/Motif and is therefore easy to port to other **system platforms**. SIMPLE++ is currently released on all popular workstations and on personal computers with SCO-UNIX. The Windows NT version will be released in the first quarter of 1995.

SIMPLE++ offers a user friendly **Graphical User Interface** for interactive and integrated operation. All functions of modelling, simulation and animation are accessible at any time

creating an 'integrated environment'. A 'non-procedural operation' means that you don't have to consider simulation flow when building SIMPLE++ models. Additionally, SIMPLE++ offers "incremental operation" which means that models or their parts can be detailed or simplified at any time. So if you have to implement changes during a simulation project, then you can do so immediately without having to change the entire model or build a new one. The productivity of modelling in this manner is usually greater by a factor of 3 or more. The integrated, non-procedural and incremental environment of SIMPLE++ allows you to work according to your own ideas and the requirements of real life and not according to procedures enforced by the simulation system. Last but not least, SIMPLE++ provides the capability to build individual, application specific dialogue masks for each object.

The **Interfaces** of SIMPLE++ allow you to exchange data in real-time, communicate with other programmes or call existing routines. The integration capabilities of SIMPLE++ are very important when using simulation to, for example, support the daily operation of an enterprise. In particular the exchange of data in real-time and the full control of SIMPLE++ by other software like MRPII can be highlighted. The following interfaces are available: ASCII file, Graphics, IPC (Interprocess Communication), SQL Database, C, C++ and Operating System. These interfaces make SIMPLE++ genuinely open and easy to integrate.

The **Object Template** contains both Basic Objects and user-defined Application Objects. All are visible as icons.

Application Objects are important for productive modelling. Any Application Object is graphical and interactively built by the user and stored in the Object Template for subsequent use. Because of the object orientation of SIMPLE++ you don't have to extensively modify an Application Object to use it for different models or applications. The user can create application specific Object Templates just by configuring Basic and Application Objects. An Application Object can be a network of Basic Objects (material and information flow objects), an entire model or sub-model and any combination of Basic Objects and models.

In the standard configuration, SIMPLE++ provides some powerful sets of Application Objects. An extraordinary feature of these Application Objects is their "user openness". The user can modify the Application Objects because they are open SIMPLE++ models built from Basic Objects. So even if the Application Objects provided do not match your

requirements, they can be adapted or used as reference objects. These help you to make a most productive start with SIMPLE++.

The following sets of Application Objects are included in the SIMPLE++ standard configuration:

SIMPLE++_controls

SIMPLE++_controls is a set of Application Objects which model standard control strategies without programming. The following Application Objects are provided in the application template for interactive use: OR, RANDOM, PERCENT, PRIORITY, SEQUENCE, ALTERNATING, SET, IF, PARAMETER and VARIO. The Basic Objects of SIMPLE++ possess a defined standard behaviour when a part enters and leaves them. If a different behaviour is required, you can use the Application Objects of SIMPLE++_controls. These are represented by graphical symbols in the template as usual, and can be used in any model. By using the connector, you define its relationship to the material flow. The Application Objects can be combined with each other in order to get additional functionality depending on the chosen combination.

SIMPLE++_SFS

SIMPLE++_SFS is a set of Application Objects which represent the most frequently used material flow components of a real material flow system and are more complex than the SIMPLE++ Basic Objects. The following Application Objects are provided in the application template: Straight, bend, branches, confluence (local and global).

SIMPLE++_staff

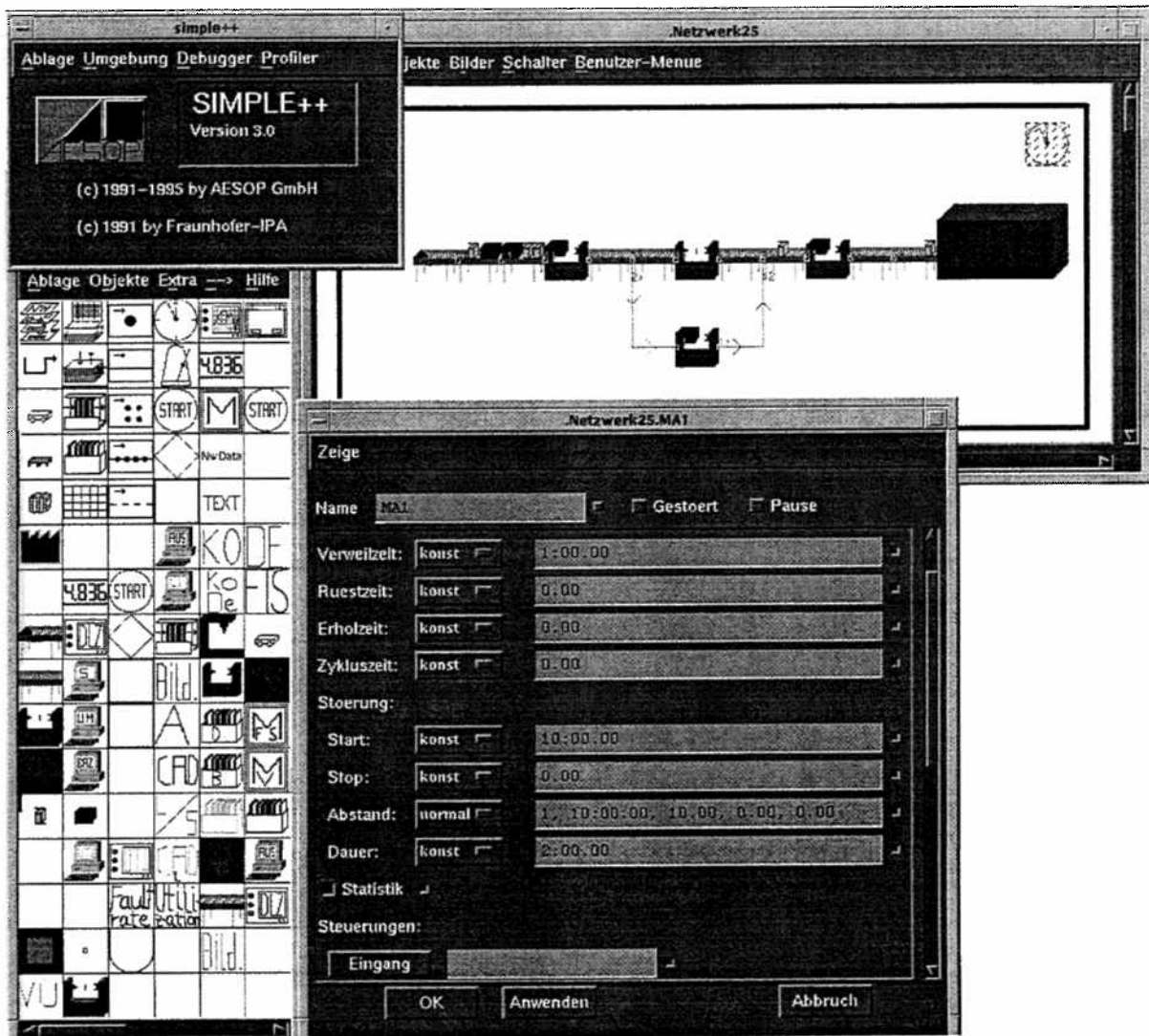
SIMPLE++_staff is a set of Application Objects to model the dispatching of production staff in an effective and structured manner. The following Application Objects and functionalities are provided in the application template for interactive use: Staff pool, qualification profile and priorities, working time model (shifts, pauses etc.), a matrix of distances and times, description of the working place, staff statistics.

SIMPLE++_AGV

SIMPLE++_AGV is a set of Application Objects to quickly and easily model automated guided vehicle systems (AGV). The following Application Building Blocks are provided in the application template for interactive use: Vehicle, path, curve, distribution, collection, intersection, central station, load station, central control and local controls.

Special sets of Application Objects like SIMPLE++_process (chemical industry), SIMPLE++_shop (Shop Control) etc. are available as optional products (refer to 2.5 Optional products).

Below you can see as a screen shot, an application template to the left-hand side. At the bottom, you can see the dialogue window in which you can type the parameters of an Object and in the upper right-hand corner, a new Object is being built.



You can position, parameterise and connect individually designed Application Objects and Basic Objects and by doing so, you can create a realistic simulation model. You determine the number, design, function and behaviour of the objects. The objects are represented by icons in the animation layout .

SIMPLE++ Basic Objects							
material flow				information flow			
movable		immovable		movable		immovable	
active	passive	active	passive	active	passive	active	passive
* vehicle	* part * container	* processor * line * conveyor	* storage * path		* data	* control * generator	* lists * displays

With the **Basic Objects for material and information flow**, the user is able to design any Application Object. The **Material flow Objects** can be divided firstly into moveable and unmoveable and secondly, into active and passive line, conveyor, storage, path, vehicle, container and part. The **Information flow Objects** are: Control, generator, lists, displays and data. The comprehensive capabilities of SIMPLE++ for information processing stem from the features of the information flow objects, the powerful built-in language SimTALK and a large number of functions, data types and operators.

Material and information flow are linked together by a sensor actor concept. A sensor e.g. at the entrance of a machine (material flow) activates a control (information flow) in order to control the related process.

A set of standard controls is provided with the SIMPLE++ standard configuration. Special controls are defined by the built-in Language **SimTALK** by the **Text Editor** or in a guided and graphical manner, without typing the syntax, by the **Symbol Editor**. Local controls for Application Objects or global controls for the total model ensure that the information processing in your simulation model is modular and well structured. Controls can be executed and tested without running the simulation. The capacity to trace and debug a model in the integrated environment is one of the most powerful features of the **debugger** built into SIMPLE++. Modelling errors can be effectively identified and rectified.

The **Simulation** can run without animation for systematic experiments which the user can easily manage. For experimentation, SIMPLE++ models can be pre-loaded with any

information and status. A model status can be stored during simulation interactively by the user or as a programmed event, which may result from a change of status. The status can be subsequently re-activated at any time. The simulation speed can be adjusted by the user or the user can 'step' through each event.

The simulation **Manager** in SIMPLE++ controls the simulation. It can be pre-set by the user or programmed during simulation.

During simulation, **Model information** can be displayed. Single parameters are shown via digital displays. Multiple parameters are shown concurrently over the time via the analogue display. SIMPLE++ offers various **Statistics over various periods of time** - interval, current and total simulation. These include utilisation, preparation time, wait time, down time etc..

The **Animation** runs on-line to the simulation. Selection of the different animation pictures for each object can be through a change of state or event driven. In this way changes can be visualised dynamically during simulation. These features are important for both, presentation purposes and verification. The pictures are defined by the user through the **Graphics Editor** or simply copied from a picture template.

By pressing the help-function the **On-line Documentation** appears on the screen. This does not render the reference manual obsolete but provides helpful information immediately and in a relevant context.

2.4 Specifications

Clear specifications will help you make the right decisions. Below are listed the specifications of SIMPLE++ V.2.3 which was released in March 1994 . You may not fully appreciate SIMPLE++ until you have seen it but a comparison of features will highlight the differences between SIMPLE++ and other products.

User interface

- object oriented, graphical user interface with window, menu and mouse technique.
- all model information is graphically represented and accessible.
- application oriented dialogue and plausibility check
- user definable dialogue masks
- Screen control by 'Zoom' and 'Scroll'
- "integrated working environment": All functions for modelling, simulation and animation are available all the time
- 'Incremental operation': Detailing or simplifying of the model step by step as requested
- 'non-procedural operation' modelling according to the user's preference and independent of the simulation flow
- on-line documentation

Modelling

- model size not limited
- detailing or simplifying of a model at any time
- Object Concept with Basic Objects and user definable Application Objects
template with Basic Objects for material and information flow:
Processor, conveyor, line, stock, vehicle, path, container, part,
lists (2D, stack, queue, random), generator, controls, display and
information objects
- templates with Application Objects for very productive modelling:
Standard Controls, Staff Pool, Automatic Guided Vehicles etc.
picture template containing various pictures to represent objects
in the animation layout
- graphics editor to define and change individual animation pictures per object which are
changed event or state driven during animation
- inheritance controlled by the user at parameter level
- unlimited model hierarchy and process structures
- fixed and free attributes for each object. All attributes can be processed
- Material- and information-flow can be modelled independently

- disruption generator per active object
- unlimited independent Random Generators
- stochastic distribution of parameters
- flexible and powerful information management. More than 50 mathematical, textual and logical functions.
- SimTALK object oriented built-in language for individual controls
- interactive test and correction of controls without running the simulation
- Debugger to debug controls and support model verification
- Information support

Simulation / Animation

- animation on-line to simulation
- animation switchable on/off
- animation speed selectable: step or variable speed
- any preloading at the beginning and storing of model status during simulation
- control by external systems like MRPII
- data exchange during simulation
- statistics for total simulation time, intervals and actual runtime:working time, preparation
- time, break down time, blocked, throughput time, capacity load (min/max)
- change of model and parameters during simulation/animation
- step mode to follow the simulation/animation events
- individual collection and presentation of model and simulation values
- free definition of the animation layout
- event driven and interactive change of the animation pictures
- visualisation of hierarchy levels by opening objects

Others:

- SIMPLE++ Reference Manual
- SIMPLE++ Tutorial
- file interface (ASCII)
- system interface
- graphics interface (PPM and TIF)
-

Optional products:	SIMPLE++_C	"C" interface
	SIMPLE++_C++	"C++" interface
	SIMPLE++_IPC	Inter process comm.
	SIMPLE++_SQL	SQL database
	SIMPLE++_shop	Shop Control Model
	SIMPLE++_gantt	Gantt Charts
	SIMPLE++_process	Chem. Industry
	SIMPLE++_GA	Genetic Algorithm

System requirements

Workstations or PC with UNIX and X-Windows/MOTIF, > 16MB memory, 100MB free disc capacity, floppy, 256 colour graphics and a resolution of > 1024 x 768, mouse and keyboard. Serial interface for SIMPLE++ security device.

Released Platforms

DECstation5000 / ULTRIX

DECalpha AXP / OSF1

HP9000-4xx and -7xx / HP-UX

IBM RS6000 / AIX

SGI indigo / IRIS





















SUN SPARCII / SUN OS and SOLARIS

PC / SCO-ODT

PC / WIN-NT

Important Features of SIMPLE++ which should be benchmarked:

We know of no other simulation software which compares to SIMPLE++. If you have knowledge of or are seriously considering purchasing a different product, then please use the form below to conduct a personal evaluation.

● Features	● Benefits	SIMPLE++	Others
hierarchy	reflect reality structuring		
Inheritance	efficiency less faults		
Integrated, graphical and object oriented enviroment	user acceptance productivity		
Specific user-defined application templates	reuseability modularity fast modelling		
User-open application objects	adaptable reference objects		
Control language with debugger	universal and flexible application		
incremental modelling	fast, actual		
Efficient modification and maintenance of models	actual models low costs		
Interfaces open architecture	integration real time data exchange		
Full object oriented technology and standards	innovation speed, security for the future		

2.5 Optional products

The standard configuration of SIMPLE++ contains some sets of Application Objects. They are SIMPLE++_controls, SIMPLE++_staff, SIMPLE++_SFS and SIMPLE++_AGV. Details are described in "2.3 Description". In addition and at extra costs the following options for SIMPLE++ are available:

SIMPLE++_C

"C" Programming Interface for SIMPLE++.

The user is able to define and integrate special C-routines. These user defined routines can be called up like internal functions (sin, round,...). All basic data types available in 'C' and SIMPLE++ can be used. It is not possible to activate the internal methods of the SIMPLE++ basic building blocks.

Prerequisite: C-compiler, SIMPLE++

SIMPLE++_IPC

Interprocess communication with SIMPLE++.

SIMPLE++_IPC allows you to communicate directly with other software systems. The appropriate SIMPLE++ functions (e.g. start, stop, delete create, data exchange etc.) can be activated by Remote Procedure Calls (RPC's). Therefore SIMPLE++ can be controlled externally. This means that SIMPLE++ can be used as a simulation module in a complex application system.

Prerequisite: SIMPLE++, TCP/IP connection

SIMPLE++_SQL

SQL-Database Interface for SIMPLE++.

SIMPLE++_SQL allows the direct, bi-directional data exchange with SQL-Databases during the simulation. For example orders or work schedules can be transferred from an MRP-system to a SIMPLE++ model and the results of simulation can be transferred back. The SIMPLE++ library provides an SQL Object block which is used to establish the communication to an SQL-server on the network. To establish more than one connection at the same time you can duplicate the SQL Object block. SQL statements and data are sent from SIMPLE++ to the database. Data from the database are available in SIMPLE++ in automatically generated lists.

Prerequisite: SIMPLE++, Network.

SIMPLE++_shop

SIMPLE++_shop is a model structure which enables you to quickly and easily build a model which corresponds to shop floor organisation. The following Application Objects are provided in the application template for interactive use: Source, Workstation, Shop Control, Part. The user puts Workstations in the model and defines the parameters. Lists of orders, operation plans and resources are prepared for application. An order list or the Source can be used to load Parts into the system. The data of the order list can be imported from an MRP system. The Shop Control Object block controls the processing of the Parts according to the operation plans. All Application Objects are built from Basic Objects and are open for the user to make individual adaptations.

Prerequisite: SIMPLE++

SIMPLE++_gantt

SIMPLE++_gantt is the software for the graphical presentation and interactive manipulation of the order operation sequence or the occupation of the resources by using bar charts and an interactive user interface. The bars can be moved and by double clicking each one, a window opens to show related data which can be changed. The updated data is immediately depicted as bar charts and stored in the corresponding file. Input and output data of the Gantt charts are well structured ASCII files.

Prerequisite: SIMPLE++

SIMPLE++_process

SIMPLE++_process is a set of Application Objects to model continuous processes in the chemical industry. The following Application Objects are provided in the application template for interactive use: Source and Sink, Pipe and Tank, Mixture and Distribution, Batch Reactor, Continuous Batch Reactor, Container Stock, Resource Management, and Control. All Application Objects are built from Basic Objects and are open for the user to make individual adaptations.

Prerequisite: SIMPLE++

SIMPLE++_GA

SIMPLE++_GA is a software that uses "genetic algorithms" for finding optimised solutions considering different and competing goal functions. For example the optimal order sequence when the competing goal function of minimum throughput time, minimum stock capital and maximum resource utilisation are defined. The quality of the calculated solutions is depicted graphically. The user can stop the calculation process at any time. The best optimisation result found to a given point in time is available. The goal functions and constraints of the optimisation are defined by the user dependent on the type of problem

that is to be solved. Input and output data of SIMPLE++_GA are well structured ASCII files.

Prerequisite: SIMPLE++

2.6 History

SIMPLE++ has been developed by AESOP together with the Fraunhofer Institute for Production and Automation (IPA) and is based on more than 20 years of experience in developing and applying simulation software.

SIMPLE++ version 1.0 was released in February 1992. From that date until the release of SIMPLE++ Version 2.0 in February 1993, the software was installed over 100 times in Germany. In July 1993 when shipping SIMPLE++ Version V.2.1, AESOP registered more than 200 installations world-wide. In December 1993, SIMPLE++ version 2.2 was shipped and this was followed by SIMPLE++ Version V.2.3 in March 1994. You can expect a much higher development speed in the future. SIMPLE++ V.3.0 is planned to be released in the 4th quarter of 1994. Software, reference manual, tutorial and training material for SIMPLE++ are available both in English and German.

3. SIMPLE++ the range of application and some examples

SIMPLE++ is used in the planning of production, logistics and engineering as well as in the daily operation of enterprises to plan and control processes.

In the **planning process**, SIMPLE++ is used to dynamically plan systems in production, logistics and engineering in most segments of the economy. In this application area the structure, the dimension and the control of systems is optimised by SIMPLE++. Manufacturing systems, assembly lines, transportation systems, warehouses, picking of goods, packaging lines, hospitals, railway networks, traffic control, entire factories and distribution centres etc. are examples where SIMPLE++ is used successfully.

SIMPLE++ is also used for **Shop Control and Monitoring** in the manufacturing industry. This is performed by the same model that is built during the planning phase. The focus of this application is to look ahead at the operations (lifelike forecasting) for next time period (e.g. next 24 hours or next week) based on plans (e.g. customer orders and worksheets from a MRP-system) and data available at the time of simulation. The output of the simulation is a fine tuned operation plan (order schedules, lot sizing, staff dispatching, throughput times, stock and operation costs etc.) to prepare the staff with better knowledge with which to tackle the real situation.

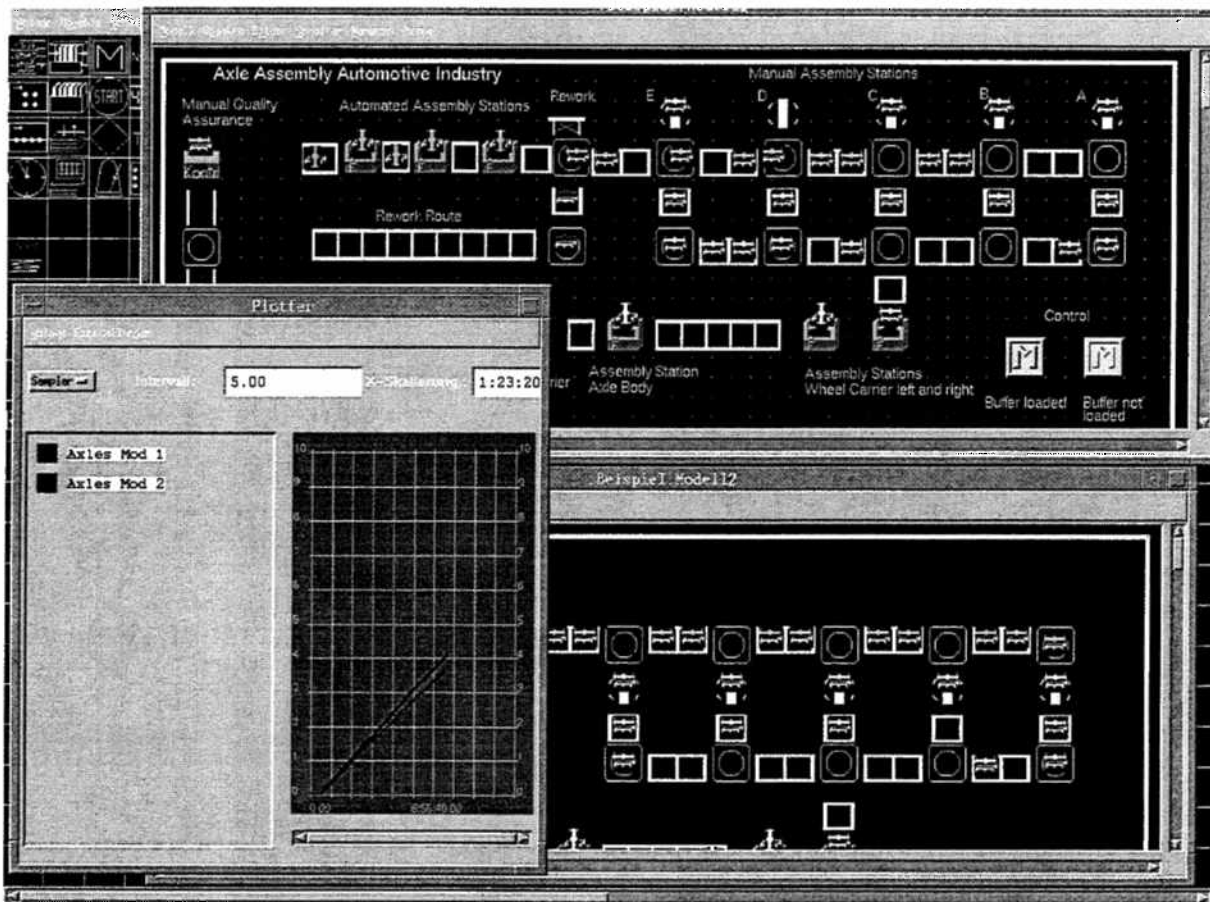
The SIMPLE++ simulation model that performs the look ahead and forecasting is also used as a monitoring system to monitor the events of the operations in the lifelike SIMPLE++ animation layout. Inter-Process-Communication and real-time data exchange of SIMPLE++ are prerequisite features for that sort of application.

The following examples of projects carried out by us should show the range of different applications of SIMPLE++.

Axle Assembly in the Automotive Industry

This example shows the optimisation by simulation by varying layout, capacity and control strategies of a back axle assembly line at **Mercedes Benz** in Stuttgart.

