

MTA Számítástechnikai és Automatizálási Kutató Intézet Budapest





MAGYAR TUDOMÁNYOS AKADÉMIA  
SZÁMITÁSTECHNIKAI ÉS AUTOMATIZÁLÁSI KUTATÓ INTÉZETE

OPERÁCIÓS RENDSZEREK ELMÉLETE  
7. VISEGRÁDI TÉLI ISKOLA  
1982

7TH CONFERENCE ON OPERATING SYSTEMS,  
VISEGRÁD

Szerkesztőbizottság:

GERTLER JÁNOS (felelős szerkesztő)

DEMETROVICS JÁNOS (titkár)

ARATÓ MÁTYÁS, BACH IVÁN, GEHÉR ISTVÁN,  
GERGELY JÓZSEF, KERESZTÉLY SÁNDOR, KNUTH ELŐD,  
KRÁMLI ANDRÁS, PRÉKOPA ANDRÁS

Felelős kiadó:

DR VAMOS TIBOR

MTA Számítástechnikai és . automatizálási Kutató Intézete  
MTA Számítástudományi Bizottsága

Konferencia szervező bizottsága:

ARATO MÁTYÁS (elnök)

KNUTH ELŐD (titkár)

VARGA LÁSZLÓ

ISBN 963 311 148 X

ISSN 0324-2951

## CONTENT

	page
Tarnay, K.: Modelling and Measuring the Communication Protocols	5
Eberbach, E., Janicki, R.: A Note on Infinite Set of Equations and Fixedpoint Semantics of Vectors of Coroutines	13
Just, J.R.: Synchronization and Communication in Distributed Computer Systems by Means of Coroutines	27
Janicki, R.: On Concurrent Systems and Concurrency Relations	43
Jomier, G.: An Overview of Systems Modelling and Evaluation Tendencies	55
Ádány, L., Micsik, J.: Program Optimization on Ryad-22 Computer	73
Prószynski, P.: Properties of Concurrent Systems	85
Hernádi, Á.: Implementation of Abstract Types in PL/I	95
Czachórski, T.: A Software for Computer System Performance Analysis - One More Effort	111
Piwowarski, M.: Data Base Performance in a Paging Environment	123
Duda, A.: Performance Evaluation of Computing System Subject to Failures	135
Kerékfy, P., Ruda, M.: Automatic Programming System Development on User Level	147



## MODELLING AND MEASURING THE COMMUNICATION PROTOCOLS

dr.Katie Tarnay

Central Research Institute for Physics

Budapest, Hungary

### Abstract

A protocol model for data link layer of OSI Reference Model is introduced and the reaction of the network on protocol behaviour is analyzed. The basic model is extended, taking the extrinsic effects of other nodes and the intrinsic effects of other layers into consideration. Finally a test program and a protocol analyzer are discussed.

### 1. INTRODUCTION

A computer network realizes a cooperation between open systems [1]. A system is subdivided into layers. Entities exist at each layer. Semantic and syntactic rules and formats determine the communication behaviour of entities. The efficiency of the network operation depends on the protocol construction and on its elements. Our aim is to analyze these interactions step by step. The first step is a simplified analysis with some neglects. Two nodes are picked out from the network, peer layers are chosen of these nodes and a typical protocol of these peer layers is examined. The influence of other nodes and traffic, just as the effect of other layers are taken into account by estimated weighting factors. The second step is the determination of weighting factors based on topology, traffic and resource demand. These results characterize the extrinsic effects. Our third step is to analyze the influence of other layers, i.e. the intrinsic effects.

Three network parameters: the throughput, the delay and the utilization of resources build the measures of network operation for all three steps.

## 2. THE BASIC MODEL

### 2.1 Selection of model components

The basic model is used in the analysis according to the above mentioned first step. The connection is characterized by a dialogue between the selected layers of any node pairs. Our choice is the data link layer, because this is better revealed than the higher layers and its behaviour, functions and services are more common with the others than those of the lower layer. The dialogue is described by a protocol. Many protocols exist in the data link layer, in our model the HDLC [2] is applied according to the Recommendation of Reference Model.

### 2.2 The protocol model

The HDLC protocol contains a basic repertoire of commands and responses, moreover optional functions. Our analysis is restricted to the basic repertoire, its elements can be seen in Table 1.

	Commands	Responses
Information	I	I
Supervisory	RR RNR	RR RNR
Unnumbered	S--M DISC	---R UA DM

Table 1. Basic repertoire of the HDLC



The interpretation of the undetermined characters / S--M,---R / depends on the operation mode.

The dialogue fulfilling the rules and formats of HDLC is generated according to the formal grammar introduced by J. Harangozó [3].

The first grammar is

$$G = (V_N, V_T, P, S)$$

where  $V_N$  means the non-terminals  
 $V_T$  means the terminals  
 $P$  is the production rule  
 $S$  is the start symbol

The  $V_T$  terminal is a set of primary and secondary commands and responses

$$V_T = V_{T1} \cup V_{T2}$$

The primary messages are

$$V_{T1} = (I, S--M, RR, RNR)$$

The secondary messages are

$$V_{T2} = (I, ---R, RR, RNR)$$

Table 2. shows the basic production rules. The first grammar is the simplest one, the others contain time relations and error generation, too.

Non-terminals	Terminal symbols										
	snrm	disc	i	rr	rnr	ua	cmdr	i	rr	rnr	
S	A	-	-	-	-	-	-	-	-	-	
A	A	-	-	-	-	B	C	L	L	L	
B	A	G	D	E	F	B	-	-	-	-	
C	A	G	-	-	-	-	C	-	-	-	
D	-	-	D	E	F	L	C	H	J	K	
E	-	-	D	E	-	L	C	H	J	K	
F	-	-	D	-	F	L	C	M	J	K	
G	-	G	-	-	-	∅	C	L	L	L	
H	A	G	D	E	F	-	-	H	J	K	
J	A	G	D	E	F	-	-	H	J	-	
K	A	G	N	E	F	-	-	H	-	K	
L	A	G	-	-	-	L	L	L	L	L	
M	A	G	D	E	F	-	-	M	M	M	
N	-	-	N	N	N	L	C	H	J	K	

Table 2. Productions for grammar G

### 2.3 Dialogue generation

Estimated weighting factors belong to the commands and responses. The estimation is based on experimental data. The elements of the dialogue are generated by the grammar generator G and follow each other according to the possible conversation. The frequency of different frames depends on their weighting factors. The results yield the throughput.

The throughput is

$$T = \frac{N_I}{N_I + N_C} = \frac{\alpha W_I}{\alpha W_I + \beta W_C}$$

where

- $N_I$  is the number of information frames
- $N_C$  is the number of control frames
- $W_I$  is the weighting factor of information frames
- $W_C$  is the weighting factor of control frames
- $\alpha$  is the sequence error of information frames
- $\beta$  is the sequence error of control frames

Two other errors can be built in the basic model: link and node errors.

### 3. EXTENDED MODELS

#### 3.1 Extrinsic effects

The activation of nodes and the proportion of information and control frames depend on extrinsic effects. These are the following:

- topology
- deterministic parameters related to topology
- stochastic parameters related to topology
- traffic

The topology can be static /one-, two-, three-dimensional or hiper-cube/ or dynamic. The nodes are active switches in the case of dynamic topology and the links are reconfigurable.

The topology is described by a channel matrix  $M$  characterizing the corresponding channel and node pairs. The connectivity-, incidence- and adjacency-matrices can be determined from the channel matrix. The weighting factor of the control frames is a function of the channel matrix, requirement matrix and routing table belonging to the shortest path:

$$W = f(M, R, RT)$$

### 3.2 Intrinsic effects

The intrinsic effects belong to the essential nature of the communication protocols and form an integral part of the information. Every information frame contains the information and control frames of higher level protocols. Among the intrinsic effects the resource allocation is of outstanding significance. The function of the resource sharing protocol within the information frame is the allocation of the resources according to the proper demand. The resource table can be applied to determine the optimum of cost, time or hop number. Thus the utilization of the resources is characterized by the generated dialogue.

## 4. TESTING AND MEASURING THE PROTOCOLS

### 4.1 Testing a protocol model

The comparator solution is selected from the protocol testing methods. The essence of the method is the following [4]. Arbitrary message series are generated by means of a random generator, these series form the input of a reference program prepared on the ground of a verbal description on one hand, while that of the model to be tested on the other hand.

The comparing analysis of the reaction of the reference program and the model is performed by a comparator program at the output of which the input and output series of the model as well as the evaluating message obtained as the result of the comparison to the reference, appears.

The advantage of the comparator solution is that it makes the automation of the checking process possible. The complexity of this method is not so much involved in its structure as in the preparation of the reference program where all the restrictions and specifications referring to the syntactics and semantics of the protocol procedure in the verbal description should be taken into consideration.

#### 4.2 A data- and protocol-analyzer

The analyzer allows for a direct monitoring of the data flow through the remote data transmission line, on a display, it monitors and counts, automatically, the important events of the physical line and the logical data link. The user can follow the wide-spread protocols by simple instructions and the special protocols in a programmed mode.

The data- and protocol-analyzer developed in the Central Research Institute for Physics is an appropriate tool to check the theoretical analysis and tests mentioned above.

#### 5. CONCLUSIONS

Our simplified model analyses the traffic between the peer layers of two nodes and serves as a reference for the extended models. The model comprising also the topologic characters gives a good approximation of the weighting factors of the control frames. The model completed with the upper layers supports a better utilization of the resources. The complex model allows the extrinsic and intrinsic effects and reveals the interactions between the frame classes and the protocol overhead as well as between the formers and the traffic.

#### REFERENCES

- [1] Data processing - Open systems interconnection - Basic reference model  
ISO/TC 97/SC16 DP 7498 March 31, 1981
- [2] High Level Data Link Control Procedures  
ISO 1254, 1255, 1256, 3309, 4335
- [3] J.Harangozó: Formal language description of a communication protocol  
Report KFKI-1977-92, Budapest, Hungary
- [4] M.Bohus private communication, Laboratory of Cybernetics József Attila University, Szeged, Hungary



A NOTE ON INFINITE SET OF EQUATIONS AND FIXEDPOINT SEMANTICS OF VECTORS OF COROUTINES

Eugeniusz Eberbach <sup>\*</sup>, Ryszard Janicki <sup>+</sup>

1. Introduction.

Proving properties of programs by means of fixpoints is such old as the theory of programming (see Blikle [3,4], Bekić [1], and many others). The structure of a program is frequently described by a finite set of equations, which can be solved either directly from Kleene Theorem on approximation or by means of the variable elimination method (see for example [1,3,4,11]).

Unfortunately, not every program can be described by a finite set of equations. For example, programs with recursive coroutines require infinite sets of equations (see Janicki [6]).

This paper deals with a method of finding the least fixpoint for an infinite set of equations.

The method is next applied to the description of the fixedpoint semantics of vectors of PD-coroutines. Vectors of PD-coroutines, introduced by Janicki [5,6], can be treated as a mathematical model for a wide class of coroutine programs. The concept of coroutine vector is also useful to describe some aspects of distributed computing systems (see Just [7,8]).

2. Basic notions and results.

Let  $(U, \leq)$  be a complete lattice fixed for the rest of this section, where  $\perp = \bigwedge U$  denotes the bottom element of the lattice.

A set  $P \subseteq U$  is said to be directed if any finite subset of  $P$  has an upper bound in  $P$ .

A function  $f: U \rightarrow U$  is said to be c-continuous if for any directed set  $P \subseteq U$ :  $f(\bigcup P) = \bigcup \{f(p) / p \in P\}$ .

Theorem 1. (Kleene [9])

If  $f: U \rightarrow U$  is c-continuous, then the least fixpoint of  $f$  exists and is equal to:

where:  $f^i(x) = f(\dots f(x) \dots)$   $i$ -times.  $\blacksquare$

$$\bigcup_{i=1}^{\infty} f^i(\perp),$$

For every function  $f:U \rightarrow U$ , its least fixpoint - if it exists - will be denoted by the symbol  $\|f\|$ .

Thus, from Theorem 1 we have that for every c-continuous function  $f$ :  $\|f\| = \bigcup_{i=1}^{\infty} f^i(\perp)$ .

Let  $U^U$  denote the set of all functions from  $U$  to  $U$ . Let  $\leq$  be the relation in  $U^U$  defined as follows:

$$(\forall F_1, F_2 \in U^U) \quad F_1 \leq F_2 \iff (\forall x \in U) \quad F_1(x) \leq F_2(x).$$

Note that  $(U^U, \leq)$  is also a complete lattice, where  $\perp = \bigcap U^U$  is the function defined by:  $(\forall x \in U) \quad \perp(x) = \perp$ .

Let  $x_1, x_2, \dots$  be an infinite sequence of elements of  $U$ . Now, we recall the following well known notions (see for example [10]):

$$\begin{aligned} \limsup_{i \rightarrow \infty} x_i &= \bigcup_{i=1}^{\infty} \bigcap_{k=0}^{\infty} x_{i+k}, \\ \liminf_{i \rightarrow \infty} x_i &= \bigcap_{i=1}^{\infty} \bigcup_{k=0}^{\infty} x_{i+k}. \end{aligned}$$

Since  $(U, \leq)$  is a complete lattice then elements  $\limsup_{i \rightarrow \infty} x_i$  and  $\liminf_{i \rightarrow \infty} x_i$  exist for any sequence  $x_1, x_2, \dots$ .

Of course  $\liminf_{i \rightarrow \infty} x_i \leq \limsup_{i \rightarrow \infty} x_i$ .

If  $\liminf_{i \rightarrow \infty} x_i = \limsup_{i \rightarrow \infty} x_i$ , then we shall write  $\lim_{i \rightarrow \infty} x_i$ , and the element  $x = \lim_{i \rightarrow \infty} x_i$  will be called the convergence of the sequence  $x_1, x_2, \dots$ .

Let  $N$  denote the set of numbers  $\{1, 2, 3, \dots\}$ .

Lemma 2.

Let  $x_1, x_2, \dots$  be a sequence of elements from  $U$ , such that:

$$(\forall i, j \in N) (\exists k \in N) \quad i \leq k \ \& \ j \leq k \ \& \ x_i \vee x_j \leq x_k.$$

Then:

$$\lim_{i \rightarrow \infty} x_i = \bigcup_{i=1}^{\infty} x_i. \quad \blacksquare$$

Corollary 3.

Let  $x_1, x_2, \dots$  be a sequence of elements from  $U$ , such that:

$$x_1 \leq x_2 \leq \dots \leq x_k \leq x_{k+1} \leq \dots$$

Then:

$$\lim_{i \rightarrow \infty} x_i = \bigcup_{i=1}^{\infty} x_i. \quad \blacksquare$$

Let  $V$  be a set, and let  $(U, \leq)$  be a complete lattice with the property:  $U \subseteq 2^V$ ,  $\leq = \subseteq$ .

In this case it can be proved that (compare [10]), the sequence of sets  $x_1, x_2, \dots \in 2^V$  is convergent to  $\lim_{n \rightarrow \infty} x_n$  if and only if the sequence of characteristic functions of those sets is convergent, in the usual sense of the mathematical



analysis, to the characteristic function of the set  $\lim_{n \rightarrow \infty} x_n$ . This fact allows frequently us to count  $\lim_{n \rightarrow \infty}$  for sets by means of the same methods as for real numbers.

3. Directed approximations of functions.

Let  $F \in U^U$  be a c-continuous function, and let  $\{F_1, F_2, \dots\} \subseteq U^U$  be a sequence of c-continuous functions. The sequence  $\{F_1, F_2, \dots\}$  is called a directed approximation of  $F$  iff:

- a)  $(\forall i, j \in \mathbb{N}) (\exists k \in \mathbb{N}) \quad i \leq k \ \& \ j \leq k \ \& \ F_i \cup F_j \leq F_k,$
- b)  $F = \bigcup_{i=1}^{\infty} F_i \quad .$

Theorem 4.

For every c-continuous function  $F$ , and every its directed approximation  $\{F_1, F_2, \dots\}$  :

$$\|F\| = \bigcup_{i=1}^{\infty} \|F_i\| \quad . \blacksquare$$

Corollary 5.

For every c-continuous function  $F$ , and every its directed approximation  $\{F_1, F_2, \dots\}$  :

$$\|F\| = \lim_{i \rightarrow \infty} \|F_i\| \quad . \blacksquare$$

Directed approximations have rather theoretical sense.

For our purposes, special kinds of directed approximations - called nondecreasing approximations, are more useful.

4. Nondecreasing approximations of functions.

Let  $F \in U^U$  be a c-continuous function, and let  $\{F_1, F_2, \dots\} \subseteq U^U$  be a sequence of c-continuous functions. The sequence  $\{F_1, F_2, \dots\}$  is called a nondecreasing approximation of  $F$  iff:

- a)  $F_1 \leq F_2 \leq \dots \leq F_k \leq \dots ,$
- b)  $F = \bigcup_{i=1}^{\infty} F_i \quad .$

Corollary 6.

For every c-continuous function  $F$ , every nondecreasing approximation of  $F$  is a directed approximation of  $F$ .  $\blacksquare$

Corollary 7.

For every c-continuous function  $F$ , and every its nondecreasing approximation  $\{F_1, F_2, \dots\}$  :

$$\|F\| = \lim_{i \rightarrow \infty} \|F_i\| \quad . \blacksquare$$

The above corollary describes a method of finding of the

least fixpoint of a c-continuous function F. Namely, one should find such nondecreasing approximation of the function F, that for every i the least fixpoint of F can be described, in a simple way, as a function of i.

Other words, we must find such nondecreasing approximation  $\{F_1, F_2, \dots\}$ , and such function  $x: N \rightarrow U^U$  that:

1.  $x(i) = \|F_i\|$  ,
2.  $x(i)$  is defined as an evident parameter of i.

Then, in order to find  $\|F\|$  it is enough to count  $\lim_{i \rightarrow \infty} x(i)$ . An application of this method will be shown in further sections.

5. Infinite sets of equations.

Let  $(U, \leq)$  be a complete lattice.

Define  $U^\infty = U \times U \times \dots$ .

Let  $\leq$  be a relation on  $U^\infty$  defined as follows:

$$(\forall \underline{a} = (a_1, a_2, \dots), \underline{b} = (b_1, b_2, \dots) \in U^\infty) \quad \underline{a} \leq \underline{b} \iff a_i \leq b_i \text{ for } i=1, 2, \dots$$

Note that  $(U^\infty, \leq)$  is also a complete lattice, and, since all results from previous sections hold for any complete lattice, then they hold also for the lattice  $(U^\infty, \leq)$ .

Consider the following infinite set of equations:

$$\begin{aligned} x_1 &= f_1(x_1, x_2, \dots) \\ x_2 &= f_2(x_1, x_2, \dots) \\ &\dots \dots \dots \\ x_k &= f_k(x_1, x_2, \dots) \\ &\dots \dots \dots \end{aligned}$$

where:  $f_i: U^\infty \rightarrow U$  for  $i=1, 2, \dots$  are c-continuous functions. Of course, this set of equations can be written as one infinite "vectorial" equation:

$$\underline{x} = F(\underline{x}) ,$$

where  $f_i: U^\infty \rightarrow U$  is a c-continuous function,  $\underline{x} = (x_1, x_2, \dots)$ ,  $F(\underline{x}) = (f_1(\underline{x}), f_2(\underline{x}), \dots)$ .

Now, we fix the equation  $\underline{x} = F(\underline{x})$  for the rest of this section.

Let  $i_1, i_2, \dots$  be an infinite sequence of natural numbers with the following property:

$$i_1 \leq i_2 \leq \dots \leq i_k \leq \dots$$

For every  $k=1,2,\dots$ , let  $F_k:U^\infty \rightarrow U^\infty$  be the following function:  $(\forall \underline{x} \in U^\infty) \quad F_k(\underline{x}) = (f_{1k}(\underline{x}), f_{2k}(\underline{x}), \dots)$ ,

where:

$$(\forall \underline{x} = (x_1, x_2, \dots, x_{i_k}, x_{i_k+1}, \dots) \in U^\infty) (\forall i \in \mathbb{N})$$

$$f_{ik}(\underline{x}) = \begin{cases} f_i(x_1, \dots, x_{i_k}, \perp, \perp, \dots) & i \leq i_k \\ \perp & i > i_k \end{cases}$$

Thus, the equation  $\underline{x} = F_k(\underline{x})$  written as the set of equations, is the following:

$$\begin{aligned} x_1 &= f_1(x_1, \dots, x_{i_k}, \perp, \perp, \dots) \\ x_2 &= f_2(x_1, \dots, x_{i_k}, \perp, \perp, \dots) \\ &\dots \dots \dots \\ x_{i_k} &= f_{i_k}(x_1, \dots, x_{i_k}, \perp, \perp, \dots) \\ x_{i_k+1} &= \perp \\ x_{i_k+2} &= \perp \\ &\dots \dots \dots \end{aligned}$$

Lemma 8.

For every  $c$ -continuous function  $F:U^\infty \rightarrow U^\infty$ , for any increasing sequence of natural numbers  $i_1, i_2, \dots$ , the set of functions  $\{F_1, F_2, \dots\}$  defined by the above procedure is a nondecreasing approximation of  $F$ , and  $\|F\| = \lim_{k \rightarrow \infty} \|F_k\|$ . ■

Consider the equation  $\underline{x} = F_k(\underline{x})$ .

Let  $\tilde{F}_k:U^{i_k} \rightarrow U^{i_k}$  be the function of the form:

$$(\forall \underline{x} \in U^{i_k}) \quad \tilde{F}_k(\underline{x}) = (\tilde{f}_{1k}(\underline{x}), \dots, \tilde{f}_{i_k k}(\underline{x})),$$

where:  $(\forall i=1, \dots, i_k) \quad \tilde{f}_{ik}(x_1, \dots, x_{i_k}) = f_i(x_1, \dots, x_{i_k}, \perp, \dots)$

Note that in many cases the equation  $\underline{x} = \tilde{F}_k(\underline{x})$  can be solved by the method of variable elimination (see [1,2,3,4, 11]).

Lemma 9.

For every  $k=1,2,\dots$ ,  $\|F_k\| = (a_1, a_2, \dots, a_{i_k}, \perp, \perp, \dots)$ ,

where  $(a_1, a_2, \dots, a_{i_k}) = \|\tilde{F}_k\|$ . ■

For every  $i=1,2,\dots$ , let  $\|f_i\|$  denote the  $i$ th coordinate of  $\|F\|$ ,  $\|f_{ik}\|$  denote the  $i$ th coordinate of  $\|F_k\|$ , and let  $\|\tilde{f}_{ik}\|$  denote the  $i$ th coordinate of  $\|\tilde{F}_k\|$ .

Lemma 10.

For every  $i=1,2,\dots$ :

$$\|f_i\| = \lim_{k \rightarrow \infty} \|f_{ik}\| = \lim_{k \rightarrow \infty} \|\tilde{f}_{ik}\|. \quad \blacksquare$$

From the above consideration it follows a method of a solu-

tion (in the sense of the least fixpoint) of the infinite set of equations. Namely, one should find such an approximation by finite sets of equations  $\{\underline{x}=\tilde{F}_1(\underline{x}), \underline{x}=\tilde{F}_2(\underline{x}), \dots \text{ and so on}\}$ , that every  $\|\tilde{f}_{ik}\|$  can easily be presented as a certain function of the parameter  $k$ . Then  $\|f_i\|$  is simply the convergence of  $\|\tilde{f}_{ik}\|$  for  $k \rightarrow \infty$ . This convergence can frequently be counted by means of methods similar to those, which are used in the classical mathematical analysis.

Note also that in many cases, the knowledge about the whole vector  $\|F\|$  is not needed, and we are only interested in a finite subset of coordinates of  $\|F\|$ .

Usually we are only interested in the form of the first coordination, i.e.  $\|f_1\|$ . This problem will be also considered in the last section of the paper.

#### 6. Vectors of coroutines.

Vectors of coroutines introduced by Janicki [5,6] can be regarded as mathematical models of programs with coroutines. A vector of coroutines is a set of components, each component is an algebraic object like the Mazurkiewicz algorithm [3,4] with a mechanism which makes an interaction possible.

Vectors of coroutines are adequate models for programs with the fixed in advance the number of components. In the case of Simula language, it is equivalent to fix in advance the number of copies of classes representing coroutines.

By a net (Blikle net) we mean an algebra:

$\text{Net} = (U, \leq, \circ, \perp, e)$ , where  $(U, \leq)$  is a complete lattice with  $\perp$  as the least element,  $(U, \circ, e, \perp)$  is a monoid with zero  $\perp$ , unit  $e$ , and with composition  $\circ$ . The operation of composition  $\circ$  is  $c$ -continuous and additive (in the sense that  $a \circ (b \vee c) = a \circ b \vee a \circ c$ ).

Basic examples of nets are the net of languages and the net of binary relations.

By a net of binary relations over a set X we mean the algebra  $(2^{X \times X}, \subseteq, \circ, \emptyset, \text{id})$ , where  $\circ$  is composition of relations, and  $\text{id}$  is the identity relation.

By a net of languages over an alphabet  $\Sigma$  we mean the algebra  $(2^{\Sigma^*}, \subseteq, \circ, \emptyset, \{\epsilon\})$ , where  $\circ$  is concatenation of

languages, and  $\varepsilon$  is an empty word.

Let  $\text{Net} = (U, \leq, \circ, \perp, e)$  be an arbitrary net.

By a vector of PD-coroutines over  $\text{Net}$  we mean any system

$$C = (i_0, A_1, \dots, A_n),$$

where:

$i_0$  is an integer ( $1 \leq i_0 \leq n$ ), and  $i_0$  is called the number of initial coroutine,

$A_i$  for  $i=1, 2, \dots, n$  are triples (called components):

$$A_i = (V_i, \sigma_i, P_i), \text{ where}$$

$V_i$  is an alphabet (of control symbols of  $A_i$ ),

$\sigma_i \in V_i$ , and  $\sigma_i$  is called the initial symbol of  $A_i$ ,

$P_i$  is a finite subset of the set:

$$(\{i\} \times \{1, \dots, n\}) \times (V_i \times V_i^*) \times U.$$

Instead of  $((i, j), (a, v), r) \in P_i$  we shall write  $(i \rightarrow j, a \rightarrow v, r)$ .

The set  $P_i$  is called the set of instructions of  $A_i$ .

Define  $VS = \{1, \dots, n\} \times V_1^* \times \dots \times V_n^*$ . This set is called the set of control states of  $C$ .