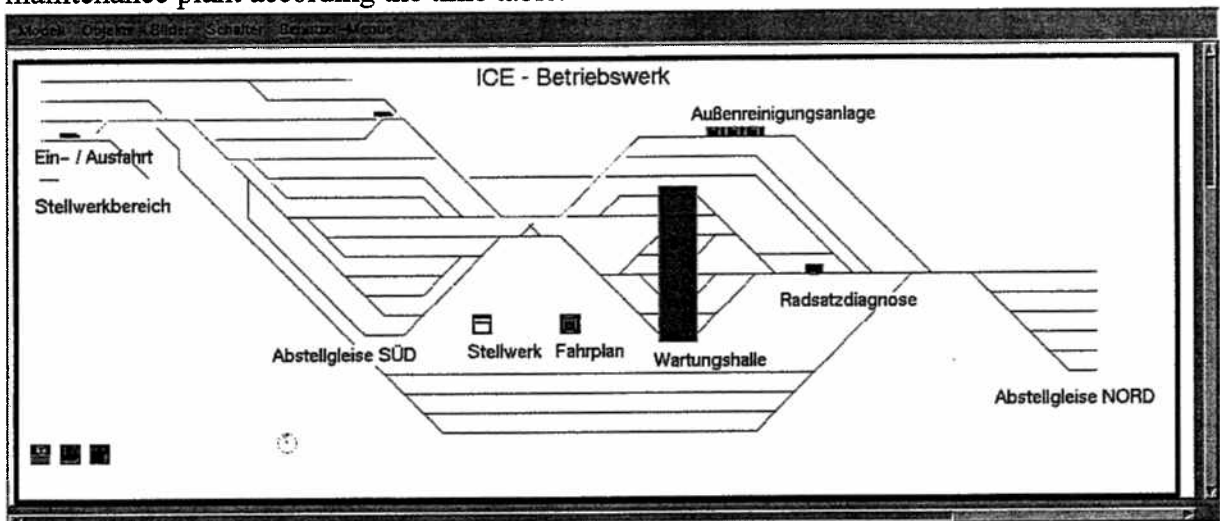
The results of changes are shown graphically. Particularly the throughput of two competing alternatives that ran against each other is monitored. Any other parameter of the model could be monitored dynamically. The picture shows two alternative layouts for the manual assembly stations - the throughput is displayed dynamically and on-line to the simulation and animation.



ICE train maintenance

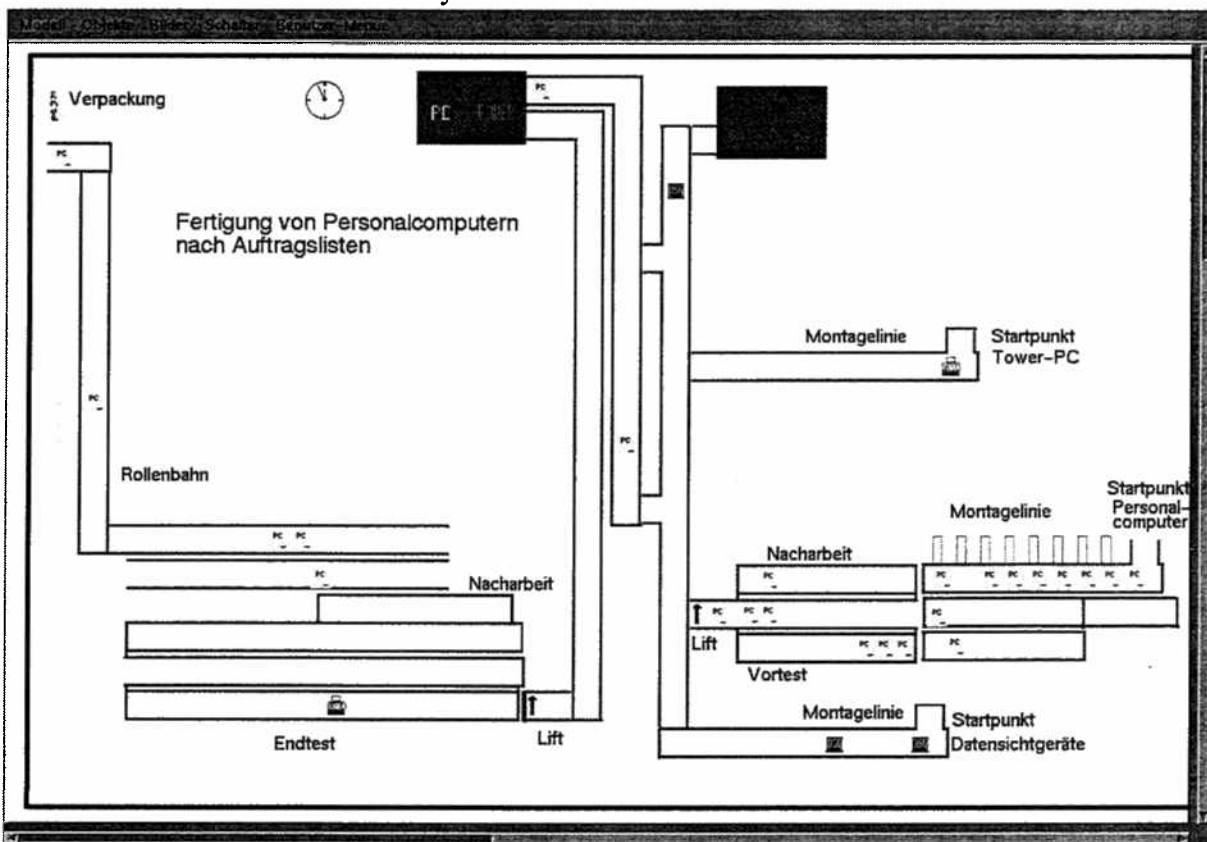
This example shows the maintenance plant for the ICE (Intercity Express) of **German Railways**. SIMPLE++ is used to optimise the scheduling of trains for all required maintenance resources. Each train costs about £21 million and thus the time for repair and maintenance should be kept to a minimum. Gantt charts are available to visualise the maintenance schedules. They can be changed interactively by the user in order to augment the scheduling and optimisation algorithms with their invaluable experience. The refined data of the Gantt Charts are used for the next simulation run to check the result of the changes. The access to their database by using the SQL-Interface of SIMPLE++ allows to exchange data in real-time.

Based on the current circumstances a realistic look-ahead is given by simulation in order to tackle the situation better. The outputs of the simulation are fine tuned operation plans and staff allocation which ensure that all maintenance is done and that the train leaves the maintenance plant according the time table.



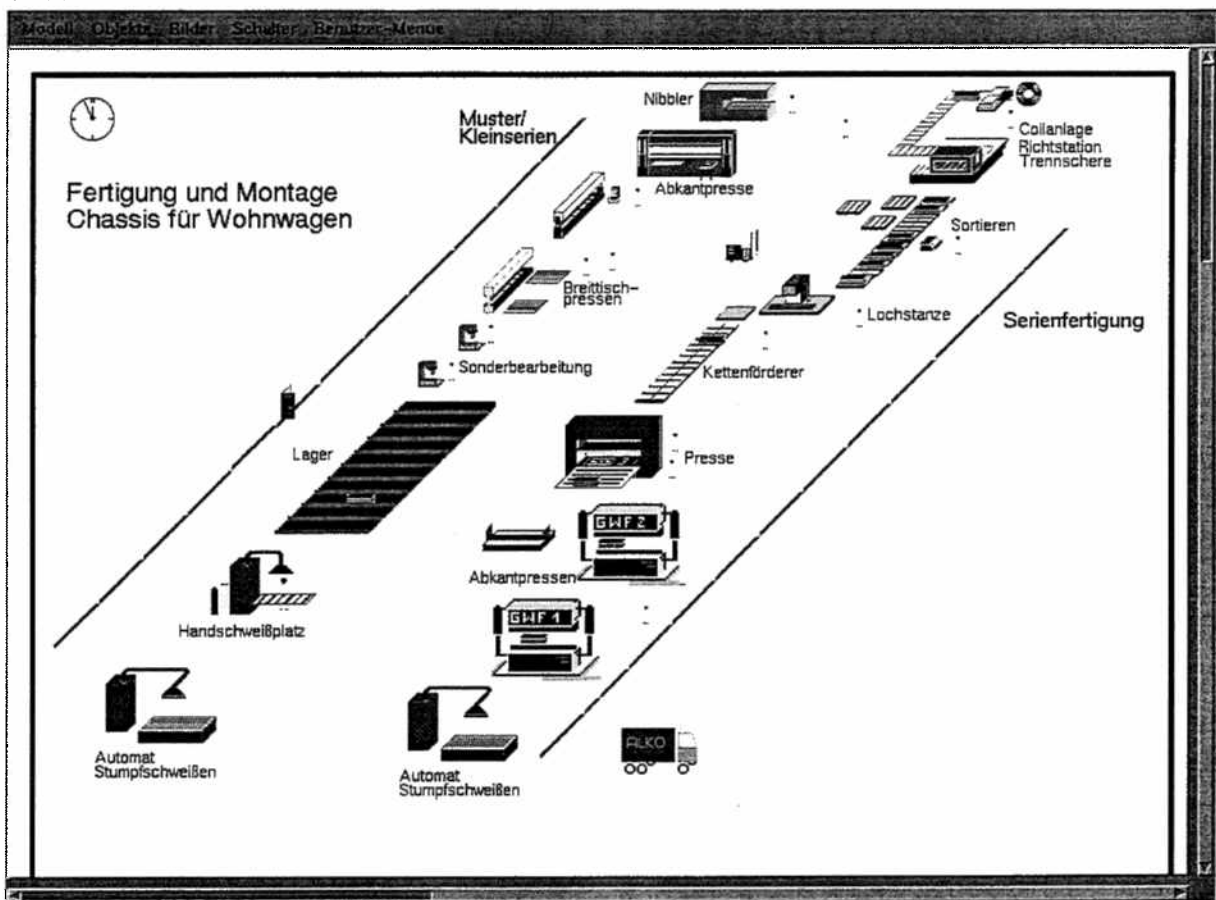
PC Assembly

The total PC assembly (labour, material flow, assembly) at **Siemens** in Augsburg is simulated in order to check the feasibility of the production plan generated by an MRP system. The daily operation is thus supported by using simulation. One result of the simulation is the list of all produced PC's during one shift with throughput times and other information. Different actions can be performed and tested by simulation before the real shift begins. This is a unique tool to get realistic forecasts and schedules with the consequence of reliable delivery dates. If quality and costs of products are at comparable level to competitors, then reliable delivery becomes a critical success factor. This critical success factor can be controlled by simulation.



Production Planning in the Manufacturing Industry

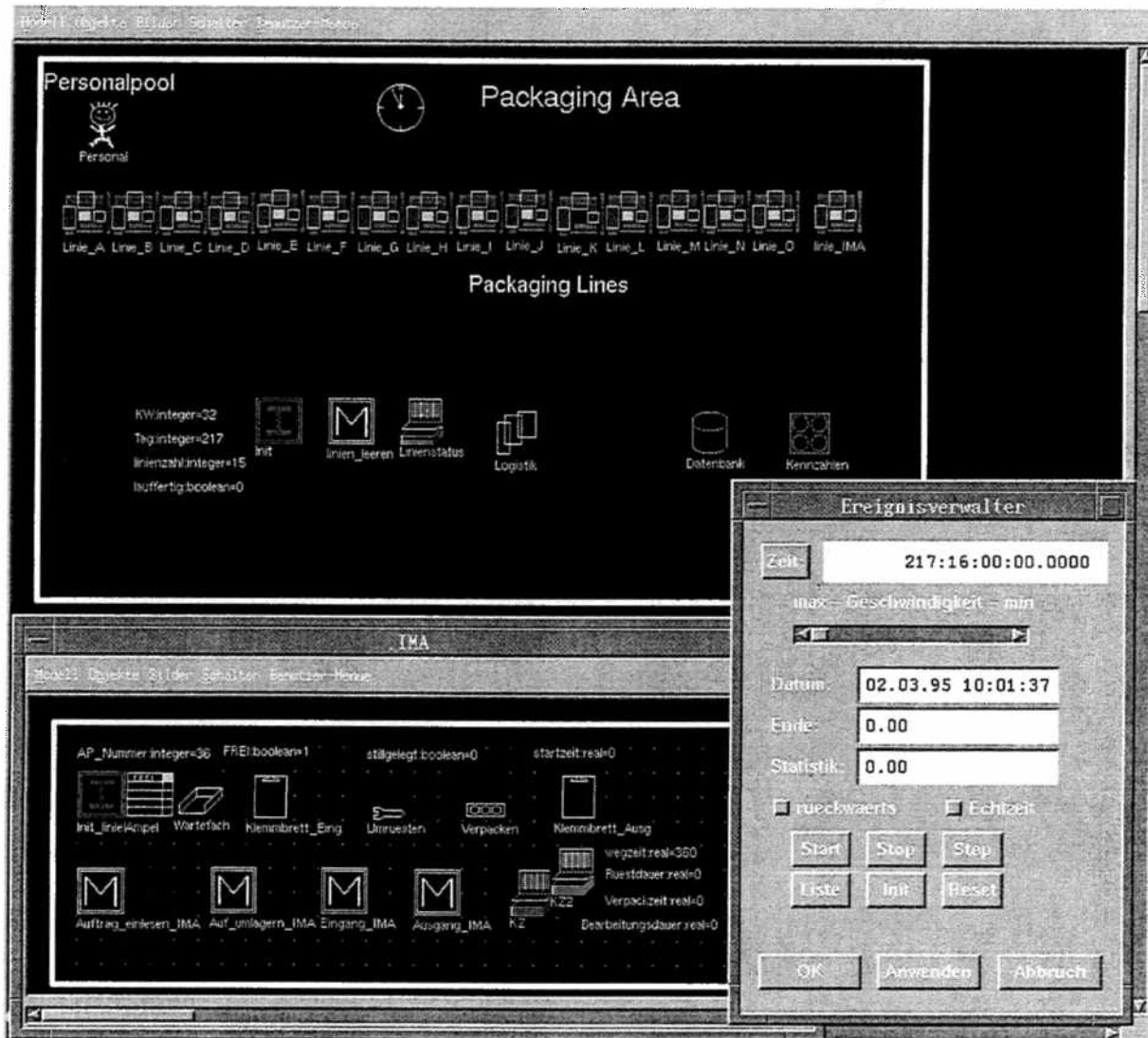
The animation layout represents the production of **Alco** chassis for caravans. The production plan of one week is simulated very quickly and animated in the attractive and realistic layout of the plant. The purpose of this simulation is to get the optimal order sequence, minimum set-up time, optimised costs and the minimum throughput time for the production programme for one week. The throughput times are decreased significantly by using SIMPLE++. The picture below shows the animation layout and the parameter sheet of one machine.



Packaging in the Pharmaceutical Industry

The packaging area with automated packaging lines, labour, material and information flow is modelled and simulated to test different production and organisation scenarios. To achieve "Lean production". All scenarios are developed and tested using SIMPLE++.

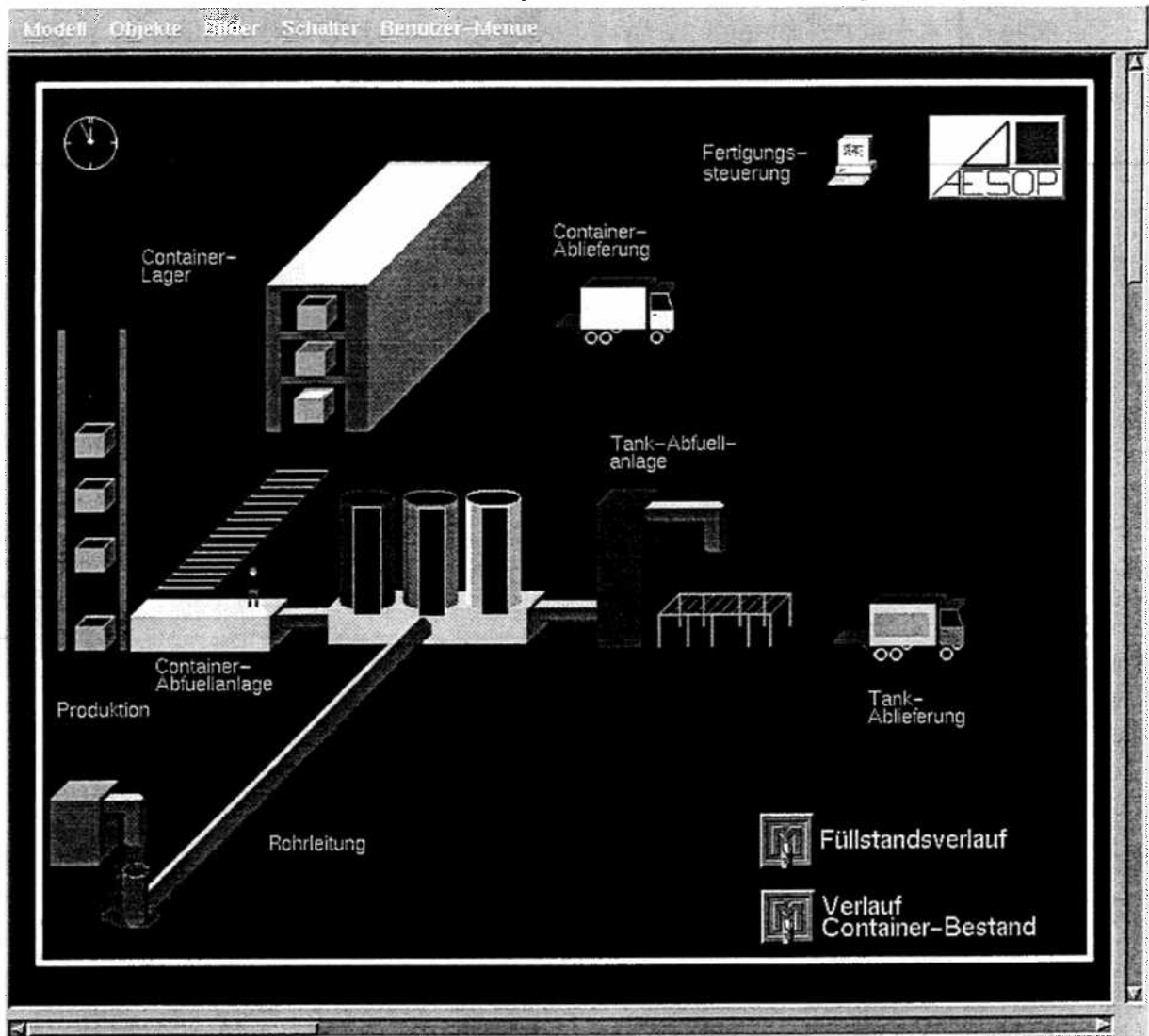
Simulation is performed in the phases of strategic planning (world-wide evaluation of locations) and of optimisation of the current facilities. Different qualification profiles of labour and varying shift models are tested to find the optimal production technique of the future. Even the local costs and specialities are considered to identify the best location for production world-wide.



Production and Logistics in the Chemical Industry

Production, logistics and particularly the control strategies are developed, optimised and tested by using the below SIMPLE++ model at **BASF** in Ludwigshafen.

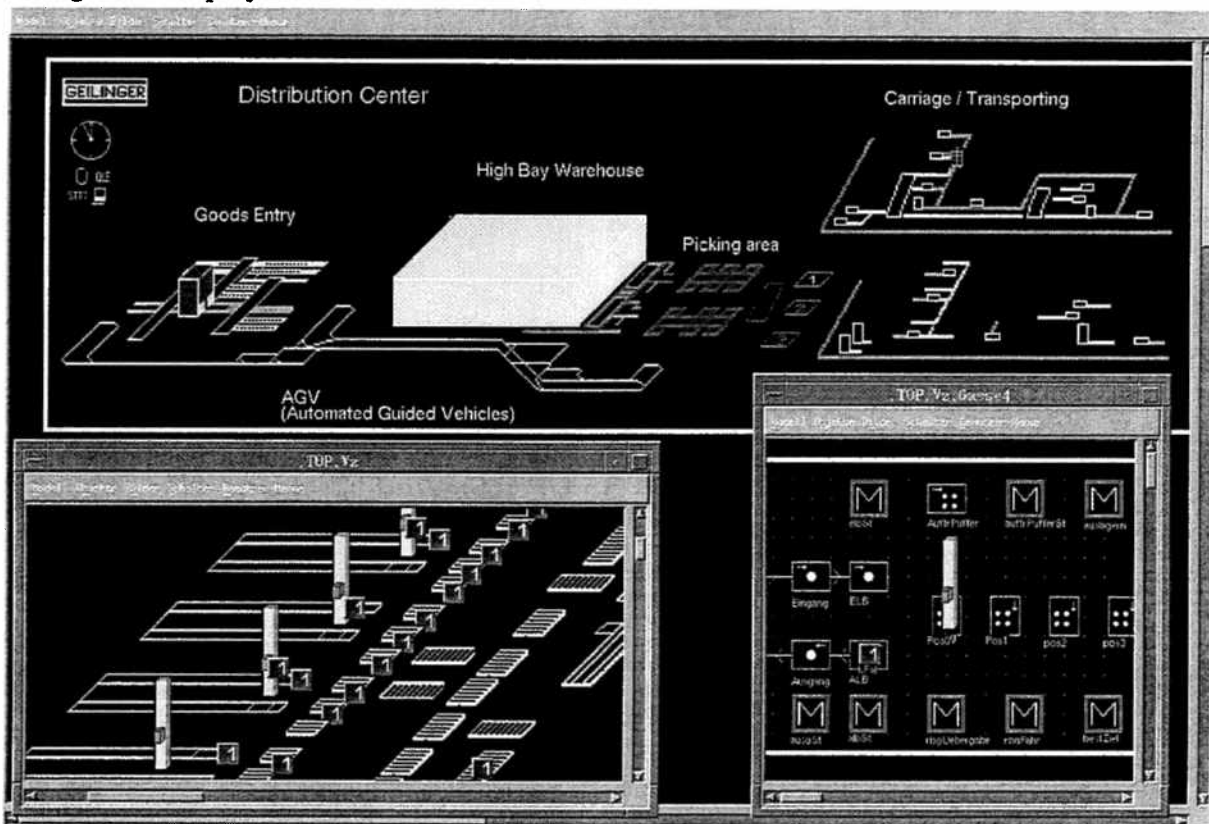
The advantages of SIMPLE++ in this application are lifelike animation, the transparent and modular structure of the controls and the dynamic visualisation of any process parameters.



Distribution Centre

A large distribution centre is modelled in a very detailed way with some levels of hierarchy by **GEILINGER engineering**, Switzerland. Simulation was used during the whole planning and implementation process in order to optimise the system, process and communication on all technical and organisational levels. SIMPLE++ supported the decision making process.