Each triple  $(i \rightarrow j, a \rightarrow v, r)$  defines the relation

$T(i \rightarrow j, a \rightarrow v, r) \in VS \times VS$  in the following way:

$$(\forall x = (i, u_1, \dots, u_n), y = (j, w_1, \dots, w_n) \in VS) (x, y) \in T(i \rightarrow j, a \rightarrow v, r) \\ \iff [(\exists w \in V_i^*) u_i = aw \ \& \ w_i = vw \ \text{and} \ u_k = w_k \ \text{for} \ k=1, \dots, i-1, \\ i+1, \dots, n].$$

Other words, if  $(x, y) \in T(i \rightarrow j, a \rightarrow v, r)$  and

$$x = (i, u_1, \dots, u_{i-1}, aw, u_{i+1}, \dots, u_n) \text{ then}$$

$$y = (j, u_1, \dots, u_{i-1}, vw, u_{i+1}, \dots, u_n).$$

Let  $VT \subseteq VS$  be a set such that  $(i, u_1, \dots, u_n) \in VT \iff u_i = \varepsilon$ .

The set  $VT$  is called the set of all terminal control states.

$$\text{Let } P_C = \bigcup_{i=1}^n P_i.$$

Consider a finite sequence of elements of  $P_C$ :

$(i_1 \rightarrow j_1, a_1 \rightarrow v_1, r_1), \dots, (i_m \rightarrow j_m, a_m \rightarrow v_m, r_m)$  such that exists a sequence of control states (i.e. elements of  $VS$ ):  $y_1, \dots, y_{m+1}$  with the following properties:

$$(1) (\forall k \leq m) (y_k, y_{k+1}) \in T(i_k \rightarrow j_k, a_k \rightarrow v_k, r_k),$$

$$(2) y_{m+1} \in VT.$$

Each such a sequence of instructions (elements of  $P_C$ ) can be considered as one particular run of the vector of coroutines  $C$ .

The corresponding sequence of actions (i.e. the sequence

$r_1, \dots, r_m$ ) is called  $y$ -trace.

Elements of the sequence  $r_1, \dots, r_m$  are actions that have been performed one after the other during the run.

The set of all  $y$ -traces will be denoted by  $Tr(y)$ .

Let  $Tr(C) = \cup \{Tr(y) / y \in VS\}$ , and let  $M:Tr(C) \rightarrow U$  be a mapping given by:

$$(\forall (r_1, \dots, r_m) \in Tr(C)) \quad M((r_1, \dots, r_m)) = r_1 \circ r_2 \circ \dots \circ r_m.$$

Each of  $y$ -traces produces its outcome, therefore the finitistic outcome, of the whole set of  $y$ -traces will be the

$$\text{join: } Tail_C(y) = \cup \{M(t) / t \in Tr(y)\}.$$

Note that  $Tail_C((i_0, \delta_1, \dots, \delta_n))$  defines the finitistic outcome of the vector  $C$ .

Define  $Res_C = Tail_C((i_0, \delta_1, \dots, \delta_n))$ . Of course  $Res_C \in U$ .

The problem is, how to find  $Res_C$  for a given vector  $C$ ?

Note that if  $Net$  is a net of binary relations then  $Res_C$  is a relation, and if  $Net$  is a net of languages then  $Res_C$  is a language.

A vector  $C = (i_0, A_1, \dots, A_n)$  is said to be a vector of FC-coroutines iff:

$$(\forall i) (i \rightarrow j, a \rightarrow v, r) \in P_i \implies v \in V_i \cup \{E\}.$$

Vectors of PD-coroutines describe properties of coroutine programs with monadic recursion, while vectors of FC-coroutines describe properties of iterative coroutine programs. The fixedpoint semantics of vectors of FC-coroutines was precisely described in [5], the fixedpoint semantics of vectors of PD-coroutines will be described in the next section.

It turns out that properties of vectors of FC-coroutines can be described by a finite sets of equations, while vectors of PD-coroutines require infinite sets of equations.

### 7. Fixedpoint semantics of vectors of PD-coroutines.

Let  $C = (i_0, A_1, \dots, A_n)$  be a given vector of PD-coroutines.

Note that  $card(VS) = \infty$ ; then elements of the set of control states of the vector of coroutines can be numbered by natural numbers.

Let  $\gamma: N \rightarrow VS$  be a one-to-one mapping such that  $\gamma(N) = VS$ ,

and  $\varphi(1) = (i_0, \bar{b}_1, \dots, \bar{b}_n)$ , where  $N = \{1, 2, 3, \dots\}$ .

The function  $\varphi$  will be called a numeration of VS.

Let  $R: VS \times VS \rightarrow U$ ,  $Q: VS \rightarrow U$  be the following functions:

$$(\forall x, y \in VS) \quad R(x, y) = \begin{cases} r & (\exists (i \rightarrow j, a \rightarrow v, r) \in P_C) \quad (x, y) \in \\ & \in T(i \rightarrow j, a \rightarrow v, r) \\ \perp & \text{otherwise.} \end{cases}$$

$$(\forall x \in VS) \quad Q(x) = \begin{cases} e & x \in VT, \\ \perp & x \notin VT. \end{cases}$$

Consider the following infinite set of equations:

$$x_1 = \bigcup_{i=1}^{\infty} R(\varphi(1), \varphi(i)) \circ x_i \vee Q(\varphi(1))$$

$$x_2 = \bigcup_{i=1}^{\infty} R(\varphi(1), \varphi(i)) \circ x_i \vee Q(\varphi(2))$$

.....

Every set of equations of the above form will be called a canonical set of equations for the vector C.

Note that all canonical sets of equations for a given vector C, are the same with exactitude to the function  $\varphi$ .

Every canonical set of equations will be written of the form of vectorial equation:

$$\underline{x} = F_{C, \varphi}(\underline{x}).$$

Lemma 11.

Let C be a vector of PD-coroutines over the net  $(U, \leq, \circ, \perp, e)$ , and let  $\varphi$  be a numeration of VS.

Then the function  $F_{C, \varphi}: U^\infty \rightarrow U^\infty$  defined as the right side of the canonical set of equations, is c-continuous. ■

From Lemma 11 it follows that  $\|F_{C, \varphi}\|$  always exists.

Let  $\|f_i\|$  denote the ith coordinate of  $\|F_{C, \varphi}\|$ .

Theorem 12.

Let C be a vector of PD-coroutines, and let  $\varphi$  be a numeration of VS.

Then:  $(\forall i \in N) \quad \text{Tail}(\varphi(i)) = \|f_i\|$  . ■

Corollary 13.

For every vector of PD-coroutines C, and every numeration  $\varphi$ :

$$\text{Res}_C = \|f_1\|$$
 . ■

Now we introduce the notion of natural numeration of VS, and we shall show how to find some  $\|f_i\|$  for a given vector of PD-coroutines.

Let  $C = (i_0, A_1, \dots, A_n)$  be a fixed vector of PD-coroutines.

Now, we must introduce a new kind of relations.

For every  $k=1,2,\dots$ , let  $T^{(k)} \subseteq VS \times VS$  be the following

relation:  $(i, u_1, \dots, u_n) T^{(k)} (j, w_1, \dots, w_n) \iff$  there is a sequence of elements of  $P_C: SQ = (i_1 \rightarrow j_1, a_1 \rightarrow v_1, r_1), \dots,$

$(i_m \rightarrow j_m, a_m \rightarrow v_m, r_m)$  such that:

1.  $(i, u_1, \dots, u_n) T_1 T_2 \dots T_m (j, w_1, \dots, w_n),$

where  $T_\ell = T(i_\ell \rightarrow j_\ell, a_\ell \rightarrow v_\ell, r_\ell)$  for  $\ell = 1, 2, \dots, m,$

2. for every  $p \in P_C$ , at most  $k$  elements of  $SQ$  is equal to  $p$ .

For every  $k=1,2,\dots$ , let  $VS_k$  denote the following set of control states:

$$VS_k = \{(i, u_1, \dots, u_n) \in VS / (i_0, \delta_1, \dots, \delta_n) T^{(k)} (i, u_1, \dots, u_n)\}$$

Note that  $VS_1 \subsetneq VS_2 \subsetneq \dots \subsetneq VS$ .

Lemma 14.

$$VS = \bigcup_{k=1}^{\infty} VS_k \quad \blacksquare$$

Define  $i_k \stackrel{k=1}{=} \text{card}(VS_k)$ . Of course  $i_1 < i_2 < \dots$ .

Let  $\varphi: \mathbb{N} \rightarrow VS$  be such a numeration that:

$$(\forall k=1,2,\dots) \quad \varphi^{-1}(VS_k) = \{1, 2, \dots, i_k\}.$$

Every numeration with the above property will be called natural.

The construction of a natural numeration is the following.

For every  $k=1,2,\dots$ , let  $\varphi_k: \{i_{k-1} + 1, \dots, i_k\} \rightarrow VS_k - VS_{k-1}$  be a one-to-one function.

Since  $\text{card}(VS_k - VS_{k-1}) = \text{card}(VS_k) - \text{card}(VS_{k-1}) = i_k - i_{k-1} = \text{card}(\{i_{k-1} + 1, \dots, i_k\})$ , then such a function always exists.

Let  $\Psi: \mathbb{N} \rightarrow \mathbb{N}$  be the following function:

$$(\forall i \in \mathbb{N}) \quad \Psi(i) = k, \text{ where } k \text{ is such number that } i_{k-1} < i \leq i_k.$$

Define  $\varphi: \mathbb{N} \rightarrow VS$  in the following way:

$$(\forall i \in \mathbb{N}) \quad \varphi(i) = \varphi_{\Psi(i)}(i).$$

Note that  $\varphi$  is a natural numeration of  $VS$ .

Let  $\varphi$  be a fixed natural numeration of  $VS$ .

In order to preserve the notation from section 5, we put

$F_{C,\varphi} = F$ . So the canonical set of equations for the vector  $C$  and a natural numeration  $\varphi$ , is of the form:

$$\underline{x} = F(\underline{x}).$$

Consider the sequence of finite sets of equations:

$\underline{x} = \tilde{F}_1(\underline{x}), \underline{x} = \tilde{F}_2(\underline{x}), \dots$  defined on the basis of  $\underline{x} = F(\underline{x})$  by the procedure from section 5, where the sequence  $i_1, i_2, \dots$



is defined by the equality  $i_k = \text{card}(VS_k)$ .

Note that every  $\tilde{F}_k(\underline{x})$  is defined precisely, so if we can present  $\|F_k\|$  as an evident function of the parameter  $k$ , then we can solve the equation  $\underline{x} = F(\underline{x})$ , because:

$$(\forall i=1,2,\dots) \|f_i\| = \lim_{k \rightarrow \infty} \|f_{ik}\| \quad (\text{we remind that } \|F\| = (\|f_1\|, \|f_2\|, \dots), \|\tilde{F}_k\| = (\|f_{1k}\|, \dots, \|f_{i_k k}\|) \text{ - see section 5}).$$

In order to illustrate the above algorithm, we consider the following example.

Example

Let  $C = (1, B_1, B_2)$  be a vector of PD-coroutines over the net  $(U, \leq, \circ, \perp, e)$ , where the symbol  $\circ$  will be omitted,

$\{r_1, r_2, r_3, r_4, s_1, s_2, s_3\} \subseteq U$ , and:

$$B_1 = (\{\delta_1, a_1, a_2, a_3, \varepsilon\}, \delta_1, P_1),$$

$$P_1 = \{(1 \rightarrow 1, \delta_1 \rightarrow \varepsilon, r_1), (1 \rightarrow 1, \delta_1 \rightarrow \delta_1 a_1, r_2), (1 \rightarrow 1, a_1 \rightarrow a_2, r_3), (1 \rightarrow 2, a_2 \rightarrow a_3, e), (1 \rightarrow 1, a_3 \rightarrow \varepsilon, r_4)\},$$

$$B_2 = (\{\delta_2, b_1, b_2, \varepsilon\}, \delta_2, P_2),$$

$$P_2 = \{(2 \rightarrow 2, \delta_2 \rightarrow \varepsilon, s_1), (2 \rightarrow 2, \delta_2 \rightarrow b_1, s_2), (2 \rightarrow 1, b_1 \rightarrow \delta_2 b_2, e), (2 \rightarrow 2, b_2 \rightarrow \varepsilon, s_3)\},$$

where  $\varepsilon$  - the empty symbol.

In order to make our considerations more intuitive, instead of  $X_i$  we shall write  $X(\gamma(i))$ , i.e. instead of  $X_1$  we shall write  $X(1, \delta_1, \delta_2)$  and so on.

A canonical set of equations defined for  $C$  by the above algorithm is the following:

$$\begin{array}{l}
 \left. \begin{array}{l}
 k=1 \\
 k=2 \\
 k=3
 \end{array} \right\} \begin{cases}
 X(1, \delta_1, \delta_2) = r_1 X(1, \varepsilon, \delta_2) \vee r_2 X(1, \delta_1 a_1, \delta_2) \\
 X(1, \varepsilon, \delta_2) = e \\
 X(1, \delta_1 a_1, \delta_2) = r_1 X(1, a_1, \delta_2) \vee r_2 X(1, \delta_1 a_1 a_1, \delta_2) \\
 X(1, a_1, \delta_2) = r_3 X(1, a_2, \delta_2) \\
 X(1, a_2, \delta_2) = X(2, a_3, \delta_2) \\
 X(2, a_3, \delta_2) = s_1 X(2, a_3, \varepsilon) \vee s_2 X(2, a_3, b_1) \\
 X(2, a_3, \varepsilon) = e \\
 X(2, a_3, b_1) = X(1, a_3, \delta_2 b_2) \\
 X(1, a_3, \delta_2 b_2) = r_4 X(1, \varepsilon, \delta_2 b_2) \\
 X(1, \varepsilon, \delta_2 b_2) = e \\
 X(1, \delta_1 a_1 a_1, \delta_2) = r_1 X(1, a_1 a_1, \delta_2) \vee r_2 X(1, \delta_1 a_1 a_1 a_1, \delta_2) \\
 X(1, a_1 a_1, \delta_2) = r_3 X(1, a_2 a_1, \delta_2) \\
 X(1, a_2 a_1, \delta_2) = X(2, a_3 a_1, \delta_2) \\
 X(2, a_3 a_1, \delta_2) = s_1 X(2, a_3 a_1, \varepsilon) \vee s_2 X(2, a_3 a_1, b_1)
 \end{cases}
 \end{array}$$

$$\begin{array}{l}
 \left. \begin{array}{l} k=3 \\ k=4 \end{array} \right\} \begin{array}{l}
 X(2, a_3 a_1, \epsilon) = e \\
 X(2, a_3 a_1, b_1) = X(1, a_3 a_1, \sigma_2 b_2) \\
 X(1, a_3 a_1, \sigma_2 b_2) = r_4 X(1, a_1, \sigma_2 b_2) \\
 X(1, a_1, \sigma_2 b_2) = r_3 X(1, a_2, \sigma_2 b_2) \\
 X(1, a_2, \sigma_2 b_2) = X(2, a_3, \sigma_2 b_2) \\
 X(2, a_3, \sigma_2 b_2) = s_1 X(2, a_3, b_2) \vee s_2 X(2, a_3, b_1 b_2) \\
 X(2, a_3, b_2) = s_3 X(2, a_3, \epsilon) \\
 X(2, a_3, \epsilon) = e \\
 X(2, a_3, b_1 b_2) = X(1, a_3, \sigma_2 b_2 b_2) \\
 X(1, a_3, \sigma_2 b_2 b_2) = r_4 X(1, \epsilon, \sigma_2 b_2 b_2) \\
 X(1, \epsilon, \sigma_2 b_2 b_2) = e \\
 X(1, \sigma_1 a_1 a_1 a_1, \sigma_2) = \dots \\
 \dots \\
 \dots \\
 \dots \\
 \dots
 \end{array}
 \end{array}$$

After a solution the part of equations for  $k=3$  we receive:

$$\| \tilde{f}_{13} \| = X(1, \sigma_1, \sigma_2) = \bigcup_{n=0}^2 r_2^n r_1 (r_3 s_2 r_4)^n \vee \bigcup_{n=1}^2 \bigcup_{m=1}^n r_2^n r_1 (r_3 s_2 r_4)^{m-1} r_3 s_1 s_3^{m-1} .$$

We can prove by induction on  $k$  that for every  $k$ :

$$\| \tilde{f}_{1k} \| = X(1, \sigma_1, \sigma_2) = \bigcup_{n=0}^{k-1} r_2^n r_1 (r_3 s_2 r_4)^n \vee \bigcup_{n=1}^{k-1} \bigcup_{m=1}^n r_2^n r_1 (r_3 s_2 r_4)^{m-1} r_3 s_1 s_3^{m-1} .$$

In the final step we count  $\lim_{k \rightarrow \infty} \| \tilde{f}_{1k} \|$ , and we obtain:

$$\| f_1 \| = X(1, \sigma_1, \sigma_2) = \bigcup_{n=0}^{\infty} r_2^n r_1 (r_3 s_2 r_4)^n \vee \bigcup_{n=1}^{\infty} \bigcup_{m=1}^n r_2^n r_1 (r_3 s_2 r_4)^{m-1} r_3 s_1 s_3^{m-1} ,$$

then:

$$Res_c = Tail(1, \sigma_1, \sigma_2) = \bigcup_{n=0}^{\infty} r_2^n r_1 (r_3 s_2 r_4)^n \vee \bigcup_{n=1}^{\infty} \bigcup_{m=1}^n r_2^n r_1 (r_3 s_2 r_4)^{m-1} r_3 s_1 s_3^{m-1} .$$

Janicki [5,6] has proved (using different method) that the above result is really  $Res_c$  of this vector of coroutines. In this way we have counted  $X(1, \sigma_1, \sigma_2)$ . Other variables of the equation  $\underline{x} = F(\underline{x})$  can be calculated in the similar way.

References

[1] Bekić H., Definable operations in general algebras and the theory of automata and flowcharts (manuscript), IBM Laboratory, Vienna 1969.

[2] Blikle A., Equational languages, Information and Control, 21 (1972), pp.134-147.

[3] Blikle A., An analysis of programs by algebraic means, In:

A.Mazurkiewicz,Z.Pawlak (ed.) ,Mathematical Foundations of Computer Science,Banach Center Publ.,vol.2,PWN,Warsaw,1977,pp.167-214.

- [4] Blikle A.,An extended approach to mathematical analysis of programs,CC PAS Reports,169,1974.
- [5] Janicki R.,Results of the theory of vectors of coroutines,ICS PAS Reports,379,1979.
- [6] Janicki R.,Analysis of vectors of coroutines by means of components,In: L.Budach (ed.),Fundamentals of Computation Theory,Math. Research,Band 2,Akademie-Verlag,Berlin,1979,pp.207-213.
- [7] Just J.R.,An algebraic model of distributed computer systems,Proc. of the 5th Conference on the Theory of Operating Systems,Tanulmanyok 100/1979,Budapest,1979,pp.311-323.
- [8] Just J.R.,Synthesis and analysis of distributed computer systems by algebraic methods,Ph.D. Thesis,Institute of Comp. Sci.,Warsaw Technical University,Warsaw,1980.
- [9] Kleene S.C.,Introduction to methamathematics,New York,1952.
- [10] Kuratowski K.,Mostowski A.,Set Theory,Nord Holland Publ. Comp.,1967,Amsterdam.
- [11] Leszczyłowski J.,A theorem on resolving equations in the space of languages,Bull. Acad. Polon. Sci.,Ser. Sci. Math. Astronom. Phys. 19 (1971),pp.967-970.

---

\* Institute of Computer Science,Warsaw Technical University,  
ul. Nowowiejska 15/19, 00-665 Warszawa/Poland

+ Institute of Mathematics, Warsaw Technical University,  
Pl. Jedności Robotniczej 1, 00-661 Warszawa/Poland



SYNCHRONIZATION AND COMMUNICATION IN DISTRIBUTED COMPUTER  
SYSTEMS BY MEANS OF COROUTINES.

Jan Rudolf Just  
Poland

1. Introduction.

The main subject of the paper are problems of the communication and the synchronization in distributed computer systems.

Since virtual distribution is realized through software support, the communication mechanism provided to realize the interaction between the different system components may be adopted to particular requirements. As a consequence, this mechanism varies considerably from one system to another. Certain primitives for interprocess communication have been incorporated into system programming language. Since they are effective and determine the order in which the actions of the system may be executed, we call such an primitives a synchronization mechanism. The important concept is coroutine.

The concept of coroutines has been known for a long time since it was firstly introduced by Conway [1]. Coroutines essentially differ from subroutines in their calling relationship. Whereas a subroutine is call and return to the point of call in the calling routine after having completed its task, a coroutines may swap / sequencing / control to another one, it is left with the current program position marked as its activation point for a subsequent entry. Only at its first activation is a coroutine entered at its head; any later exchange of control enters its body at the resumptive activation point of the previous activation.

To gain a theoretical understanding of distributed systems, it is necessary to find mathematical models which reflect the essential feature of these systems while abstracting away irrelevant details. Such models allows problem to be stated precisely and make them amenable to mathematical analysis. In papers [8,9] it has been introduced a mathematical model of distributed computer systems and a mathematical model of their input/output behavior.

Our description of a distributed system include coroutine mechanism, in order to process communication and synchronization.

Formally, our model is based on the notion of so called vector of coroutines. This notion has been introduced by Janicki [5], in order to describe the semantics of programs with coroutines.

The main subject of paper being presented is the problem of the synthesis of processes in distributed systems / DS /. The synthesis of processes in DS problem solution will be a distribution of processes in the system - an allocation of actions to particular processors -, and a synchronization of their actions and a design of their communication mechanism, such that the execution of these processes will be feasible in required manner.

In our approach communication and synchronization are accomplished through the input and output constructs.

## 2. The model.

In this chapter basic facts, usefull for the problem examined, below will be presented. For more details the reader is advised to refer to [8].

For every  $n=1,2,\dots$ , let  $[n] = \{1,2,\dots,n\}$ . For every alphabet  $\Sigma$  let  $\Sigma^* = \Sigma \cup \{\epsilon\}$  where  $\epsilon$  denotes an empty word,  
 $\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i$ ,  $\Sigma^+ = \Sigma^* \Sigma$ .

The remaining notations are either standard or defined in suitable sections.

By a model of distributed computer systems we shall mean 3-tuple:

$$DCS = ( S , MP , AL ) , \text{ where:}$$

S - a structure of a system,

MP - a set of processes in a system,

AL - a mapping  $AL:MP \rightarrow S$ .

### 2.1. Structure of DCS.

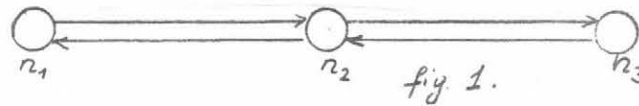
By a structure of DCS we mean a directed graph:

$$S = ( N , n_0 , LT ) , \text{ where:}$$

N - a set of nodes / stations of a computer network,

$n_0 \in N$  - an initial node,  
 $LT \subseteq N \times N$  - a set of edges / transmission lines /.

Example 2.1.



2.2. Processes in DCS.

A task realization in DCS is the result of the activity of processes distributed in the system and connected asynchronously. During the task realization a user of the system creates so called virtual network of processes. The virtual network of processes consists of a set of logically connected processes. Each of coprocess for a given virtual process is executed in different pr-ocessor of DCS.

In order to describe the set of processes in DCS we shall introduce a mathematical object, called a matrix of coprocesses. This object describes the algorithmic structure / semantics / of DCS.

2.2.1. Matrix of coprocesses.

By a matrix of coprocesses we mean a system:

$$MP = (\mathcal{A}, I_0) \text{ , where:}$$

$$\mathcal{A} = \{A_{ij}\}_{\substack{i \in [m] \\ j \in [n]}} \text{ , } I_0 \in [m] \times [n] \text{ .}$$

$A_{ij}$  - are coprocesses / see below /,  $I_0$  - indicates the start process.

$A_{ij}$  - a 4-tuple which represent j-th coprocess in i-th process.

$$A_{ij} = (\Sigma_{ij}, V_{ij}, \sigma_{ij}, P_{ij}) \text{ or } A_{ij} = (\emptyset, \emptyset, \{\epsilon\}, \emptyset)$$

1/  $\Sigma_{ij}$  - an alphabet / of action names symbols /,

2/  $V_{ij}$  - an alphabet / of control symbols of  $A_{ij}$  /,

3/  $\sigma_{ij} \in V_{ij}$  - the start symbol of  $A_{ij}$ ,

4/  $P_{ij}$  - a finite subset of the set:

$$\{(i, j)\} \times ([m] \times [n]) \times (V \times V^*) \times \Sigma_{ij}^*$$

This mean that  $P_{ij}$  is a finite set of 4-tuples of the form:

$$(i \rightarrow r, j \rightarrow s, a \rightarrow b, R) \text{ , where:}$$

1/  $i \rightarrow r \in \{i\} \times [m]$  , 3/  $a \rightarrow b \in V \times V^*$  ,

2/  $j \rightarrow s \in \{j\} \times [n]$  , 4/  $R \in \Sigma_{ij}^*$  .

$P_{ij}$  is called the set of instructions of  $A_{ij}$ .

Let  $P = \bigcup_{i=1}^m \bigcup_{j=1}^n P_{ij}$ .

Each of instructions consists of four parts:

- 1/  $i \rightarrow r$  indicates the process which will be active after the execution of the instruction /  $r$ -th process will be active /,
- 2/  $j \rightarrow s$  indicates the number of coprocess which will be active after execution of the instruction,
- 3/  $a \rightarrow b$  indicates the way of execution of the component  $A_{ij}$ . This part of the instruction indicates the current and the next point of  $A_{ij}$  component,
- 4/  $R$  is the "action" of that instruction. It is an action name.  $R$  in respect to an abstract character, we shall mean as the program, the part of the program or an activity of the operating system.

Every matrix of coprocesses can be represented graphically using graphs:

$\xrightarrow{R}$                        $\xrightarrow{ij}$                        $\xrightarrow{ij}$

to denote instructions:  $(i \rightarrow i, j \rightarrow j, a \rightarrow b, R)$ ,  $(i \rightarrow i, j \rightarrow s, a \rightarrow b, R)$  and  $(i \rightarrow r, j \rightarrow s, a \rightarrow b, R)$  respectively.

Put  $\Sigma = \bigcup_{i=1}^m \bigcup_{j=1}^n \Sigma_{ij}$ . The set  $\Sigma$  is called the set of action names of MP.

Let  $ms = \prod_{i=1}^m \prod_{j=1}^n V_{ij}^*$  /  $\times$  - a cartesian product /.  
 For each element  $\alpha \in ms$   $\alpha = \prod_{i=1}^m \prod_{j=1}^n a_{ij}$ , where:  $i \in [m], j \in [n]$   
 $a_{ij} \in V_{ij}^*$ .

The set  $MS = [m] \times [n] \times ms$  is called the set of control states of the matrix MP.

Let  $co: [m] \times [n] \times ms \rightarrow \prod_{i=1}^m \prod_{j=1}^n V_{ij}^*$  be a function such that:  
 $co(i, j, \alpha) = a_{ij}$ .

Each  $(i \rightarrow r, j \rightarrow s, a \rightarrow b, R)$  can be regarded as a relation in the set  $Rel(MS)$  / where by  $Rel(X)$  we denote the set of  $Rel(X) = \{R | R \subseteq X \times X\}$  /, defined in the following way:

$y_1(i \rightarrow r, j \rightarrow s, a \rightarrow b, R) y_2 \Leftrightarrow \{ \alpha, \beta \in ms \mid y_1 = (i, j, \alpha), y_2 = (r, s, \beta) \}$   
 and  $co(i, j, \alpha) = a$ ,  $co(r, s, \beta) = b$ .

The set  $MT = \{(i, j, \alpha) \in MS \mid co(i, j, \alpha) = \varepsilon\}$  is called the set of terminal control states of MP.

The set  $ST = MS \times \Sigma^*$  is called the set of states of MP.

Let  $Tr \subseteq ST \times ST$  be the relation defined by the equivalence:



$(y_1, u_1) \text{Tr}(y_2, u_2) \Leftrightarrow [ \exists (i \rightarrow r, j \rightarrow s, a \rightarrow b, R) \in P \ (y_1, y_2) \in MS \ \& \ u_2 = u_1 R ]$ .

We put  $y_0 = (i_0, j_0, \alpha_0)$ , where:

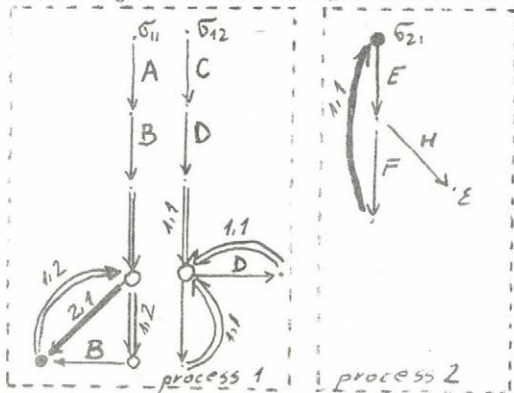
$$co(i, j, \alpha_0) = \begin{cases} \sigma_{ij} & \text{for } A_{ij} \neq \emptyset \\ \varepsilon & \text{for } A_{ij} = \emptyset \end{cases}$$

/By  $\emptyset$  we denote of the empty coprocess of the form  $(\emptyset, \emptyset, \{\varepsilon\}, \emptyset)$  /  
The control state  $y_0$  is called the start control state of MP.

Put:  $L(MP) = \{ w \in \Sigma^* \mid (\exists y \in MT) (y_0, \varepsilon) \text{Tr}^*(y, w) \}$ .

The language  $L(MP)$  is called the language generated by the matrix of coprocesses MP. This language is interpreted as a description of the semantics of the matrix MP. In our the model the language  $L(MP)$  expresses the outcome of the virtual process.

Example 2.2.1. Consider the system which consists of two processes such the first process consists of two coprocesses. Let this system be represented by the below flowdiagram.



• - the reactivation point of process,

o - the reactivation point of coprocess.

It can be proved that:

$$L MP = ABCD \ EF \ GB \ D \ EH .$$

### 2.3. Allocation function AL.

The mapping AL is the third element of the DCS model. To describe the particular system it is necessary to specify:

- 1/ how to allocate processes to processors,
- 2/ how to allocate communication lines between that processors.

It is specified by the mapping AL. The mapping AL is a certain homomorphism among structure of DCS - the graph S and the graph of given virtual process. This means that the structure of logical channels between components of the given virtual process, must be adequate to the structure of connections between processors of DCS.

### 3. Communication and synchronization in DCS.

Proving properties of the system of processes /in our the model/ is proving properties of the language  $L(MP)$ . Properties

of these language can be analysed by means of fixt-point methods / see [8,9] /. The language  $L(MP)$  does not contain much information about the structure of the matrix of coprocesses. If we know this language only we do not know anything about the number and the form of components. We do not know anything about of the component / coprocesses / synchronization and the communication in the system. Now we define a language which defines the language  $L(MP)$ , the number of components, sublanguages defined by components and contain an information about the communication and synchronization in the system.

Note that every component can be interpreted as certain right-linear grammar.

Let  $MP = (\mathcal{A}, I_0)$ , where:  $\mathcal{A} = \{A_{ij}\}_{i \in [m], j \in [n]}$ ,  $I_0 \in [m] \times [n]$  and

$A_{ij} = (\Sigma_{ij}, V_{ij}, \sigma_{ij}, P_{ij})$  be a  $j \in [n]$  matrix of coprocesses.

We define the following alphabets:  $T_j^i = \{t_{js_1}^{ir_1}, \dots, t_{js_n}^{ir_m}\} - \{t_{jj}^{ii}\}$ ,

$\hat{T}_j^i = T_j^i \cup \{t_j^i\}$ , for  $i, r \in [m], j, s \in [n]$ .

Let also:  $T = \bigcup_{i=1}^m \bigcup_{j=1}^n T_j^i$ ,  $\hat{T} = \bigcup_{i=1}^m \bigcup_{j=1}^n \hat{T}_j^i$ .

The set  $T$  in our model represents the set of transmission actions names. For example  $t_{js}^{ir} \in T$  denote the action of transmission from  $j$ -th coprocess of  $i$ -th process to  $s$ -th coprocess of  $r$ -th process.

Let  $T(MP)$  be the matrix of coprocesses defined as follows:

$T(MP) = (\mathcal{A}^T, I_0)$ , where:  $\mathcal{A}^T = \{A_{ij}^T\}_{i \in [m], j \in [n]}$ ,  $I_0 \in [m] \times [n]$ ,

$A_{ij}^T = (\Sigma_{ij} \cup T_j^i, V_{ij} \cup \{\sigma_{ij}'\}, \sigma_{ij}', P_{ij}^T)$  and

$P_{ij}^T = \{(i \rightarrow r, j \rightarrow s, a \rightarrow b, R) \mid (i \rightarrow r, j \rightarrow s, a \rightarrow b, R) \in P_{ij} \ \& \ (i \neq r \vee j \neq s)\} \cup$   
 $\cup \{(i \rightarrow r, j \rightarrow s, a \rightarrow b, R) \mid (i \rightarrow r, j \rightarrow s, a \rightarrow b, R) \in P_{ij} \ \& \ (i = r \ \& \ j = s)\} \cup$   
 $\cup \{(i \rightarrow i, j \rightarrow j, \sigma_{ij} \rightarrow \sigma_{ij}, \nu^n) \mid \text{if } (i, j) = I_0 \text{ then } \nu^n = t_{j0}^i, \text{ elsewhere } \nu^n = \varepsilon\}$ .

The language  $L(T(MP))$  consists all necessary information about the structure of the matrix of coprocesses.

Let  $h_T: (\Sigma \cup \hat{T})^* \rightarrow \Sigma^*$  be the following homomorphism:

$$(\forall A \in \Sigma \cup \hat{T}) \quad h_T(A) = \begin{cases} A & A \in \Sigma \\ \varepsilon & A \notin \Sigma \end{cases} .$$

Corollary 1.  $L(MP) = h_T(L(T(MP)))$ .

A component  $A_{ij}$  of  $MP$  is called final if there exists such an

instruction  $(i \rightarrow r, j \rightarrow s, a \rightarrow b, R) \in P_{ij}$  that  $b = \varepsilon$ .  
 The set of all final components of MP will be denoted by FINAL.  
 We restrict our attention to the matrix MP with the property  $\text{card}(\text{FINAL}) = 1$ . For  $i=1, \dots, m$ ,  $j=1, \dots, n$ , let  $G(A_{ij})$  be a right linear grammar defined as follows:

$$G(A_{ij}) = ( \sum_{ij} \cup T_{ij}^i, V_{ij}, \sigma_{ij}, \hat{Q}_{ij} )$$

$$\hat{Q}_{ij} = \{ a \rightarrow Rb \mid (i \rightarrow r, j \rightarrow s, a \rightarrow b, R) \in P_{ij} \} \cup$$

$$\cup \{ a \rightarrow R t_{js}^{ir} b \mid (i \rightarrow r, j \rightarrow s, a \rightarrow b, R) \in P_{ij} \ \& \ (i \neq r \vee j \neq s) \} \cup$$

$$\cup \{ a \rightarrow R t_{js}^{ir} \mid (i \rightarrow r, j \rightarrow s, a \rightarrow b, R) \in P_{ij} \ \& \ A_{ij} \in \text{FINAL} \} .$$

Let  $L(G(A_{ij}))$  denotes the language generated by the grammar  $G(A_{ij})$ . It can be prove, that if we know grammars  $G(A_{ij})$  / it's mean that we know the languages  $L(T(A_{ij}))$  / we can define the language  $L(T(\text{MP}))$ .

#### 4. Synthesis of processes in distributed computer system.

From the point of view our paper the synthesis of processes in DCS problem solution, will be a distribution of processes in the system - an allocation of actions to particular processors, a synchronization of their actions and a design of their communications mechanism, such that the execution of these processes will be feasible in required manner.

To solve this problem in the formal way, we shall present some properties of regular expressions and regular language.

##### 4.1. Some properties of regular expressions and regular languages.

In this section we recall some theory from [7] and [9]. By the set of regular expressions over an alphabet  $\Sigma$ ,  $\text{REX}(\Sigma)$  we shall mean the least set of terms which fulfils the following conditions:

1.  $\Sigma \cup \{ \varepsilon \} \subseteq \text{REX}(\Sigma)$
2.  $(\forall u \in \text{REX}(\Sigma)) \ u^* \in \text{REX}(\Sigma)$
3.  $(\forall u_1, u_2 \in \text{REX}(\Sigma)) \ u_1 u_2, u_1 \cup u_2 \in \text{REX}(\Sigma)$ .

For every the regular expression WR, let  $|WR|$  denote the regular language defined by WR.

Assume that symbols  $\Sigma, \Sigma_1, \dots, \Sigma_z$  denote alphabets and

1.  $\Sigma = \Sigma_1 \cup \dots \cup \Sigma_z$
2.  $k \neq l \Rightarrow \Sigma_k \cap \Sigma_l = \emptyset$
3.  $(\forall k \in [z]) \ \Sigma_k \neq \emptyset$ .

By a segment of a word  $w$  we mean any word  $u$  such that:  $(\exists v_1, v_2) w = v_1 u v_2$ . By a factor of a regular expression  $WR$  we mean every maximal segment of  $WR$  which does not contain characters  $(, )$ ,  $\cup$  and does not begin from the character  $*$ . Let  $\text{fac}(WR)$  denote the set of all factors of the regular expression  $WR$ . For example, if  $WR = ABCD((EF \cup GB)D)^*EH$  then  $\text{fac}(WR) = \{ABCD, D, EF, GB, EH\}$ .

Thus, every factor is a word over the alphabet  $\Sigma \cup \{*\}$ .

Let  $\bar{\Sigma}$  be the alphabet defined in the following way:

$\bar{\Sigma} = \Sigma \cup \{A^* \mid A \in \Sigma\}$ , where every two characters  $A, *$  written of the form  $A^*$  are treated as one symbol.

Let  $u \in \bar{\Sigma}^+$ . The word  $u$  can unambiguously be represented as the concatenation:  $u = u_1 \dots u_p$ , where for every  $h = 1, \dots, p$ ,  $u_h$  is a maximal segment consisting of symbols from  $\bar{\Sigma}_{k_h}$  only.

If  $u_1 u_2 \dots u_p$  is a decomposition of the word  $u$  of the form described above, and for  $k = 1, \dots, z$ :  $\Lambda_k = \{\lambda_{k1}, \dots, \lambda_{kk-1}, \lambda_{kk+1}, \dots, \lambda_{kz}\}$ ,  $\Lambda = \bigcup_{k=1}^z \Lambda_k$ , then let  $\lambda(u) \in (\bar{\Sigma} \cup \Lambda)^+$  be the word defined in the following way:

$$\lambda(u) = u_1 \lambda_{k_1 k_2} u_2 \lambda_{k_2 k_3} \dots u_{p-1} \lambda_{k_{p-1} k_p} u_p.$$

Let  $k, l \in [p]$ , and let  $u$  have the decomposition as the above. Then  $f_{kl}(u)$  denotes the word defined as follows:

$$f_{kl}(u) = \begin{cases} \lambda(u) & k_1 = k \text{ and } k_p = l \\ \lambda_{kk_1} \lambda(u) & k_1 \neq k \text{ and } k_p = l \\ \lambda(u) \lambda_{k_1 l} & k_1 = k \text{ and } k_p \neq l \\ \lambda_{kk_1} \lambda(u) \lambda_{k_p l} & k_1 \neq k \text{ and } k_p \neq l. \end{cases}$$

Let  $WR$  be any regular expression over an alphabet  $\Sigma$ . Note that there always exists a regular expression  $WR'$  with the following properties:

1.  $|WR'| = |WR|$
2.  $WR' = \begin{cases} q_1 p_1 \cup \dots \cup q_p p_p \cup \varepsilon & \varepsilon \in |WR| \\ q_1 p_1 \cup \dots \cup q_p p_p & \varepsilon \notin |WR| \end{cases}$ ,

where  $p_k \in \text{fac}(WR')$ ,  $q_k \in \text{REX}(\Sigma)$  for  $k = 1, \dots, p$ .

Let  $\text{NREX}(\Sigma) = \{WR' \mid WR' \in \text{REX}(\Sigma)\}$ , where  $WR'$  is an expression of the form defined above by points 1, 2.

For every  $k, l \in [z]$ , let  $H_{kl}$  be the mapping of the form:

$H_{kl}: \text{NREX}(\Sigma) \rightarrow \text{REG}(\bar{\Sigma} \cup \Lambda)$  which acts under the rules described below.

Assume that  $WR' = q_1 p_1 \dots q_p p_p \vee \varepsilon$ , where  $p_s \in \text{fac}(WR)$  for  $s=1, \dots, p$ . We replace each of factors  $p_s / s=1, \dots, p /$  by  $f_{kl}(p_s)$ , and in every fragment  $q_s / s=1, \dots, p /$  we replace every factor  $u = \text{fac}(q_s)$  by  $f_{kk}(u)$ . If  $k \neq 1$  then we replace the symbol  $\varepsilon$  by  $\lambda_{kl}$ . The result of these transformations equal to  $H_{kl}(WR')$ .

For  $WR = q_1 p_1 \vee \dots \vee q_p p_p$  we proceed analogously.

#### 4.2. Synthesis of communicating coprocesses.

Problem of communicating coprocesses synthesis - from the point of view of our model - is the problem of synthesis of the matrix of coprocesses. It can be formally expressed in the following way. The process given is in the form of the regular language / regular expression /  $WR$ . Taking into considerations a technical constrains as a result of impossibility of the execution some particular parts of the process by a given processor we define the mapping  $AL$ . This mapping creat the partition of an alphabet  $\Sigma, \{\Sigma_{11}, \Sigma_{12}, \dots, \Sigma_{mn}, \Sigma\}$  for  $\Sigma_{11} \vee \Sigma_{12} \vee \dots \vee \Sigma_{mn} = \Sigma$  and

$$\forall (i, j), (r, s) \in [m] \times [n] \quad (i, j) \neq (r, s) \Rightarrow \Sigma_{ij} \cap \Sigma_{rs} = \emptyset.$$

For given the problem we should build a matrix of coprocesses which contains of  $mn$  components and generates the language  $|WR| = L(MP)$  and the alphabet / of action symbols / of  $(i, j)$ -th component is contain in  $\Sigma_{ij}$ .

Now we fix  $I_0$  - number of an initial coprocess,  $I_0 = (i_0, j_0) \in [m] \times [n]$ , and  $I_F = (i_F, j_F) \in [m] \times [n]$  - number of the final coprocess.

For  $i=1, 2, \dots, m, j=1, 2, \dots, n$  we define the family of process synthesis transformations

$$FPST = \{ PST_{11}^{11}, \dots, PST_{js}^{ir}, \dots, PST_{nn}^{mm} \}$$

described on  $NREX(WR)$ , such that:

$$(\forall PST_{js}^{ir} \in FPST) \quad PST_{js}^{ir} = H_{\text{num}(i, j) \text{ num}(r, s)}$$

where:  $\text{num}: [m] \times [n] \rightarrow [mn]$  is defined in the following way:  
 $\text{num}(i, j) = (i-1)m + j$ .

/ The transformation  $H$  was defined in section 4.1 /

$$\text{Define } \overline{WR} = PST_{j_0 j_F}^{i_0 i_F} (WR).$$

Note that every the regular expression is a word over the alphabet  $\Sigma \cup \{ ( , ) , \cup , * , \varepsilon \}$ . Thus, it can be an argument of the homomorphism  $h_\Omega$  / definrd in section 3 /, which effaces characters belonging to  $\Omega$ .

From the formal language theory it is follows that:

$$(\forall \Omega \subseteq \Sigma)(\forall WR \in \text{REX}(\Sigma)) \quad |h_\Omega(WR)| = h_\Omega(|WR|) \quad .$$

Let  $\{\overline{WR}_{11}, \dots, \overline{WR}_{mn}\}$  be the set of regular expressions of the following form:  $\forall \text{num}(i, j) = \text{num}(m, n) \quad \overline{WR}_{ij} = h_{\Omega_j^i}(\overline{WR})$ ,

where:  $\Omega_j^i = (\Sigma \cup T) - (\Sigma_{ij} \cup T_j^i)$  .

Example 4.2.1.

For the regular expression  $WR = ABCD((EF \cup GB)D)^*EH$  and for the partition of the alphabet  $\Sigma = \{A, B\} \cup \{C, D, G\} \cup \{E, H, F\}$ , let  $I_0 = (1, 1)$ ,  $I_F = (2, 1)$ . In this case  $WR' = WR$ .

$$\overline{WR} = ABt_{12}^{11}CDt_{21}^{11}((t_{11}^{12}Eft_{11}^{21} \cup t_{12}^{11}Gt_{21}^{11}B)t_{12}^{11}Dt_{21}^{11})^*t_{11}^{12}EH$$

$$\Omega_1^1 = \{A, B, t_{12}^{11}, t_{11}^{12}\}, \quad \Omega_2^1 = \{C, D, G, t_{21}^{11}, t_{21}^{12}\}, \quad \Omega_1^2 = \{E, H, F,$$

$$\overline{WR}_{11} = ABt_{12}^{11}((t_{11}^{12} t_{12}^{11}B) t_{12}^{11})^*t_{11}^{12}, \quad t_{11}^{21}, t_{12}^{21}\}$$

$$\overline{WR}_{12} = CDt_{21}^{11}((Gt_{21}^{11} \cup \varepsilon)Dt_{21}^{11})^*$$

$$\overline{WR}_{21} = (Eft_{11}^{21} \cup \varepsilon)EH.$$

From the formal language theory it also follows that for every  $\overline{WR}$ ,  $i=1, \dots, m, j=1, \dots, n$  we can build a right linear grammar:  $G_{ij} = (\Sigma_{ij} \cup T_j^i, V_{ij}, \sigma_{ij}, \tilde{Q}_{ij})$  such, that:

1.  $L(G_{ij}) = |WR_{ij}|$
2.  $\forall a \rightarrow Rb \in \tilde{Q}_{ij} \quad R \in \Sigma_{ij} \cup T_j^i.$

Of course, the above construction is ambiguous, and for every  $i=1, \dots, m, j=1, \dots, n$  if  $\overline{WR}_{ij}$  contains the character  $\varepsilon$  then there exists an infinite number of grammars which fulfils such conditions.

For a given regular expression  $WR, I_0, I_F$  and for given partition of the alphabet  $\Sigma$  let GRAM denote the family of all sets of grammars  $\{G_{11}, \dots, G_{mn}\}$ .

Let TAB be the following set of matrices of coprocesses

$$MP \in \text{TAB} \Leftrightarrow (\exists \{G_{11}, G_{12}, \dots, G_{mn}\} \in \text{GRAM}) G_{ij} = (\Sigma_{ij} \cup T_j^i, V_{ij}, \sigma_{ij}, \tilde{Q}_{ij})$$

and  $MP = (A, I_0)$ ,  $A = \{A_{ij}\}_{\substack{i \in [m] \\ j \in [n]}}$ ,  $I_0 \in [m] \times [n]$

$$A_{ij} = (\Sigma_{ij} \cup T_j^i, V_{ij}, \sigma_{ij}, P_{ij})$$

$$P_{ij} = \{ (i \rightarrow i, j \rightarrow j, a \rightarrow b, R) \mid a \rightarrow R b \in \tilde{Q}_{ij} \text{ and } R \in \Sigma_{ij}^* \} \cup$$

$$\cup \{ (i \rightarrow r, j \rightarrow s, a \rightarrow b, \varepsilon) \mid a \rightarrow t_{js}^{ir} b \in \tilde{Q}_{ij} \text{ and } b \notin \varepsilon \} \cup$$

$$\cup \{ (i \rightarrow r, j \rightarrow s, a \rightarrow \varepsilon, \varepsilon) \mid a \rightarrow t_{js}^{ir} \in \tilde{Q}_{ij} \text{ and } (i, j) = (i_F, j_F) \}.$$

Theorem. For every matrix of coprocesses  $MP = (A, I_0) \in \text{TAB}$

$$A = \{ A_{ij} \}_{\substack{i \in [m] \\ j \in [n]}}, \quad I_0 = [m] \times [n] :$$

1.  $(\forall G_{ij} \in \{G_{11}, \dots, G_{mn}\} \in \text{GRAM} \quad G_{ij} = G(A_{ij}) .$
2.  $L(MP) = |WR| .$

Example 4.2.3. / compare example 4.2.1. /. The matrix of coprocesses for this example of system can be graphically expresses by the graph from fig.2 .

#### 4.2.1. The graph of the DCS transmmision.

Let's obtain so called the graph of transmission  $G_T$ , for a given matrix of coprocesses. The graph of transmmision will represent the structure of logical connections between - allocated to separate processors - interacted coprocesses. Let  $G_T = (N_T, L_T)$ , where:  $N_T$  - the subset of nodes of DCS structure - determined by mapping AL.  $L_T \subseteq N_T \times N_T$  - the set of logical channels between coprocesses of MP. Having the expression  $\overline{WR}$  / see section 4.2. / we can obtain the set  $L_T$ .

Let  $\mathcal{L}(x)$  be the function defined as follows:

$$(\forall x \in T^*) \mathcal{L}(x) = \{ t \in T \mid \exists u, v \in T^*, utv = x \} .$$

Now we define the set of symbols  $t_j^i \in T$  / actions of transmission / belongs to the expression  $\overline{WR}$ . This set will be denoted by  $\text{LOCH}_{\overline{WR}}$ .  $\text{LOCH}_{\overline{WR}} = \mathcal{L}(h_{\Sigma}(\overline{WR})) .$

We shall present an example illustrating the construction described above.

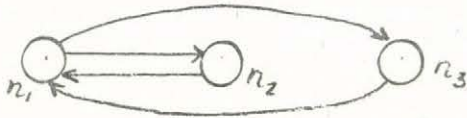
Example 4.2.1.1. For expression  $\overline{WR}$  was defined in Examp.4.2.1.

$$\text{LOCH}_{\overline{WR}} = \{ t_{12}^{11}, t_{21}^{11}, t_{11}^{12}, t_{11}^{21} \} .$$

Let  $N_T = \{ n_1, \dots, n_q \}$ . Note that for a given matrix of coprocesses and a given allocation function AL we have:

$$(\forall i \in [m], j \in [n]) \quad L_T = \{ (n_i, n_j) \in N_T \times N_T \mid \exists t_{js}^{ir} \in \text{LOCH}_{\overline{WR}} \} .$$

Example 4.2.1.2. The graph of transmission for the matrix of coprocesses from an Example 2.2.1. has the form:



The interconnection structure of the graph  $G_T$  must be adequate to the interconnection structure of the graph  $S$  - the structure

of physical connections between processors of the computer network. There have to exist the isomorphism between the graph  $G_T$  and the graph  $S$ . This requirement isn't satisfied in our example system / compare Ex. 2.1. and Ex. 4.2.1.2. / . Thus, the matrix of coprocesses must be modified - we must change both the communication and synchronization mechanisms.

4.2.2. The adaptation of the matrix of coprocesses to the structure of the system.

The definition of the graph of transmission of DCS is based on the expression  $\overline{WR}$ . Now this expression must be modified, such that the requirement of an isomorphism among graphs  $G_T$  and  $S$  will be satisfied. This modification will be discussed below.

Every the regular expression  $\overline{WR}$  is an element of the set  $REX(\Sigma^* \cup \hat{T})$ . Let, for  $1 \leq i_q \leq m$ ,  $1 \leq j_v \leq n$ ,  $\hat{T}$  be the alphabet:

$$\hat{T} = \left\{ t_{j_1 j_2}^{i_1 i_1}, t_{j_2 j_3}^{i_1 i_1}, \dots, t_{j_{n-1} j_n}^{i_1 i_1}, t_{j_1 j_1}^{i_1 i_2}, t_{j_1 j_2}^{i_1 i_2}, \dots, t_{j_1 j_2}^{i_{m-1} i_m}, t_{j_2 j_3}^{i_{m-1} i_m}, \dots, t_{j_v j_v}^{i_{q-1} i_q}, t_{j_1 j_1}^{i_q i_q}, \dots, t_{j_{v-1} j_v}^{i_q i_q}, t_{j_n j_n}^{i_m i_m} \right\}$$

such that:

$$(\forall t_{cd}^{ab} \in \hat{T}) \exists ((n_{num(a,c)}, n_{num(b,d)}) \in N_T \times N_T) \text{ and } (a,c), (b,d) \in [m] \times [n].$$

Let  $\omega$  be the set of any sequences:

$num(i, j), num(i_1, j_1), num(i_1, j_2), \dots, num(i_k, j_l), num(r, s)$  for  $(i, j), (r, s) \in [m] \times [n]$  such, that every pair:

$(num(i, j), num(i_1, j_1)), (num(i_k, j_l), num(r, s))$  and  $(num(i_p, j_h), num(i_{p+1}, j_{h+1}))$  for  $p=1, \dots, k-1, h=1, \dots, l-1$  - is an edge of the graph  $S$ .

The sequence  $num(i, j), num(i_1, j_1), \dots, num(i_k, j_l), num(r, s)$  can be interpreted as the path from  $num(i, j)$  to  $num(r, s)$ .