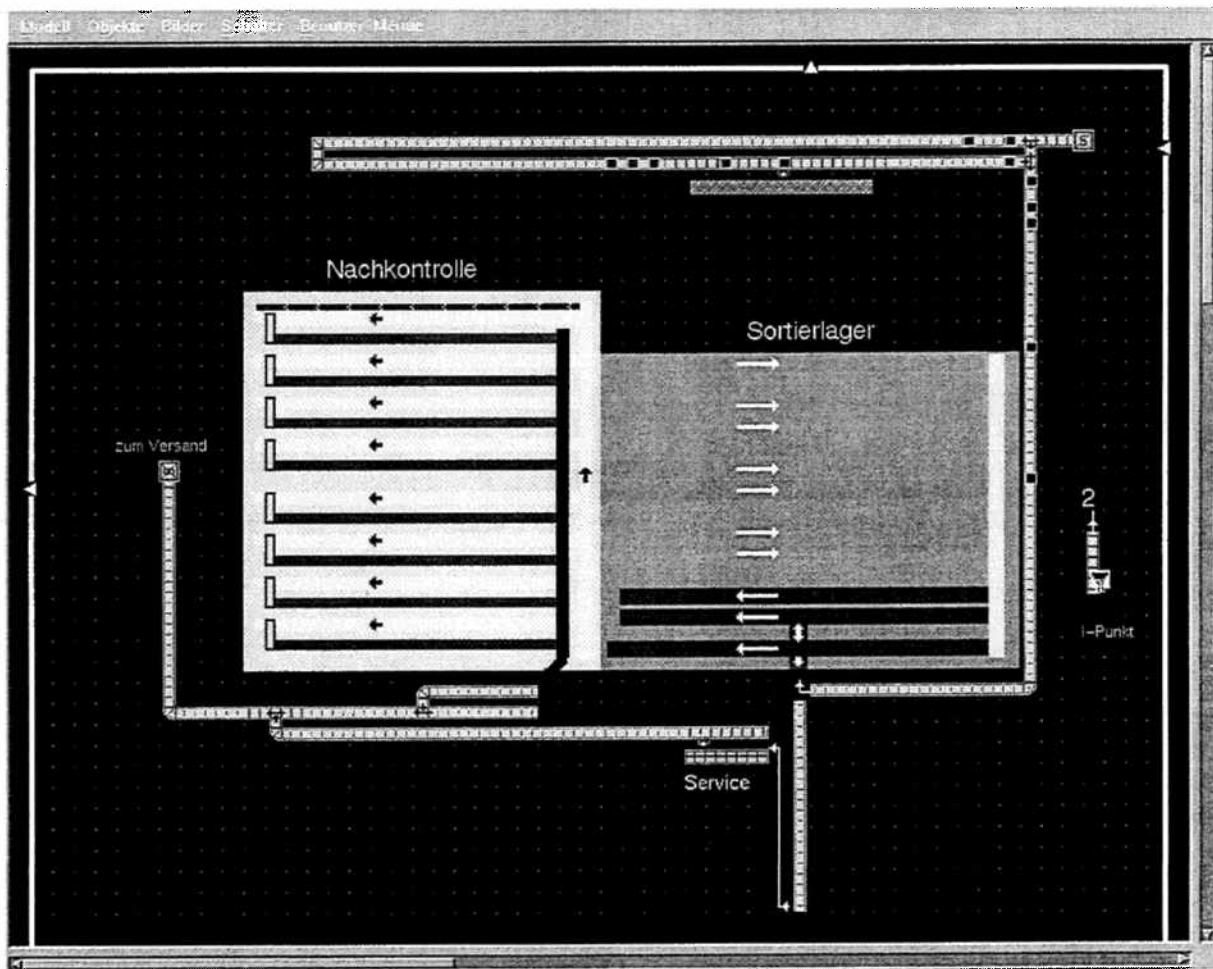
To achieve the shortest modelling time for such a complex project, several people worked in parallel developing different sub-models. Each was tested independently before incorporation into the overall model. Hierarchy and inheritance had been invaluable throughout this project.



Planning of a new Sorting Storage

The objective was to simulate the commissioning process to the point of dispatch for a finished parts warehouse holding 13 commissioning areas on two floors.

The commissioning areas, a sorting area including 105 sorting lines, the combined final inspection and packaging area, the service area, dispatch area and a extensive driven transportation system make up the main components of the model. Another task assigned to simulation was to examine the functionality of the overall system renouncing 35 sorting lines. Further on, several strategies for the work flow management have been investigated. A testing was done on the employment of flexibly assignable comissionneers. Their assignment had to be considered according to the state of the overall system and queues in the particular commissioning areas.

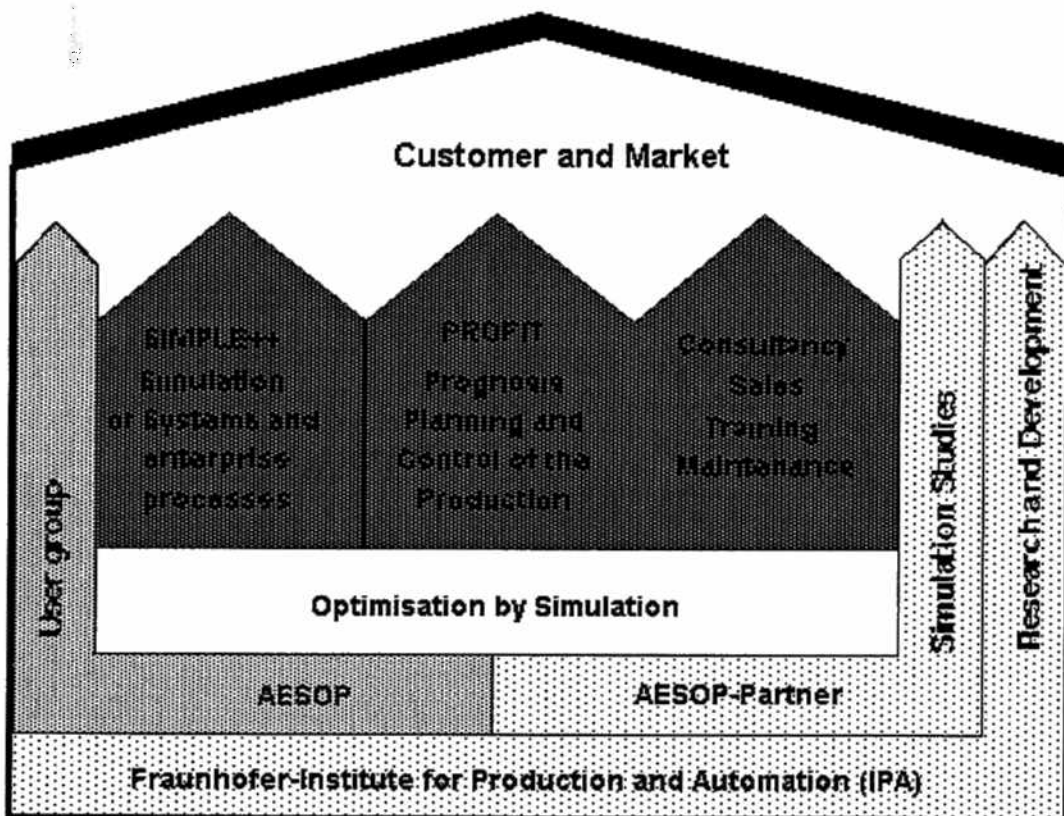


4. AESOP - the vendor of SIMPLE++

AESOP is a privately owned company and was founded in May 1990 in order to develop, produce, market and support simulation software for optimisation. Since that time AESOP has been growing continuously each year. All investments and growth are financed by the stock capital and the operating profit without any loan from a bank. Most of the profit is being reinvested in the company.

The management has many years of experience in developing software - particularly simulation software - applying simulation software and marketing of information systems. The staff is highly educated and motivated to keep SIMPLE++ the most attractive simulation software at the market and to provide first class service to our customers. All employees participate in the financial success of AESOP.

We tried to draw our activities into one solid building. At the top are the customers and the market. They are most important and drive our activities in the outlined frame.



At the foundation level is the Fraunhofer-Institute for Production and Automation (IPA). SIMPLE++ has been developed by AESOP together with the IPA and the mutual co-

operation with the IPA is now even closer. The results found in research projects by IPA can be transferred to commercial products immediately by AESOP. In addition to the formal co-operation, we are happy to have Professor Dr. Ing. H.J. Warnecke, the president of all Fraunhofer-Institutes and the former president of IPA, in our advisory board.

Today AESOP and about 20 AESOP-Partners provide consultancy services based on SIMPLE++ and market the software SIMPLE++ world-wide.

SIMPLE++ is the only product line of AESOP. All resources are focused on it to provide solutions for "Optimisation by Simulation". We offer SIMPLE++ as a standard simulation package with application oriented options for the optimisation of systems and business processes. PROFIT is a modular software system based on SIMPLE++ for the prognosis, planning and control of production to optimise daily operations. Our solutions should help our customers to dynamically analyse systems and processes for better and safer decisions and increased profit. For all the products and solutions we provide first class consultancy, training and maintenance. "Optimisation by Simulation" is our mission.

5. Unseld + Partner

Since 1991 Unseld + Partner is a Partner of AESOP focusing on Simple++ as simulation software. Unseld + Partner has become the service company with the highest turnover of simulationsoftware in Austria.

The occupation of Unseld + Partner include planing and controlling of production lines and depots.

Together with FhG-IPA innovative project are worked out, for example refined production planing in steel construction. Furthermore, the themes metal-working and commissioning are in the centre of interest. Co-operative software developing companies and institutes work in the area of order volume optimising in der Fensterproduktion und Einlastungssteuerung in der Schleifmittelindustrie.

In 1995 following goals will be realised:

1. Construction of a net for demonstration centres for SIMPLE++ and PROFIT
2. Development of specific SIMPLE++ software modules in order to determine the optimal strategy for production planing and control bei complex Fertigungsabläufen
3. Installation of PROFIT at customers in Austria.

In addition to simulation on the industrial sector Unseld + Partner is engaged in simulation of freight centres and the logistics thereof within the ITF-Umbrellaproject „Logistics and Transportation“

Profit from PROFIT

I.A Walls

AESOP Ltd., PTMC, Marsh Lane, Preston, UK

Tel. (44) 01772 200380

fax (44) 01772 200404

Abstract: PROFIT is the commercial realisation of two ESPRIT supported projects. It is an integrated simulation based system for planning, controlling and monitoring production. The core of any application is an accurate, company specific model of the production process, constructed using the object oriented simulation tool, SIMPLE++.

T&D Automotive have recently commissioned a new factory, to Supply In the correct Line Sequence (SILS) car bumpers to the adjacent Vauxhall Motors factory. PROFIT was chosen to provide the complete solution for all information processing between Vauxhall, the group-wide MRP and the shop floor PLCs. It will schedule, monitor, control and trace the production flow and processes at T&D Automotive. A special focus was put on the integration of shop floor personnel in the planning and control of the factory. A key concept of PROFIT is that of "user openness" through which it is possible to involve people and to integrate T&D Automotive in the design and construction of their own solution. This paper provides an insight into this first implementation of PROFIT in the UK.

1. Background

PROFIT is the commercial realisation of the system initially developed during two projects receiving funding from the EC under the ESPRIT programme. It is an integrated simulation based system for planning, controlling and monitoring production. The core of any PROFIT application is an accurate, company specific model of the production process, constructed using the object oriented simulation tool, SIMPLE++.

This paper introduces PROFIT and the research from which it was developed. It then goes on to discuss the first implementation of PROFIT in the UK, at the factory of T&D Automotive.

1.1 *ESPRIT - Shop Control / Real-I-CIM*

Shop Control (ESPRIT project 5478) had as its objective the development of a suite of standard modules that would allow companies to quickly install and configure complete manufacturing and control systems. Its success was such that a follow-on project called Real-time Interactive CIM (Real-I-CIM) was commissioned (ESPRIT project 8865).

1.2 Partners and Pilots

The PROFIT product arising from these ESPRIT projects is a modular system with individual components developed by AESOP GmbH, DiCONSULT GmbH, Fraunhofer-Institut für Produktionstechnik und Automatisierung (FhG-IPA), Instituto de Engenharia de Sistemas e Computadores (INESC) and TECHNOTRON.

Pilot sites for the initial implementations were established at ARJAL, an automotive component supplier, and GROWELA a shoe factory. In both cases the results were positive. The current Real-I-CIM project features a pilot application at the BMW factory in Lohhof.

2. PROFIT

SIMPLE++

The discrete event simulation system SIMPLE++ forms the heart of any PROFIT installation. SIMPLE++ is to our knowledge unique in being the only fully object oriented, graphical and interactive simulation system commercially available. An accurate customised model of the factory is used for on-line monitoring and scheduling and off-line capacity planning. The object oriented nature of SIMPLE++ means that very complex systems can be modelled in an incremental and hierarchical way. A library of re-usable application specific objects ensures that the model can be built rapidly and, just as importantly, it can be easily modified and maintained.

PROFIT Characteristics

Figure 1 illustrates the modular architecture of PROFIT. The key features of PROFIT are:

- "user openness", which for a CIM system means the maximum involvement of the customer in the design phase and flexibility for the customer to make future changes and adaptations to the system themselves;
- excellent system ergonomics through the use of object oriented modelling techniques, benefiting users both on the shop floor and in the planning department;
- modularity based on a comprehensive set of CIM functions such as management information tools, scheduling, exception handling, monitoring, maintenance, DCE software administration, personnel management, quality management, traceability, shop floor data acquisition, shop floor control etc.;
- connectivity to legacy systems, especially MRP, PPC, CAP, SCADA etc.;

- fast design and implementation of a customer specific solution through highly productive object oriented modelling techniques, client server data integration and open architectures.

3. T&D Automotive - Requirements for the PPC system

3.1 Scenario

The T&D Automotive factory makes plastic car bumpers, having the capability to produce up to 2500 bumpers per day. Its main customer is the Vauxhall Motors plant situated next to the factory. T&D Automotive are contracted to supply bumpers in the correct line sequence (SILS) in a just in time manner. An order for a bumper pair of a particular colour and build-level arrives approximately once per minute through an electronic data link. These orders are received some three hours before they have to be delivered to the Vauxhall Motors line-side.

There is a multitude of variants possible. Each bumper can be one of 13 basic colours (with special order colours also possible); in addition each bumper can have head lamp washer jets, fog-lamps, two grille options, and can be for a saloon or estate car variant. It would be impractical to keep this number of variants in stock, yet three hours is insufficient time in which to make a bumper from scratch. The solution is to produce intermediate stocks of unpainted bumpers and painted bumpers, and only to proceed to the final assembly on receipt of an order from Vauxhall. A central requirement of the Production Planning and Control (PPC) System is to minimise the size of these intermediate stocks while ensuring that parts are always available to fulfil orders.

3.2 First make a bumper

The manufacturing process is as follows: the bumpers are injection moulded, loaded onto pallets and transported by overhead monorail to the unpainted bumper store. From there they are called-off under the direction of the planning and control system to be painted. The painting process consists of a plasma surface treatment followed by a two stage paint application, which is performed by robots. The painted bumpers are then held in store until the corresponding order appears on the call-off list, following which they are brought to the assembly areas (one each for front and rear bumpers). Once assembled to the appropriate level they are loaded on trolleys and shipped to the Vauxhall Motors line-side by electric vehicle.

3.3 *Simulation*

In the pre-production planning phase the designers of the factory and its various systems required simulation in order to prove that the different sub-systems would work together as planned and to investigate the operation of the factory in various contingency scenarios in which the production capability is impaired in some way.

Once the factory is operational, the simulation model continues to be of importance. At any time the model can be automatically initialised with the current work in progress and plant status, and a simulation run performed to investigate the future behaviour of the plant. In addition, the simulation model developed is re-used to provide the graphical front end for monitoring and, in an adapted form, forms the basis of the scheduling function.

3.4 *Monitoring*

A crucial aspect of operating such a factory is to have a detailed and up-to-the-minute knowledge of the work in progress and the status of various resources. This plant monitoring role also had to be integrated with the other management functions such as scheduling. Access to this monitoring function was required at various strategic locations around the factory.

An additional requirement was to construct an archive containing a record for each bumper produced in which the conditions at each stage of the process are recorded. This database formed part of the quality assurance system, giving complete traceability on individual parts.

3.5 *Scheduling*

The scheduling of bumpers through the plant centres on the minimisation of set-up (at injection moulding and at paint) while ensuring that sufficient intermediate stocks are available to fulfil orders. Should a bumper be scrapped at the final assembly stage where it has been allocated to a particular customer order, the scheduling system must expedite its replacement so that the fundamental SILS requirement can be met.

3.6 *Flexibility*

The entire planning and control system required sufficient flexibility to be re-configured easily to accommodate production route changes and product additions and changes. Examples of modifications that are likely in the future include the addition of further moulding machines to produce bumpers for different car types and the supply of unfinished bumpers and spare parts to other manufacturers and resellers.

4. The Team

The team formed to address the requirements of T&D Automotive, outlined above, consisted of the Fraunhofer IPA, DiCONSULT GmbH and AESOP, with AESOP (UK) Ltd. as the prime contractor.

4.1 Fraunhofer IPA

The Fraunhofer IPA have established themselves as one of the leading consultancy and applied research organisations in the area of factory planning, control and optimisation. Their experience of simulation and its application in on-line systems, is extensive. It was here that the predecessors of SIMPLE++ were devised and developed to the point where their commercial potential was evident, leading to the establishment of AESOP GmbH.

4.2 AESOP

AESOP GmbH was formed in 1989 to develop and exploit the simulation activity of FhG-IPA. AESOP (UK) Ltd was formed in 1994 to market and support SIMPLE++ and integrated simulation based systems in the UK and Ireland.

4.3 DiCONSULT

DiCONSULT, who are based near Munich, specialise in Shop Floor Production Planning and Control Systems. Their expertise lies in the acquisition and communication of production information from and to the shop floor. They use their own rugged industrial terminal system to act as the interface between people and machines on the shop floor and the higher level planning and control system.

5. The Solution

5.1 General Overview

The principal objective of the Manufacturing Planning and Control system at T&D Automotive is to control and enable the production of the bumpers. To achieve this a bumper pair (front and rear) meeting stringent quality requirements must be available for assembly at Vauxhall Motors just as the car to which they are to be fitted reaches the attachment station on the assembly line. To ensure that this objective was met, the entire T&D Automotive factory was first simulated using SIMPLE++, to identify potential bottlenecks and appropriate operating regimes.

The implementation consisted of the following modules:

- scheduling and forecasting system
- monitoring system
- traceability archive system
- shop floor control system
- interface to the machines, shop floor and planning personnel.

5.2 *Off-line simulation*

The first stage in implementing the production control system was to simulate the entire T&D Automotive factory so that its various operating modes could be validated. The simulation model, built using SIMPLE++, showed that part of the automatic transport system within the factory was incapable of achieving the required throughput with the parameters as specified. It is worth noting that due to the complex interaction of carriers competing for access to track sections and the rules governing how transport orders are allocated to carriers, it is impossible to predict the detailed behaviour of such a system using static design calculations. By experimenting with the model it was possible to show the implications of various suggested modifications to the system, helping to ensure that the final configuration met the requirements of T&D Automotive.

5.3 *Scheduling and forecasting system*

The use of simulation as the basis of a planning and scheduling system is rapidly gaining acceptance as the approach of choice in many manufacturing situations (see for example [1]). The scheduling and forecasting system plans the production of the required bumpers. It provides the information required to oversee production and to make decisions in the case of contingencies. In the normal working mode the primary route of material flow from the delivery at Vauxhall Motors back through the system to the injection moulding section is automatically scheduled and controlled. Orders are received through EDI link direct from the Vauxhall motors plant. Three different time horizons are used in planning the production at the T&D plant. These are: a three-week rolling forecast of anticipated production; firm orders for three shifts ahead (whose sequence is only about 80% correct); a 3 hour call-off signal, updated every 50 seconds, of actual bumper pairs required at line-side.

Scheduling optimises batch sizes at the injection moulding and paint plant while meeting the various competing system constraints (such as limited storage space for partially complete bumpers). Additionally, the SIMPLE++ system will incorporate human decisions for certain well defined contingencies and exceptions. The planning system also forecasts the use of material, future bottlenecks and implications of human decisions.

5.4 *Monitoring*

The monitoring sub-system presents to the customer information about the exact status of his plant and of the orders flowing through it. This information was presented through a graphical display derived from the simulation model of the factory. Colour encoded icons represented the production status of the various resources in the system. Each resource icon could in turn be opened to present further information about the production history for that resource and the work currently in progress there. A third, more detailed, level of information was available to the user by directly accessing the databases associated with the machines, orders and produced parts. Access to this data was made available through a GUI employing data filters for selectivity.

5.5 *Low Level Control & Data Acquisition*

Information about the status of each machine in the factory and information about the bumpers themselves as they move through their production process, were captured by dedicated industrial terminals from DiCONSULT. Data was acquired direct from the machine PLCs, through bar-code readers, through manual input and through direct digital signal inputs. In this way the high level production planning and control system can access up-to-date information about the exact status of the factory, presenting this to the operators and management through a graphical display. The planning system uses the information about resource availability and inventory at various production stages to produce a viable schedule. This schedule information and the consequent direct control of the various machines and transport systems were relayed to the shop-floor through the same DiCONSULT terminals.

The system has been designed in such a way that it is fault tolerant. Should the planning computer or the connecting network go off-line, the factory can continue to operate under the direct control of the terminals until the fault is rectified.

5.6 *Operating Modes*

The normal operating mode for the T&D Automotive factory is where all the resources are available and customer orders are coming into the system by EDI. In this situation the entire planning process is fully automatic, and the factory can deliver all bumpers required in the correct sequence. In addition various contingency scenarios were identified in which the ability of the factory to produce bumpers was impaired. Typical contingencies are due to machine failure (e.g. where one of the two moulding machines breaks down), or where raw materials are in short supply. In the former case the planning system was able to predict the time at which orders would fail to be fulfilled allowing the customer to be forewarned of potential problems. In the latter case the planning system

recognised when such a raw material shortfall was likely and suggested to the production manager the best product mix in order to maximise the time before orders could not be fulfilled.

6. Implementation

The project to design and successfully implement the production Planning and Control System for the T&D Automotive factory was characterised by a number of key elements.

6.1 Time Scales

The time scales for the project were extremely compressed, with only some seven months from the inaugural project meeting to having a fully functional planning and control system. It is the author's contention that this was only made possible through the use of the modular object oriented software that forms PROFIT.

6.2 Customer Participation

An essential success factor in the implementation of the planning and control system was the close working relationship established with the customer. The key person responsible for the system implementation, the computer services manager, was seconded to AESOP for the duration of the contract. This enabled him to be involved in the day-to-day decision making process and to learn the details of the system's inner workings. This last point means that not only do T&D have the skills necessary to run their plant, they also have the ability to react to changing market situations by updating and modifying the system themselves - a rare feature in a planning and control system.

In addition a series of regular (approximately monthly) workshops were held at the factory site, which key personnel from T&D Automotive (including the Operations Director, Technical Director and the Technical Manager) and the AESOP consortium attended. These workshops were invaluable in reaching a rapid consensus on the various technical issues that arose during the project.

6.3 Functional Design Specification

The Functional Design Specification (FDS) was a detailed document defined and agreed during the early workshop sessions. It set out the functionality of the planning and control system - what it would do, what inputs were expected, what contingencies were considered, what the outputs were to be and the specification of the interfaces to the machinery supplied by T&D and other third party suppliers.

Although a mutually agreed version of the FDS was used as the formal basis of the implementation, the document itself continued to evolve as more detail was added. In this way the FDS also served as the basis for the technical documentation of the system.

6.4 Liaison with 3rd Party Suppliers

The equipment specified for the individual production steps can all, to some extent, stand alone. For an integrated factory the planning and control system acts as the bridge between these islands of automation. The interface to all the machinery therefore is a crucial aspect of the planning and control system. In an ideal world these would all be specified at the procurement phase; unfortunately for various reasons much of the equipment had already been purchased (without too much consideration of interface capabilities). Consequently, the need to have timely and effective communication with these suppliers was critical. The experience of this project was that this best achieved through face-to-face meetings at which the technical issues were quickly resolved.

6.5 Maintenance & Upgrading

AESOP's philosophy in implementing a production planning and control system is to give the customer the necessary skills to allow him to implement changes to the system himself (with recourse to AESOP should this be needed).

Routine system upgrades are performed remotely through ISDN routing giving access to the T&D Automotive computers from team member sites in Germany and the UK.

7. Where Do We Go From Here ?

The Production Planning and Control System implemented at T&D Automotive is a success in at least two ways. T&D have a flexible solution to their operational requirement, which because of the nature of PROFIT was implemented within tight time scales, while at the same time being of lower cost than any of the "conventional" solutions proposed for the factory. Secondly, the project has demonstrated how the results of ESPRIT funded research projects can be effectively commercialised.

In terms of the implementation of the system some important points can be discerned. In particular it is difficult to understate the importance of developing a Functional Design Specification, of the type described above, in a project of this nature.

The future looks secure for simulation based Production Planning and Control Systems and in particular PROFIT. The final word should go to T&D Automotive's computer services manager:

"PROFIT has provided T&D Automotive with a manufacturing system that gives us three main advantages in a highly competitive automotive market.

Firstly, it is a complete manufacturing solution, with all parts of a traditionally disjointed system seamlessly integrated. The hierarchical nature of the installation, gives flexibility allowing changes to be accommodated easily and very quickly. However of almost greater importance it gives us easy access to the critical data being collected and processed on site.

Secondly, the ability of the whole manufacturing operation to run in a truly lean manner, with minimal stocks of all components held, directly reduces our costs to the customer ensuring that we maintain our competitiveness within the marketplace.