Define the mapping  $\varphi_\omega : \hat{T} \rightarrow (\hat{T})^*$  in the following way:

$$(\forall t_{js}^{ir} \in \hat{T}) \quad \varphi_\omega(t_{js}^{ir}) = t_{j j_1}^{i i} \dots t_{j_1 s}^{i_k r}$$



Let  $\Psi_\omega : (\Sigma \cup \hat{T}) \rightarrow (\Sigma \cup \hat{T})^*$  be the homomorphism defined as follows:

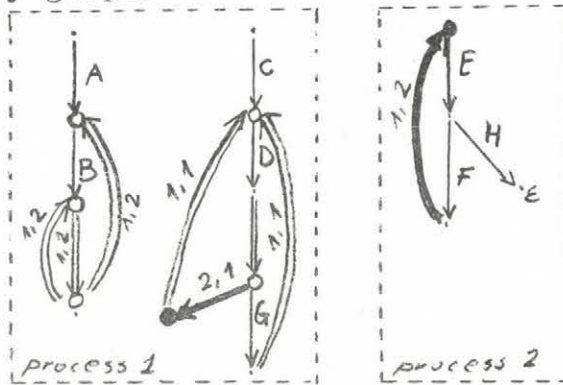
$$\Psi_\omega(a) = \begin{cases} \Psi_\omega(a) & \text{for } a \in \hat{T} \\ a & \text{elsewhere} \end{cases}$$

Now, the matrix of coprocesses should be build on the basis of the expression  $\Psi_\omega(\overline{WR}) = \overline{WR}$ , instead of the expression  $\overline{WR}$ . As a result we obtain a matrix of coprocesses, which satisfies isomorphism among  $G_T$  and  $S$ , requirement.

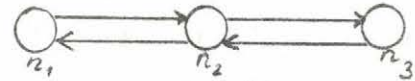
Example 4.2.2.1. For our an example system we can obtain:

$$\begin{aligned} \Psi(\overline{WR}) &= ABt_{12}^{11}CDt_{21}^{11}((t_{12}^{11}t_{21}^{12}Eft_{12}^{21}t_{21}^{11}vt_{12}^{11}Gt_{21}^{11}B)t_{12}^{11}Dt_{21}^{11})^*t_{12}^{11}t_{21}^{12}EH \text{ and} \\ \overline{WR}_{11} &= ABt_{12}^{11}((t_{12}^{11}vt_{12}^{11}B)t_{12}^{11})^*t_{12}^{11}, \\ \overline{WR}_{12} &= CDt_{21}^{11}((t_{21}^{12}t_{21}^{11}vt_{21}^{11}Gt_{21}^{11})Dt_{21}^{11})^*t_{21}^{12}, \\ \overline{WR}_{21} &= (Eft_{12}^{21}vt_{12}^{11})^*EH. \end{aligned}$$

The modified matrix of coprocesses can be graphically expressed by graphs:



The graph of transmission  $G_T$  has the form:



Both graphs  $G_T$  / see above fig./ and graph  $S$  / see fig.1/ are isomprhic.

4.3. The virtual graph of transmission in DCS.

Let's obtain so called the virtual graph of transmission in DCS. Varius properties of DCS can be expresses in terms of virtual graph of transmission / abbr. VGT /, which characterize all potential communications. For example, if VGT is a tree then DCS obviously a deadlock is impossible.

In order to obtain this graph we use the expression  $\overline{WR}$ . This expression represents all possible behaviors of the system / all sequences of both processing actions and communicating actions /.

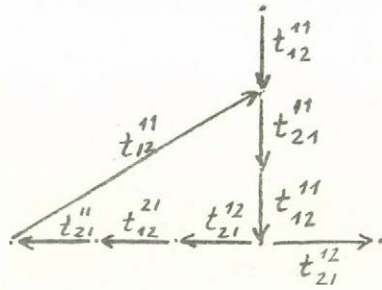
First, we define the regular expression which represent the virtual communication. This expression we denote by VC.

$$VC = h_\Sigma(\overline{WR})$$

Example 4.3.1. For  $WR$  defined in EX.4.2.2.1. VC has form:

$$VC = t_{12}^{11} t_{21}^{11} ((t_{12}^{11} t_{21}^{12} t_{12}^{21} t_{21}^{11} \vee t_{12}^{11} t_{21}^{11}) t_{12}^{11} t_{21}^{11})^* t_{12}^{11} t_{21}^{12}$$

This expression can be represented graphically by below graph:



Expressions described local graphs of communication /abbr. LGC/ - for particular coprocesses, we can obtain by following way:

$$\forall (i \in [m], j \in [n]) \quad LGC_{ij} = h_{\Sigma_{ij}}(\overline{WR}_{ij})$$

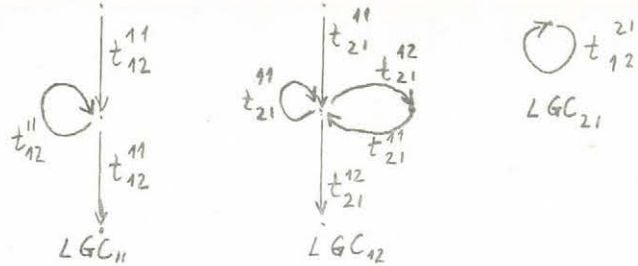
Example 4.3.2. For our an example system:

$$LGC_{11} = t_{12}^{11} (t_{12}^{11})^* t_{12}^{11}$$

$$LGC_{12} = t_{21}^{11} (t_{21}^{12} t_{21}^{11} \vee t_{21}^{11})^* t_{21}^{12}$$

$$LGC_{21} = (t_{12}^{21})^*$$

and graphically:



5. Final comment.

Treating distributed systems as the superposition of sequential subsystems is, not only to author's mind, the natural way of analysis and synthesis of those systems. This paper is an attempt to formal approach to this problem. Similar problems, but from a different point of view are considered in Hennesy, Plotkin [3], Francez, Hoare, Lehan, de Roever [2], Hoare [4].

References.

- [1] Conway M.E., Design of a separable transition-diagram compiler. Comm. of the ACM, vol.6,7,1963, pp.396-408.
- [2] Francez N., Hoare C.A.R., Lehmann D., de Roever W., Semantics of nondeterminism, concurrency, and communication. J. of comp. and System Sci., 19,1979, pp.290-308.
- [3] Hennesy M.C.B., Plotkin G.D., A term model for CCS. Lecture Notes in comp. Sci., vol. 88,1980, pp.262-274.
- [4] Hoare C.A.R., Communicating sequential processes. Comm. of the ACM, 21,1978, pp.666-677.

- [5] Janicki R., Vector of coroutines. Lecture Notes in Comp. Sci., vol.45, 1976, pp.377-384.
- [6] Janicki R., Results of the theory of vectors of coroutines. Fundamenta Informaticae, vol.2,3, 1979, pp.289-316.
- [7] Janicki R., Analysis of coroutines by means of vector of coroutines. ICS PAS REPORTS, 379, 1979 .
- [8] Just J.R., An algebreic model of the distributed computer system. 5-th Conf. on the Theory of Oper. Syst., Visegrad 79.
- [9] Just J.R., Analysis and synthesis of distributed computer systems by algebraic means. 6-th Conf. on the Theory of Oper. Syst., Visegrad 80.

Jan R. Just  
Warsaw Technical University  
Institute of Electronic Fundamentals  
ul. Nowowiejska 15/19 p. 230A  
00-665 Warsaw  
POLAND



ON CONCURRENT SYSTEMS AND CONCURRENCY RELATIONS

Ryszard Janicki

Poland

1. Introduction.

The notion of concurrency relation, introduced by Petri in a paper [7] for so called nets of occurrences, seems to be one of the basic notions of the concurrency theory. If we say that two objects are concurrent, in fact we define the concurrency relation for these objects. Properties of this relation were developed by Best [1], Petri [7,8] - on the process level, and by Janicki [3,4,5], Prószyński [9] - on the system level. This paper is a continuation of [3,4] and the complement of [5], although it can be read independently. In the paper we restrict our attention to nets decomposable into sequential finite state machines. This follows from two reasons. Firstly, the author is convinced that people think sequentially (cf. Brinch Hansen [2]), and the composition of concurrent systems from sequential ones is one of the natural methods of the construction (see Lauer and others [6]). Secondly, it was proved in a paper [5] that if a well defined marked net satisfied Petri's postulate that every sequential subsystem and every global system state had one common element, then this net could be decomposed into a set of sequential finite state machines.

2. Marked s-nets.

In this section we recall some necessary notions introduced in [3,4].

For every set  $X$ , let  $\text{left}: X \times X \rightarrow X$ ,  $\text{right}: X \times X \rightarrow X$  be the following functions:

$$(\forall (x,y) \in X \times X) \text{left}((x,y))=x, \text{right}((x,y))=y.$$

By a simple net (abbr. s-net) we mean any pair

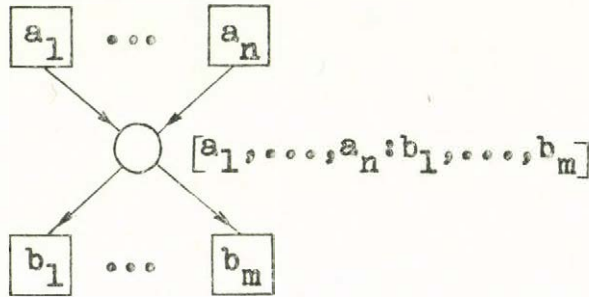
$$N = (T, P)$$

where:  $T$  is a set (of transitions),

$P \subseteq 2^T \times 2^T$  is a relation (interpreted as a set of places),

$$(\forall a \in T) (\exists p, q \in P) \quad a \in \text{left}(p) \cup \text{right}(q).$$

We shall only consider finite s-nets. Every s-net  $N=(T,P)$  can be graphically represented using the graph:



to denote the fact that  $(\{a_1, \dots, a_n\}, \{b_1, \dots, b_m\}) \in P$ .

This definition differs from the standard definition of Petri nets (cf. [7,8]). The approach presented here is luckier in the sense that it makes more easy to handle operation among nets (cf. [3,4]).

Let  $N_1=(T_1, P_1)$ ,  $N_2=(T_2, P_2)$  be s-nets. It can be proved (cf. [3]) that the pair  $(T_1 \cup T_2, P_1 \cup P_2)$  is also a s-net.

Thus we can define the following operation:

$$N_1 \cup N_2 = (T_1 \cup T_2, P_1 \cup P_2).$$

Let  $N=(T,P)$  be a s-net, and let  $F \subseteq T \times P \cup P \times T$  be the following relation:  $(\forall x, y \in T \cup P) (x, y) \in F \iff x \in \text{left}(y) \text{ or } y \in \text{right}(x)$ .

The relation  $F$  is called the flow-relation. Note that the triple  $(T, P, F)$  is a standard representation of the net  $N$  (see [7,8]).

For every  $x \in T \cup P$ , let  $\cdot x = \{y \mid yFx\}$ ,  $x^\circ = \{y \mid xFy\}$ .

A s-net  $N=(T,P)$  is said to be connected iff

$$(\forall x, y \in T \cup P) (x, y) \in (F \cup F^{-1})^*$$

Other words, a s-net is connected if its suitable graph is connected.

A s-net  $N=(T,P)$  is said to be quasi-elementary iff

$$(\forall a \in T) \quad \text{card}(\cdot a) = \text{card}(a^\circ) = 1.$$

A s-net  $N$  is said to be elementary iff it is quasi-elementary and connected.

Elementary nets are equivalent with totally labelled sequential finite state machines, and can be treated as a model of sequential systems.

For every s-net  $N=(T,P)$ , let:

$$\text{elem}(N) = \{N' \mid N'=(T',P') \text{ is an elementary s-net} \ \& \ T' \subseteq T \ \& \ P' \subseteq P\}.$$

We are interested in nets decomposable into a set of sequential finite state machines, called proper in this approach.

A s-net  $N$  is said to be proper iff  $N = \bigcup_{N' \in \text{elem}(N)} N'$ .

Note that  $N$  is proper if there is a set  $\{N_1, \dots, N_m\}$  of elementary s-nets and  $N=N_1 \cup \dots \cup N_m$  (see also [3,4]).

Let  $N=(T,P)$  be a s-net, and let  $R_1, CR_1 \subseteq 2^P \times 2^P$  be the following relations:

$$(M_1, M_2) \in R_1 \iff (\exists a \in T) M_1 - \cdot a = M_2 - a \cdot \ \& \ \cdot a \in M_1 \ \& \ a \cdot \in M_2, \text{ and}$$

$$(M_1, M_2) \in CR_1 \iff (\exists A \subseteq T) (\forall a, b \in A) (a \neq b \implies \cdot a \cap \cdot b = a \cdot \cap b \cdot = \emptyset) \ \& \\ M_1 - \bigcup_{a \in A} \cdot a = M_2 - \bigcup_{a \in A} a \cdot \ \& \ \bigcup_{a \in A} \cdot a \in M_1 \ \& \ \bigcup_{a \in A} a \cdot \in M_2.$$

The relation  $R_1$  is called the forward reachability in one step, and  $CR_1$  is called the concurrent forward reachability in one step. It can be proved that for finite s-nets:

$(R_1 \cup R_1^{-1})^* = (CR_1 \cup CR_1^{-1})^*$ . Let  $R = (R_1 \cup R_1^{-1})^*$ . This relation is called the forward and backward reachability of  $N$  (cf. [7,8]).

Note that  $R$  is an equivalence relation. For every  $M \in 2^P$ , let  $[M]_R$  denote the equivalence class of  $R$  containing  $M$ .

By a marked simple net (abbr. ms-net) we mean any triple

$$MN = (T, P, \text{Mar}),$$

where:  $N=(T,P)$  is a s-net,

$\text{Mar} \subseteq 2^P$  is a set of markings of  $MN$ ,

$$\text{Mar} = \bigcup_{M \in \text{Mar}} [M]_R.$$

A ms-net  $MN=(T,P,\text{Mar})$  is said to be compact iff

$$(\forall M \in \text{Mar}) \text{Mar} = [M]_R.$$

Note that Petri's condition/event systems (cf. [9]) are compact ms-net.

A transition  $a \in T$  is said to be fireable iff

$$(\exists M_1, M_2 \in \text{Mar}) \cdot a \in M_1 \ \& \ a \cdot \in M_2.$$

A ms-net  $MN=(T,P,Mar)$  is said to be safe iff:  
 $(\forall A \in 2^P)(\forall a \in T)$   
 $(a \cap A = \emptyset \ \& \ (\exists M \in Mar) \ a \cup A \subseteq M) \iff (a \cap A = \emptyset \ \& \ (\exists M' \in Mar) \ a \cup A \subseteq M')$ .  
 For more details the reader is advised to refer to [3,4].

### 3. Sir-relations.

Our approach is based on the notion of a symmetric and ir-reflexive relation defined by a fixed covering of a set. Elements of a covering will represent sequential components of a system.

Let  $X$  be a set.

A relation  $C \subseteq X \times X$  is said to be the sir-relation (from symmetric and irreflexive) iff:

$$(\forall a, b \in X) \ (a, b) \in C \iff (b, a) \in C \ \& \ (a, b) \in C \implies a \neq b.$$

Let  $C$  be a sir-relation,  $id = \{(x, x) \mid x \in X\}$ , and let  $kens(C)$ ,  $\overline{kens}(C)$  be the following families of subsets of  $X$ :

$$kens(C) = \{A \mid ((\forall a, b \in A) \ (a, b) \in C \cup id \ \& \ (\forall c \notin A) (\exists a \in A) \ (a, c) \notin C)\},$$

$$\overline{kens}(C) = \{A \mid ((\forall a, b \in A) \ (a, b) \notin C \ \& \ (\forall c \notin A) (\exists a \in A) \ (a, c) \in C)\}.$$

Note that  $kens(C)$ ,  $\overline{kens}(C)$  are coverings of  $X$ . From the viewpoint of the graph theory, the set  $kens(C)$  is the set of all cliques of the undirect graph representing  $C$ , while the set  $\overline{kens}(C)$  is the set of all cliques of the complement of that graph.

Let  $cov$  be a covering of  $X$ .

Let  $sir(cov) \subseteq X \times X$  be the relation defined as follows

$$(\forall a, b \in X) \ (a, b) \in sir(cov) \iff a \neq b \ \& \ (\forall A \in cov) \ a \notin A \ \text{or} \ b \notin A.$$

In this approach, a covering  $cov$  represents an arbitrary set of sequential system components, and the relation  $sir(cov)$  represents the concurrency structure defined by that set.

Let  $Mar \subseteq 2^X$  be a covering of  $X$  satisfying the following property  $Mar \subseteq kens(sir(cov))$ .

The family  $Mar$  represents the set of "global" system states (marking class). The pair  $D = (cov, Mar)$  will be called the double covering (abbr. d-covering) of  $X$ .

Thus, we have the following interpretations:

$sir(cov)$  - the concurrency relation,

$cov$  - the set of sequential system components,



$\text{Mar}$  - the set of all global system states,

$\overline{\text{kens}}(\text{sir}(\text{cov}))$  - the family of all maximal locally dependent sets, where by a locally dependent set we mean any set  $A$  such that for every two elements  $a, b \in A$ , the pair  $(a, b) \in \text{sir}(\text{cov})$ ,

$\text{kens}(\text{sir}(\text{cov}))$  - the family of all maximal locally concurrent sets, where by a locally concurrent set we mean any set  $A$  such that for every two different elements  $a, b \in A$ ,  $(a, b) \in \text{sir}(\text{cov})$ .

The family  $\overline{\text{kens}}(\text{sir}(\text{cov}))$  is a set of sequential system components only if  $\text{cov} = \overline{\text{kens}}(\text{sir}(\text{cov}))$ , and  $\text{kens}(\text{sir}(\text{cov}))$  is a set of global system states only if  $\text{Mar} = \text{kens}(\text{sir}(\text{cov}))$ .

A sir-relation  $\text{sir}(\text{cov})$  is said to be consistent iff  $\text{cov} = \overline{\text{kens}}(\text{sir}(\text{cov}))$ , and it is said to be semiconsistent iff  $\text{cov} \subseteq \overline{\text{kens}}(\text{sir}(\text{cov}))$ .

The property of consistency means that the concurrency relation describes precisely the set of sequential components, while the property of semiconsistency means only that every sequential component is defined by the concurrency relation (compare [5,10]). In fact, the above properties are rather properties of the covering  $\text{cov}$  than the relation  $\text{sir}(\text{cov})$ , because many coverings can define the same relation. Nevertheless, in further considerations the covering will usually be fixed, whereas speaking about consistency and semiconsistency as properties of the relation enable more uniform consideration. The same remark concerns notions of KM-, and CM-density introduced below.

Considering nets of occurrences, Petri [7] has postulated that for every real process, every sequential component and every "case" (global state) have one element in common. This is a generalization of the well known postulate of physics that every time sequence and every space must have one common element. Petri has called this property as K-density.

Although K-density is formally defined as a property of the concurrency relation (see [7,8]), in reality, as it was justly noticed by Best [1], it is a property of occurrence nets. The K-density is formally defined as follows:

A sir-relation  $C \subseteq X \times X$  is said to be K-dense iff

$$(\forall A \in \overline{\text{kens}}(C)) (\forall B \in \overline{\text{kens}}(C)) \quad A \cap B \neq \emptyset .$$

In the case of occurrence nets, the notion of K-density is adequate (cf. [1,7,9]), but in our approach it has a good interpretation only if  $\text{cov} = \overline{\text{kens}}(\text{sir}(\text{cov}))$  and  $\text{Mar} = \text{kens}(\text{sir}(\text{cov}))$ .

Therefore, we have to replace one by more adequate notions.

Let  $D = (\text{cov}, \text{Mar})$  be a d-covering of X.

A sir-relation  $\text{sir}(\text{cov}) \subseteq X \times X$  is said to be KM-dense iff

$$(\forall A \in \text{Mar})(\forall B \in \overline{\text{kens}}(\text{sir}(\text{cov}))) A \cap B \neq \emptyset.$$

A sir-relation  $\text{sir}(\text{cov}) \subseteq X \times X$  is said to be CM-dense iff

$$(\forall A \in \text{Mar})(\forall B \in \text{cov}) A \cap B \neq \emptyset.$$

If  $\text{Mar} = \text{kens}(\text{sir}(\text{cov}))$  then KM-density is equivalent to K-density, and CM-density is equivalent to so called C-density developed in [9]. In the approach presented, CM-density describes Petri's postulate on a common element. It will be proved that CM-density is a strong property. KM-density has no such a good interpretation, although it is also a strong property.

#### Corollary 1.

1.  $\text{cov} = \overline{\text{kens}}(\text{sir}(\text{cov})) \Rightarrow (\text{KM-density} \Leftrightarrow \text{CM-density}),$
2.  $\text{cov} \subseteq \overline{\text{kens}}(\text{sir}(\text{cov})) \Rightarrow (\text{KM-density} \Rightarrow \text{CM-density}). \blacksquare$

#### Theorem 2.

If Mar is a covering of X and  $\text{sir}(\text{cov})$  is CM-dense, then  $\text{cov} \subseteq \overline{\text{kens}}(\text{sir}(\text{cov})). \blacksquare$

Other words, if Mar covers X then CM-density of  $\text{sir}(\text{cov})$  implies its semiconsistency.

Now we are going to come back to marked s-nets.

#### 4. Seminaturally marked s-nets.

In this section we shall deal with relationship between static net structure (i.e. the pair  $(T, P)$ ), and the properties of marking class (i.e. the set Mar). Results of this section are generalizations of those from [3,4].

Let  $N = (T, P)$  be a proper s-net, and let

$C = \{N_1, \dots, N_m\} \subseteq \text{elem}(N)$  be a set of elementary nets such that:  $N = N_1 \cup \dots \cup N_m$ .

Assume that  $N_i = (T_i, P_i)$  for  $i=1, \dots, m$ .

Every set C of the above form is said to be an elementary covering of N (abbr. e-covering).

Let  $\text{cov}_C = \{P_1, \dots, P_m\} \subseteq 2^P$ . Note that  $\text{cov}_C$  is a covering of  $C$ .

Let  $\text{coex}_C \subseteq P \times P$  be the following relation

$$\text{coex}_C = \text{sir}(\text{cov}_C).$$

Other words:  $(a, b) \in \text{coex}_C \iff a \neq b \ \& \ (\forall P_i \in \text{cov}_C) \ a \notin P_i \ \text{or} \ b \notin P_i$ .

The relation  $\text{coex}_C$  is said to be the coexistency defined by the e-covering  $C$  of  $N$ .

It turns out that the triple  $(T, P, \text{Mar})$ , where  $\text{Mar} = \text{kens}(\text{coex}_C)$ , is a ms-net with regular properties. Properties of such ms-nets were developed in a paper [4]. The case when  $\text{Mar} = \text{kens}(\text{coex}_C)$  and  $C = \text{elem}(N)$  was considered in [3, 9].

A ms-net  $MN = (T, P, \text{Mar})$  is called seminaturally marked with respect to a set of elementary nets  $C$  iff:

1.  $C$  is an e-covering of  $N = (T, P)$ ,
2.  $\text{Mar} = \text{kens}(\text{coex}_C)$ ,
3. every element of  $T$  is fireable.

Now we are going to characterize seminaturally marked s-nets.

Let  $MN = (T, P, \text{Mar})$  be a fixed seminaturally marked s-net with respect to the set  $C$ . Let also  $N = (T, P)$ .

Theorem 3.

$MN$  is safe. ■

Note that the notions: KM-density and CM-density can be defined in terms of this section. Namely, the relation  $\text{coex}_C$  is KM-dense iff  $(\forall A \in \text{Mar})(\forall B \in \overline{\text{kens}(\text{coex}_C)}) \ A \cap B \neq \emptyset$ , and  $\text{coex}_C$  is CM-dense iff  $(\forall A \in \text{Mar})(\forall B \in \text{cov}_C) \ A \cap B \neq \emptyset$ .

Corollary 4.

$\text{coex}_C$  is CM-dense  $\implies \text{cov}_C \subseteq \overline{\text{kens}(\text{coex}_C)}$ . ■

Let QEL denote the family of all quasidelementary s-nets.

Theorem 5.

If  $\text{coex}_C$  is KM-dense then:  $(\forall A \in \overline{\text{kens}(\text{coex}_C)}) \ N_A = (^*A \cup A^*, A) \in \text{QEL}$ . ■

Of course, by the construction we have that every element of  $\text{cov}_C$  describes an elementary s-net, but we do not know anything about elements of  $\overline{\text{kens}(\text{coex}_C)}$ . Note that, in general, we do not assume the property  $\text{cov}_C \subseteq \overline{\text{kens}(\text{coex}_C)}$ .

The above theorem means that if  $\text{coex}_C$  is KM-dense then every element of  $\overline{\text{kens}(\text{coex}_C)}$ , i.e. every maximal locally dependent

set, creates a sequential finite state machine (not necessarily connected). It can be proved that the symbol QEL cannot be replaced by  $\text{elem}(N)$ .

Compactness is the property, which is frequently required from concurrent systems. For example, Petri has assumed that every condition/event system is compact (see [8]). For compact seminaturally marked s-net, we can formulate the following theorems.

Theorem 6.

$MN$  is compact  $\Rightarrow \text{coex}_C$  is CM-dense. ■

Thus, if seminaturally marked s-net is compact then every sequential subsystem and every global state have one common element.

Corollary 7.

$MN$  is compact  $\Rightarrow \text{cov}_C \subseteq \overline{\text{kens}(\text{coex}_C)}$ . ■

This means that in that case, every sequential subsystem can be described as a clique of the relation  $\text{coex}_C$ .

Corollary 8.

$MN$  is compact and  $\text{cov}_C = \overline{\text{kens}(\text{coex}_C)}$   $\Rightarrow \text{coex}_C$  is KM-dense. ■

It turns out that for compact nets the result of Theorem 5 can be strengthened.

Theorem 9.

If  $MN$  is compact and  $\text{coex}_C$  is KM-dense then:

$$(\forall A \in \overline{\text{kens}(\text{coex}_C)}) N_A = (^*A \cup A^*, A) \in \text{elem}(N). \blacksquare$$

Of course, if  $MN$  is compact then  $\text{cov}_C = \overline{\text{kens}(\text{coex}_C)}$ , and every element of  $\text{cov}_C$  generates - by the definition - an elementary s-net. From Theorem 9 it follows that elements of  $\overline{\text{kens}(\text{coex}_C)}$  generate also elementary nets.

In the case of  $C = \text{elem}(N)$ , we can replace Corollary 8 by the following theorem.

Theorem 10.

Let  $C = \text{elem}(N)$ . Then:

$MN$  is compact  $\Rightarrow (\text{coex}_C \text{ is KM-dense} \Leftrightarrow \text{cov}_C = \overline{\text{kens}(\text{coex}_C)})$ . ■

As it was pointed out, the above results are a generalization of theorems and lemmas from [3,4,9]. Under the assumption  $\text{Mar}=\text{kens}(\text{coex}_C)$  we obtain results from [4], under the assumption  $\text{Mar}=\text{kens}(\text{coex}_C) \ \& \ C=\text{elem}(N)$  we obtain results from [3,9].

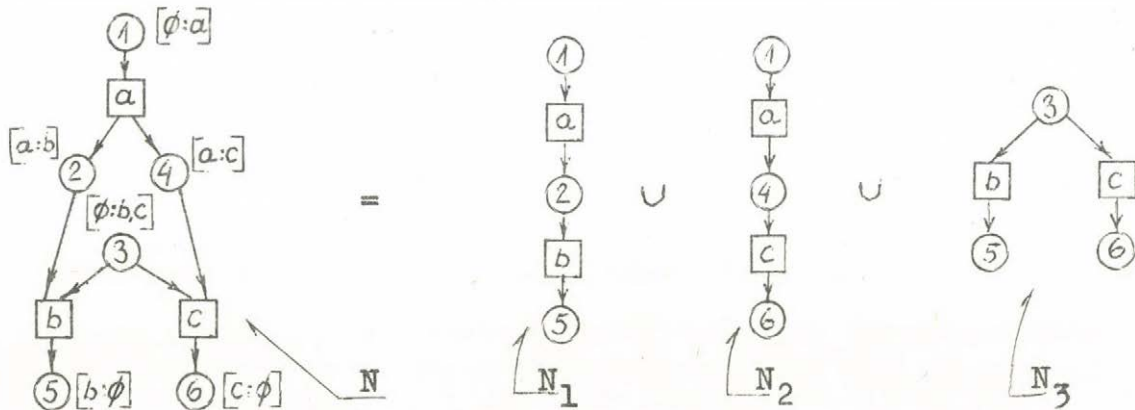
Seminaturally marked s-nets seem to be very interesting class of marked s-nets. On the one hand this class is large (for instance it contains the class of nets generated by  $\text{GE}^*$ -paths [6]), on the other it has convenient properties, since every seminaturally marked s-net is composed from a set of sequential finite state machines and its marking class is strictly connected with this composition. Furthermore, from the paper [5] it follows that if a compact marked net satisfies the mentioned above Petri's postulate on a common element, then this net can be treated as seminaturally marked.

We are now going to consider some examples, which illustrate the above notions and results.

Let  $\overline{\text{coex}}_C = (\text{P P-coex}_C) - \text{id}$ .

Example 1.

Let  $N=(T,P)$ ,  $N_i=(T_i,P_i)$  for  $i=1,2,3$  be the following s-nets.



Note that  $N = N_1 \cup N_2 \cup N_3$  and  $\text{elem}(N) = \{N_1, N_2, N_3\}$ .

Let  $C = \{N_1, N_2, N_3\}$ . Of course,  $C$  is the e-covering of  $N$ . The graphs of  $\text{coex}_C$  and  $\overline{\text{coex}}_C$  are of the following form.



Note that:  $\text{kens}(\text{coex}_C) = \{\{1,3\}, \{2,3,4\}, \{2,6\}, \{4,5\}\}$ ,  
 $\overline{\text{kens}(\text{coex}_C)} = \{\{1,2,5\}, \{1,4,6\}, \{3,5,6\}, \{1,5,6\}\}$ ,  
 $\text{cov}_C = \{\{1,2,5\}, \{1,4,6\}, \{3,5,6\}\}$ ,  
 thus  $\text{cov}_C \overline{\text{kens}(\text{coex}_C)} = \text{cov}_C \cup \{1,5,6\}$ .

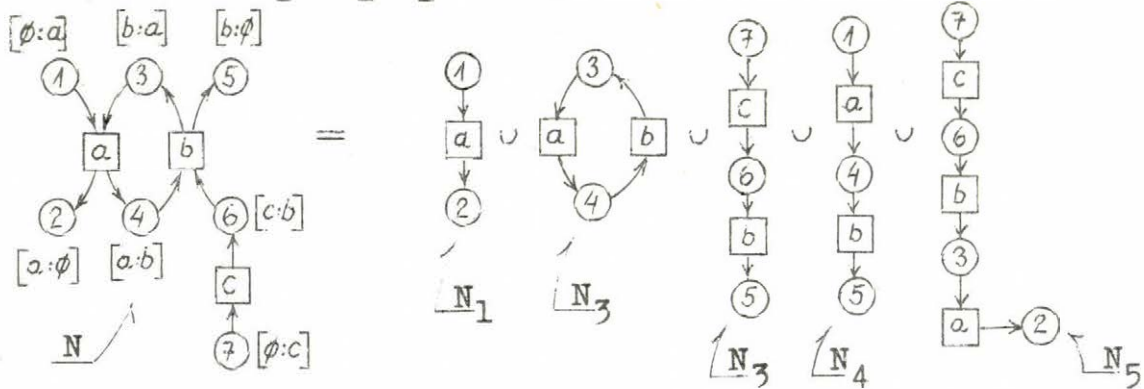
Let us define  $\text{Mar} = \text{kens}(\text{coex}_C)$ .

Note that  $\text{MN} = (\text{T}, \text{P}, \text{Mar})$  is a seminaturally marked s-net with respect to the set  $C = \{N_1, N_2, N_3\}$ . The ms-net MN is safe and compact, and every element of T is fireable. The relation  $\text{coex}_C$  is CM-dense, but it is not KM-dense, because  $\{1,5,6\} \cap \{2,3,4\} = \emptyset$ . The statement  $(\forall A \in \overline{\text{kens}(\text{coex}_C)}) N_A \in \text{QEL}$ , where  $N_A = (^*A \cup A^*, A)$ , is not true, because the set  $\{1,5,6\} \in \overline{\text{kens}(\text{coex}_C)}$  does not define any s-net.

Since  $\text{Mar} = \text{kens}(\text{coex}_C)$  then KM-density is equivalent to K-density and CM-density is equivalent to C-density (cf. [9]).

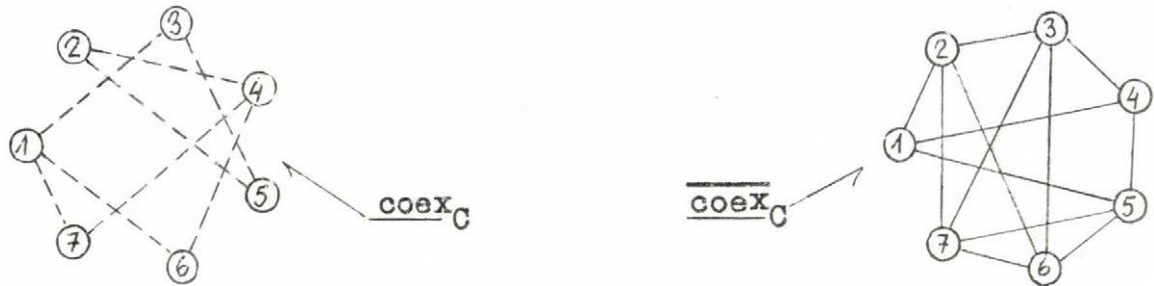
Example 2.

Let  $N = (\text{T}, \text{P})$ ,  $N_i = (\text{T}_i, \text{P}_i)$  for  $i=1, \dots, 5$  be the following s-nets.



Note that  $N = N_1 \cup \dots \cup N_5$ , and  $\text{elem}(N) = \{N_1, \dots, N_5\}$ .

Let  $C = \{N_1, \dots, N_5\}$ . The family C is obviously an e-covering of N. The graphs of  $\text{coex}_C$  and  $\overline{\text{coex}_C}$  are the following.



Here we have:  $\text{kens}(\text{coex}_C) = \{\{1,3\}, \{1,6\}, \{1,7\}, \{2,4\}, \{2,5\}, \{3,5\}, \{4,6\}, \{4,7\}\}$ ,

$$\overline{\text{kens}}(\text{coex}_C) = \{\{1,2\}, \{3,4\}, \{5,6,7\}, \{1,4,5\}, \{2,3,6,7\}\},$$

$$\text{cov}_C = \overline{\text{kens}}(\text{coex}_C).$$

Let  $\text{Mar} = \{\{1,3\}, \{2,4\}, \{3,5\}, \{4,6\}, \{4,7\}\} \not\subseteq \text{kens}(\text{coex}_C)$ .

Note that  $\text{MN}=(\text{T},\text{P},\text{Mar})$  is a seminaturally marked s-net with respect to the set  $C=\{N_1, \dots, N_5\}$ . The ms-net is safe, but not compact, and every element of  $\text{T}$  is fireable.

Since  $\text{cov}_C = \overline{\text{kens}}(\text{coex}_C)$  then KM-density is equivalent with CM-density, but the relation  $\text{coex}_C$  is not CM-dense, because for example  $\{1,3\} \cap \{5,6,7\} = \emptyset$ . Note that every element of  $\overline{\text{kens}}(\text{coex}_C)$  defines an elementary s-net (because  $\text{cov}_C = \overline{\text{kens}}(\text{coex}_C)$ ), although  $\text{coex}_C$  is not KM-dense.

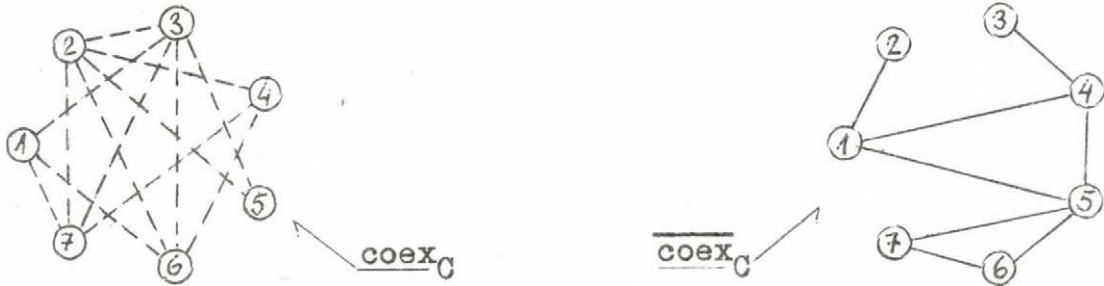
Example 3.

Let  $N=(\text{T},\text{P})$ ,  $N_i=(\text{T}_i,\text{P}_i)$  for  $i=1, \dots, 5$  be the same s-nets as in Example 2.

Let  $C = \{N_1, N_2, N_3, N_4\} \not\subseteq \text{elem}(N) = C \cup \{N_5\}$ .

Of course  $N = N_1 \cup N_2 \cup N_3 \cup N_4$ , so  $C$  is an e-covering of  $N$ .

Here, the relations  $\text{coex}_C$  and  $\overline{\text{coex}}_C$  are of the below form.



In this case:

$$\text{kens}(\text{coex}_C) = \{\{1,3,7\}, \{1,3,6\}, \{2,4,6\}, \{2,4,7\}, \{2,3,5\}, \{2,3,6\}, \{2,3,7\}\},$$

$$\overline{\text{kens}}(\text{coex}_C) = \{\{1,2\}, \{3,4\}, \{5,6,7\}, \{1,4,5\}\},$$

$$\text{cov}_C = \overline{\text{kens}}(\text{coex}_C).$$

Let  $\text{Mar} = \{\{1,3,7\}, \{1,3,6\}, \{2,4,6\}, \{2,4,7\}, \{2,3,5\}\} \not\subseteq \text{kens}(\text{coex}_C)$ .

Note that  $\text{MN}=(\text{T},\text{P},\text{Mar})$  is a seminaturally marked s-net with respect to the set  $C=\{N_1, N_2, N_3, N_4\}$ . The ms-net  $\text{MN}$  is safe, compact, and every element of  $\text{T}$  is fireable. The relation  $\text{coex}_C$  is CM-dense and KM-dense.

References.

Abbreviations:

LNCS - Lecture Notes in Computer Science,

GMD - Gesellschaft für Mathematik und Datenverarbeitung.

- [1] E. Best, The relative strenght of K-density, LNCS 84, Springer 1980, 261-276.
- [2] P. Brinch Hansen, Operating Systems Principles, Prentice Hall, Inc., New Yersey, 1973.
- [3] R. Janicki, An algebraic structure of Petri nets, LNCS 83, Springer 1980, 177-192.
- [4] R. Janicki, On atomic nets and concurrency relations, LNCS 88, Springer 1980, 320-333.
- [5] R. Janicki, Analysis of concurrent schemes by means of concurrency relations, Proc. of the AFCET Symposium on "Mathematics for Computer Science", Paris, 1982, to appear.
- [6] P.E. Lauer, M.W. Shields, J.Y.Cotronis, Formal behavioural specification of concurrent systems without globality assumptions, LNCS 107, Springer 1981, 115-151.
- [7] C.A. Petri, Non-sequential processes, ISF Report 70-01, GMD, Bonn 1977.
- [8] C.A. Petri, Concurrency, LNCS 84, Springer 1980, 251-260.
- [9] P. Prószyński, Petri nets and concurrency-like relations, LNCS 107, Springer 1981, 471-478.

---

Institute of Mathematics  
Warsaw Technical University  
Pl. Jedności Robotniczej 1  
00-661 Warszawa / Poland



AN OVERVIEW OF SYSTEMS MODELING

AND EVALUATION TENDENCIES

Geneviève JOMIER

I . INTRODUCTION

During the last ten years a lot of papers about modeling and performance evaluation of computer systems and computer networks have been published. Now new distributed systems are being built, integrating computers and communications. This involves new modeling and evaluation problems. The goal of this paper is to present the main developments and trends in this domain.

II. MODELING AND PERFORMANCE EVALUATION OF COMPUTER SYSTEMS

On Figure 1 we show the different steps needed to evaluate a system, and two ways to procede.

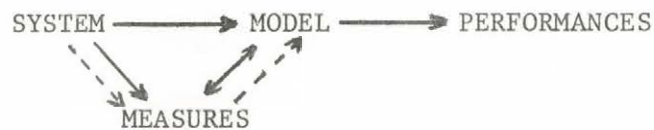


Fig.1

In the first one (solid arrows) a model is deduced from the system. The model is used to determine parameters which are measured or estimated from the system. There the measures are integrated in the model to obtain performances which are supposed to be representative of the system. An illustration of this procedure is the modelling of computer networks or computer systems (or part of them as central [Cour 77, Jomi 81] or secondary memory [Ge Mi 80, Bran 81, Arti 81] management etc...) using queues

and queueing networks [Klei 76] scheduling models [Coff 76] or Petri nets [Zube 81, Br FN 82].

In the second case (dashed arrows) empirical models are deduced from measures [Ferr 78, Svob 76, Fe Sp 80], using statistical techniques like regression, multidimensional analysis [Rala 79] chronological series, etc... They are mainly used in the study of systems workload.

In this part we will concentrate on computer systems modeling using queueing networks because this approach has been very successful and has produced a lot of important results. We begin by the fundamental theorem of Basket Chandy Muntz and Palacios [BCMP 75] which characterizes queueing networks with "product form solution". This means that these networks of queues with  $n$  nodes possess a steady state probability distribution  $P_s$  of the network state  $s$  of the form :

$$P_s = d \cdot f(s) \cdot \prod_{i=1}^n P_{s(i)} \quad (1)$$

where  $d$  is a normalizing constant,  $P_{s(i)}$  is the steady state probability of the corresponding state  $s(i)$  of the queueing system in the node  $i$ , and  $f(s)$  is a function of the number of customers depending on the state  $s$  [Ja Ko 80].

Two complementary approaches are developed to extend this type of solution to other networks. In the first one it is shown that some particular queueing networks admit product form solution. A.Hordijk and N.Van Dijk use it, for instance in [Ho VD 81] for certain cases of exponential queueing networks with blocking. We find it too in [Ja Ko 80] where U.Jansen and D.Konig use insensitivity properties to characterize an important family of open, closed or mixed networks admitting product form steady state probabilities. These results are based on the complementary approach in which more powerful new mathematical tools are developed. The outstanding works in this area have been made by Kelly [Kell 79] on reversibility and quasi-reversibility, and by Schassberger [Scha 77, He & 79] on insensitivity, connected with the last developments on point processes theory. A synthesis of this theory based on Palm's measure is presented, in [FKAS 79] by Franken, Köning, Arndt and Schmidt.

Due to the large number of states in the system, the computation of the normalization constant  $d$  in (1) may be untractable for real networks. As a result some computational algorithms have been presented by Chandy

and Sauer [ChSa 80] and by Bruell and Balbo [BrBa 80]. Other algorithms, possibly approximate, for large networks have been proposed by Mackenna and Mitra [MKMi 81] and Lavenberg [Lave 80]. Approaching it another way, for product form solution networks it is possible to directly obtain some parameters, thereby avoiding the normalization constant computation by the use of "mean value analysis" [ReLa 80].

The product form solutions [Pujo 80] are connected with an idea of "independence" between the different queues. This does not happen in some cases of computer systems or computer networks modelling which involve dependencies between queues. The exact analytical solution for some of these problems has been established, for instance, when it is possible to come to bidimensional markovian processus models [FaKM 80] and for a particular case of two coupled queues networks [Fayo 79].

However, at the present, in most cases the practical solution of such systems may be studied using :

- 1) numerical techniques [Stew 79, KiMi 80]
- 2) approximations of the model by decomposition and equivalence [Bran 80] or by decomposability-aggregation [Cour 77, VaGL 80]
- 3) or approximate solutions of the model by diffusion method [Koba 78], or isolation [LaPu 80], or by iterative techniques [DoAS 81, Mari 78].

Different methods may be used simultaneously.

Some packages providing the facility of describing and solving (with exact or approximate methods) queueing networks have been developed, such as QNAP at INRIA [PoVe 79], QMOD [Gron 81], RESQ and QNET4 at IBM [ReSa 78, SaMS 80]. Among other solution techniques QNAP and RESQ offer the possibility of obtaining results using simulation.

To satisfy the needs of performance evaluation, important improvements in simulation [Lero 80, BaSa 81] have occurred. The latest deal with

- 1) the simulation inputs : how to build random numbers generators, and how to generate correlated number sequences [Bade 79]
- 2) the analysis of the simulator outputs : a lot of papers have been published on the regenerative method [IgSh 80, Igle 78, LaMS 79] and on the confidence interval accuracy [HeWe 81] .

### III. MODELING AND PERFORMANCE EVALUATION OF COMPUTER NETWORKS

The same tools are often used in computer systems and computer networks modeling, so it is difficult to draw a clear (and artificial) boundary between them.

The use of queueing networks to model computer networks is widespread. The importance of priorities, blocking (e.g. due to the limited size of buffers), the possible packet desequencing, etc., often need the use of approximate solutions or simulation, they can be used only when the number of system states is rather small. As a result other modeling techniques are used [TGPM 78] such as the stochastic processes theory (renewal theory, Markov chains, semi-markov processes, regenerative processes) and the markovian theory of decision.

M. Reiser, in a very interesting report [Reis 81], classifies the performance evaluation studies of data communication systems into four categories, (the first one being the most numerous) :

- 1) evaluation of a given protocol
- 2) design and configuration of real networks
- 3) performance evaluation of "products" (packets) of communication networks
- 4) performance evaluation of real networks based on their "products" and on workload measures.

It appears that the recent improvements in the performance evaluation of general networks have occurred in the modeling phase (transition from model to the expression of performance). The most important aspect in this approach has been the structuring of protocols into layers (7 for ISO) and their normalization [SRWG 80, ISO, PoZi 78, ZiPo 81]. Such structuring has been very useful in understanding their functioning and, as a result, in modeling them. Thus there are now results on the performance evaluation of different level protocols, such as HDLC (layer 2) [Sere 81, LaPu 79] or of a set of layers such as X.25 (CCITT) [GiJM 81] which integrates the three lowest layers. A special attention must be given to the interrelationship between the different level protocols [BuSc 81].

For the local networks the normalization is in progress, and the situation is characterized by a very wide variety of supports (and, as a consequence of theoretical throughput), topologies and access protocols.

A taxonomy and comparison of random access protocols for computer networks have been proposed in [Mi Na 81]. Fixed and dynamic schemes are distinguished, and for the dynamic one they are separated in centralized, centralized polling, contention networks and decentralized. A new distinction is made in the decentralized dynamic assignation schemes between the random access (different types of ALQHA and CSMA) and the non-random access (decentralized reservation, polling, round robin, alternating priorities, random order, minislotted). Many papers have been published on that subject : references and protocol comparisons may be found in [Reis 81, Mina 81, Bux 81], and studies on particular protocols in [To Hu 80, Ge Mi 81, Span 81].

Some special topics of networks gave rise to studies, such as the messages resequencing , a synthesis of which is in [Ba GP 81]. Yet, in other domains the emphasis is placed on faisability more than on performance evaluation : the network interconnection [ISCA 80, ISCA 81, PWIN 80, FaMi 81] is an example of situation where very few papers appear on performance evaluation [Bern81] despite a real need. New performance evaluation problems arise with the use of networks to transport not only data but voice or pictures, in applications like burotics (office automation) or telematics. These uses involve different constraints in quality, speed and volume of tranfered data.

#### IV . DISTRIBUTED SYSTEMS :

The evolution of technology particularly the miniaturization (microprocessors) and the communications development (buses, local networks), and the fall of hardware prices involves the development of distributed systems. Beyond the versatility of such systems the idea is to use some small cooperating machines to perform tasks formerly devoted to large centralized systems [QED 78].

Therefore new systems oriented toward applications (like office automation, robotics, computer assisted instruction) are created. They are completely different form the universal centralized systems of the preceding generation.

These new distributed systems are sets of processors, specialized (like data bases machines) or universal, tightly coupled by buses or loosely

coupled by network (especially local network). In an application it is possible to distribute the computation and / or the data, and / or the control. For each of these cases a great variety of choices is possible in distributing and in managing the distribution.

The diversity of distribution choices is superimposed on the diversity of applications. The performance evaluation must take into account these two aspects : thus in a distributed system every site requires the application software and the modules necessary to manage the communications and the distribution. The different parts of the software are in conflict for access to some resources of the system (memory and computation time). This has an impact on performance, particularly when synchronization between processes involves forced idleness of some processors.

As a result the studies on modeling and performance evaluation evolve along two axes : the evaluation of specific applications and the evaluation of the distribution.

The evaluation of applications poses the problem of the generality of the studied applications. Also numerous papers published on this subject are devoted to data base management systems (DBMS) because they are widespread and increasingly used in the heart of new systems [DWHa 81, Tsic 81]. They take an interest in the DBMS as a whole [Sevc 81, LoMa 81, HeWY 81] or in some specific point such as the concurrency control [RiSt 77 ShSp 81, Ries 81, CHGM 81, PoLe 80], the access paths to data [AsKS 80], and, when the relational model is used, the size of operations results [GeGa 82, Rich 80] and the query optimization [Kim 81] etc. The large variety of types of DBMS is an obstacle their modeling.

The quantitative evaluation of distributed systems is limited by lack of tools to model the synchronization. However evaluation studies are published on tightly connected architectures [Pate 81, Balk 80, Gele 80, GnPa 80] and on loosely connected ones [BaFl 80] with a particular interest for distributed data bases [CoGP 80, Garc 79, Wilm 79]. Those papers are mainly based on theoretical algorithms to manage distributed systems and not on existing systems. Thus the count of messages necessary for the correct execution of a two phase commit in a distributed data base is interesting to compare different algorithms for maintaining concurrency [Garc 79, Wilm 79] but it is clearly insufficient to determine the intrinsic performance of one particular algorithm. So a lot of work is to be

done to obtain a clear idea of distributed system performance. This will be possible with the implementation of systems and the development of experimental concrete models [BCEJK81] which will point out the crucial performance problems by measures.

#### V. CONCLUSION

In the course of this study we have seen how the theoreticians began with the modeling of computer systems and computer networks using queueing networks, and how they were obliged to improve more and more their mathematical tools. Simultaneously the evolution of technology and the creation of systems of increasing complexity, integrating processors and communications, raised new problems necessitating the development of new modeling tools. Many problem problems are still open in these different areas.

BIBLIOGRAPHIE

- [ACMS 81] ACM/SIGMETRICS  
Conference on Measurement and Modelings of computer Systems -  
Sept. 1981 - Perf. Eval Review Vol 10, n° 3, Fall 1981 ,  
pp 175-215.
- [AdBD 81] G.ADORNI, A.BOCCALATTE, M.DIMANZO  
"Evaluation of Scheduling algorithms in the multiprocessor  
environment"  
Computer Performance Vol 2 n° 2 Juin 81 pp 70-76
- [Arti 81] H.P. ARTIS  
"Predicting the behavior of secondary storage management systems  
for IBM Computer Systems"  
Performance 81 - Amsterdam Nov 81 pp 435-444.
- [AsKS 80] M.M. ASTRAHAN, W.KIM, M.SCHKOLNICK  
"Evaluation of the System access path selection mechanism"  
IBM Research Laboratory, San Jose Californie 95 193 -  
RJ 2797 - 4/10/80 - 18 pages.
- [BaFl 80] F.BACCELLI et Th. FLEURY  
"On parsing in a multiprocessing environment"  
Rapport INRIA - 78153. Le Chesnay France, 1980 - To appear in  
Acta Informatica.
- [BaGP 81] F.BACCELLI, E.GELENBE, B.PLATEAU  
"An End to End Approach to the resequencing problem"  
Rapport de Recherche INRIA n° 97 - Nov 1981 - To appear in  
JACM.
- [Bade 79] M.BADEL  
"Generation de nombres aléatoires correlés"  
Mathematics and Computer in Simulation (IMACS) Vol XX, n° 4,  
1979.
- [BaWS 80] E.E. BALKOVICH, G.WHITBY-STREVEWS  
"On the Performance of decentralized software"  
Performance 80\* - Toronto - May 80 pp 175-180.
- [BaSa 81] Y.BARD and C.H. SAUER  
"IBM contributions to Computer Performance Modelling"  
IBM J. Res. Develop. Vol 25 . n°5 Sept 1981 p 562-570.



- [BCMP 75] F.BASKET, K.MANI CHANDY, R.R. MUNTZ, F.G. PALACIOS  
"Open, Closed, and Mixed Networks of Queues with Different classes of customers"  
JACM Vol 22, n° 2, April 75, pp 248-260.
- [Bern 81] G.BERNARD  
"Interconnection of Local Computer Networks : Modeling and optimization problems"  
Submitted to IEEE transactions on Software Engineering  
Rapport de Recherche - LRI, Université Paris Sud - n° 77 91405 - Orsay France
- [BCFJK 81] P. BOUCHET, A. CHESNAIS, J.M. FEUVRE, G. JOMIER, A. KURINCKX  
"PEPIN : An experimental multimicrocomputer database management system"  
2nd International Conference on Distributed Computing Systems  
IEEE/IFIP/AFCEC - Paris April 1981 pp 211-217.
- [Bran 80] A. BRANDWAJN  
"Further results on equivalence and decomposition in queueing network models"  
Performance 80\*, ACM Symetrics Vol 9, 2, Summer 80, p 93-104.
- [Bran 81] A. BRANDWAJN  
"Multiple paths versus memory for improving DASD subsystem performance"  
Performance 81 - Amsterdam Nov 81 - pp 415-434.
- [BrFN 82] G. BREGER, G. FLORIN, S. NATKIN  
"Un outil d'aide à l'évaluation de la sûreté et des performances de systèmes informatiques"  
Colloque AFCEC "Mathematics for Computer Science"  
Mars 1982 - CNAM - Paris France
- [BrBa 80] S.C. BRUELL and G. BALBO  
"Computational Algorithms for Closed Queueing Networks"  
North Holland, NY 1980.
- [BuSc 81] A. BUTRIMENKO, G. SCOLLO  
"Protocol parameters and network characteristics : classification and some interrelations"  
Proc. of the meeting "Evolution of Computer Networks Theory and Experience" Ed. Petrentko Sept. 1981 -  
IIASA - Laxenburg Austria. 1e Dec 1979.
- [Bux 81] W. BUX  
"Local Area Subnetworks : a Performance Comparison Research Report RZ 1057, 2/24/1981.  
IBM Zurich Research Laboratory, CH-8803 Ruschlikon Switzerland  
Or Proc fo the IFIP WG 6.4 International Workshop on local Area Networks Zurich - 27.29 Aug 1980, to be published by North-Holland, Amsterdam 1981.