Thirdly, it provides a greater understanding of the major benefits that simulation can provide. This will allow us to plan the expansion of the current site , or the development of any new sites for new customers, with the confidence that we have gained from this installation. Using SIMPLE++ as the simulation tool has one major advantage over other conventional simulation systems in that the final simulation is not discarded but provides a core element of the implemented PROFIT system. This of course means that people are more inclined to invest the time in simulation knowing that it will directly benefit the end result.

I believe that the future of modern manufacturing systems is in complete integrated solutions, with simulation providing core functionality. No other system that I reviewed before finding PROFIT came close to the functionality that it has provided us."

8. References

[1] "New Ways of Planning Production in the Process Industries", B-D Becker and S Nestler, FhG-IPA, 1995

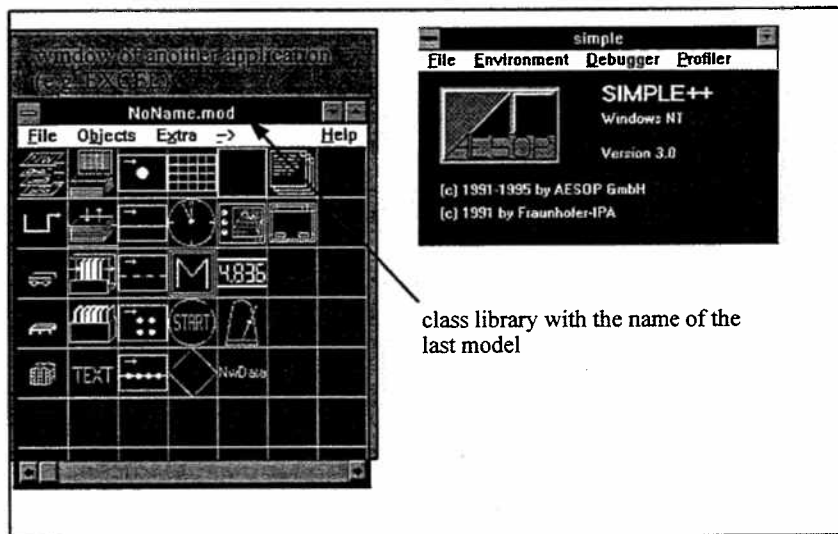
SIMPLE ++



EUROSIM'95

The Graphic User Interface of SIMPLE++

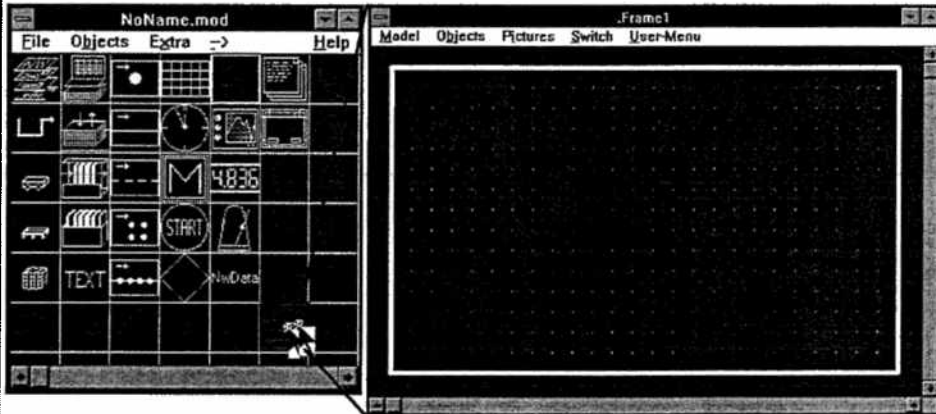
After starting SIMPLE++, a class library containing the basic objects will be displayed (unless otherwise specified) and, as an example, the following screen appears:



EUROSIM'95

Opening Objects or Models

A model is opened by double clicking on the desired object in the library with the left mouse button.

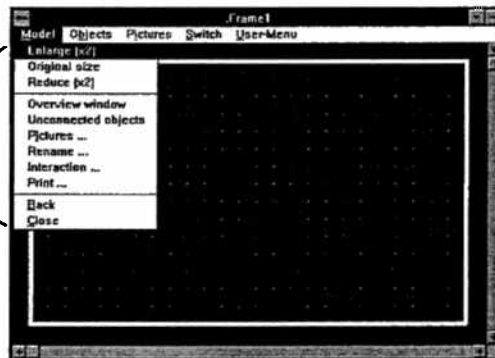


EUROSIM'95

Using Menus

menu bar

menu items

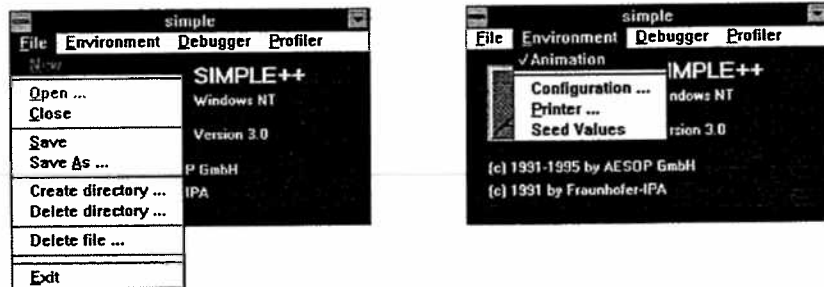


To activate a menu click on the desired menu title with the left mouse button. A list of menu items will appear. To select one of these items also click on it with the left mouse button.

Alternatively you could press the "Alt" - key and the underlined letter of the menu title simultaneously. This will also open the menu. One of these items can be activated by entering the underlined letter. Please note that these letters are casesensitive.

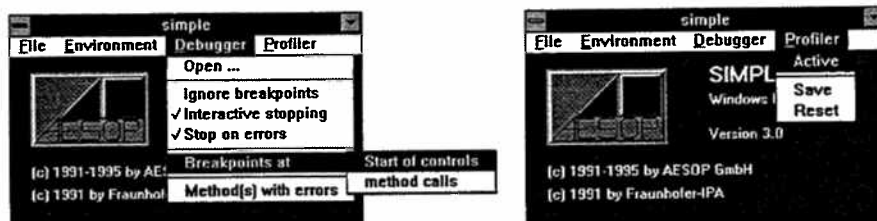
EUROSIM'95

The File and the Environment Menus



EUROSIM'95

The Debugger and Profiler Menus

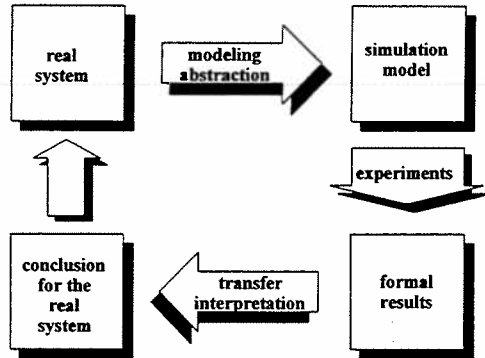


EUROSIM'95

What is Simulation?

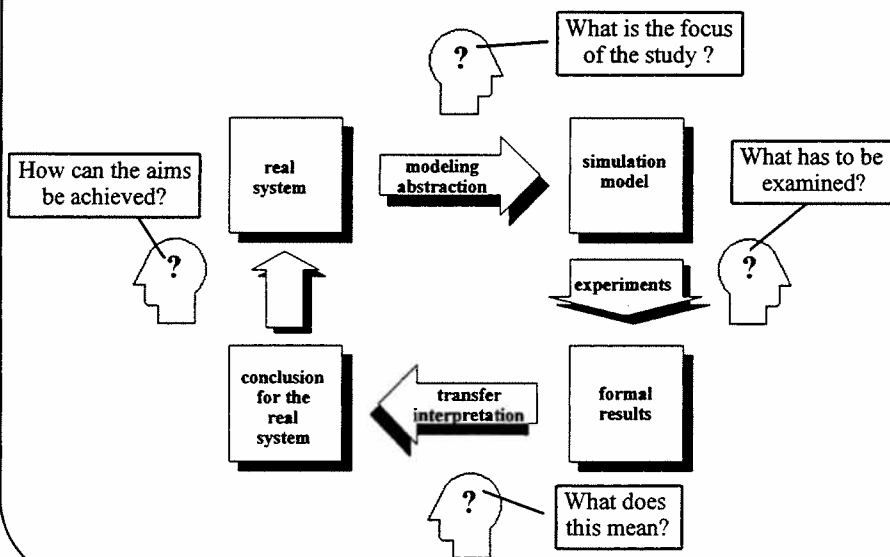
Simulation is defined by the German Institute of Engineers (Directive 3633) as:

Simulation is the depiction of a dynamic process in a model to gain knowledge that can be transferred to the real system.



EUROSIM'95

Simulation and the User



EUROSIM'95

Application Areas for Simulation

Simulation is used for the development and examination of

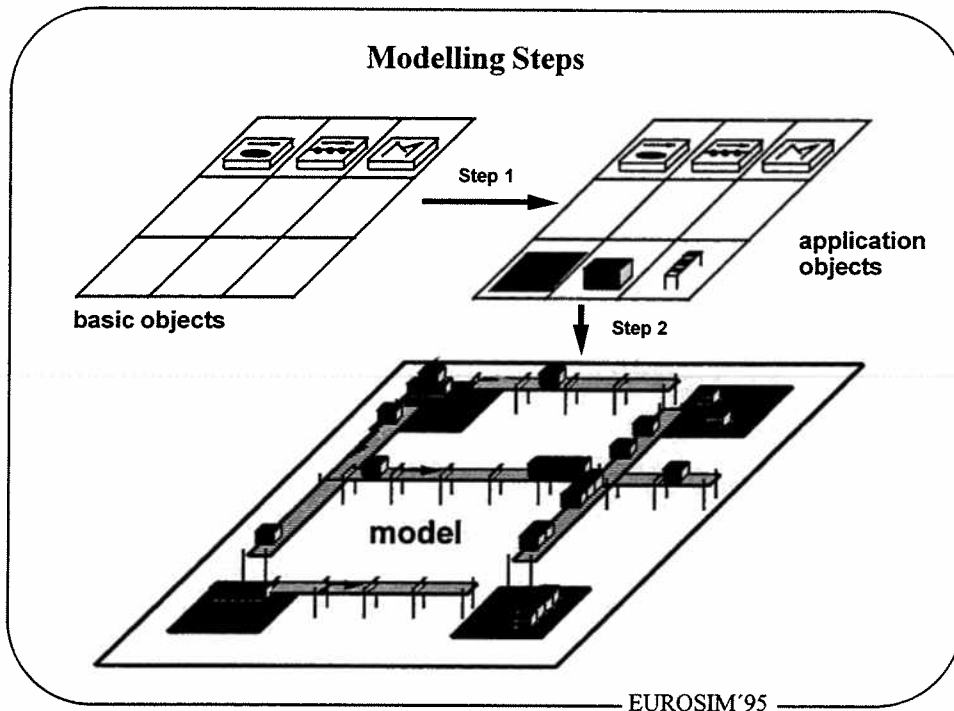
- environmental systems
- biological systems
- vehicles and aeroplanes
- manufacturing systems
- robot systems
- business processes
- chemical processes
- transportation systems
- etc.

EUROSIM'95

Overview: Modelling with SIMPLE++

- Principles of modelling processes
- The class library
- Example: building a model with SIMPLE++

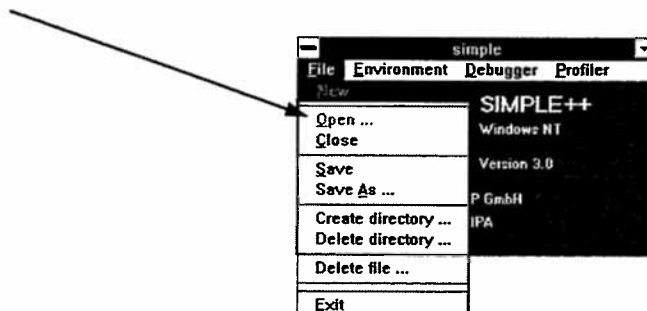
EUROSIM'95



Opening an Existing Model

Clicking on the *Open...* menu item in the SIMPLE++ main window an existing model can be opened.

Note: if a class library is already opened, this must be closed first.



Step1:
Open the model named *Transp.mod* in your training directory.

EUROSIM'95

The Class Library



basic objects

specific application objects
built from basic objects

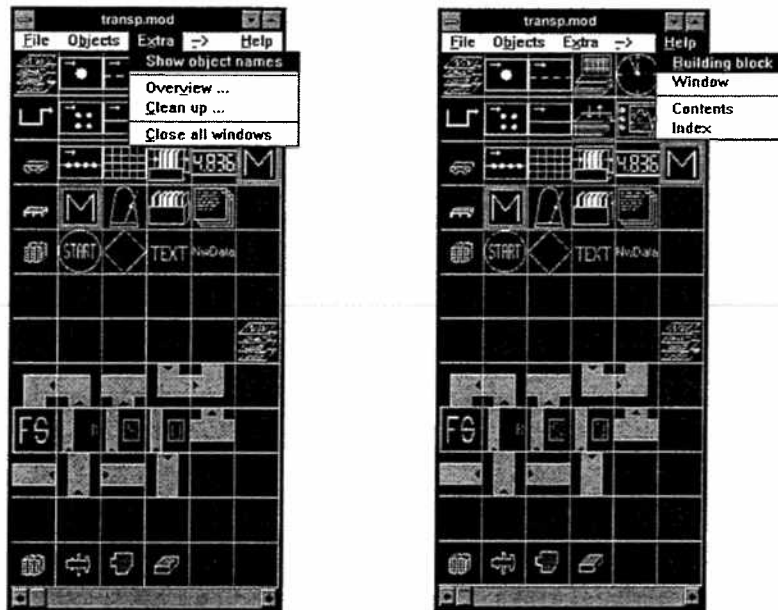
EUROSIM'95

Menus of the Class Library (1)



EUROSIM'95

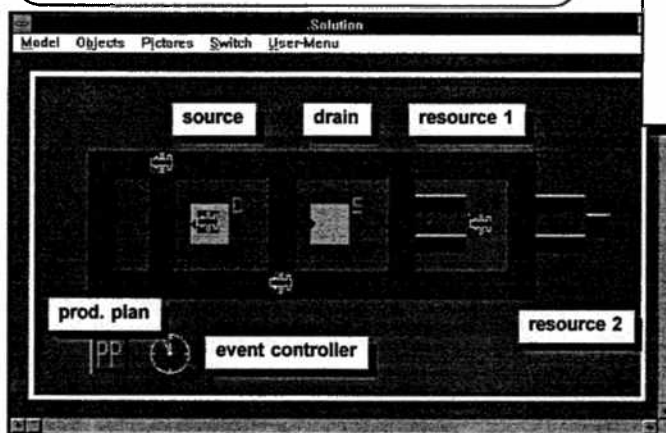
Menus of the Class Library (2)







EUROSIM'95

Modelling with SIMPLE++

+ Creating a model using pre-defined application objects

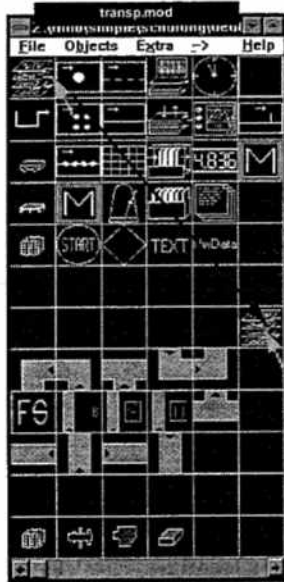


List of products

-  box
-  shaft
-  plate
-  case

EUROSIM'95

Copying an Empty Network

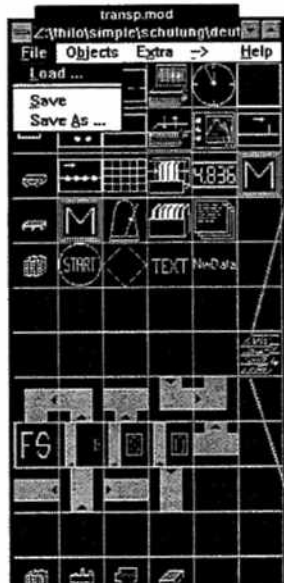


+ Step2:
Copy an empty network into an empty area of the library by clicking on the frame object then pressing the "shift" key and the left mouse button simultaneously and dragging the network to an empty field.

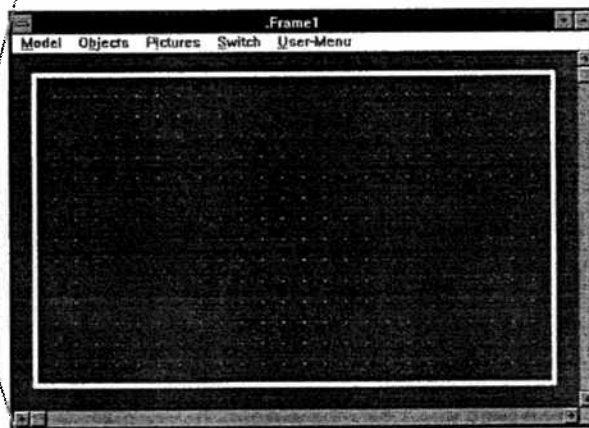
Remark:
The creation of an empty network is always the first step in creating a new model. The empty frame offered in the class library cannot be opened.

EUROSIM'95

Opening the Network

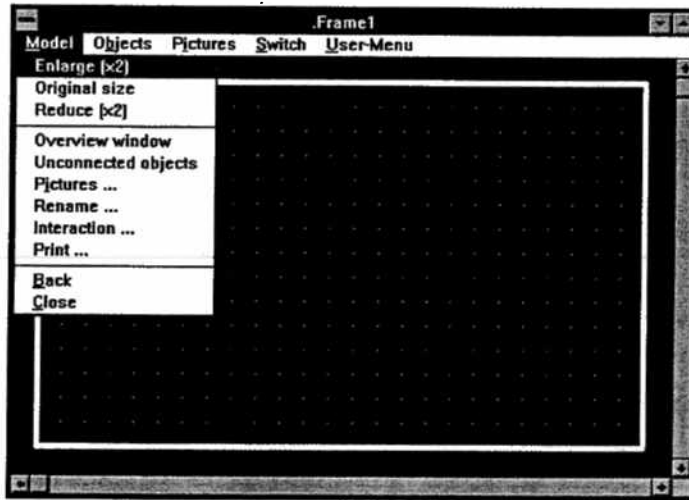


+ Step 3:
Open the copy of the empty network by double clicking on the network symbol in the library with the left mouse button.



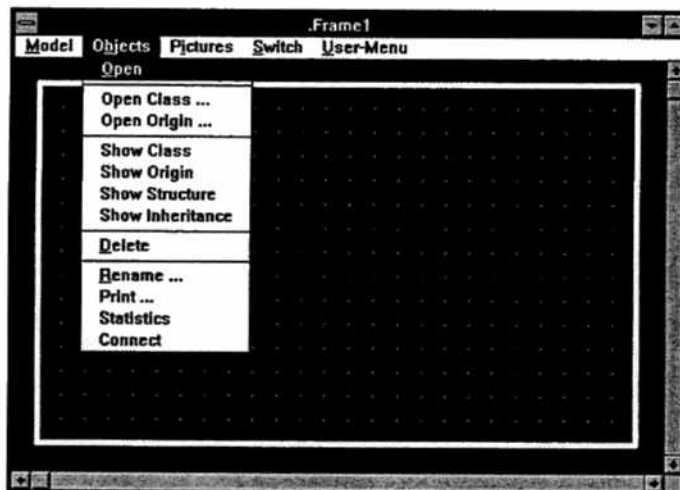
EUROSIM'95

The Frame Menu (1)



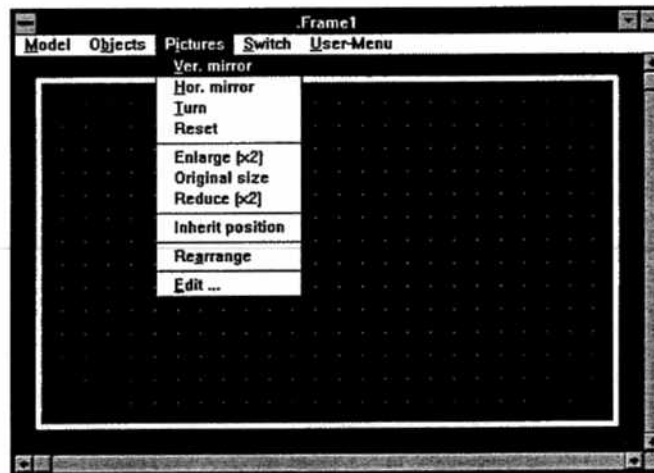
EUROSIM'95

The Frame Menu (2)



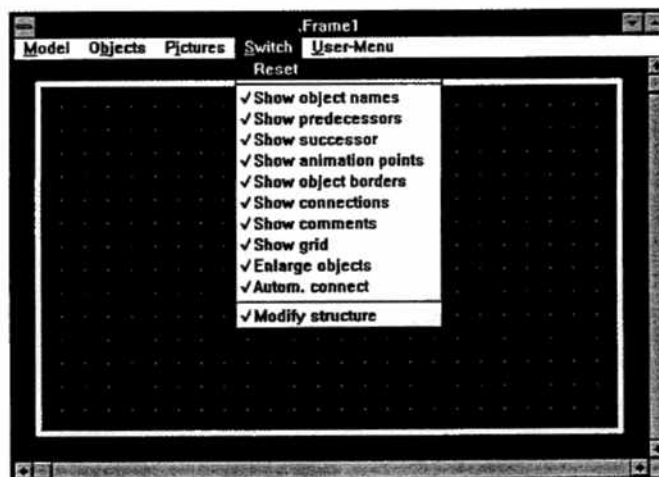
EUROSIM'95

The Frame Menu (3)



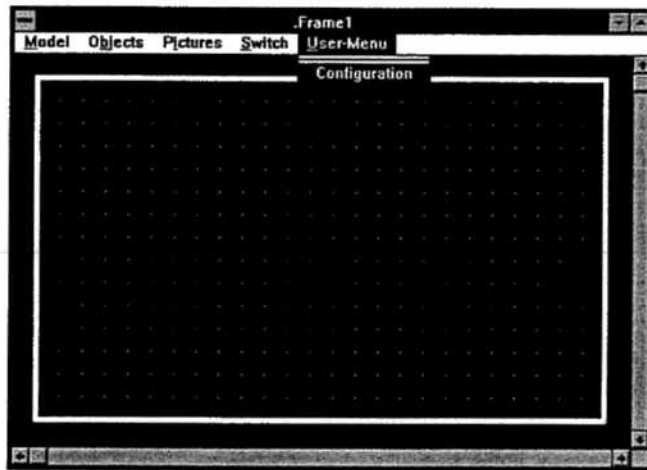
EUROSIM'95

The Frame Menu (4)



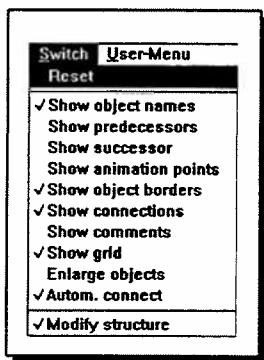
EUROSIM'95

The Frame Menu (5)



EUROSIM'95

Configuration

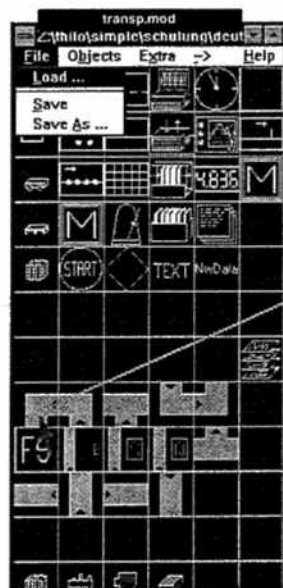


Step 4:
Before starting to model, toggle the options below the menu *Switch* as shown in the picture.