- [ChSa 80] K.M. CHANDY and Ch.H SAUER  
"Computational Algorithms for Product Form Queueing Networks"  
Toronto May 1980 Performance 80\*  
Performance Evaluation review  
Vol 9 n°2 - pp 1 - 10.
- [ChGM 81] A.CHESNAIS, E.GELENBE, I.MITRANI  
"On the Modeling of concurrent access to shared data"  
Applied Probability / Computer Science : the Interface  
Boca Raton, Florida , Jan 1981.
- [Coff 76] E.G. COFFMAN  
"Computer and Job-shop Scheduling Theory"  
John Wiley 1970.
- [CoGP 80] E.G. COFFMAN, E.GELENBE, B.PLATEAU  
"Optimization of the number of copies in a Distributed Data  
base System"  
Performance 80\* - pp 257-264.
- [Cour 77] P.J. COURTOIS  
"Decomposability : queueing and computer system applications"  
Academic Press - New York 1977.
- [Deno 81] L.A. DENOIA  
"An Approach to the Cost / Performance comparison of distribu-  
ted Systems"  
Proc - 5th Berkeley Workshop on Distributed Data Management  
and Computer Networks.  
University of California Berkeley Feb 3-5, 1981 - pp 38-66.
- [DWHa 81] D.J. De WITT, P.B. HAWTHORN  
"A Performance Evaluation of Data base Machine Architecture"  
Proc 7 the Very Large Data Bases - Cannes France 9.11 Sept. 81  
Ed Zaniolo Delobel - pp 199-213.
- [DoAS 81] S.L. DODD, D.F. Mc ALLISTER, W.J. STEWART  
"An Iterative Method for the Exact Solution of Coxian Queueing  
networks"  
Performance Evaluation review Vol 10 n°3 Fall 1981 - p 97-  
104.
- [FaMe 81] A.FARO and G.MESSINA  
"Internetworking analysis"  
Computer communications Vol 4 n°4 August 81 - pp 169-173.
- [Fayo 79] G.FAYOLLE  
"Methodes analytiques pour files d'attente couplées"  
Thèse d'Etat - Université Paris 6, Paris France, Nov 79.

- [FaKM 80] G.FAYOLLE, P.J.B. KING, I.MITRANI  
"The solution of Certain two Dimensional Markov Models"  
Performance 80\* - Toronto - pp 283-289.
- [Ferr 78] D.FERRARI  
"Computer systems performance evaluation"  
Prentice Hall, Englewood cliffs, NJ, 1978.
- [FeSp 80] Ed. D.FERRARI and M.SPADONI  
Experimental Computer Performance Evaluation  
Lecture Notes of the 2nd Summer School on Computer Systems  
Performance Evaluation - Sogesta, Urbino Italy  
16-27 June 1980 - North Holland 1981.
- [FKAS 79] P.FRANKEN, D.KONIG, U.ARNDT, V.SCHMIDT  
"Queues and Point Processes"  
Akademie - Verlag - Berlin 1979.
- [Garc 79] M.GARCIA MOLINA  
"Performance of update algorithms for related data in a dis-  
tributed data base"  
Stanford University . 1977, Ph D Thesis, USA
- [Gele 80] E.GELENBE  
"Parallel computation of partial differential equations a mo-  
deling approach"  
19 th IEEE Conf. on Decision and Control - Nov 1980.
- [GeGa 82] E.GELENBE and D.GARDY  
"On the size of Projection I"  
"On the size of ProjectionII"  
Researchreport to be published  
LRI Univ. Paris-Sud Orsay France - 1982
- [GeMi 80] E.GELENBE and I.MITRANI  
"Analysis and Synthesis of Computer System"  
Academic Press - London - 1980.
- [GeMi 81] E.GELENBE and I.MITRANI  
"Analysis of retransmission control policies in CSMA local  
area networks"  
Rapport de recherche INRIA, 78153 Le Chesnay Cedex France  
1981.
- [GiSM 81] A.GLISSLER, A.JAGEMANN and E.MASER  
"Simulation of an X25 Network Providing Throughput Guarantees"  
Performance of Data Communication Systems and their Applications  
G.Pujolle Ed. Paris Sept 81 - North Holland - pp 279-290.
- [GrPa 80] D.H.GRIT and R.L. PAGE  
"Performance of a multiprocessor for Applicative Programs"  
Performance 80\* - ACM Sigmetrics Vol 9, n°2, Summer 80 -  
pp 181-190 - Toronto.

- [Grow 81] L.H. GRONER  
"QMOD : a system for automatically generating and solving analytical queueing network models"  
APL 81 - Conf. San Fransisco - 21.23 Oct. 81 - pp 125-128.
- [Grub 81] J.GRUBER  
"Performance considerations for integrated Voice and data networks"  
Computer Communications Vol 4 n°3 - Juin 1981 - pp 105-126.
- [HeWe 81] P.HEIDELBERGER and P.D. WELCH  
"A spectral method for confidence interval generation and run length control in simulations"  
C.A.C.M. 1981 24, pp 233-245.
- [HeWY 81] P.HEIDELBERGER, P.D.WELCH and P.C. YUE  
"Statistical analysis of data base systems measurements"  
Performance 81 - Amsterdam - Nov 81 - pp 335-344.
- [HeSc 79] W.E HELM and R.SCHASSBERGER  
"Insensitive Generalized Semi-Markov Schemes with Point Process Input"  
Preprint n°502 October 1979.
- [HoVD 81] A.HORDIJK and N. VAN DIJK  
"Networks of queues with blocking"  
Performance 81 - Amsterdam - Nov 1981 - pp 51-65.
- [Igle 78] D.L. IGLEHART  
"The Regenerative Method for Simulation Analysis"  
Current Trends in Programming Methodology"  
Vol III : Software Modeling and its Impact on Performance - Chandy-Yeh Editors - Prentice hall - 1978.
- [IgSh 80] D.L. IGLEHART and G.S. SHEDLER  
"Regenerative Simulation of Response Times in Networks of Queues"  
Springer-Verlag , New York 1980.
- [ISCA 80] ISCA 7  
7th Symposium on Computer Architecture.  
La Baule France - 6.8 May 1980.
- [ISCA 81] ISCA 8  
8th Symposium on Computer Architecture.  
Mimeapolis (Minesota) USA - May 1981  
in SIGARCH Newsletter vol 9 n°3 - ACM.
- [ ISO ] ISO  
"Data Processing - Open - System interconnection Basic Reference Model"  
ISO TC 97 - sous comité 16  
Proposition de standard ISO/DP 7498.

- [JaKo 80] U.JANSEN and D.KONIG  
"Insensitivity and Steady - State Probabilities in Product Form  
for Queueing Networks"  
Journal of Information Processing and Cybernetics  
EJK 16 (1980) 8/9 pp 385-397.
- [Jomi 81] G.JOMIER  
"A Mathematical Model for the Comparison of Static and Dynamic  
Memory Allocation in a Paged System"  
IEEE Transactions on Software Engineering  
Vol SE-7 - n°4, July 1981 - pp 375-385.
- [Kell 79] KELLY  
Reversibility and Stochastic Networks  
Wiley - 1979.
- [Kim 81 ] W.KIM  
"Queueing Optimization for Relational Data base Systems"  
IBM Research Report RJ 3081 - San José (Ca) USA - 3/10/81.
- [KiMi 80] P.J.B. KING and I.MITRANI  
"Numerical methods for Infinite Markov Processes"  
Performance 80\* - Toronto - pp 277-282.
- [Klei 76] L.KLEINROCK  
"Queueing Systems" Vol 1 and 2  
Wiley - 1976.
- [Koba 78] H.KOBAYASHI  
"Modeling and Analysis : an introduction to system performance  
evaluation methodology  
Addison Wesley - 1978.
- [LaPu 79] J.LABETOULLE , G.PUJOLLE  
"Modeling and Performance evaluation of the protocol HDLC"  
Proc. Symp. Flow Control in Computer Networks  
Versailles, France 12.14 Fev 1979  
J.L. Grangé, M.Gien - Eds - North Holland.
- [LaPu 80] J.LABETOULLE and G.PUJOLLE  
"Networks of queues"  
IEEE Trans. Software Engineering , 6, 4, 1980.- 373-381.
- \*\*
- [LaMS 79] S.S LAVENBERG, T.L. MOELLER, C.H. SAUER  
"Concomitant Control Variables Applied to the Regenerative Si-  
mulation of Queueing System"  
Open. Res. 27, 1979 - pp 134-160.
- [Lero 80] J.LEROUQUIER  
"La Simulation à événements discrets"  
Ed. Hommes et Techniques - (France) 1980.

- [LoMa 81] B.J. LOWNDES and J.W. MARTIN  
"A Comparative Study of form data base management systems"  
Data bases - Proc 1st Brit. Nat. Conf. on data bases  
Cambridge 13.14/7/81 - pp 187-205.
- [MKMi 81] J. Mc KENNA and D.MITRA  
"Integral representations and asymptotic expansions for closed  
markovian queueing networks : normal usage"  
Amsterdam - Nov 1981 - Performance 81 - pp 67-84.
- [Mari 78] R.MARIE  
"Methodes itératives de résolution de modèles math. de systèmes  
informatiques"  
RAIRO Informatique 12,2,1978.
- [MiNa 81] D.MINOLI and W.NAKAMINE  
"A Taxonomy and Comparison of Random Access protocol, for Com-  
puter networks"  
Data communication and Computer Networks  
S.Ramani (Edition) - North Holland  
CSI/IFIP - 1981 - PP 187-206.
- [pate 81] J.H. PATEL  
"Performance of Processor-Memory Interconnctions for Multiproces-  
sors"  
IEEE on Computers Oct 81 - Vol c30 n°10 - p 771-780.
- [PoLe 80] D.POTIER and Ph.LEBLANC  
"Analysis of Locking policies in data bases management systems"  
ACM 23,10 (Oct 1980) pp 584-593.
- [PoVe 79] D.POTIER and M.VERAN  
"QNAP : a Modeling Tool for computer Performance Evaluation"  
7th European Conference on Computer Measurement, Paris, Oct 79.
- [PoZi 78] L.POUZIN and H.ZIMMERMANN  
"A Tutorial on Protocols"  
Proc IEEE - Vol 66 - n°11 - Nov 1978 - pp 1346-1370.
- [PWIN 80] Proceedings of the Workshop on Interconnection Networks for par-  
allel and distributed processing  
April 21-28 1980 - Purdue University Howard Jay Siegel Editor.
- [Pujo 80] G.PUJOLLE  
"Réseaux de files d'attente en forme produit"  
RAIRO - Vol 14, n°4 - Nov 1980 - pp 317-330.
- [QED 78 ] "Distributed Processing : current practice and future develop-  
ments" Vol 2 Technical Report  
QED Information Sciences, 141 Lindon Street, Wellesley MA 02181

- [Rala 79] H.RALAMBONDRAINY  
"Application de l'analyse multidimensionnelle à l'étude de la charge d'un ordinateur"  
Thèse - Université Paris 6, France - Juin 1979.
- [Reis 81] M.REISER  
"Performance Evaluation of Data Communication Systems"  
RZ 1092 - 8/19/81 - IBM Zurich Research Laboratory.
- [Rela 80] M.REISER and SS. LAVENBERG  
"Mean Value Analysis of Closed multi Chain Queueing Networks"  
JACM - Vol 27 - n°2 - April 1980 - pp 313-322.
- [ReSa 78] M.REISER and C.H. SAUER  
"Queueing Network Models : Methods of Solution and their Program Implementation"  
Current Trends in Programming Methodology -  
Vol III : Software Modeling and its Impact on Performance.  
Chandy-Yeh Editors - Prentice Hall - 1978 - pp 115-167.
- [Rich 80] Ph.RICHARD  
"On the Evaluation of the size of the answer of a relational query"  
Rapport INRIA 78153 Le Chesnay France - n°51 - Dec 80.
- [Ries 81] D.R. RIES  
"The effects of concurrency control on the performance of data base management system"  
Rapport Univ. California Berkeley, CA, USA  
Electronics Research Laboratory - 222p.
- [RiSt 77] D.R. RIES, M.STONEBRAKER  
Effects of Locking Granularity in Data base Management system"  
ACM Transaction Data base Systems - Vol 2 n°3 - Sept 1977 -  
pp 233-246.
- [SaMS 80] C.H. SAUER, E.A. MACNAIR, S.SALZA  
"A Language for Extended queueing Network Models"  
IBM J. Res. Develop., 24 , 1980, pp 747-755.
- [Scha 77] R.SCHASSBERGER  
"Insensitivity of Steady - State Distributions of Generalized semi-Markov Processes"  
The Annals of Probability  
1977, Vol 5, n°1 - pp 87-99.
- [SRWG 80] G.D. SCHULTZ , D.B.ROSE, CH.WEST and J.P.GRAY  
"Executable Description and Validation of SNA"  
IEEE Transactions on Communications, COM.28,  
n°4, April 1980, pp 661-677.

- [Sere 81] D.SERET  
"Influence du degré d'anticipation sur les performances de la  
procédure HDLC"  
Performance of Data communication Systems and then Applications.  
G.Pujolle (ed) - Paris - Sept 1981 - pp 291-304 - North Holland.
- [Sevc 81] K.C. SEVCIK  
"Data base System Performance Prediction Using an Analytical  
Model"  
Proc. 7th Very Large Data Base - Cannes, France - 9.11 Sept 1981  
Ed Zaniolo-Delobel - pp 182-198.
- [ShSp 81] A.W. SHUM and P.G. SPIRAKIS  
"Performance analysis of concurency control methods data base  
systems"  
8th International symp. on Computer Performance modeling  
and Evaluation  
4.6. Nov. 1981 - Amsterdam - p 1-20.
- [Span 81] O.SPANIOL  
"Analysis and Performance Evaluation of Hyperchannel Access Pro-  
tocols"  
Proc 2nd International Conf. on Distributed Computing Systems,  
Paris, France - April 8-10 1981.
- [Stew 79] W.J. STEWART  
"A Direct Numerical method for queueing networks"  
4th International Symposium on Modeling and Performance Evalua-  
tion of Computer Systems  
WIEN (Austria) - Feb 1979.
- [Svob 76] L.SVOBODOVA  
"Computer Performance Measurement and Evaluation methods :  
Analysis and Applications"  
Elsevier - New York - 1976.
- [TGPM 78] F.A. TOBAGI, M.GERLA, R.W. PEEBLES, E.G. MANNING  
"Modeling and Measurement Techniques in Packet Communication  
Networks"  
Proc. of the IEEE - Vol 66, n°11, Nov 78, pp 1423-1447.
- [ToHu 80] F.A. TOBAGI, V.B. HUNT  
"Performance Analysis of Carrier Sense Multiple Access with  
Collision detection"  
Computer Network - Vol 4, n°5, Oct.Nov 1980.
- [Tsic 81] TSICHRITZIS  
"Integrating data bases and message Systems"  
Proc. Very Large Data Bases  
Cannes - Sept 1981 - pp 356-362.



- [VaGL 80] H.T. VANTILBORGH, R.L. GARNER, E.D. LAZOWSKA  
"Near complete Decomposability of Queueing Networks"  
Toronto May 1980 - Performance 80 - pp 81-92.
- [Wilm 79] P.WILMS  
"Etude et Comparaison d'algorithmes de maintien de la cohérence  
dans les bases de données réparties"  
Thèse - I.N.P. Grenoble , France - 23/11/1979.
- [ZiPo 81] H.ZIMMERMANN and L.POUZIN  
"The standard Network Architecture Developed by ISO"  
Data Communication and Computer Networks  
S.Ramani Editor - North Holland  
CSI/IFIP 1981 - pp 1-15.
- [Zube 81] ZUBEREK  
"Timed-Petri Nets and Preliminary Performance evaluation"  
Perf. 81.
- \*\* [Lave 80] S.S. LAVENBERG  
"Closed Multichain Product Form Queueing Networks with Large  
Population Sizes"  
IBM Research report - RC 8496, Sept 80.
- Performance 80\* - Toronto - May 80  
ACM Sigmetrics - Vol 9, n°2  
Summer 1980.



## Program Optimization on Ryad-22 Computer.

L. Adámy and J. Micsik, Hungary

### Abstract

The author's aim was to optimize the main memory usage of a program written by a third person.

The original program which was written in PL/I Level F language was recompiled using PL/I optimizing compiler to exploit its main memory requirement optimizing options.

With both versions the large program was split into smaller program parts without rewriting the whole program and compiled by applying the overlay structure.

Another experiment was to form from these program parts independent OS jobsteps, the necessary linkage between them being supplied by a parameter file.

All program versions were run on Ryad-22 /EC 1022/ computer /its main memory was 512 kbyte/ in OS/MFT Rel 21.8F and HASP/.

Because of the constraints of this task /an already existing large program with complicated control/ the optimal solution was found to be the program variant with independent OS jobsteps, whose performance was only approached by the versions with the overlay structure.

### 1. The Original Program

The original DISCNT program is part of the SAA /System Analyzing and Accounting/ program package. Its task is to generate the cumulative utilization and account values sorted by "systems" /i.e. type of the computer, of the operating system e.g. R-22/OS, R-22/DOS, RC3600, etc./, by account

numbers and by departments, and to make the desired discount on the basis of the already analysed universal resource utilization and accounting records /in this system the so called Q records/.

The DISCNT program is able to initially generate the so called file C /which contains the cumulative account information organized index-sequentially/ and to update it in the sense of above mentioned procedure.

The result of the necessary discount or transfer /from an account number to another account number/ entry /record "EROL"/ is the logical deletion of the original "Q" record and, - depending upon the total or partial discount of the resource utilization - one or two new "Q" records are generated. During discount the cumulative account informations are updated appropriately. The flowchart of the original DISCNT program is on Fig. 1. The control function in the program is performed by the five-bits variable FLAG using the label array TEDD /FLAG/.

The bits of the FLAG/5/ variable have the following meanings  
Bit F5 is 1 at end of file of sorted EROL /ESTROUT/  
Bit F4 is 1 if there is any "EROL" record  
Bit F3 is 1 at end of file of sorted Q /QSRTOUT/  
Bit F2 is 1 if there is any "Q" record  
Bit F1 is 1 if there is any "C" record

## 2. Partitioning the Program

It is impossible to split the program into small new parts /subprograms/ owing to the application of the TEDD /FLAG/ label array. Only the reading and the sorting activities of the "EROL" and "Q" records can be separated from the whole program, the overwhelming majority of the program remaining unchanged.

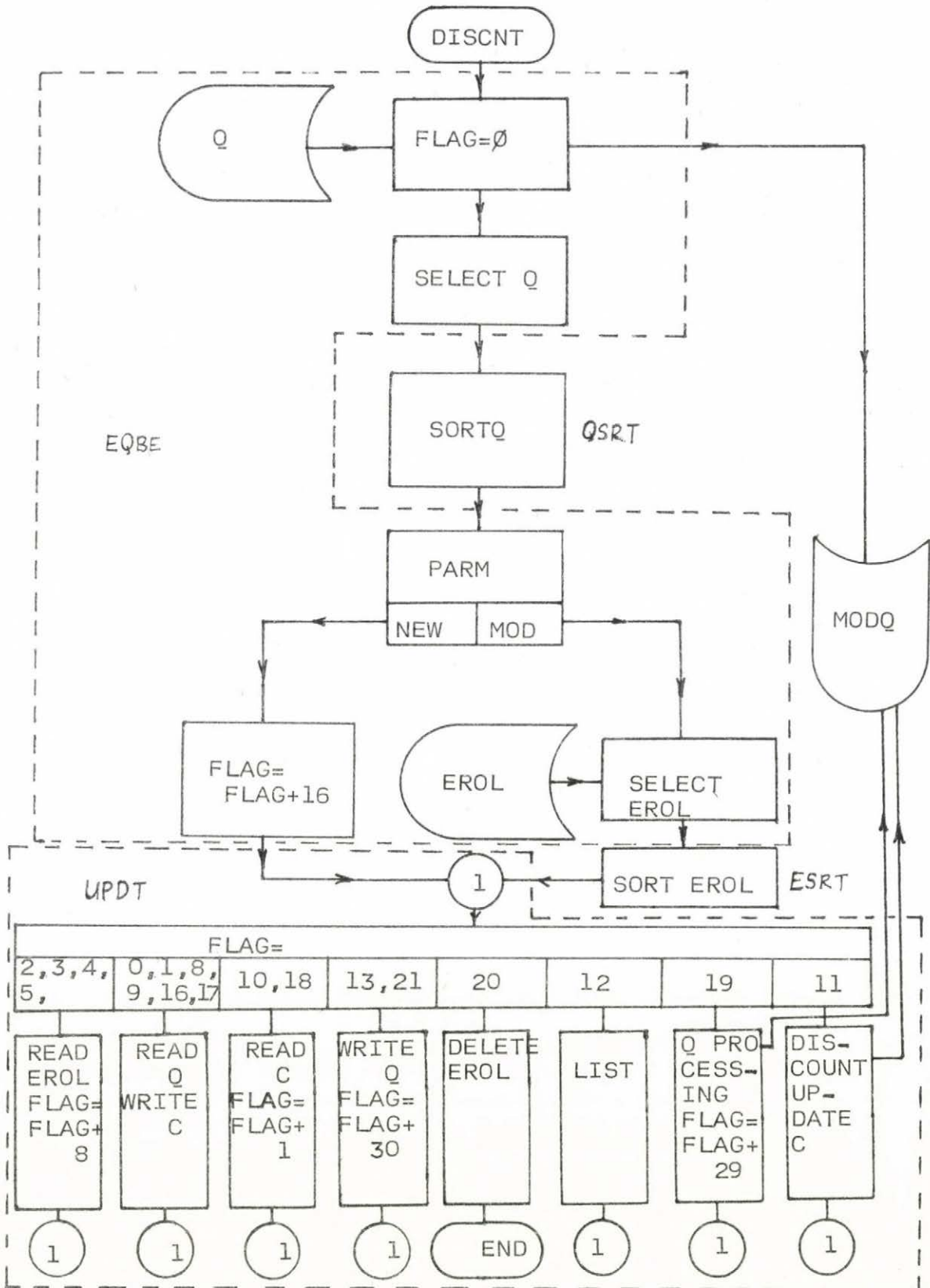


Fig.1. The Original Program

The subprogram doing the read selection and verification of the "EROL" and "Q" records is called EQBE. The sorting of the selected "EROL" and "Q" records is made by the subprograms ESRT and QSRT respectively.

The subprogram called UPDT performs the majority of the tasks i.e. generates cumulative account values, discounts and generates or updates the file "C".

Our original intention was not to rewrite the whole program but to examine how its main memory usage can be reduced: by

- splitting the program into smaller subprograms,
  - recompiling the program applying the overlay structure,
  - at PL/I Optimizing compiler using the main memory requirements optimizing option /OPT(TIME)/, and
  - transforming the subprograms into independent OS jobsteps.
- The files of the subprograms EQBE, ESRT, QSRT and UPDT can be seen on Fig. 2.

### 3. The Program Version Using Overlay Structure

The program version compiled applying the overlay structure is on Fig. 3.

The root segment /called VEZER/ gets the parameters PARM /can be "NEW" or "MOD"/ and PHC /Ø1..12/ at run time and it is here, that the declarations necessary for calling the SORT/MERGE program can be found.

First, the root segment /VEZER/ calls subprogram EQBE then the SORT/MERGE program and finally the processing subprogram UPDT.