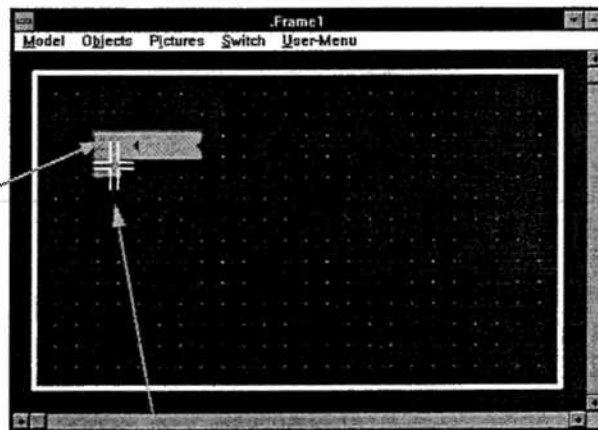
Note:
The automatic connection of building blocks is possible if there is an obvious connection between those building blocks.

EUROSIM'95

Inserting Objects into the Frame



- + Place application objects in the frame. Arrange the application objects according to the model shown previously so that they touch each other.



When inserting an object the cursor becomes a double cross.

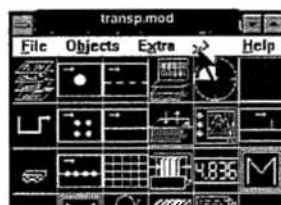
EUROSIM'95

To leave the INSERT Mode

As long as an object in the library is, an instance of this object will be placed in the frame each time you click the left mouse button within the model window.

You can deselect INSERT mode by:

- Clicking the right mouse button in the frame
- Clicking on an empty field in the class library (left mouse button)
- Clicking on the arrow in the menu bar of the class library (left mouse button)



EUROSIM'95

Connecting Objects

1. possibility

graphical, interactive with the connector tool



2. possibility

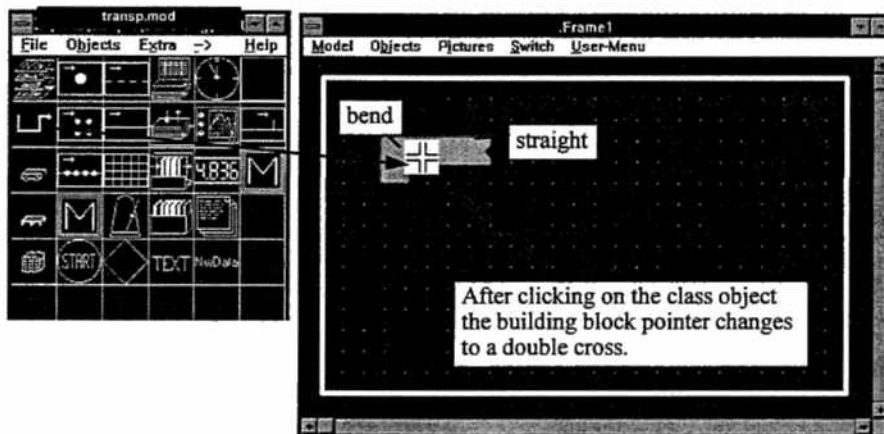
automatically while inserting building blocks
(if the Switch - Autom. connect is on)

3. possibility

automatically after inserting the building blocks
frame-menu: Objects - connect

EUROSIM'95

Interactive Connection of Objects(1)

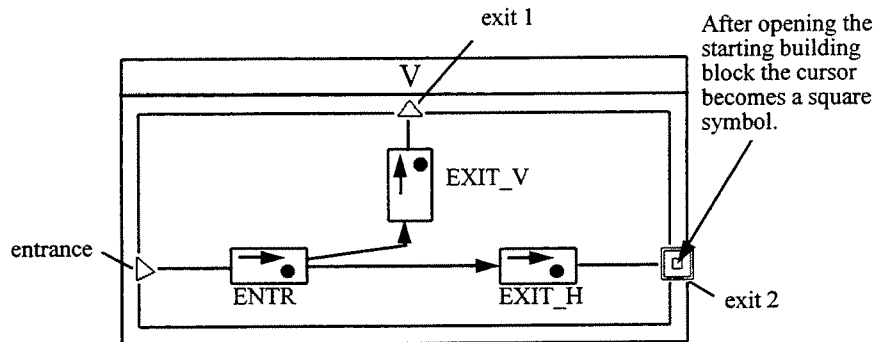


Application objects are connected manually by first selecting the *Connector* tool in the class library, then clicking on the starting building block and then on the goal building block of the desired connection.

EUROSIM'95

Interactive Connection of Objects(2)

After selecting the starting object in the model, the object is opened and, in the example below, the following picture appears, displaying the entrances and exits and the subelements of the building block. This picture only appears if the building block has more than one unconnected exit. If there is only one unconnected exit the connection is unambiguous and will be automatically closed.

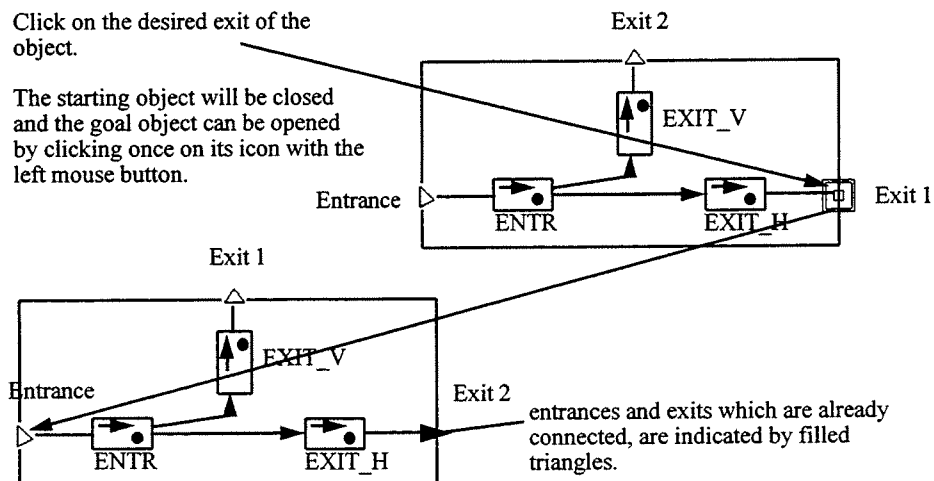


EUROSIM'95

Interactive Connection of Objects(3)

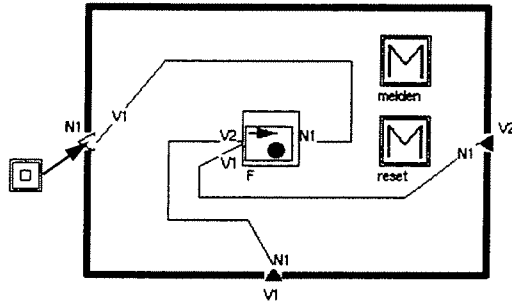
Click on the desired exit of the object.

The starting object will be closed and the goal object can be opened by clicking once on its icon with the left mouse button.



EUROSIM'95

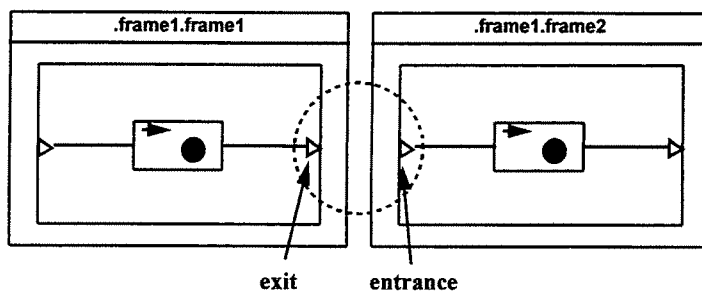
Interactive Connection of Objects (4)



The connection is established by clicking on an entrance of the goal building block. The goal building block will be closed automatically. This process may be repeated until all the entrances and exits of the model are connected.

EUROSIM'95

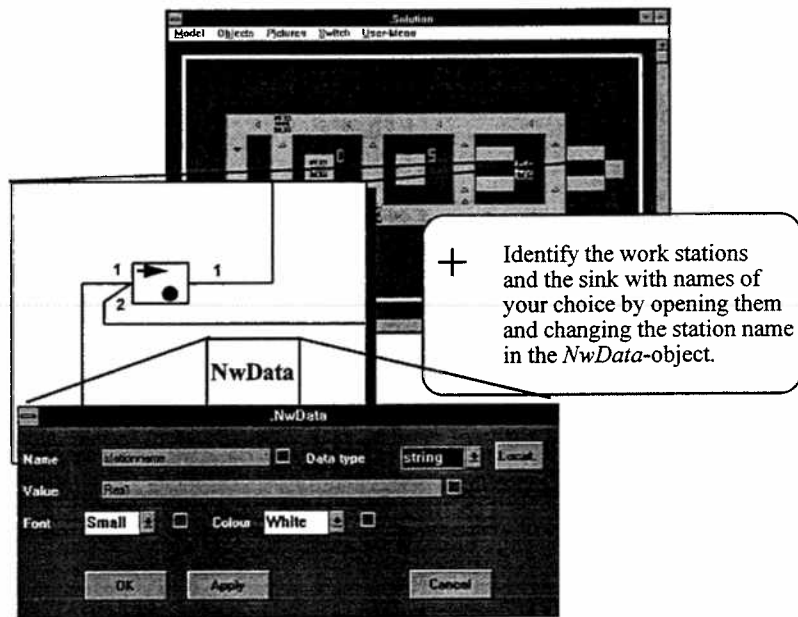
Automatic Connection of Objects



The automatic connection of building blocks is done, when the entrance and the exit of the building blocks being inserted are close together. The circular catch window has a 3 pixel radius.

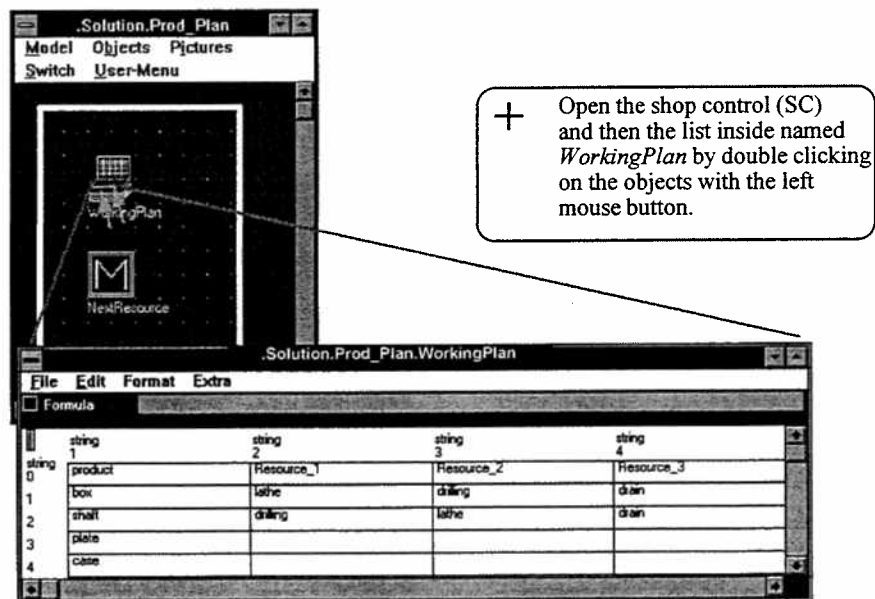
EUROSIM'95

Changing the Names of the Stations



EUROSIM'95

Building the Working Plan (1)



EUROSIM'95

Building the Working Plan (2)

- + Build the working plan using the machine names you have chosen.

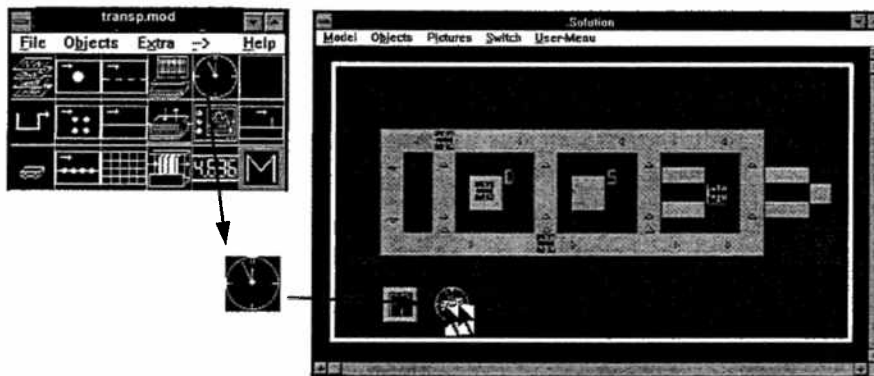
	string 1	string 2	string 3	string 4
0	product	Resource_1	Resource_2	Resource_3
1	box	lathe	drilling	drill
2	shaft	drilling	lathe	drill
3	plate	drilling	lathe	drill
4	case	drilling	lathe	drill

manufacturing sequence →

Be sure that the last work station is a drain. Otherwise the parts will not be deleted!

EUROSIM'95

Placing the *EventController* in the Model



- + Place an *EventController* in the model and open it by double clicking on it with the left mouse button.

EUROSIM'95

Starting the Simulation with the *EventController*



Step 6:

1. Click on the *Reset* button to bring the model into a defined state.
2. Click on the *Init* button to initialize the model.
3. Start the simulation by clicking the *Start* button

EUROSIM'95

Inheritance (1)

The difference between *class* and *instance*

class: each model/object in the class library. A class passes all its characteristics to its instance.

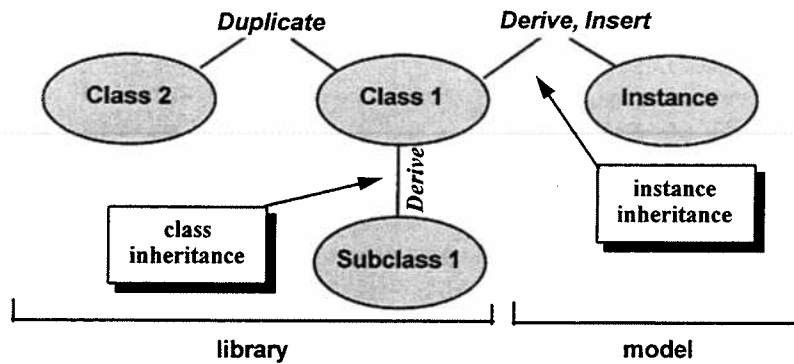
instance: specimen (child) of the class (parents) , e.g. the inserted object *Bend* is the instance of the object *Bend* in the class library.
 Building block *Bend* in library: *class*
 Inserted building block *Bend*: *instance*

Important: if changes of properties are to effect all specimens they must be changed for the *class*.

EUROSIM'95

Inheritance (2)

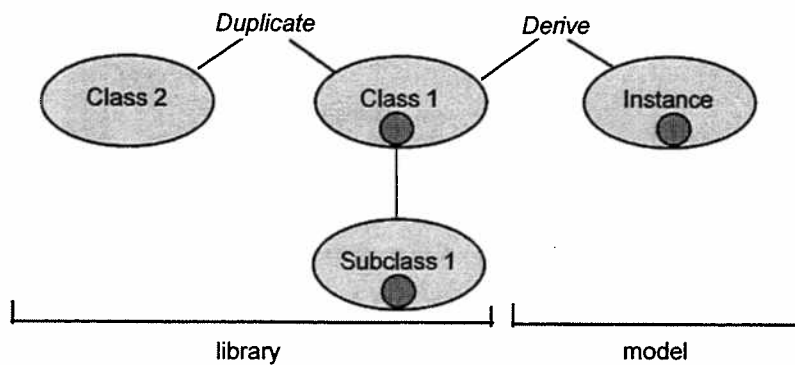
Difference between *Duplicate*(=Copy) and *Derive*



EUROSIM'95

Inheritance (3)

Situation after the change of a property



EUROSIM'95

Basic Objects -> Application Objects

- Comparison of the different generations of simulation systems
- Derivation and description of the basic objects in SIMPLE++
- The default behaviour of the SIMPLE++ - building blocks

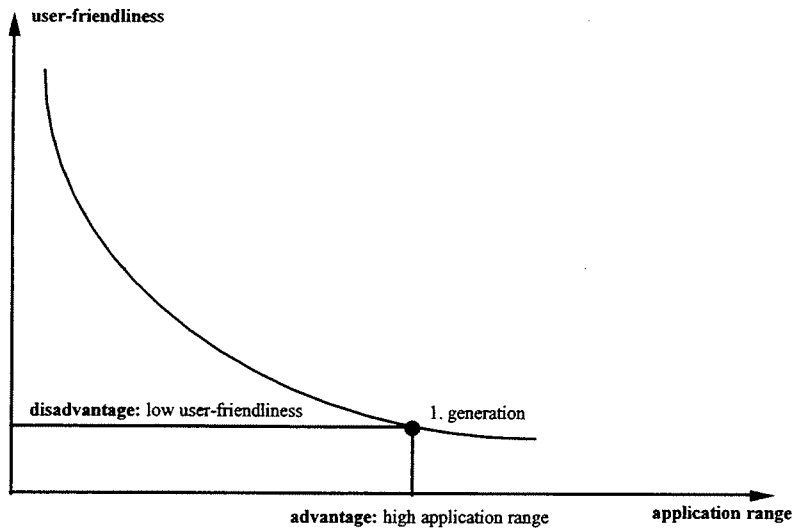
EUROSIM'95

Generations of Simulation Systems

- 3. generation:** object oriented modeling
 - + objects closely match the reality
 - + predefined basic building blocks
 - + user specific modelling by change of attributes and flexible control strategies
 - + graphical object oriented user surface
 - > SIMPLE++
- 2. generation:** parametrical modelling
 - + simplified modelling
 - only simple control strategies from a catalog
 - user control strategies have to be programmed
 - > SIMULAP, DOSIMIS III, etc.
- 1. generation:** linguistic modelling
 - + general modelling
 - + flexible use
 - knowledge in programming necessary
 - > SIMAN, SLAM, GPSS, SIMULA, etc.

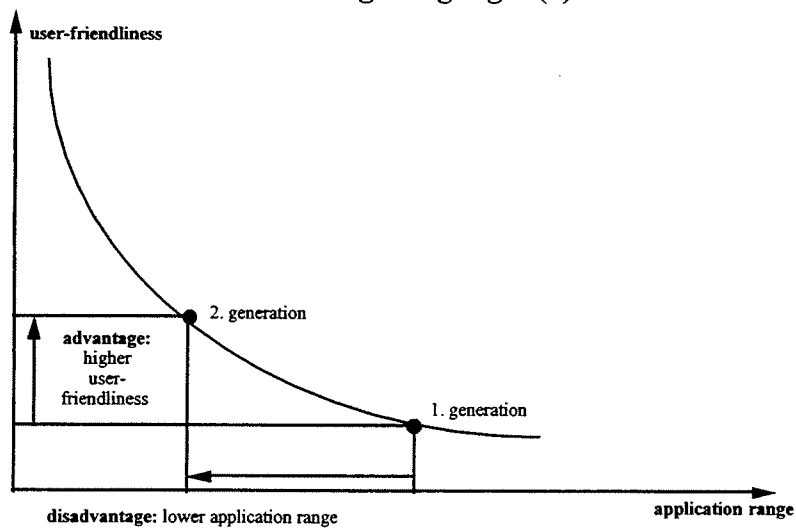
EUROSIM'95

Application Range and User-Friendliness of Modelling Languages (1)



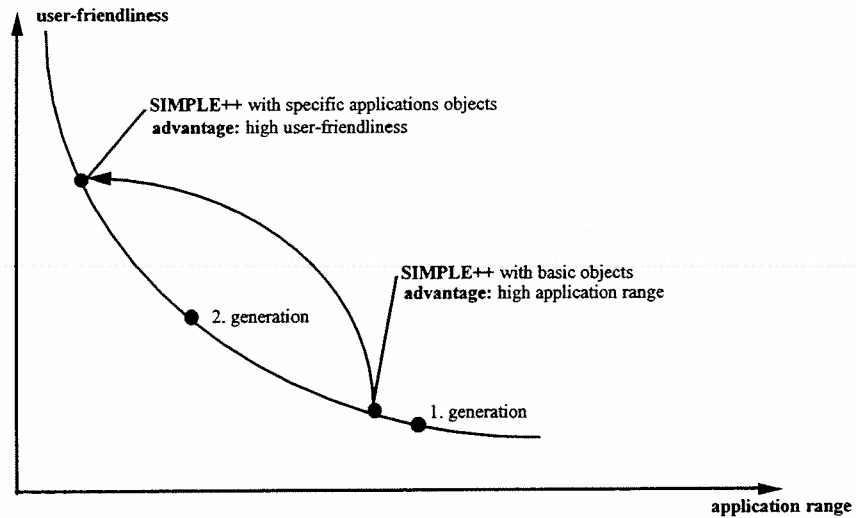
EUROSIM'95

Application Range and User-Friendliness of Modelling Languages (2)



EUROSIM'95

Application Range and User-Friendliness of Modelling Languages (3)



EUROSIM'95

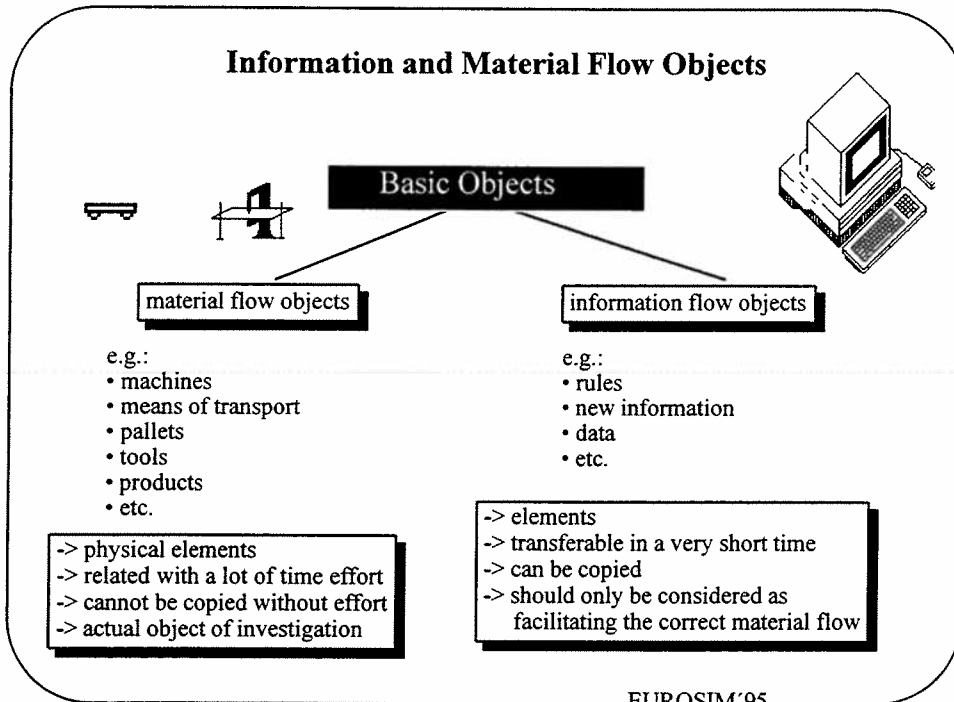
Structure of the Basic Objects

Basic Objects							
material flow objects				information flow objects			
movable		immovable		movable		immovable	
active	passive	active	passive	active	passive	active	passive
• Transporter	• Container • Entity	• SingleProc • ParallelProc • SerialProc • Line	• Track • Warehouse		• Attributes	• Method • Generator	• TableFile • StackFile • QueueFile • CardFile • SQL-interface • FileInterface

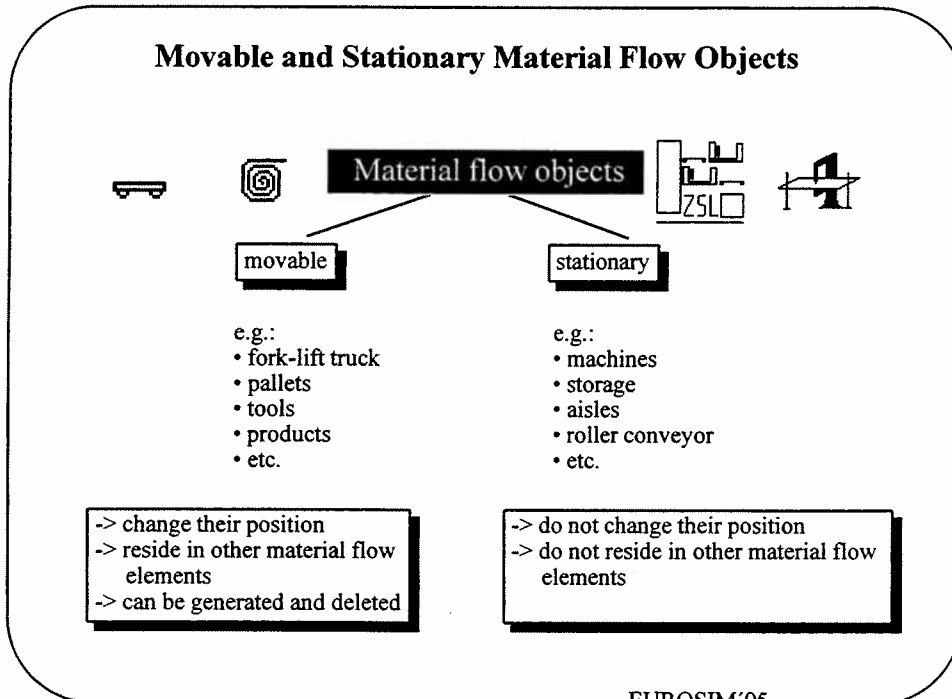
Additional Service Objects: *Plotter, Gauge, Dialog, Text, Connector, EventController*

EUROSIM'95

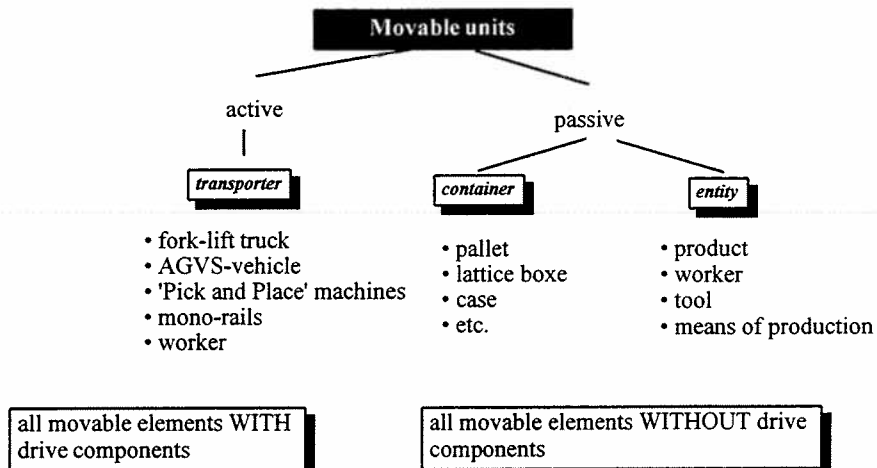
Information and Material Flow Objects



Movable and Stationary Material Flow Objects

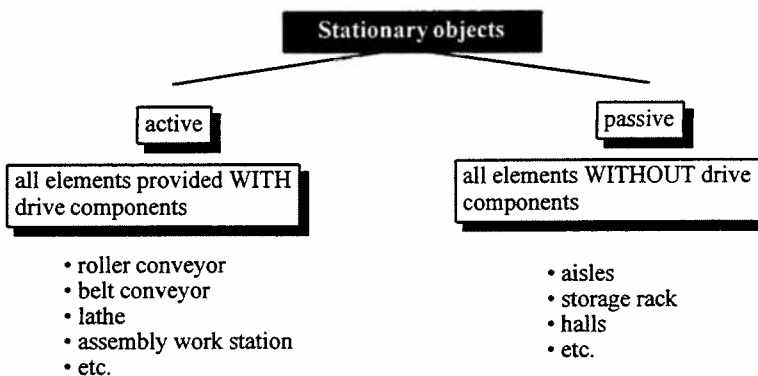


Movable Units (MU)



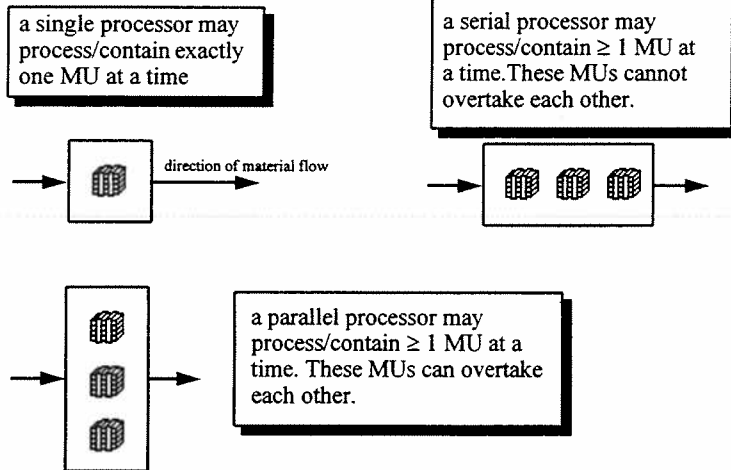
EUROSIM'95

Stationary Material Flow Objects



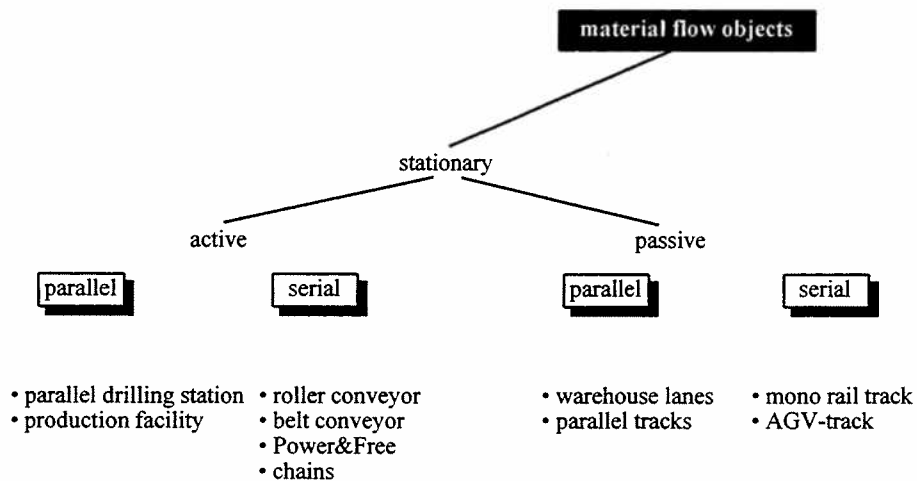
EUROSIM'95

Serial and Parallel Stationary Objects (1)



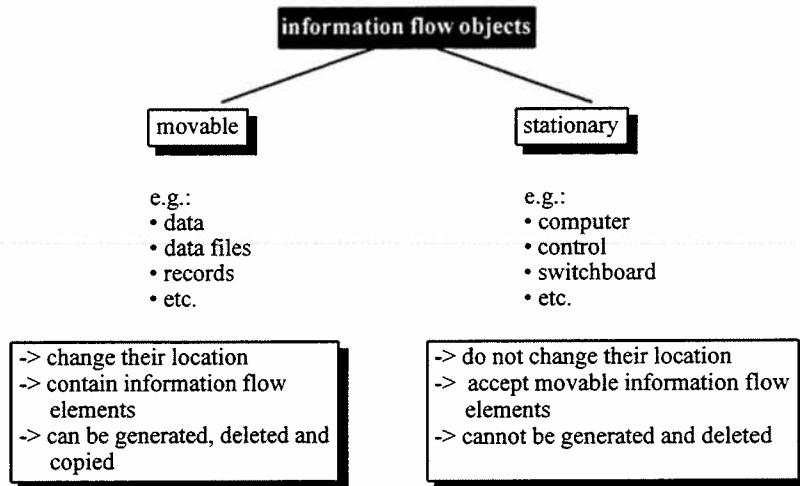
EUROSIM'95

Serial and Parallel Stationary Objects (2)



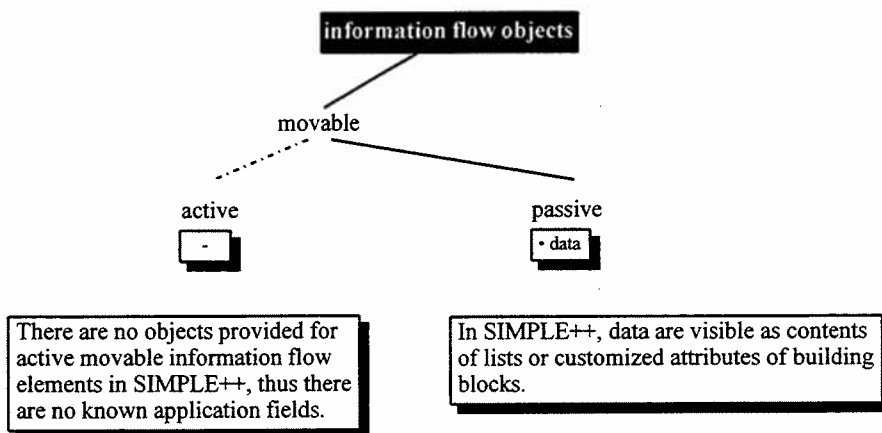
EUROSIM'95

Information Flow Objects



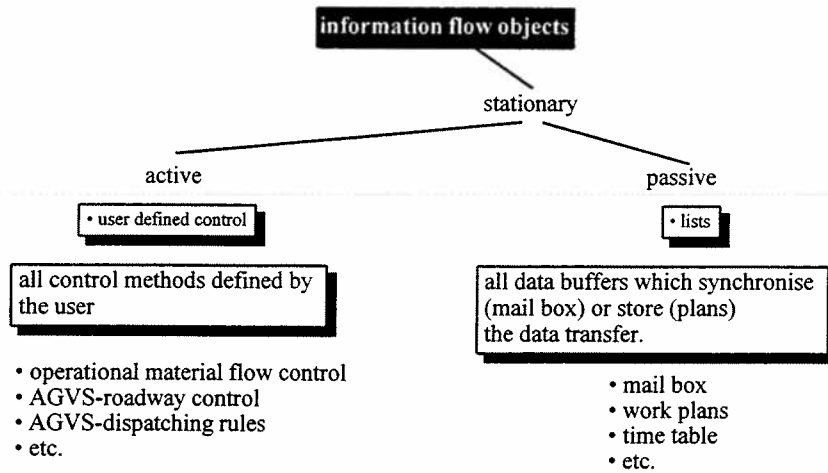
EUROSIM'95

Movable Information Flow Objects



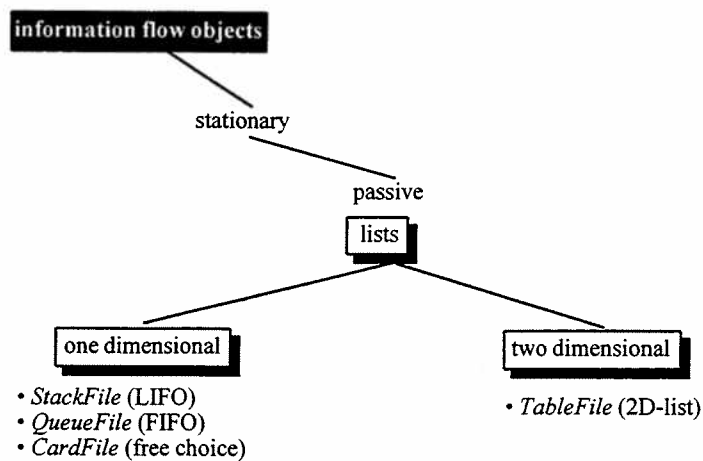
EUROSIM'95

Stationary Information Flow Objects















EUROSIM'95

Lists















EUROSIM'95

Names of the Basic Objects (1)

 <i>Frame</i>	 <i>Warehouse</i>
 <i>SingleProc</i>	 <i>Entity</i>
 <i>ParallelProc</i>	 <i>Container</i>
 <i>SerialProc</i>	 <i>Transporter</i>
 <i>Line</i>	 <i>Connector</i>
 <i>Track</i>	 <i>EventController</i>

EUROSIM'95

Names of the Basic Objects (2)

 <i>StackFile (LIFO)</i>	 <i>Generator</i>
 <i>QueueFile (FIFO)</i>	 <i>Method</i>
 <i>CardFile</i>	 <i>Comment</i>
 <i>TableFile</i>	 <i>Display</i>
 <i>NwData</i>	 <i>Plotter</i>
 <i>FileInterface</i>	 <i>Dialog</i>

EUROSIM'95

Basic Material Flow Objects (1)



SingleProc

immovable	cannot change its place during a simulation run
active	takes in an MU and tries to pass it on the successor after the processing time
capacity = 1	only one MU can stay in the SingleProc
place oriented	the length of an MU is disregarded
possible application	machine with capacity one, segment of a conveyor, buffer, etc.

EUROSIM'95

Basic Material Flow Objects (2)



ParallelProc

immovable	cannot change its place during a simulation run
active	takes in an MU and tries to pass it on the successor after the processing time
capacity = n	more than one MU can stay in the ParallelProc at the same time. It's possible to set a different processing time for each MU. An MU can over-take another one.
place oriented	the length of an MU is disregarded
possible application	machine with capacity > 1, etc.

EUROSIM'95

Basic Material Flow Objects (3)



SerialProc

immovable active	cannot change its place during a simulation run
capacity = n	the number of parallel lines can be defined. An MU cannot over-take another one in the same line.
place oriented	the length of an MU is disregarded
jamable/un-	user defined setting
possible application	conveyor, buffer, etc.

EUROSIM'95

Basic Material Flow Objects (4)



Line

immovable active	cannot change its place during a simulation run
capacity	the length of the line can be freely defined. An MU cannot over-take another one.
length oriented	the length of an MU is regarded, i.e. the max. number of MUs on a line depends on the length of the line and the length of the MUs.
accumulating	user defined setting
possible application	conveyor bend with segments of different lengths, etc.

EUROSIM'95

Basic Material Flow Objects (5)



Track

immovable

passive used to realise a track of an active MU
 (e.g. a transporter).

capacity the length of the track can be freely defined.
 An active MU cannot over-take another one.

length oriented

possible application AGV-track, etc.

EUROSIM'95

Basic Material Flow Objects (6)



Warehouse

immovable

passive stores passive MUs (e.g. a container)
capacity the storage capacity could be freely defined in
 a x/y-matrix

place oriented

possible application aisle, shelf, etc.

EUROSIM'95

Basic Material Flow Objects (7)



Entity

movable	moves during a simulation run. An entity's location can be an immovable basic building block, a transporter or a container. The length of an entity can be freely defined.
passive	an entity has no own drive
capacity=0	cannot hold another part
possible application	parts to be produced or transported, etc.

EUROSIM'95

Basic Material Flow Objects (8)



Container

movable	
passive	
capacity=n	the capacity of a container can be freely defined as a x/y-matrix.
place oriented	
possible application	pallet, box, etc.

EUROSIM'95

Basic Material Flow Objects (9)



Transporter

movable

active the transporter has its own drive. The velocity is freely definable

capacity=n the capacity of a transporter can be freely defined as a x/y-matrix.

place oriented

possible application AGV, EOM, fork lifting truck, etc.

EUROSIM'95

The Application Objects *SingleProc*

material flow objects			
movable		immovable	
active	passive	active	passive
• transporter	• container • entity	• <i>SingleProc</i> • <i>ParallelProc</i> • <i>SerialProc</i> • <i>Line</i>	• track • warehouse

Characteristics:

• active material flow object

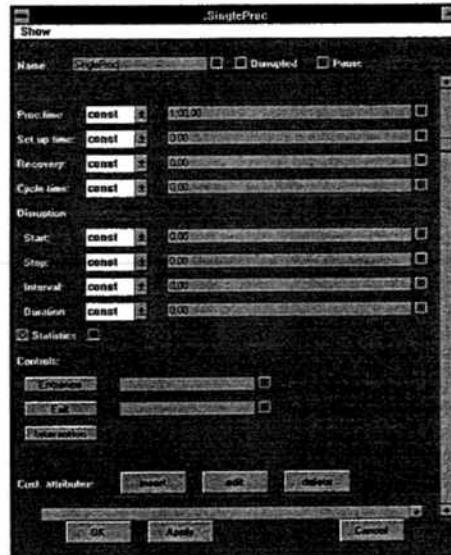
• capacity: 1

• icon:



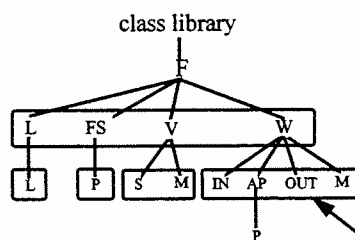
EUROSIM'95

The Dialog Window of the *SingleProc*



EUROSIM'95

Name Scope



- All objects having a common parent element in the model hierarchy (this means that they are included in the same object) constitute a *name scope*.

- Within a name scope, the names have to be unique.
- But is possible to use the same names in different name scopes.

EUROSIM'95

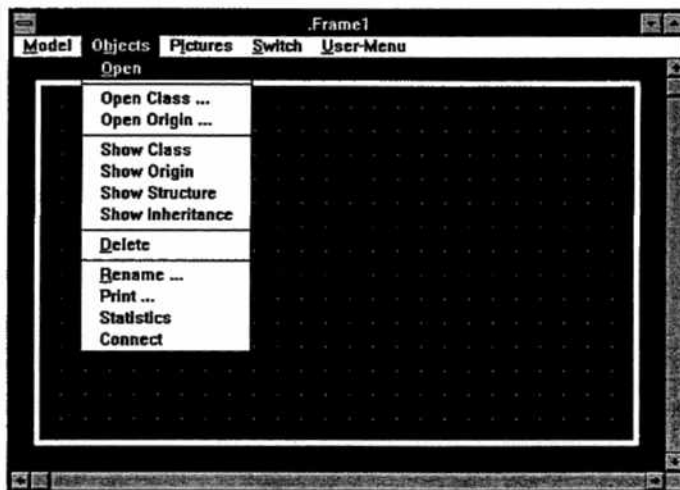
Renaming Objects in the Library

Objects placed in the library can be renamed by selecting the object and then clicking on the menu item *Rename*.



EUROSIM'95

Renaming already Inserted Objects



Already inserted objects can be renamed by selecting the object and clicking on the menu item *Rename* in the model *Objects* menu.

EUROSIM'95

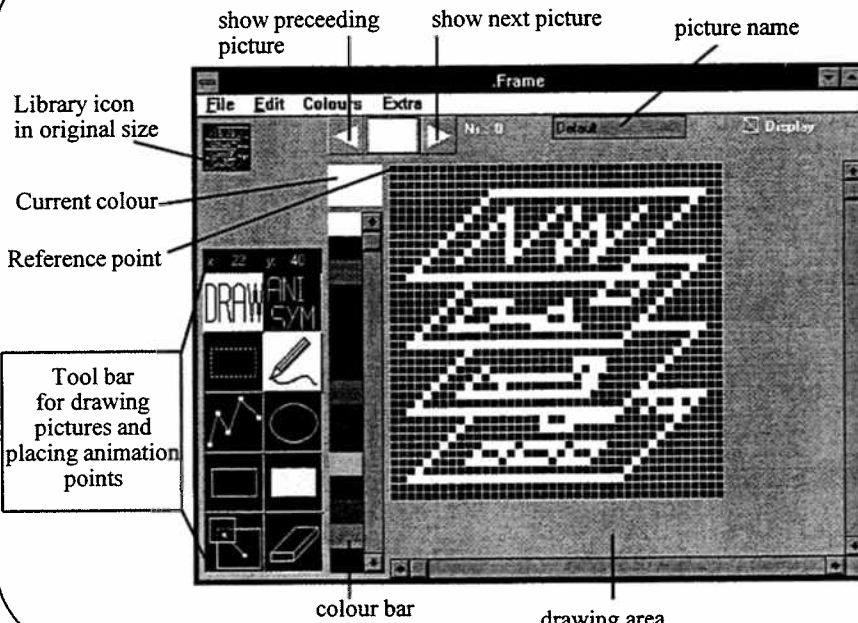
Opening the Picture Editor

The picture editor is opened by first clicking on the desired object and then selecting the menu item *Pictures...*



EUROSIM'95

The Picture Editor



EUROSIM'95

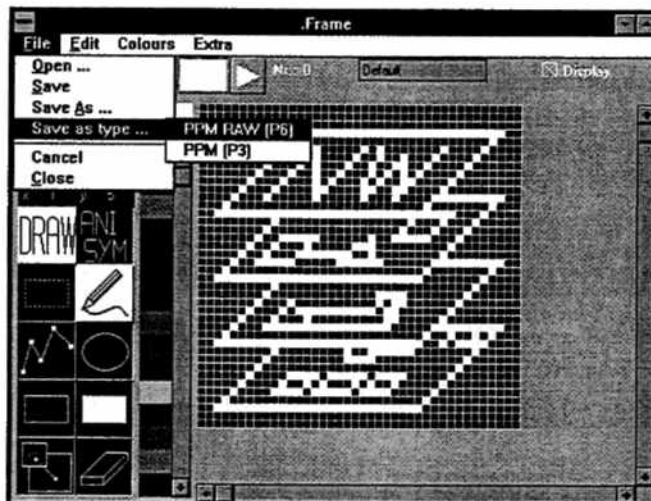
Pictures

- One or more icons for each* object in the class library can be defined.
- Each icon gets a number from the system and can be named by the user. The icon names for one object must be unique.
- The number of icons is unlimited.
- The maximum size of an icon is 999x999 pixels.
- Each object has an icon numbered with 0 and named *Default*. This is the icon the object is shown in the library. Therefore it has a maximum size of 40x40 pixels.
- Icons require a lot of memory. Therefore they are associated with the class and not with the instances of the class. Consequently, if the icons of an instance of a class are changed, the icons of all instances of the class, including the parent class, are changed too.