### 4. Program Version Using Independent OS Jobsteps

All the subprograms /EQBE, UPDT, ESRT, QSRT/ are the same one as at the earlier program versions users. The necessary

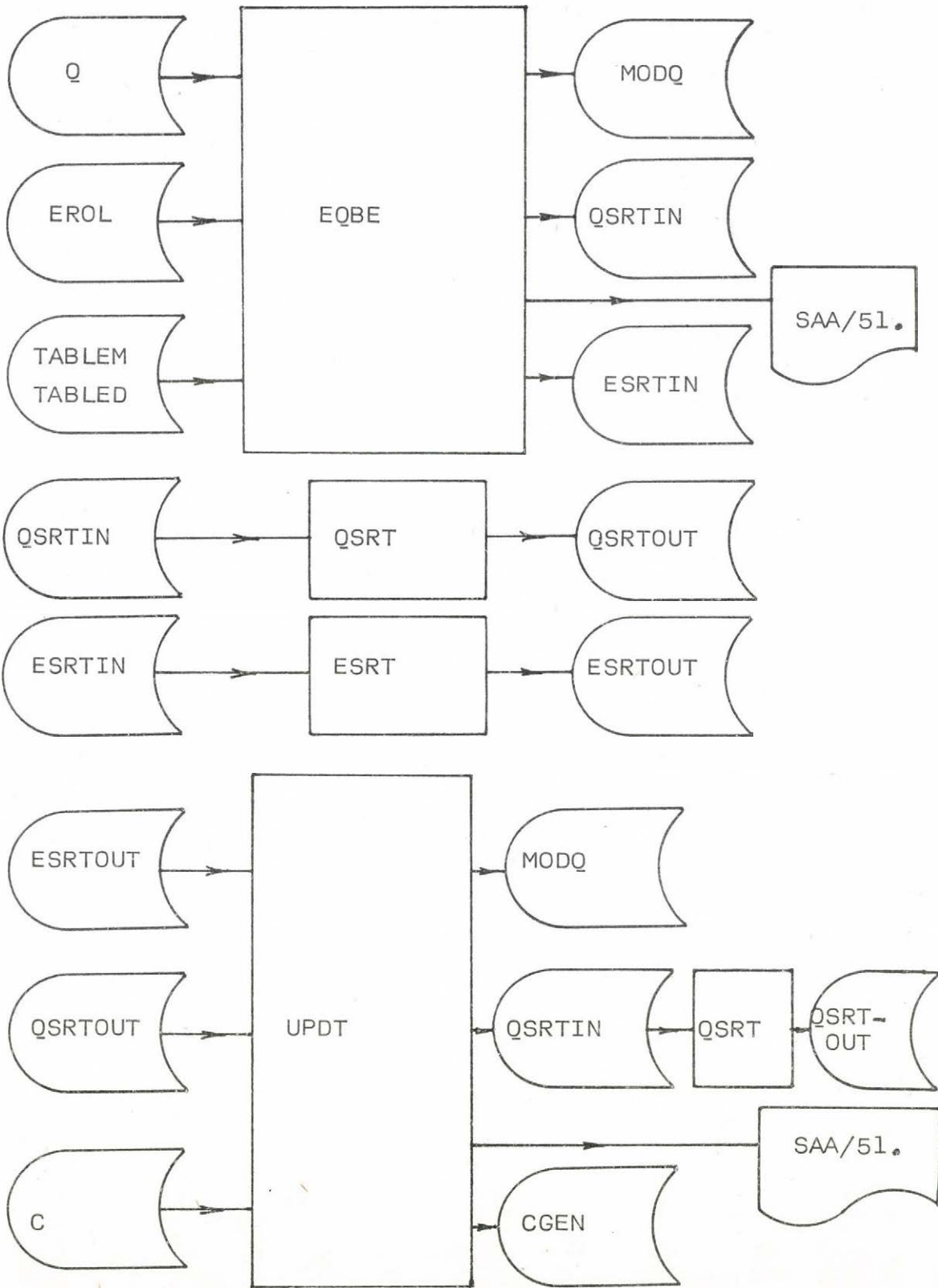


Fig. 2. Data Sets of EQBE, QSRT, ESRT, UPDT subprograms

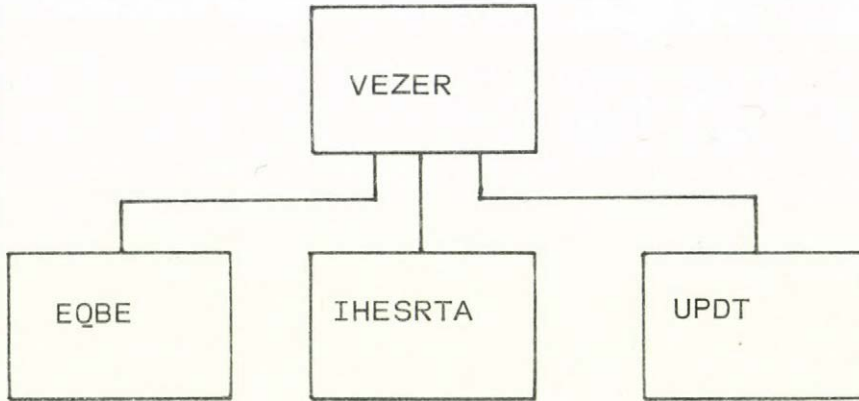


Fig. 3. Program version using overlay structure

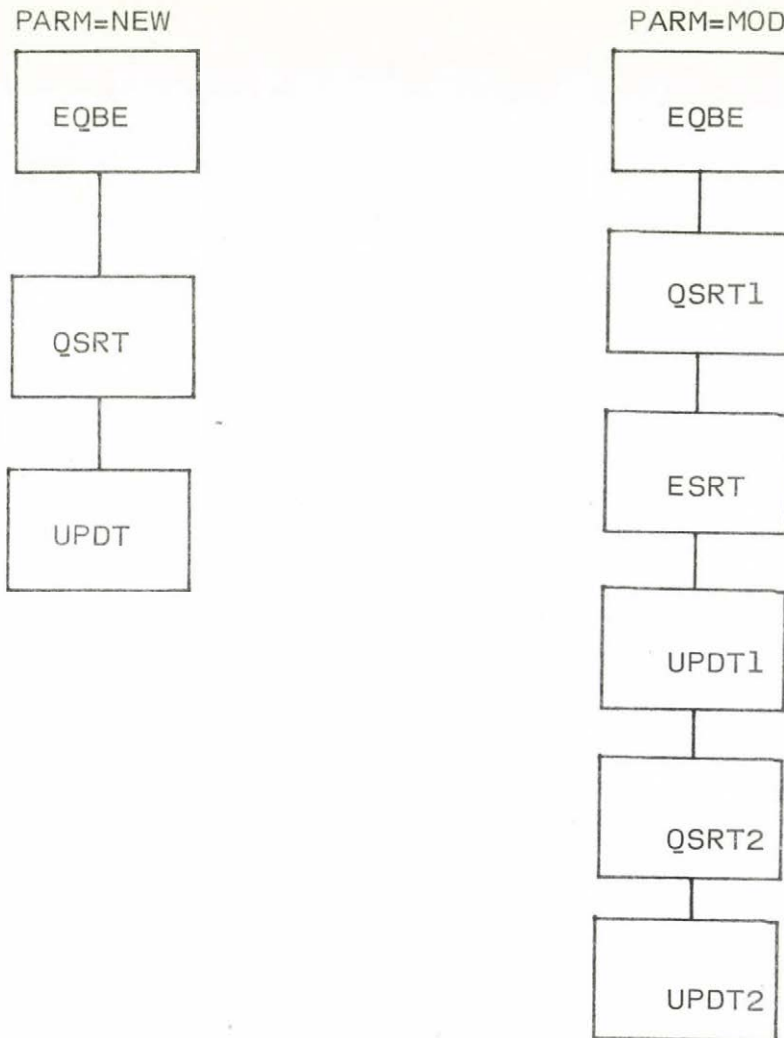


Fig. 4. The independent OS jobsteps



linkage between subprograms EQBE and UPDT is supplied by a parameter file /called PARF/ and its elements are all the common /PARM, PHO, FLAG, QMARK and QR/ variables. At the end of the subprogram EQBE the values of the above variables are written into file PARF and subprogram UPDT starts with reading file PARF. The independent OS jobsteps in case of PARM=NEW /to generate file "C"/ and PARM=MOD /to update file "C"/ can be seen on Fig. 4.

## 5. Experimental Results

All program-versions were run on Ryad-22 computer with main memory of 512 kbyte under OS operating system /MFT Release 21.8F and HASP/. The compilers were the PL/I-F version 5.5 and the PL/I Optimizing version 1 Release 2.2. The results of the program runs under PL/I-F compiler are shown in Tables 1. and 2. and those under PL/I Optimizing compiler in Tables 3. and 4.

On the tables can be seen the following run data for the selected best program-versions:

version - original: the original program

overlay: compiled with overlay structure

jobsteps: version with independent OS  
jobsteps

job-identifier - the identifier /name,date/ of the job in  
which the program was run

number of jobsteps - number of jobsteps in the job

BUFNO - the value of the BUFNO /number of buffers/ parameters  
of the DCB in the DD card

The following data characterize the sort properties of file  
"Q":

QS/cyls/ - the primary space in cylinders of the QSRTOUT file

QSW/cyls/ - the primary space in cylinders of the QSRTWK work  
file

QSW/pcs/ - number of QSRTWK work files

Version	Job identifier	Number of jobsteps	BUFNO	Q SORT					Console time	CPU time	main memory
				QS cyls	QSW cyls	QSW pcs	RCD IN	CORE kbyte			
									min	min sec	kbyte
Original	TDSCNT 81.feb.	1	2	12	5	6	9514	55	25	5m 11.48s	154
Overlay	DSC/3o	1	1	30	12	3	10752	25	24	7m 20.38s	110
Jobsteps	ON/22	3	2	24	10	3	10752	60	19	6m 37.92s	108

Table 1. Results with PL/I-F and PARM=NEW

Version	Job identifier	Number of jobsteps	BUFNO	Q SORT					Console time min	CPU time min sec	main memory kbyte
				QS cyls	QSW cyls	QSW pcs	RCD IN	CORE kbyte			
Original	TDSCNT	1	2	12	5	6	5973	55	32	6m 54.18s	180
Overlay	DSC/36	1	1	40	16	3	5619	18	32	9m 11.82s	128
Jobsteps	ON/23	6	2/1	30	12	3	5619	60	20	7m 32.70s	120

Table 2. Results with PL/I-F and PARM=MOD

Version	Job identifier	Number of Jobsteps	BUFNO	Q SORT					Con- sole time min	CPU time min,sec	ISA/ OUTSIDE ISA kbyte/ kbyte	main memory kbyte
				QS cyle	QSW cyls	QSW pcs	RCD IN	CORE kbyte				
Original	OPTD/23	1	2	48	24	3	10752	40	22	7m 19.68s	12/0	134
Overlay	FATDP/34	1	2	30	12	3	10752	18	39	8m 23.88s	8/0	96
Jobsteps	ONOP/13	3	2	48	10	3	10752	60	16	6m 03.84s	6/0	92

Table 3. Results with PLIOPT and PARM=NEW

Version	Job identifier	Number of jobsteps		Q SORT					Con- sole time	CPU time	ISA/ OUTSIDE ISA	main memo- ry	
				BUENO	QS	QSW	QSW	RCD IN					CORE
					cyls	cyls	pcs						kbyte
								min	min,sec	kbyte/ kbyte	kbyte		
Original	OPTD/23	1	2	48	24	3	5619	40	17	5m 16.80s	12/0	140	
Overlay	FATOP/35	1	2	30	12	3	5376	18	36	8m 44.22s	8/3	116	
Jobsteps	ONOP/14	6	2	48	24	3	5376	60	19	7m 10.16s	10/0	110	

Table 4. Results with PLIOPT and PARM=MOD

RCD IN - the number of "Q" records read in by the SORT/MERGE program

CORE - the main memory necessary for the SORT/MERGE program in thousands of bytes

Console time - the console time of the job in minutes

CPU-time - the actual CPU-time of the job in minutes and seconds

ISA/OUTSIDE ISA - /only at PL/IOptimizing compiler/

Initial Storage Area /ISA/ and the amount of storage obtained outside ISA in kbytes

main memory - the amount of the main memory occupied by the job in kbytes

The data in Tables 1. and 3. are for PARM=NEW /generate "C" file/ and in Tables 2. and 4. for PARM=MOD /update file "C"/.

Considering the amount of main memory occupied by the job, the best results are given in all cases by the program-version using independent OS jobsteps.

Enterprise for Computing Application, Hungary  
H-1536 Budapest. P.O.B. 227.

PROPERTIES OF CONCURRENT SYSTEMS

Piotr Prószczyński, Poland

Introduction.

Our goal is to present some results concerning the properties of concurrent systems. These results was obtained on the basis of investigation of concurrency-like relations defined for the systems.

We want to pay attention that the concurrency relation notion was introduced by C.A.Petri in the case of concurrent processes (occurrence nets) [2]. In that case concurrency relation can be defined as the complement of partial order. Thus, the most interesting results follow from the partial orders theory. An extention of concurrency relation notion to the case of systems (represented by non-deterministic and cyclic nets) and methods of its investigation are not obvious.

The present paper is based on the approach presented in [1], and uses some notions and results described in [1,5,6].

1.Basic notions.

In this section we recall some notions and results from [1,5], discarding the formal notation, if it is possible. The reader interested in more formal approach is advised to refer to these articles.

1.1.Simple and proper nets.

Let  $X$  be a set and let  $left: X \times X \rightarrow X$ ,  $right: X \times X \rightarrow X$  be the functions:

$$(\forall (x,y) \in X \times X) \quad left((x,y))=x, \quad right((x,y))=y.$$

Df. By a simple net we mean any pair  $N=(T,P)$ , where:

$T$  - is a set of transitions,

$P \subseteq 2^T \times 2^T$  is a relation (interpreted as a set of places),

$$(\forall a \in T) (\exists p, q \in P) \quad a \in left(p) \cap right(q).$$

We shall consider finite simple nets only.

We accept the standard graphical representation of Petri net. In this representation places will be numbered or described by  $[a_1, \dots, a_n; b_1, \dots, b_m]$  to denote the fact  $\{\{a_1, \dots, a_n\}, \{b_1, \dots, b_m\}\} \in P$  ( $a_1, \dots, a_n$  - are input, and  $b_1, \dots, b_m$  - are output transitions for a given place).

We will say that the net  $N_1=(T_1, P_1)$  is a subnet of  $N=(T, P)$  and write  $N_1 \subseteq N$  iff  $P_1 \subseteq P$ . It was proved that " $\subseteq$ " is a partial order relation and that the set of all simple nets with this relation is a lattice.

We have:  $N_1 \vee N_2 = (T_1 \cup T_2, P_1 \cup P_2)$ .

Df. A simple net  $N=(T,P)$  is called elementary iff

- 1)  $(\forall a \in T) |^{\cdot}a| = |a^{\cdot}| = 1$  (every transition has exactly one input and exactly one output place)
- 2)  $N$  is connected (a graph representing the net  $N$  is connected).

An elementary net (equivalent to finite state machine) represents sequential system - in general non-deterministic and cyclic one.

Df. A simple net  $N=(T,P)$  is said to be proper iff it is an union of its elementary subnets.

Our further considerations will be restricted to proper nets only. Furthermore we will consider also the dynamic structure of the net.

### 1.2. Marked nets.

Let  $N=(T,P)$  be a simple net and  $R1 \subseteq 2^P \times 2^P$  be the following relation:

$$(M_1, M_2) \in R1 \iff (\exists a \in T) M_1 - \cdot a = M_2 - a^{\cdot} \text{ \& \ } \cdot a \in M_1 \text{ \& \ } a^{\cdot} \in M_2.$$

The above relation is called reachability relation in one step.

The relation  $RN = (R1 \cup R1^{-1})^*$ , called reachability relation of the net  $N$ , is an equivalence relation and for every  $M \in 2^P$  the equivalence class of  $RN$  containing  $M$  will be denoted by  $[M]_{RN}$ .

More formally we shall define a different reachability relation in one step [3]. The relation  $CR1 \subseteq 2^P \times 2^P$  defined as follows:

$$(M_1, M_2) \in CR1 \iff (\exists A \subseteq T) ((\forall a, b \in A) a \neq b \Rightarrow \cdot a \wedge \cdot b = a^{\cdot} \wedge b^{\cdot} = a^{\cdot} \wedge b^{\cdot} = a^{\cdot} \wedge b^{\cdot} = \emptyset) \\ \text{\& \ } M_1 - \bigcup_{a \in A} \cdot a = M_2 - \bigcup_{a \in A} a^{\cdot} \\ \text{\& \ } \bigcup_{a \in A} \cdot a \in M_1 \text{ \& \ } \bigcup_{a \in A} a^{\cdot} \in M_2$$

is called concurrent reachability relation in one step.

It can be easily proved that the relations  $RN$  defined on the basis of the  $R1$  and  $CR1$  are the same, i.e.  $RN = (R1 \cup R1^{-1})^* = (CR1 \cup CR1^{-1})^*$ .

Df. By a marked simple net we mean any triple  $MN=(T,P,Mar)$ , where:

- $N=(T,P)$  is a simple net,
- $Mar \subseteq 2^P$  is a set of markings and
- $Mar = \bigcup \{ [M]_{RN} \mid M \in Mar \}$ .

We will say that:

A transition  $a \in T$  is fireable iff  $(\exists M_1, M_2) \cdot a \in M_1 \text{ \& \ } a^{\cdot} \in M_2$ .

A marked net is locally fireable iff every transition of this net is fireable.

A marked net is safe iff  $(\forall C \subseteq 2^P) (\forall a \in T)$

$$(\cdot a \wedge C = \emptyset \text{ \& \ } (\exists M \in Mar) \cdot a \cup C \subseteq M) \iff (a^{\cdot} \wedge C = \emptyset \text{ \& \ } (\exists M \in Mar) a^{\cdot} \cup C \subseteq M).$$

### 1.3. Coexistancy relations.

The relations we are going to use as a model of concurrency-like relations are symmetric and irreflexive (called in [1] sr-relations).



For any proper net  $N=(T,P)$  we can define these relations in the following way:

Let  $N_i=(T_i,P_i)$  ( $i=1,2,\dots,m$ ) be all elementary subnets of  $N$  and let  $\text{cov}_E = \{P_{k_1}, \dots, P_{k_n}\}$  ( $k_n \leq m$ ) be a covering of  $P$ .

Df. The relation  $\text{coex}_E$  defined as follows:

$$(\forall p, q \in P) (p, q) \in \text{coex}_E \Leftrightarrow (\forall A \in \text{cov}_E) p \notin A \text{ or } q \notin A$$

is called the coexistancy relation defined by the covering  $\text{cov}_E$ .

In words: Places  $p$  and  $q$  are in the relation  $\text{coex}_E$  (are "coexisting") iff they don't belong to the same subnet of  $N$ .

According to [1,2] the family of maximal sets of places being in the relation  $\text{coex}_E$  we denote by  $\text{kens}(\text{coex}_E)$ .

Df. A marked simple net  $MN=(T,P,Mar)$  is called naturally marked iff

$N=(T,P)$  - is a proper net, and

$Mar = \text{kens}(\text{coex}_E)$  (for any covering  $\text{cov}_E$  defining the relation  $\text{coex}_E$ ).

Theorem [1] .

Every naturally marked net is safe and locally fireable. ■

Let us observe that every marking of naturally marked net and every elementary subnet may have at most one common place. Thus, in this case, the set of markings represents the set of all permissible global states of such the system, whose sequential subsystems cannot be in two different states at the same time.

In the case of naturally marked nets there was introduced the notion of C-density, which save the interpretation of Petri's K-density for this, wider class of nets:

Df. (R.Janicki, see also [5,6] )

The relation  $\text{coex}_E$  is C-dense for covering  $\text{cov}_E$  defining this relation iff

$$(\forall A \in \text{cov}_E) (\forall B \in \text{kens}(\text{coex}_E)) A \cap B \neq \emptyset .$$

We can say that if the relation  $\text{coex}_E$  is C-dense for the given covering  $\text{cov}_E$ , then every global state of the system represented by naturally marked net is an union of local states of all sequential subsystems.

The following, very useful property of covering was defined in [5,6]:

Df. Let  $\text{cov}$  be a covering of  $X$ , where  $X$  is finite set.

The covering  $\text{cov}$  is said to be replete set covering (abbr. RS-covering) iff

$$(\forall S \in \text{cov}) (\forall R \subset \text{cov}) [((\forall A \in R) A \cap S \neq \emptyset) \& S \subset \bigcup_{A \in R} A \Rightarrow \bigcap_{A \in R} A \subset S] .$$

The following theorem is valid:

Theorem [5] .

If a covering  $\text{cov}_E$  is not RS-covering, then the relation  $\text{coex}_E$  defined by this covering is not C-dense. ■

2. Various notions of fireability.

In the previous section we have recalled from [1] the definition of locally fireable net. Note that the local fireability is very important property. The lack of local fireability means that the static and the dynamic structure of the system are inconsistent: there exist transitions which never be fired. For naturally marked nets, however, the request of local fireability is always fulfilled. On the other hand, the request of local fireability is too weak; it is possible that there is no firing sequence, that markings allowing to fire different transitions belong to different equivalence classes of reachability relation.

In [1] it was defined notion of the fireability of the net - the fireable net that is the net which is locally fireable and the set of markings consist of exactly one equivalence class. However, such the condition is difficult to satisfy and, in addition, in more cases it cannot be assumed because of interpretation of the net.

Let us now introduce another notions, which allow us to describe desirable properties of nets.

Df. A marked simple net  $N=(T,P,Mar)$  is called weakly fireable iff

$$(\exists M \in Mar)(\forall a \in T)(\exists M_a) M_a \in [M] \ \& \ \cdot a \in M \ \& \ \underline{a} \cap (M - \cdot a) = \emptyset .$$

Df. A marked simple net  $N=(T,P,Mar)$  is called semifireable iff

$$(\forall M \in Mar)(\forall a \in T)(\exists M_a) M_a \in [M] \ \& \ \cdot a \in M \ \& \ \underline{a} \cap (M - \cdot a) = \emptyset .$$

(If the net is safe, then underlined conditions can be omitted.)

The weak fireability means that there exists marking which enable us to reach (in the meaning of forward and backward reachability relation) the possibility of firing of every transition of the net. If the net is semifireable, then each marking has such the property.

Let us observe that the net is "coherent" - from the point of view of the dynamic structure of the net - if it is at least weakly fireable.

The necessary conditions of weak fireability and semifireability are the following:

Theorem 1.

If the covering  $\text{cov}_E$  defining the relation  $\text{coex}_E$  is not RS-covering, then the naturally marked net  $(T,P,\text{kens}(\text{coex}_E))$  is not weakly fireable. ■

Theorem 2.

If the relation  $\text{coex}_{\mathbb{E}}$  is not C-dense, then the naturally marked net  $(T, P, \text{kens}(\text{coex}_{\mathbb{E}}))$  is not semifireable. ■

Note that C-density is neither necessary nor sufficient condition of weak fireability.

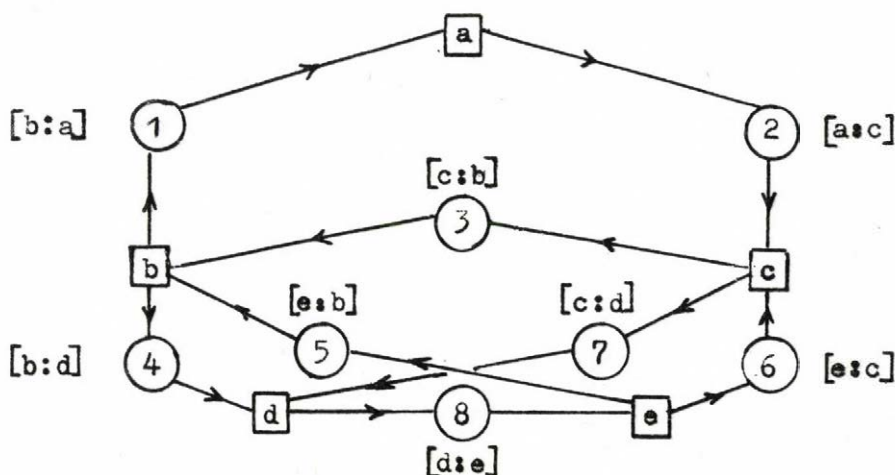
Postulate. The net representing "well-defined" concurrent system should satisfy the following conditions:

- 1) it should be weakly fireable
- 2) every elementary subnet representing real sequential subsystem and every marking (global state of the system) should have common element. ■

Of course, in the case of naturally marked nets the second condition is equivalent to C-density. So, a naturally marked net may represent the well-defined system only if the places of its subnets corresponding to real sequential subsystems form a RS-covering.

Example.

Let us consider the following net:



The decomposition into elementary subnets gives us five subnets with the following sets of places:

$$P_1 = \{1, 2, 3\}, \quad P_2 = \{4, 5, 8\}, \quad P_3 = \{3, 4, 6, 8\},$$

$$P_4 = \{6, 7, 8\}, \quad P_5 = \{1, 2, 5, 7, 8\}.$$

a). At first let us consider the covering  $\text{cov}_N = \{P_1, P_2, P_3, P_4, P_5\}$ .

This covering is not RS-covering, because:

$$P_1 \cap P_3 \neq \emptyset \quad \& \quad P_1 \cap P_5 \neq \emptyset \quad \text{and:}$$

$$P_1 \not\subseteq P_3 \cup P_5 \quad \& \quad P_3 \cap P_5 = \{8\} \not\subseteq P_1.$$

Thus the net  $(T, P, \text{kens}(\text{coex}_N))$ , where  $\text{coex}_N$  is defined by the covering  $\text{cov}_N$ , is not weakly fireable and is not C-dense.

b). Now, let the covering defining relation  $\text{coex}_E$  be the following one:  
 $\text{cov}_E = \{P_1, P_2, P_4\}$ . This covering is a minimal covering, and we can state immediately that  $\text{coex}_E$  is C-dense (see [5,6]).

The investigation of semi- and weak fireability is arduous because of the lack of sufficient conditions. At first we have to construct the family  $\text{kens}(\text{coex}_E)$  on the basis of the graph representing the relation  $\text{coex}_E$  (see [1]). This family is of the form:

$$\text{kens}(\text{coex}_E) = \{ \{1,4,6\}, \{1,5,6\}, \{1,5,7\}, \{1,4,7\}, \{1,8\}, \{2,4,6\}, \{2,4,7\}, \\ \{2,5,6\}, \{2,5,7\}, \{3,4,6\}, \{3,4,7\}, \{3,5,7\}, \{2,8\}, \{3,8\} \}.$$

Further we have to investigate the reachability and fireability.

We can observe that there are four equivalence classes of RN:

$$K_1 = \{ \{3,4,6\} \}$$

$$K_2 = \{ \{1,5,7\}, \{2,5,7\} \}$$

$$K_3 = \{ \{1,4,6\}, \{2,4,6\}, \{3,4,7\}, \{3,5,6\}, \{3,8\} \}$$

$$K_4 = \{ \{1,5,6\}, \{1,4,7\}, \{2,4,7\}, \{2,5,6\}, \{1,8\}, \{2,8\}, \{3,5,7\} \}.$$

Because  $K_3$  (or  $K_4$ ) allows us to fire all transitions of the net - the net is weakly fireable. It is not semifireable, however the net  $(T, P, \text{Mar})$ , where  $\text{Mar} = K_3 \vee K_4$  (it is not naturally marked net) - is.  $\square$

### 3. Concurrency.

The concurrency-like relations, which have been considered in previous sections of this paper, describe only the coexistence of the local states of the system. In this way, however, we are able to describe all permissible global states of the system, and consequently, to describe the notion of transitions' concurrency. We will accept the following definition:

Df. The transitions  $a_1, \dots, a_n$  of marked simple net  $(T, P, \text{Mar})$  are said to be concurrent iff

$$1). (\forall a_i, a_j \in T) i \neq j \Rightarrow \underline{a_i \wedge a_j = a_i \wedge a_j = a_i \wedge a_j = a_i \wedge a_j = \emptyset}.$$

$$2). (\exists M \in \text{Mar}) \bigcup_{i=1}^n a_i \subseteq M \quad \& \quad \underline{\bigcup_{i=1}^n a_i \wedge (M - a_i) = \emptyset}.$$

(If the net is safe one, then the underlined parts of the above definition can be omitted).

Let us observe that first condition of the definition concerns the static structure of the net, second - the dynamic structure of it. Because in the case of naturally marked nets the dynamic structure is also built on the basis of the static one, thus the concurrency of transitions in this case is completely described by the static structure of net. (However, it is dependent on the choice of the covering  $\text{cov}_E$ ).

Let  $\text{coex}_E$  and  $\text{coex}'_E$  be the coexistancy relations defined by the coverings (of the net  $(T,P)$ )  $\text{cov}_E$  and  $\text{cov}'_E$  respectively.

Theorem 3.

If  $\text{cov}_E \subseteq \text{cov}'_E$  then all transitions concurrent for the net  $(T,P,\text{kens}(\text{coex}'_E))$  are concurrent for the net  $(T,P,\text{kens}(\text{coex}_E))$  too. ■

Theorem 4.

The greatest number of concurrent transitions (which can be simultaneously concurrent) of the net  $(T,P,\text{kens}(\text{coex}_E))$  is not greater than cardinality of the least covering  $\text{cov}'_E \subseteq \text{cov}_E$ . ■

Theorem 5.