* with the exception of the objects with fixed icons - *EventController*, *Comment*, *Connector* and *NwData*

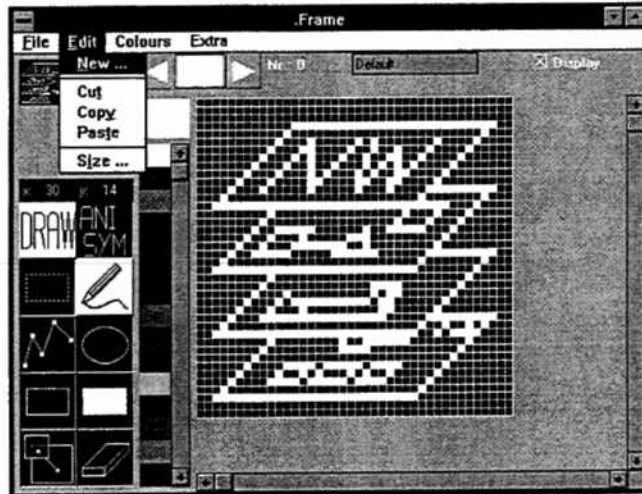
EUROSIM'95

The Menus of the Picture Editor (1)



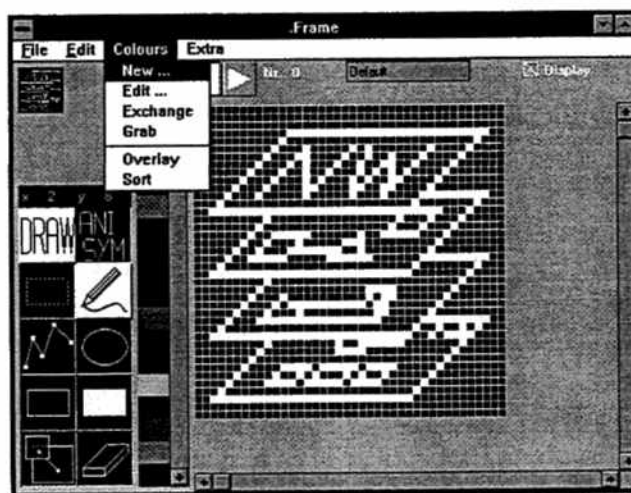
EUROSIM'95

The Menus of the Picture Editor (2)



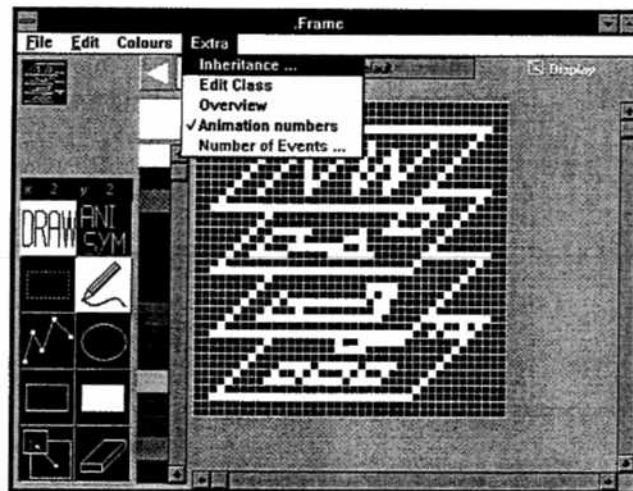
EUROSIM'95

The Menus of the Picture Editor (3)



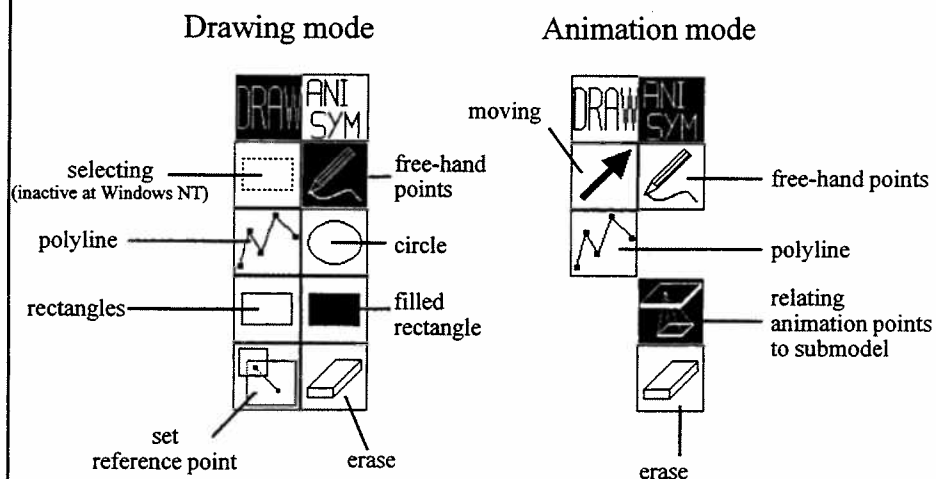
EUROSIM'95

The Menus of the Picture Editor (4)



EUROSIM'95

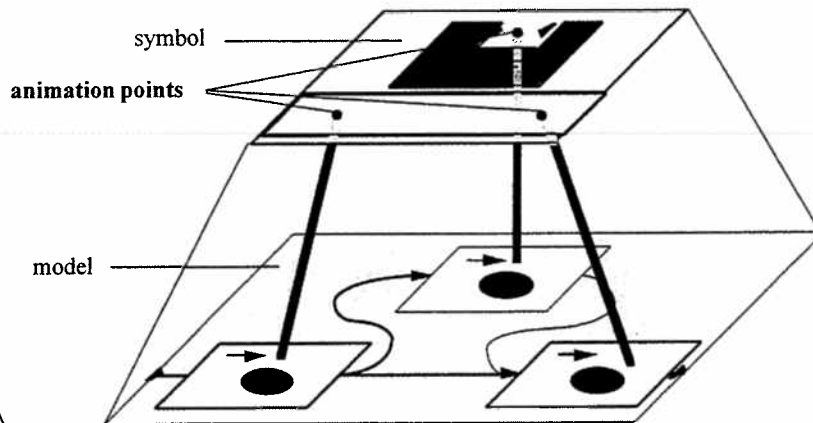
The Drawing Tool Bar



EUROSIM'95

Symbols and Animation Points

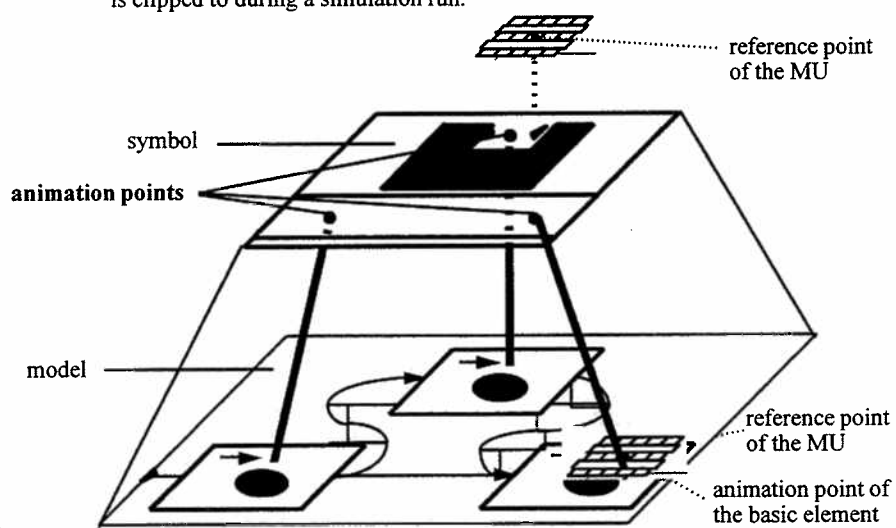
To visualise the MUs, which are moving through the logical components of the model, **animation points** have to be placed in the picture of the symbolic level. Correct animation requires an assignment of these animation points to the logical level.



EUROSIM'95

Animation Points

Animation points are 'attachment points', where the picture of a MU is clipped to during a simulation run.

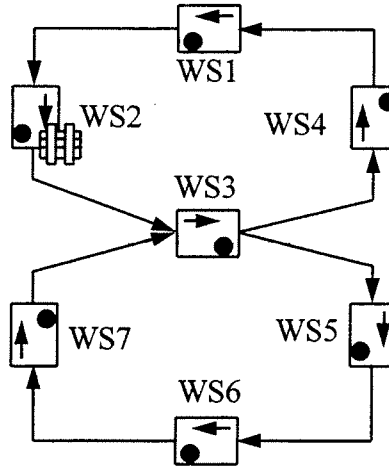


EUROSIM'95

Default Material Flow Behaviour at a Divergence Point (1)

+ Create a model according to the following picture:

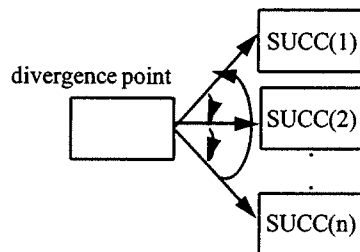
+ Start the simulation and observe the transfer behaviour at the divergence point.



EUROSIM'95

Default Material Flow Behaviour at a Divergence Point (2)

If no user control is specified, the departing entities are transferred to the succeeding building blocks in the sequence in which these were connected to the divergence point

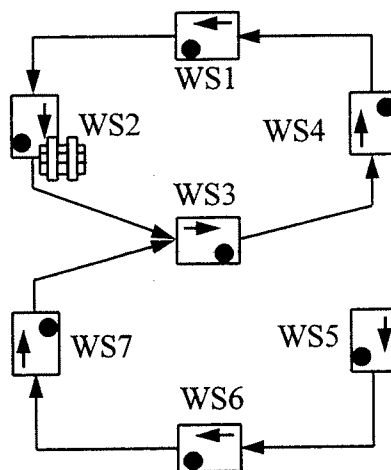


Thereby Simple models can be constructed without developing special control rules.

EUROSIM'95

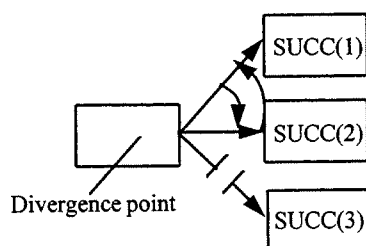
Exercise

+ Delete one connector, start the simulation again and describe the transfer behaviour.



EUROSIM'95

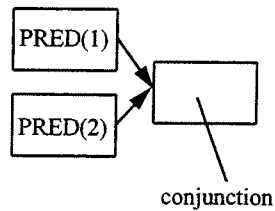
Default Material Flow Behaviour at a Divergence Point (3)



In SIMPLE ++ the departing entities are transferred alternately to the successors which still exist.

EUROSIM'95

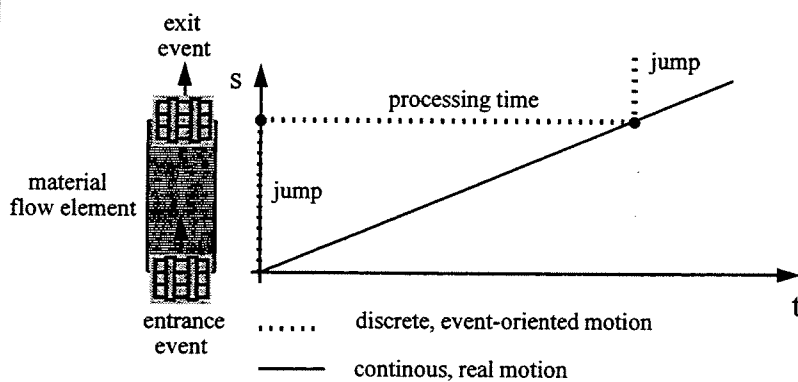
Default Material Flow Behaviour at a Convergence Point



In SIMPLE ++ entities are transferred to the successor following the FIFO-rule (first come, first served).

EUROSIM'95

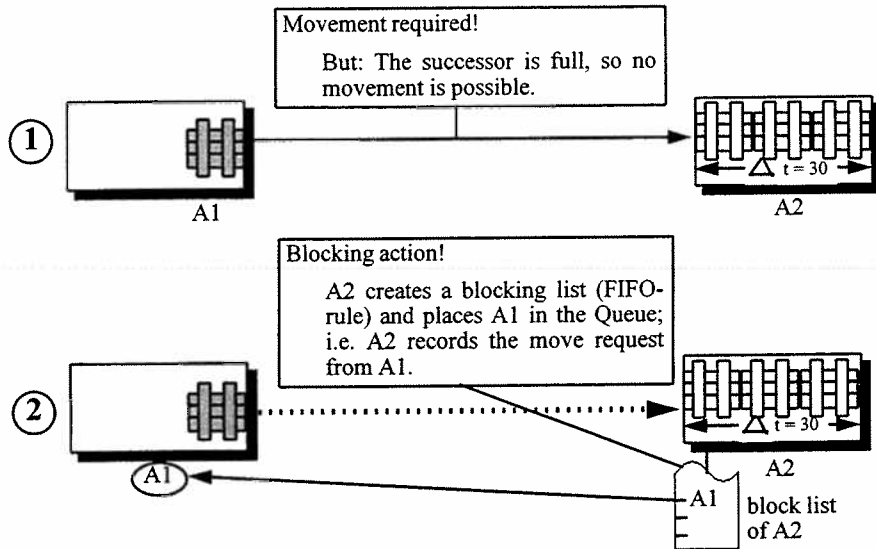
Continuous vs. Discrete Movement



Note: On current digital computers, only discretised simulation is possible. The larger the jump in time, the more calculation time is available for simulation.

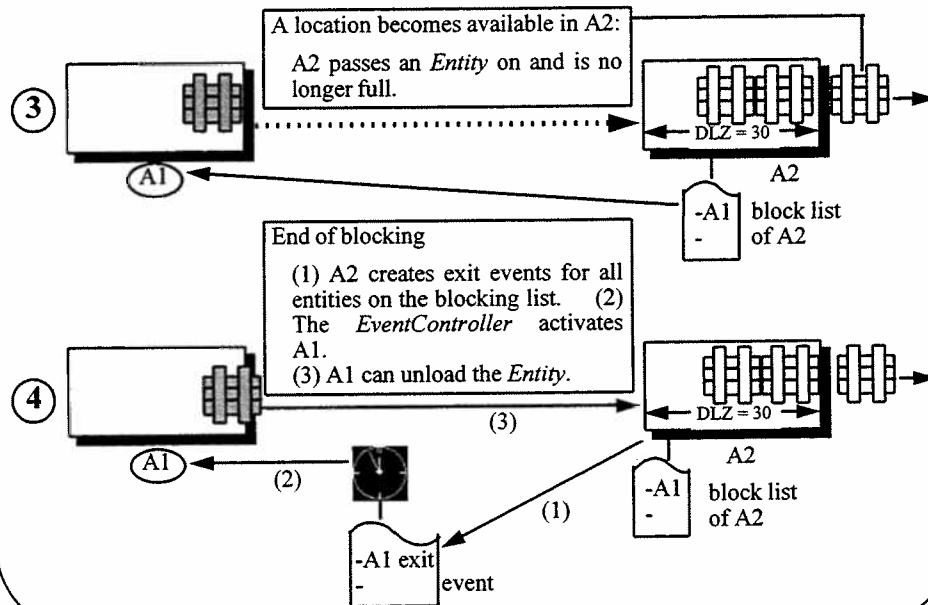
EUROSIM'95

Internal Realisation of the Push-Block Concept (1)



EUROSIM'95

Internal Realisation of the Push-Block Concept (2)

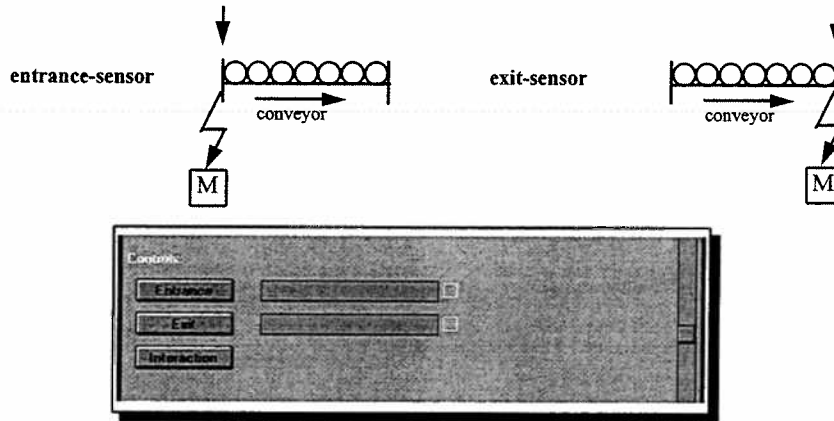


EUROSIM'95

The Sensor-Actor Concept (part 1)

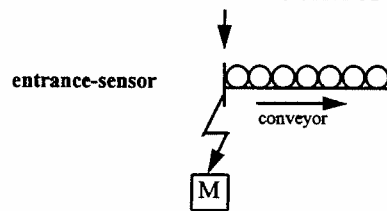
The basic material flow elements (*SingleProc*, *ParallelProc*, ...) have an *Entrance-Control* and an *Exit-Control*.

A method specified as an entrance or exit control will be activated, whenever a MU wants to enter or leave that material flow element.



EUROSIM'95

The Sensor-Actor Concept The Entrance Control



Activation:

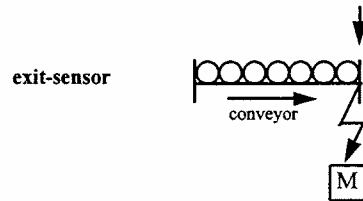
- the entrance control is activated when the MU successfully enters the building block

Important:

- when the entrance control is activated, the MU has already moved onto that building block, i.e. the change of the processing time within the entrance control does not affect the present MU.
- the entrance control does overwrite the default behaviour of the building block
- the entrance control can be activated only once for each entering MU

EUROSIM'95

The Sensor-Actor Concept The Exit Control



Activation:

- the exit control is activated whenever an MU wants to leave its location

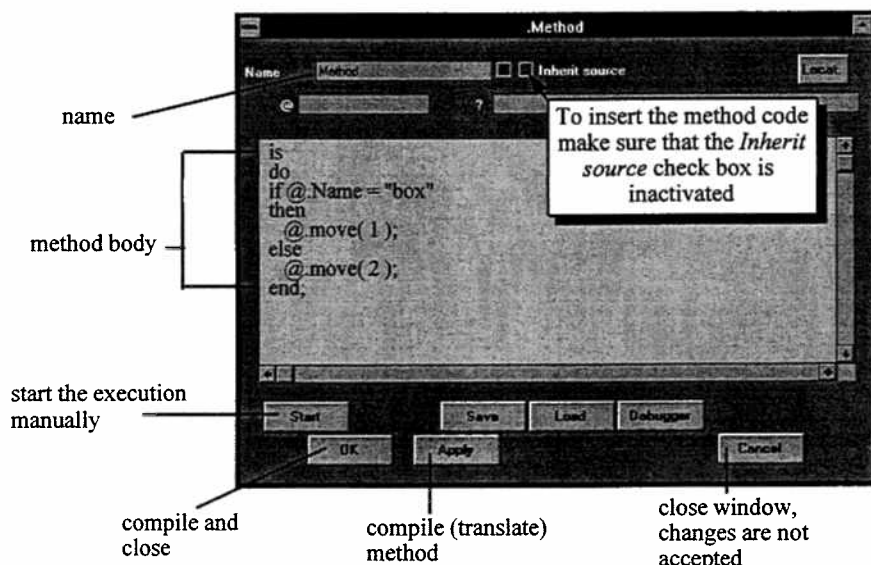
Important:

- the exit control overwrites the default behaviour of the building block, i.e. the MU will **not be moved** to the successor **automatically**
- the exit control may be activated more than once by a single MU.

Reason: If the MU is blocked and cannot leave its location, the exit control will be activated a second time when the MU becomes operational again.

EUROSIM'95

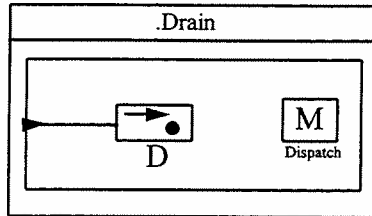
Construction of a Method (Control)



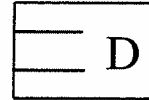
EUROSIM'95

The Application Object *Drain* (1)

model:



symbol:



Exercise:

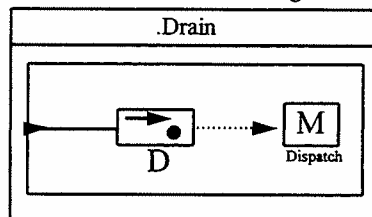
1. open the pre-defined building block *Term*
2. insert the basic object *SingleProc*
3. insert the basic object *Method*
4. connect the model entrance with the *SingleProc* (manually)
5. rename the the building block
6. draw a separate picture for the model
7. define an animation point
8. assign the animation point to the *SingleProc*

familiar

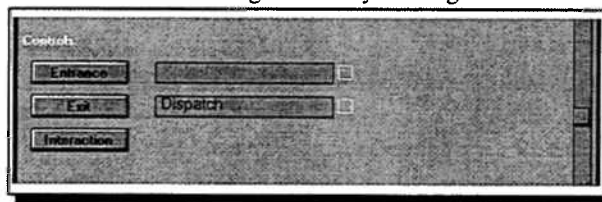
EUROSIM'95

The Application Object *Drain* (2)

9. define the control method *Dispatch* as the exit control of the user-defined building block *D*.



- a. open the dialog window of the *SingleProc*
- b. enter the name of the exit control method
- c. close the dialog window by clicking on the *OK*-button



new

EUROSIM'95

The Application Object *Drain* (3)

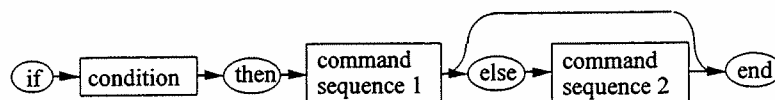
10. define the exit control method
- open the window of the *Dispatch* method with a double click.
 - switch off the *Inherit source*-button
 - enter the method the code shown below
 - close the window with the *OK*-button

new

EUROSIM'95

Flow Control Structures - Conditional Branch

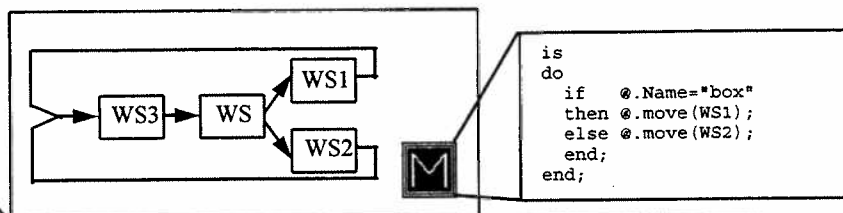
The conditional branch is used to make the execution of command sequences depending on a certain condition.



If the condition is fulfilled (TRUE), then command sequence 1 is executed, otherwise command sequence 2 is executed, if an else branch is specified.

The *else* -branch is optional.

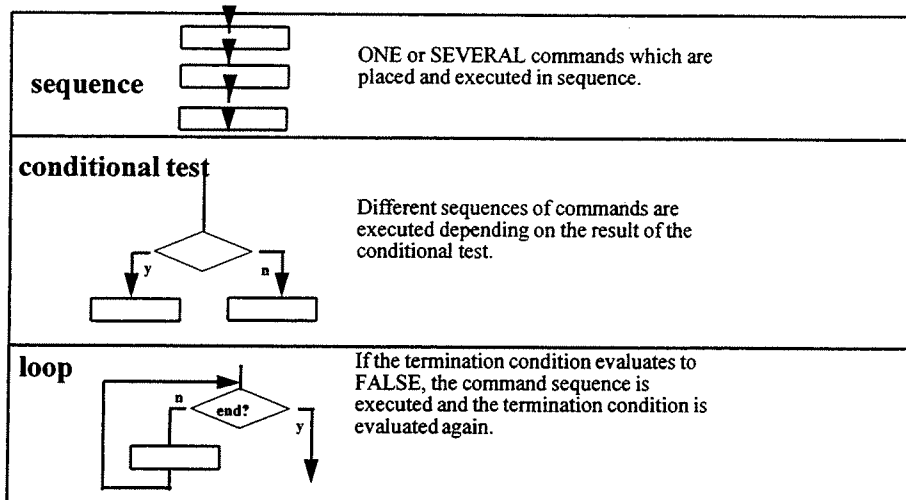
- + Build a model with a method using the conditional check, e.g.:



EUROSIM'95

Flow Control Structures

Flow control structures are used to control the sequence of commands executed in the information flow language *SimTalk*. *SimTalk* offers three different flow control structures:



EUROSIM'95

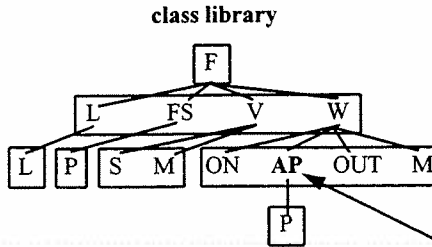
How to create Addresses?

Object addresses - overview

- relative address
- absolute address
- anonymous names
- navigating
 - through the model hierarchy
 - through the logical network

EUROSIM'95

Relative Address



A relative address is always relative to the name scope, i.e. the hierarchical level of the model where it is used (as opposed to an absolute address stating with the class library). In the example below, SIMPLE++ searches the name scope of method *M* for an object with the name *AP*.

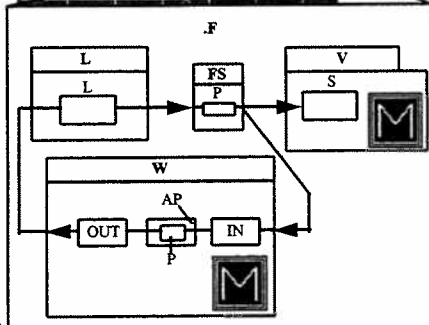
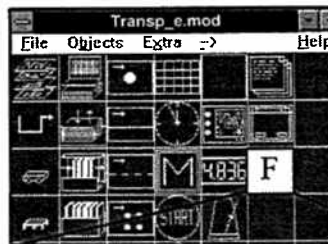
- + Insert the following lines into the method Method *.F.W.M*:


```
is
do
  print AP;
end;
```
- + Click *Apply* and *Start* and watch the result in the standard output.

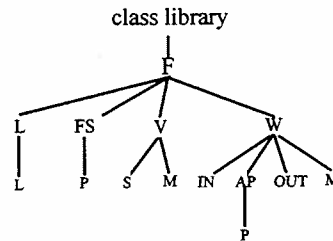
EUROSIM'95

Absolute Address

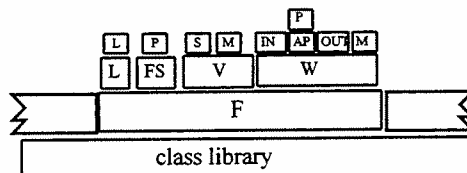
absolute address



model hierarchy



"construction pyramid"



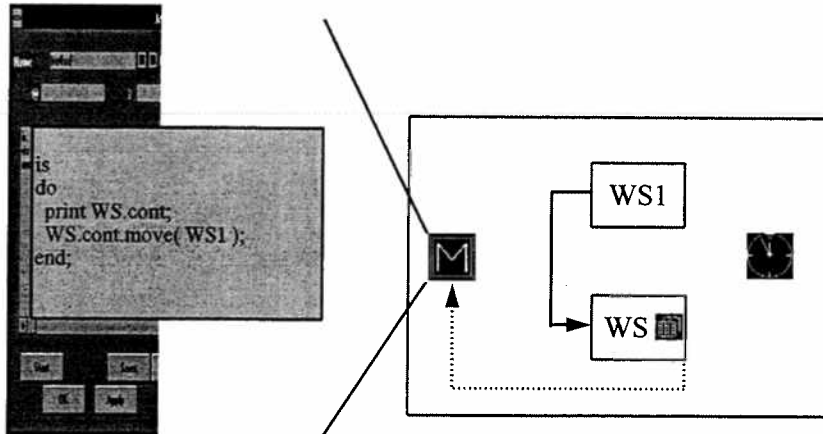
EUROSIM'95

The Anonymous Name *cont*

Cont returns the address of the MU being loaded on an object.

+

- Build the model shown below (WS and WS1 have to be *SingleProcs*).
- Insert M as the exit control of WS.
- Start the simulation and watch the animation and the standard output.



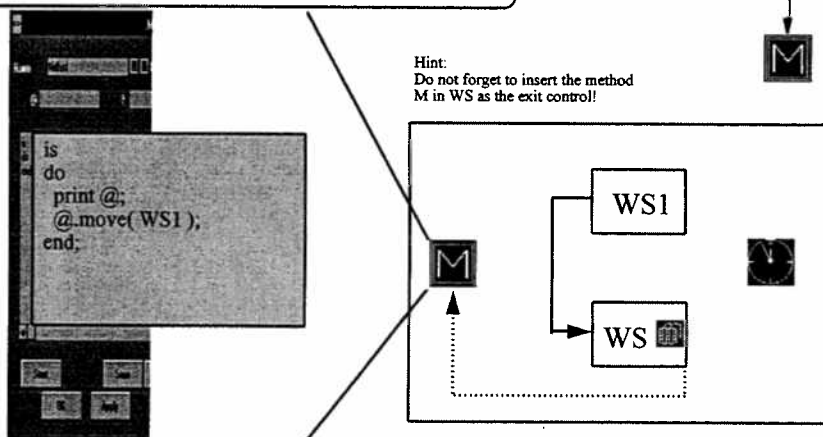
EUROSIM'95

The Anonymous Name @

@ is the address of the movable unit which triggered the current event.

+

- Build the model described below.
- Start the simulation, watch the standard output and the animation.

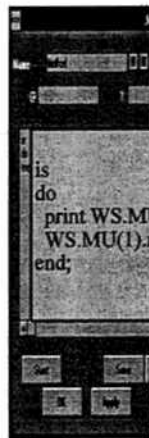


Hint:
Do not forget to insert the method
M in WS as the exit control!

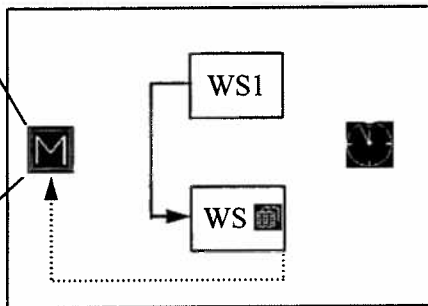
EUROSIM'95

The Anonymous Name $MU(n)$ (1)

$MU(n)$ returns the address of the n-th MU in a stationary material flow object (e.g. *SingleProc*, *ParallelProc*,...). (The MUs are arranged in the departure order).



- Change the latest model to the model shown below.
- Start the simulation and watch the animation and the standard output.



EUROSIM'95

The Anonymous Name *current*

current is a pointer to the location of the current method.



Insert the following lines in a method, click *Apply* and *Start* and watch the standard output:

```
is
do
  print current;
end;
```

EUROSIM'95

Further Anonymous Names

In the navigating chapter you've got to know some more anonymous names:

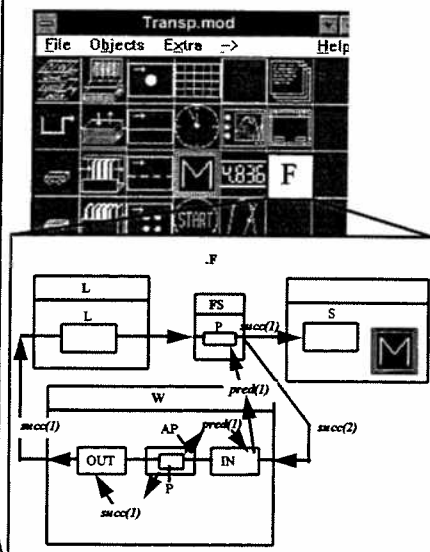
- *location*
- *pred(n)*
- *succ(n)*
- *root*



Think of an example for each construct. Make sure that you are able to use them in the right way.

EUROSIM'95

Navigating through the Logical Networks



- Insert the following lines in the method building block *F.W.M.*

```
is
do
  print AP.succ(1);
  print AP.succ(1).succ(1);
  print AP.pred(1);
  print OUT.pred(1).pred(1);
  print OUT.succ(1).succ(1).succ(2);
end;
```

- Click on *Apply* and *Start* and watch the result in the standard output.

From a given object you can move through the logical network. With ... *succ(n)* you access the n-th successor and with ... *pred(n)* the n-th predecessor.

EUROSIM'95

Customised Attributes

Customised attributes can be defined for all material flow objects and lists. The customised attributes can be used to store information like part type, part routing, order number, etc.

To insert a customised attribute, open the data dialog box of the basic objects and move the scroll bar to the bottom. The field *Cust. attributes* will appear. After clicking the *insert*-button a new window is displayed in which you can insert the name, type and value of the customised attribute.

The number of attributes is not limited.

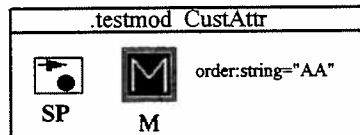
A customised attribute consists of a name, a data type and a value.

EUROSIM'95

Accessing the Value of a Customised Attribute (1)

+ Create the following test model

layout

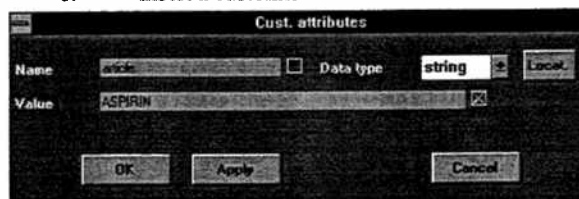


Exercise:

familiar

new

1. copy an empty model in the library
2. open the model
3. insert a basic object *Method*
4. insert a basic object *SingleProc*
5. rename the basic object to *SP*
6. insert a customised attribute for *SP*



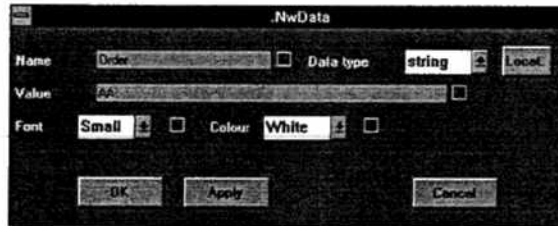
7. close the dialog window with the *OK*-button

EUROSIM'95

Accessing the Value of a Customised Attribute (2)

8. insert an instance of the building block *NwData*
9. open the dialog window with a double click and fill it out

familiar



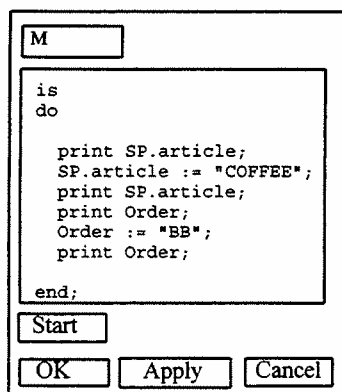
10. close the window with the *OK*-button

EUROSIM'95

Accessing the Value of a Customised Attribute (3)

11. enter the following method code and click on the *Apply*-button

familiar



12. click on the *Start*-button and watch the standard output (the window from which Simple++ was started)
13. open the *SP*-dialog window and look for the value of the customised attribute
14. close the model

EUROSIM'95

Data Types

Customised attributes consist of a **name**, a **data type** and a **value**.

SIMPLE++ offers a range of different data types:

- *boolean*: truth value; can be "TRUE" or "FALSE"
- *integer*: whole number, e.g. 1, 127, -3566
- *real*: floating point number, e.g. 3.141, -382.777
- *string*: string of characters or numbers; e.g. "This is text", "123-ABC"
- *object*: address of a model element (absolute address), e.g. .F.W.AP
- ...

EUROSIM'95

Boolean Operators

The following operators can be used with the data type *boolean*:

operator	example
<ul style="list-style-type: none">• AND• OR• NOT• =• /=	<p>TRUE AND FALSE. Always FALSE if one of the operands is FALSE.</p> <p>TRUE OR FALSE. Always TRUE if one of the operands is TRUE.</p> <p>NOT TRUE is FALSE. Reverses the value of the boolean operator.</p> <p>FALSE = FALSE. TRUE if the operands are equal.</p> <p>FALSE /= TRUE. TRUE if the operands are not equal.</p>
<ul style="list-style-type: none">• <i>bool_to_num</i>:• <i>bool_to_str</i>:	<p>conversion operator, e.g. <i>bool_to_num</i> (FALSE) returns 0.</p> <p>conversion operator, e.g. <i>bool_to_str</i> (FALSE) returns "FALSE".</p>

EUROSIM'95

Integer Operators

+	addition		
-	subtraction		
*	multiplication		
//	integer division		
\%	modulo operator		
/	division		
=	equal		
/=	not equal		
>	greater than		
<	less than		
>=	greater than or equal		
<=	less than or equal		

integer

integer

real

boolean

num_to_bool, num_to_string: type conversion operator

+ Perform different experiments with the conversion operators.

EUROSIM'95

REAL Operators

+	addition
-	subtraction
*	multiplication
/	division
=	equal
/=	not equal
>	greater than
<	less than
>=	greater than or equal
<=	less than or equal

num_to_bool, num_to_string: type conversion operators

+ Perform different experiments with the REAL operators.

EUROSIM'95

STRING Operators

+	concatenation	—	string
=	equal	—	boolean
/=	not equal	—	
toLower		—	string
toUpper		—	
copy		—	
incl		—	
omit		—	integer
strlen		—	
pos		—	object
sprint		—	

num_to_bool, num_to_string: type conversion operators



Perform different experiments with the STRING operators.

EUROSIM'95

The Basic Object *TableFile* 2D list (1)

WorkingPlan:

	1	2	3
1	a1	a2	...
2	b1	b2	
3	c1	c2	drain
4	d1	d2	drain

columns → x

rows ↓ y

Specific fields are accessed via an index, e.g.:

read entry: *WorkingPlan*[2, 3]; → c2

column | row

write entry: *WorkingPlan*[3, 4] := "drain";

assignment

EUROSIM'95

The Basic Object *TableFile* 2D list (2)

In addition to the system defined index (integer), the user can define indices for both the columns and the rows, e.g.:

WorkingPlan:

		1	2	
		station1	station2	column index
1	box	source	drill	
2	shaft	source	lathe	
row index				

example: `print WorkingPlan["station2", "shaft"] -> lathe`

EUROSIM'95

The Basic Object *CardFile*, *StackFile*, *QueueFile*

- *QueueFile*



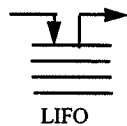
`QF.push("lathe");`

`QF.pop;`

`QF.top;`



- *StackFile*



`SF.push("lathe");`

`SF.pop;`

`SF.top;`



- *CardFile*



`CF.insert(2, "lathe");`

`CF.cutRow(4);`

`CF.read(4);`



EUROSIM'95

Searching a 1D-list

- each list has an internal cursor which points to the current field of the list.
- the list can be searched automatically. The result of the search can be used in boolean expression to determine whether a certain value was found or not.
- the search starts at the current cursor position and continues until the value is found or the end of the list is reached.
- if the correct value is found, the internal cursor points to the field containing this value and the boolean result *TRUE* is returned. If a search is unsuccessful, the cursor position is unchanged.
- if you want to search the complete list, the cursor must be set to the first field of the list prior to searching.
- **commands (e.g.):**

```
CF.Cursor := 1;
print CF.find( "lathe" );
print CF.Cursor;
```

TRUE
3

CF
drill
mill
lathe

EUROSIM'95

Searching a 2D-list

- **example:**

```
TF.CursorX := 1;
TF.CursorY := 1;
print TF.find( "lathe_old" );
print TF.CursorY;
print TF.CursorX;
```

TRUE
3
2

		columns	
			X
rows	↓	drill	mill
		grind	check
		sew	lathe_old
		lathe_new	

EUROSIM'95

Results / Statistics

- statistics of immovable units
- statistics of movable units
- activate / inactivate statistics
- reset statistics
 - eventcontroller
 - methods
- accessing separate statistic values
- write statistic values to the hard disc

EUROSIM'95

Special Items

- *Reset* method
- *Init* method
- *EndSim* method
- saving objects
- printing a model and model elements
- the basic object *Display*
- the basic object *Plotter*
- the SIMPLE++ debugger
- ASCII-interface of SIMPLE++

EUROSIM'95

Standard Interfaces of SIMPLE++

- ASCII-interface
- File-interface
open, close, write, writeln, readln
- interface to the operating system
example: system("ls"); -> UNIX
 system("dir"); -> WindowsNT
- Icon-interface (PPM-format)
- Exchange of data and icons via clipboard
(WindowsNT only!)

EUROSIM'95

Basic Objects Libraries of SIMPLE++

included in Development Licence:

- | | |
|---------------------|------------------------------|
| • SIMPLE++_control | standard strategies |
| • SIMPLE++_personal | staff |
| • SIMPLE++_conveyor | steady transportation system |
| • SIMPLE++_AGV | automatic guided vehicles |

optional templates:

- | | |
|--------------------|------------------------------|
| • SIMPLE++_EOM | electrical overhead monorail |
| • SIMPLE++_HBW | high bay warehouse |
| • SIMPLE++_process | chemical industry |
| • SIMPLE++_shop | shop floor control |

optional products:

- | | |
|------------------|--|
| • SIMPLE++_C | C programming interface |
| • SIMPLE++_SQL | database interface |
| • SIMPLE++_Gantt | Gantt chart visualization tool |
| • SIMPLE++_RPC | remote process communication |
| • SIMPLE++_GA | genetic algorithm |
| • SIMPLE++_DDE | dynamic data exchange
(WindowsNT only!) |

EUROSIM'95

Procedure to carry out a Simulation Study (1)

1. Problem definition and aim of study
2. Analysis of the system
3. Collecting the necessary data
4. Building the model
5. Implementation and testing
6. Validation of the model
7. Simulation runs: planning and carrying out
8. Evaluation of the results
9. Change and optimisation of the system

EUROSIM'95

Procedure to carry out a Simulation Study (2)

1. Problem definition and aim of study

Exercise: Demand definition by formulating
a question sheet

Example: What is the expected throughput of
the facility?

Are the buffers right dimensioned?

Which dispatch is meaningful?

What is the optimal number of
employed workers?

EUROSIM'95

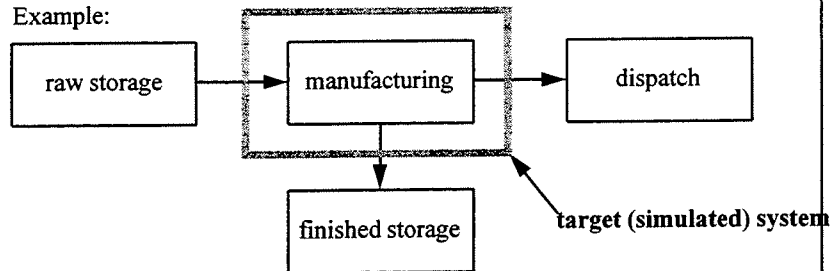
Procedure to carry out a Simulation Study (3)

2. Analysis of the system (1)

Exercise: Describe the system to be simulated

Outline of the system vs. environment

Example:



EUROSIM'95

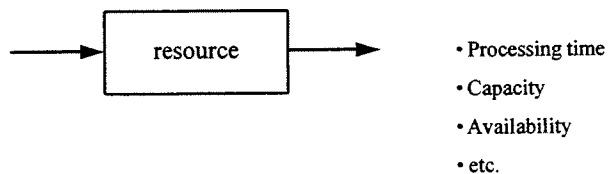
Procedure to carry out a Simulation Study (4)

2. Analysis of the system (2)

Exercise: Analysing the elements of the system

Determine the attributes of the elements

Example:



EUROSIM'95

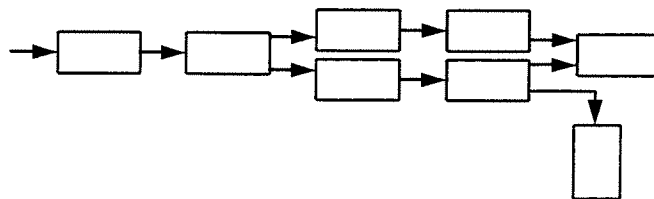
Procedure to carry out a Simulation Study (5)

2. Analysis of the system (3)

Exercise: Determine the structure of the system

Mutual interface specification of the system elements

Example:



EUROSIM'95

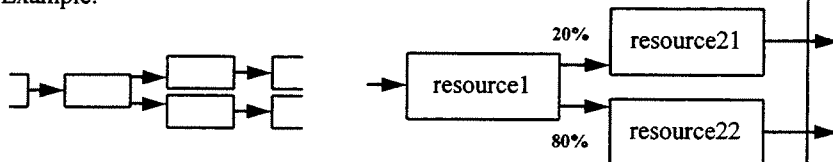
Procedure to carry out a Simulation Study (6)

2. Analysis of the system (4)

Exercise: Analysing the dispatch and process rules

Determine this rules affecting the entities and the facility

Example:



EUROSIM'95

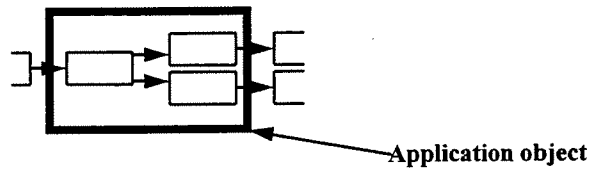
Procedure to carry out a Simulation Study (7)

2. Analysis of the system (5)

Exercise: Defining the required application objects

Is it possible to use pre-defined application objects?

Example:



EUROSIM'95

Tuning Tips for Simulation Runs

- Switch off the animation
- Switch off the interactive stopping
- Close the *EventController*
- Activate the fast index access (*TableFile*)
- Animation activated:
 - hide grid
 - hide connectors
 - hide object names
 - use pictures in original size
 - as few animation points as possible
 - as few animation events as possible (animation line)

EUROSIM'95

How to model with SIMPLE++

- Identify the system elements
- Encapsulate these elements to application objects
- Is it possible to use pre-defined application objects?
- Design the application objects on paper
 - define the interfaces
 - determine the structure of the building blocks
 - formulate the controls in spoken language
- Design the application objects
- Test the objects one by one
- Build up the complete model and test it

EUROSIM'95

Doing Simulation Studies (1)

1. System definition/goals

Compile a description of the real system and fix the borders of it.
 Keep asking questions until the functionality of single elements and the general behaviour of the overall system are clear.
 Formulate the goals of the simulation study.
 This is a very important step as the goals determine the necessary level of detail for modelling and thereby the effort which is necessary.

2. Design

Make a list of all elements of the system to be modelled. Evaluate which elements have a similar or identical functionality.
 From that a list of building blocks to be modelled can be created.
 Try to think of already existing objects which may be used.
 Specify and design the remaining objects on paper. Define and describe explicitly the interfaces for material and information flow.
 If necessary specify *Reset* and *Init* methods.

EUROSIM'95

Doing Simulation Studies (2)

3. Data

Make sure that the data necessary for the simulation of the real system are available or are captured if necessary. This often causes considerable effort and may be a longer process. A competent and skilled contact person who is responsible for organising the data is important.

4. Implementation

According to the design, the building blocks are created in Simple++. The model is then built using these building blocks. However, the model may also be structured into several hierarchical levels. Do not forget the documentation for the model.

EUROSIM'95

Doing Simulation Studies (3)

5. Verification

This step is to make sure that the single building blocks have the correct functionality and behave according to the specification. The single objects of the model usually are tested on their own, in conjunction with further building blocks and in the entire model. Verify that all parameters are set to correct values.

6. Validation

Check that the model corresponds to the planned or to the existing system. Develop estimates for important results and check if the simulation results are in a corresponding range. Discuss the model, the behaviour of the model and the results with an expert.

7. Plan for simulation experiments

Make a plan for the number of simulation runs, the variation of parameters, variation of random distributions, etc.

EUROSIM'95

Doing Simulation Studies (4)

8. Simulation experiments

Perform the necessary simulation runs and collect the results.

9. Evaluation of results

The results gained by the experiments are evaluated in this step and are documented.

10. Variation, Optimisation

Check if the goals as specified in step 1 are reached by the experiments. If necessary, adjust the model and perform further simulation runs until you reach an optimum result.

Simulation studies usually are an evolution process. A first design will often be revised a few times where new findings or results will lead to modifications in earlier steps of the entire process. Repeated loops will thereby lead from a fairly rough first draft to a final detailed model.

— EUROSIM'95 —

Why Simulation of Manufacturing Systems?

A manufacturing system consists of a variable number of interacting elements. The interactions consist of parallel and serial processes which exchange information and material. The complex nature of modern production systems means that with only a few elements the number of factors which need to be considered is so vast that rational and exact planning with conventional planning tools is impossible.

To obtain measures for optimisation of the facility during the planning phase, some techniques are necessary to model the interaction of the physical and logical elements of each system precisely, i.e. in its full complexity.

— EUROSIM'95 —

The Aim of Manufacturing Processes

The aim of all manufacturing processes should be the profitability of the entire system.

The profitability can be achieved by different aims:

- minimum lead time
- maximum production facilities utilization
- minimum work-in-progress and stock
- minimum energy consumption
- etc.

These criteria may be emphasised differently for individual processes in the system.

EUROSIM'95

Simulation and Profitability

One precondition for economic flexibility is planning that takes into consideration dynamic situations at present and in the future.



The possibility to perform experiments during the planning stages makes **simulation** an ideal tool for optimising manufacturing processes.

EUROSIM'95

The Necessity of Adapting a Manufacturing System to Changing Conditions

A manufacturing system has to be adapted to changing conditions which are permanent, short-term and a long-term.

New conditions arise through, e.g.:

- increasing variety of products
- shorter cycles of innovation
- personnel costs
- changes of
 - working time
 - energy costs
 - environmental regulations
 - rates of exchange
 - etc.

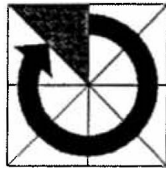
The effects of these changing conditions on the system can only be assessed completely by **simulation**.

EUROSIM'95

Significant Changes V 2.3 -> V 3.0

- **new interpreter -> increase of speed and performance**
- **better memory management (lists)**
- **event debugger**
- **profiler**
- **modelling with SimTALK commands**
- **battery-powered vehicles with charge-monitoring**
- **front/rear-control of material flow elements**

EUROSIM'95



Unseld + Partner

Business- and Marketing Consulting Simulation!

What we can offer our customers !

Unseld + Partner :

- focuses on simulation topics only and provides efficient industrial simulation experience, gained in many industrial projects
- consists of a group of highly educated simulation experts representing the highest concentration of such know-how in Austria
- offers simulation expertise in industrial projects, while being receptive for sophisticated new strategies in freight centres and intercompany logistics, tackling especially multimodal aspects.
- provides advice on EU-programs (concerning railways, IT and Telematics) to leading members of Austrian industrial organisations and government bodies
- is a completely independent Austrian company
- has co-operation contracts and contacts with many prominent international research and university institutes

Of course state-of-the-art simulation software technology for instance SIMPLE++ and VISIO are used and professional project management skills are provided.

➔ Without Simulation no Innovation ➔

Unseld + Partner Business- and Marketing Consulting Simulation!
A-1080 Vienna Lerchenfelderstraße 44/9 phone +43-1-4030371-0* fax +43-1-4030371-90