If every two transitions belonging to a set  $\{a_1, \dots, a_k\} \subseteq T$  are concurrent and the net  $(T,P,\text{Mar})$  is naturally marked, then all transitions  $a_1, \dots, a_k$  are concurrent. ■

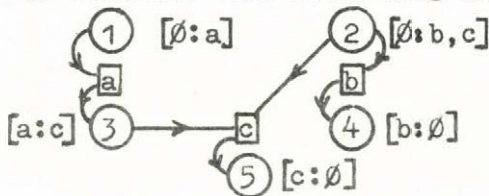
Note that in the case non-naturally marked net Theorem 5 is not true, so in this sense the natural marking gives us the "maximal" concurrency. We want to pay attention that the fact that  $k$  transitions can be non-concurrent - in spite of every  $n < k$  of them are concurrent - points out that binary relation of transitions' concurrency cannot be (in general) extended to more complex structure.

At the end we want to present some comments concerning the definition of transitions' concurrency accepted here.

First comment. The fact that transitions  $a$  and  $b$  are concurrent means only the possibility of concurrent executions of them. However, for some initial states this possibility may be lost. This situation is shown by Example, where we can find two pairs of concurrent transitions:  $a, d$  and  $a, e$ . But concurrent execution is possible only for markings belonging to equivalence class  $K_4$ . If we start from any marking belonging to  $K_3$ , then no transitions can be fired concurrently, although all of them will be executed. ■

Second comment. We will pay special attention to the understanding of concurrency phenomenon in the case of non-deterministic systems.

Let us consider the following nets:



Here we have: - only one covering  $\text{cov}_E = \{\{1, 3, 5\}, \{2, 4, 5\}\}$  and  
 - the family  $\text{kens}(\text{coex}_E) = \{\{1, 2\}, \{2, 3\}, \{1, 4\}, \{3, 4\}, \{5\}\}$ .

On the basis of the definition we can state that transitions a and b are concurrent. However, looking closer the philosophy of concurrency we can say that this result is not so clear, as seems at first.

In [4] the authors consider differences between non-determinism and concurrency on the basis "how the decisions are taken". They suggest that concurrency - interleaving of many interactions - "is not decision at all, since the actual interleaving cannot have any influence on the future behaviour of the system". From this point of view we can dispute if the transitions a and b are really concurrent.

These considerations arise the following question:

Maybe the stronger definition of transitions' concurrency is necessary?

The author propose the following explanation, which let us save the definition in the previous form:

The global nondeterminism [4] is represented by the (non-deterministic) evolution of a global state of the system. Thus, the behaviour of the system is the result of the global decision, how the actual state will be transformed. The single decision, of course, can concern only the transformation of the actual state in another state reachable in one step.

In our example we have:

the actual state -  $\{1,2\}$  (any action - a or b - can be fired) and the states reachable in one step -  $\{1,4\}$ ,  $\{2,3\}$  and  $\{3,4\}$ .

Note that the last one is reachable by the simultaneous firing of transitions a and b (it is reachable in the meaning of the relation CK1).

When the decision that the output state is the state  $\{3,4\}$  has been taken, then a and b can be fired concurrently (simultaneously or in arbitrary sequence).

So, we can say that in the case of non-deterministic system the possibility of concurrent executions of transitions may depend on the (global) decision of the system. ■

#### 4. Final comment.

The results presented here, obtained for proper simple nets, seem to be helpful in the case of synthesis of concurrent system on the basis of the sequential components of this system. However, it is obvious that some sufficient conditions of weak fireability and C-density will be more convenient (the necessary and sufficient condition of C-density has been described in [5]). To resolve this problem we have to look closer the description of places for simple net - till this description is used only during the decomposition of the net into elementary subnets.

Acknowledgement.

The author would like to thank R. Janicki for invaluable discussions, stimulating papers and for help in the formulation of many problems.

References.

- [1] Janicki R., On Atomic Nets and Concurrency-like Relations,  
Lecture Notes in Comp. Sci., vol.70, Springer-Verlag,  
1980, pp. 320-333.
- [2] Petri C.A., Non-Sequential Processes,  
ISF Report 70-01, GMD, Bonn, 1977.
- [3] Petri C.A., Concurrency as a Basis of System Thinking,  
ISF Report 78-06, GMD, Bonn, 1978.
- [4] Montanari U., Simonelli C., On distinguishing Concurrency  
from Nondeterminism,  
Progetto Finalizzato Informatica C.N.R. 7,  
Cnet, ETS/Pisa, 1980.
- [5] Prószyński P., Petri Nets and Concurrency-like Relations,  
Lecture Notes in Comp. Sci., vol.107, Springer-Verlag,  
1981, pp. 471-478.
- [6] Prószyński P., Remarks on the Notion of Concurrency Relation  
in the case of Systems,  
Lecture Notes in Comp. Sci., vol.117, Springer-Verlag,  
1981, pp. 311-320.

Mailing address: Piotr Prószyński  
Institute of Mathematics  
Warsaw Technical University  
Pl. Jedności Robotniczej 1  
00-661 Warsaw / Poland





IMPLEMENTATION OF ABSTRACT TYPES IN PL/I

Á. Hernádi

Hungary

1. Introduction

A main problem in the design and development of large software systems is reducing the amount of complexity or detail that must always be considered. Two common and effective solution methods are:

- decomposition, that is factoring a task into separable sub-tasks, and
- abstraction, that is providing a mechanism for separating attributes relevant in a given context from those which are not.

Procedural abstraction enables defining new mechanisms operating on old values.

Type abstraction enables defining new values and mechanisms to operate them.

A program should be able to define data of any type - either a primitive type of the base language /such as real, integer or string/ or a user-defined type such as a set, graph or other more complex objects such as a personal file, symboltable or the like. With these types certain operations are defined. For the primitive types the operations are defined by the language itself. For the user-created types the user must also specify the available operations. For example, the operations on a variable of type symboltable may be to add a new name, search or delete from the table.

Most of the recent work on embedding abstract data types into programming languages has emphasized the use of strong typing

and class-like constructions to provide isolation for the implementors of abstract types. Only those operations defined for a certain type, and no others, are permitted on data of that type. This is where most existing languages fail. Generally speaking, not only the creator, but any other routine has access to the contents of a variable of a new type. For example in FORTRAN the type stack may be simulated as an array. This array may be passed to subroutines which implement the various operations allowed on stacks. However, there is nothing to prevent any routine from modifying the array, and hence accessing the contents of a user-defined type.

To avoid these problems new languages have been devised, for much of which the class construction of SIMULA 67 has been used /e.g., CLU, Alphard and Euclid/.

## 2. Specification of Abstract Types

There are many possible approaches to specifying the semantics of the operations of abstract types. By a well-known classification most of them, however, can be placed in one of the two categories: operational or definitional.

In an operational specification, instead of describing the properties of the abstract type, a model should be built up for the type in terms of some well-understood language or discipline. The operational specifications often force one to overspecify the abstraction by introducing extraneous detail. In operational specification one must infer the properties of the abstract type from the properties of the operational model, so there is the risk of inferring unnecessary properties.

In a definitional specification one explicitly lists the properties that the values and operations forming the abstract type are to have. Only essential characteristics need be specified, thus the specification is an abstraction encompassing a relatively large class of implementations. In addition the absence of superfluous detail tends to increase the clarity of

the specification. The ability to state explicitly the properties of the operations makes the specification a better tool for formal reasoning.

One of the most prominent approaches to construct definitional specifications is the axiomatic specification of Hoare [HO'69], which is the most widely used as well. We shall look at two axiomatic approaches to the specification of abstract types: the approach suggested by Hoare [HO'72] and the algebraic axioms [GU'78a].

### 2.1. The Hoare Approach

Hoare's approach has enjoyed widespread use. Most of the users have departed in some ways from the notation originally used by Hoare. Here we shall use the notation of Alghard. We begin by looking at an example, a definition of the abstract notion of a stack independently of the kinds being stacked [WU'76]. In this case we shall allow only four operations:

"push" makes a new entry at the top of the stack,  
"pop" deletes the current top element of the stack,  
"top" returns the value of the current top element of the stack, and

"empty" returns "true" iff the stack is empty.

Alghard's abstraction mechanism, the form, provides encapsulation and support of type abstraction. The definition of the objects of an abstract type and the operations on them consists of three parts:

- the specifications, which constitutes the user's sole source of information about the form,
- the representation, which describes the representation and related properties of an object of this type, and
- the implementation, which contains the definitions of the functions that can be applied to an object.

All of the representational information in a form is inaccessible to the abstract program using the newly defined notion;

only those properties defined in the formal specification are accessible.

```

form stack (T:form<=>,n:integer)= (4)
beginform
specifications
  requires n>0;
  let stack=<...xi...> where xi is T;
  invariant 0<length(stack)<n;
  initially stack=nullseq;
  function push(s:stack,x:T) pre 0<length(s)<n post s=s'~x,
    pop(s:stack) pre 0<length(s)<n post s=leader(s') ,
    top(s:stack) returns (x:T) pre 0<length(s)<n
      post x=last(s') ,
    empty(s:stack) returns (b:boolean) post b=(s=nullseq);
representation
  unique v:vector(T,l,n),sp:integer init sp<-0;
  rep(v,sp)=seq(v,l,sp);
  invariant 0<sp<n;
  states mt when sp=0,
    normal when 0<sp<n,
    full when sp=n,
    err otherwise;
implementation
  body push out (s.sp=s.sp'+1^s.v=α(s.v',s.sp,x))=
    mt,normal::(s.sp<-s.sp+1;s.v[s.sp]←x);
    otherwise::FAIL;
  body pop out (s.sp=s.sp'-1)=
    normal,full::s.sp<-s.sp-1;
    otherwise::FAIL;
  body top out (x=s.v[s.sp])=
    normal,full::x←s.v[s.sp];
    otherwise::FAIL;
  body empty out (b=(sp=0))=
    normal,full::b←false;
    mt::b←true;
    otherwise::FAIL;
endform;

```

The relation between a concrete object and the abstract object represented may be expressed by the representation function, rep. Once a stack form is defined, instances of abstraction may be introduced into Alphard programs, for example, by declarations such as

```

  local si:stack(integer,35),sr:stack(real,14);

```

which makes "si" a stack of integers and "sr" a stack of reals.

-----  
 (4)"n" is the maximum permissible depth of a stack. The primed symbols in the post conditions and out assertions refer to the value of the symbol prior to the execution of the operation.

The important property of the language is the ability to separate the use of an abstraction from its concrete representation. The verification technique exploits this separation and permits the implementation /the form/ to be verified independently of the abstract program in which it is used.

Let us consider how to define this abstract type in CLU. In this language the cluster supports type abstraction. The first part of a cluster definition provides a very brief description of the interface which the cluster presents to its users. The remainder of the cluster definition contains three parts:

- the object representation,
- the code to create objects and
- the operation definitions.

```

stack=cluster[t:type] is create,push,pop,top,empty
  at=array[t]
  rep=record[sp:int,v:at]
  create=proc(n:int) returns (cvt)
    return(rep§{sp:n,v:at§new()})
  end create
  push=proc(s:cvt,x:t) signals (stackoverflow)
    if s.sp=at§size(s.v) then signal stackoverflow
    else at§addh(s.v,x)
    end
  end push
  pop=proc(s:cvt) signals(stackunderflow)
    at§remh(s.v)
    except when bounds: signal stackunderflow
    end
  end pop
  top=proc(s:cvt) returns (t) signals (stackunderflow)
    return(at§top(s.v))
    except when bounds: signal stackunderflow
    end
  end top
  empty=proc(s:cvt) returns(bool)
    return(at§size(s.v)=0)
  end empty
end stack
  
```

Within a cluster cvt can be used to "convert" the viewpoint between the abstract type being defined and the internal representation. We can introduce an object of stack, for example, by

```
is:stack [int]:=stack[int]§create (35)
```

which is equivalent to

```

is:stack[int]
is:=stack[int]§create(35).

```

CLU uses compound names for operations. The first part of the compound name identifies the type to which the operation belongs while the second component identifies the operation. As it seems CLU declarations include just the information that the compiler can check with reasonable efficiency. Other information required for proofs should be expressed in a separate "specification" language. Various specification language processors could be added to the system. Verification is decomposed: one module at a time is studied to determine whether it implements its abstraction.

Despite the fact that the various operations of type stack are intricately related to one another, these relationships are not directly expressed in the specifications of the type. Rather, stand-alone pre- and postconditions are supplied for each operation. This leads us to introduce a third domain of discourse in which to express the meanings of the operations.

## 2.2. Algebraic Specifications

An algebraic specification of an abstract type consists of three parts:

- a syntactic specification to provide syntactic and type checking information: the names, domains and ranges of the operations associated with the type;
- a semantic specification which is a set of axioms to define the meaning of abstractions by stating their relationships to one another; and
- a restriction specification which deals with preconditions and exception conditions.

```

type Stack [t:Type,n:Natural number] where ( )
syntax
newstack:           →Stack
push:      Stack × t →Stack
pop:       Stack    →Stack
top:       Stack    →Stack

```

```

empty:      Stack      → Boolean
*depth:     Stack      → Integer

```

(2)

semantics

```

declare s:Stack, x:t
1/ pop(push(s,x))=s
2/ top(push(s,x))=x
3/ empty(newstack)=true
4/ empty(push(s,x))=false
5/ depth(newstack)=0
6/ depth(push(s,x))=1+depth(s)

```

restrictions

```

pre(pop,s)=¬empty(s)
empty(s) ⇒ failure(top,s)
failure(push,s,x) ⇒ depth(s) ≥ n

```

The questions of consistency and completeness of axioms are discussed in [GU'78a]. The axioms are used as rewrite rules and proofs can be established via a series of reduction [GU'78b]. It is important to note that the techniques developed in the above papers are essentially independent of a specific programming language.

Thus the presence of axiomatic definitions of abstract types, either Hoare-like or algebraic specifications are used, provides a technique for factoring the proof into manageable sections. The main program expressed in terms of operations on abstract objects natural to the problem is verified by traditional methods, treating the specifications of the abstract objects and operations as if they were primitive. Then we have to verify whether the concrete implementation of each abstraction is consistent with its specification.

The Hoare-like approach is more convenient when the type abstraction is closely related to a type available in the underlying specification language.

The algebraic technique is more convenient for a type abstraction that is not readily represented or modelled by a well-known type.

-----  
(2) The \* indicates that depth is an auxiliary /"hidden"/ function, which may not appear as part of programs using the abstraction. Auxiliary functions are part of the specification of the abstraction but not of the abstraction itself.

### 3. Proposal for Incorporating Abstract Data Types into Language PL/I

While CLU, Alphard, Euclid, Ada and other such languages hold great promise experience, however, proves that a new language to get widely spread takes 15 to 20 years. It could help programmers today to add such facilities to existing languages - where possible.

In the socialist countries most commercial programming is done in PL/I. We defined a data definitional facility for the PL/I language that preserves most of the desirable features of data abstraction. Since our ultimate goal is to implement this with a preprocessor, we designed this mechanism to involve as few changes to PL/I as possible.

Now we describe the programming object whose preprocessing provides an implementation of a type. Let us consider the abstract data type stack again. In the extended language DEF\_TYPE modules support abstraction mechanism.

```
STACK: DEF_TYPE (N);
      /* LOCAL VARIABLES FOR SHARED INFORMATION AND
      OBJECT INITIALIZATION IF REQUIRED */
      DCL N BIN FIXED;
      DCL 1 STACK,
           2 SIZE BIN FIXED,
           2 ELEMENT (SIZE REFER (N)) ,
           2 SP BIN FIXED INIT (0);
      DCL S TYPE (STACK),
           E;
PUSH: FUNCTION (S,E);
      IF S.SP >= S.SIZE
      THEN SIGNAL CONDITION (STACKOF);
      S.SP = S.SP + 1;
      S.ELEMENT (S.SP) = E;
      ENDFUNCTION;
POP: FUNCTION (S);
      IF NOT EMPTY (S)
      THEN S.SP = S.SP - 1;
      ENDFUNCTION;
TOP: FUNCTION (S) RETURNS (DEC FLOAT (6));
      IF EMPTY (S)
      THEN SIGNAL CONDITION (STACKUF);
      RETURN (S.ELEMENT (S.SP));
      ENDFUNCTION;
```



```
EMPTY: FUNCTION (S) RETURNS (BIT (1)) ;
        RETURN (S.SP<=0);
        ENDFUNCTION;
END_DEF STACK;
```

The abstraction module specifies the representation as a structure variable having the same name as the type abstraction. Each FUNCTION-ENDFUNCTION pair defines a separate function /excluding GOTO statement/. So the abstraction mechanism is defined as several functions with shared information via any local variables of the DEF\_TYPE module.

Abstraction modules can have generic parameters too. Generic parameters play the same role in DEF\_TYPE modules as macro variables in macro definitions, and must be enclosed in brackets, [ ], immediatly after the keyword DEF\_TYPE.

```
STACK: DEF_TYPE[T] (N);
        /* LOCAL VARIABLES FOR SHARED INFORMATION AND
           OBJECT INITIALIZATION IF REQUIRED */
        DCL T;
        DCL N BIN FIXED;
        DCL I STACK,
            2 SIZE BIN FIXED,
            2 ELEMENT (SIZE REFER (N)) TYPE (T),
            2 SP BIN FIXED INIT (0);
        DCL S TYPE (STACK [T]),
            E TYPE (T);
PUSH: FUNCTION (S,E);
        ...
        ENDFUNCTION;
POP: FUNCTION (S);
        ...
        ENDFUNCTION;
TOP: FUNCTION (S) RETURNS (TYPE (T));
        ...
        ENDFUNCTION;
EMPTY: FUNCTION (S) RETURNS (BIT (1)) ;
        ...
        ENDFUNCTION;
END_DEF STACK;
```

These abstractions cannot be used directly compared to ordinary ones. Instances /that is copies/ of such an abstraction are obtained by binding the generic parameters to generic arguments in a special declare statement of the abstract program using the type:

```
DCLT STACK_I STACK [BIN FIXED];
DCL SI TYPE (STACK_I (35)) ;.
```

To declare names for abstract data types and to bind generic parameters to generic arguments in PL/I programs we introduced the DECLARETYPE /abbreviation DCLT/ statement, which has the following general format:

DECLARETYPE identifier typename  $\left[ \left[ g1 \left[ , g2 \right] \dots \right] \right]$   
 $\left[ , identifier \text{typename} \left[ \left[ g1 \left[ , g2 \right] \dots \right] \right] \right] \dots ;$  (3)

Syntax rules:

- Any number of identifiers may be declared in one DECLARETYPE statement.
- The name "identifier" is to be used in the PL/I program for the user-defined type "typename" with generic parameters substituted with the specified generic arguments, if there are any. A DECLARETYPE statement is valid iff "identifier" is a unique name and there is an abstract type "typename" defined, which has at least so many generic parameters as many generic arguments appear in this declaration.
- One need not supply generic arguments for each generic parameters, because an empty string is assumed as a default value. If no generic argument is to be specified, brackets may be ignored.
- The corresponding generic argument must match whenever a generic parameter is declared to be a type. If the corresponding generic argument is an abstract type itself, only the COPY operation of that type is available in the invoking abstraction, if there is any.
- For generic parameters not declared to be types generic arguments are not checked to match, and generic arguments are managed as if they were character strings starting with the first character after terminators [ or comma and ending at the next terminator comma or ].
- For each abstract type used there must be a DECLARETYPE statement, even if it has no generic parameters, and the program uses the name of the abstraction module.

We added a new attribute TYPE to specify the abstract type and arguments matching the parameters, if there are any in the

<sup>(3)</sup> [ ] brackets denote options, because square brackets, [ ] , enclose generic arguments.

DEF\_TYPE module. Dimension attribute which specifies bounds evaluable at preprocessing time, alignment attributes /ALIGNED, UNALIGNED/, scope attributes /INTERNAL,EXTERNAL/, and storage attributes /AUTOMATIC,STATIC,BASED,CONTROLLED/ may be supplied with the attribute TYPE.

The extended language, like CLU, uses compound names for the operations. For example:

```
DCLT STK_I STACK[BIN FIXED (31)] ;
      STK_R STACK[DEC FLOAT];
DCL SI TYPE(STK_I(35)), SR(5) TYPE(STACK_R(14))STATIC;
...
CALL STK_I$PUSH(SI,2);
...
A=STK_R$TOP(SR(3));
...
```

To implement the above facility the preprocessor must match two considerations:

- to generate correct PL/I code for correct programs
- to detect errors in the improper use of type abstractions.

We use the PL/I procedure as the basic structure to group data definitions and operations defined with ENTRY statements. The skeleton of a data abstraction modul produced by the preprocessor is the following:

```
typename: PROCEDURE ..., RECURSIVE RETURNS(POINTER);
          ALLOCATE typename SET $PTR ;
          /* LOCAL VARIABLES FOR SHARED INFORMATION AND
             OBJECT INITIALIZATION IF REQUIRED */
          ...
          DCL l typename BASED $PTR ,
          ...
          RETURN($PTR);
op1: ENTRY...;
      BEGIN;
          ...
          END;
          RETURN;
op2: ENTRY...;
      BEGIN;
          ...
          END;
          RETURN;
op3: ENTRY...;
      BEGIN;
          ...
          END;
          RETURN;
END typename;
```

In this case "typename" is the name of the abstract type being created and the entry points /op1, op2, and op3/ define the operations on this new data. The main entry point of the abstraction module has the special significance for the initialization of the abstract objects. Since the ALLOCATE statement is inserted immediately after the preprocessed DEF\_TYPE statement, and RETURN statement is inserted on scanning the first FUNCTION statement, additional initialization specified by the user, will be performed for each allocation of an abstract object. Information required for creating an object is passed in the parameterlist of the main entry point. To prevent control to pass around ENTRY statements in normal sequential flow /by forgetting the RETURN before the ENDFUNCTION/, the preprocessor inserts a RETURN statement for ENDFUNCTION. The representation of the data abstraction is restricted to this procedure, so other modules may only manipulate the object via ist defined operations, and may not alter its representation in any other manner.

The basic data abstraction is nothing more than a pointer variable. To prevent the "outside world" from gaining an access to the storage referenced by such pointers, their values are hidden in a static table. To add protection to TYPE pointers, it was necessary to store some more information with these pointers to identify the allocating abstraction module at run time. As a result a TYPE(....) attribute is substituted with CHAR(8) INITIAL((....)) attributes. The INITIAL attribute calls the main entry point of the abstraction module, thus TYPE variables are all initialized at the start of procedures, and each represents a unique storage structure. However the basic mechanism of PL/I, that is storage allocation on procedure entry and deletion on procedure exit in the case of AUTOMATIC variables, cannot be implemented as a whole, because upon exit from a procedure, storage allocated for abstract objects and not referenced any more cannot be freed. Since abstract objects are really pointers, in order to eliminate several variables sharing the same representation we do not allow abstract objects as target variables of assignment statements.

Instead a COPY operation should be specified for each abstract type - if desirable.

The preprocessor implementing these facilities accepts a series of modules as input. A module will usually be a DEF\_TYPE abstraction module or a PL/I procedure.

In the course of preprocessing PL/I procedures

- two ON-units are inserted to maintain run time errors such as incorrect use of an abstraction /when PL/I compiling and running is forced despite the error signalled during preprocessing/ and overflow of the static table hiding TYPE pointers;
- DECLARETYPE statements are ignored, and declarations of the entry points of abstract types defined correctly are inserted instead, if there are any. The original names appear in comment only, because internal names must be generated to satisfy the restrictions for external names;
- TYPE attributes are substituted as above mentioned, but INITIAL attributes are not given for parameters;
- names of abstract operations are substituted with the corresponding internal names;
- type checking is performed for TYPE variables;
- instances of abstract types defined correctly are attached to the PL/I program produced by the preprocessor as external procedures.

In the course of preprocessing DEF\_TYPE modules

- "DEF\_TYPE" is substituted with "PROCEDURE", generic parameters are ignored, and RECURSIVE and RETURNS attributes are added;
- ALLOCATE and RETURN statements are inserted as above discussed;
- "FUNCTION" is substituted with "ENTRY", and a BEGIN statement is inserted immediately after each ENTRY statement;
- ENDFUNCTION is implemented by END; RETURN; statements;
- abstract types other than the one being defined may be involved, and are managed in the same way as in PL/I modules;
- pointer values referring to storage allocated for abstract objects must be handled through the static table hiding

TYPE pointers;

- a description-unit must be built to contain information needed during preprocessing a module involving this type.

The preprocessor implementing these features is under development. The version which currently exists maintains only the PL/I procedures using abstract types. For this first approach we introduced some restrictions such as

- a name declared for an abstract data type may not appear as a generic parameter, so an abstract data type may not refer to another one;
- description-units containing information about the abstraction modules are in an index-sequentially organized file. A description-unit holds complete information about the generic parameters of the abstraction module, all parameter and returned value types of each operation in the abstraction module;
- the source code of the abstraction modules is in a partitioned file, in semi-preprocessed form;
- TYPE attribute is valid only for variables which are of first level with a level number implicitly declared. TYPE attribute cannot be factored, and EXTERNAL attribute may not be specified for TYPE variables;
- some keywords are considered to be reserved words, and their declaration as an identifier would lead to a meaningless result. These are the following:

```
BEGIN  
  DECLARE and DCL  
  DECLATETYPE and DCLT  
  DO  
  ELSE  
  END  
  ENTRY  
  IF  
  ON  
  PROCEDURE and PROC  
  RETURNS  
  THEN.
```

### Conclusion

It is certainly possible to use abstract types as a programming tool without actually making provision for them in the programming language. There are, however, several advantages to be gained from having a facility for the definition of abstract types within a programming language: the programs

which result are more modular, easier to understand, modify, maintain and prove correct.

### References

- DA'70 Dahl, O.J., Myhrhaug, B., Nygaard, K.:  
SIMULA 67 COMMON BASE LANGUAGE  
Publication S-22 Oct. 1970. Norwegian Computing  
Center. Forskningsvien 1B Oslo 3, Norway.
- GU'77 Guttag, J.V.:  
Abstract Data Types and the Development of Data  
Structures  
CACM, Vol. 20, pp.396-404, June 1977
- GU'78a Guttag, J.V., Horning, J.J.:  
The Algebraic Specification of Abstract Data Types  
Acta Informatica, Vol. 10, pp. 27-52, 1978
- GU'78b Guttag, J.V., Horowitz, E., Musser, D.R.:  
Abstract Data Types and Software Validation  
CACM, Vol. 21, pp. 1048-1064, Dec. 1978
- GU'80 Guttag, J.V.:  
Notes on Type Abstraction /Version 2/  
IEEE Trans. Software Eng., Vol. SE-6, pp. 13-23.  
Jan, 1980
- HO'69 Hoare, C.A.R.:  
An Axiomatic Basis for Computer Programming  
CACM, Vol. 12, pp. 576-580, Oct. 1969
- HO'72 Hoare, C.A.R.:  
Proofs of Correctness of Data Representations  
Acta Informatica, Vol. 1, No. 1, pp. 271-281, 1972
- LI'77 Liskov, B., Snyder, A., Atkinson, R., Schaffert, C.:  
Abstraction Mechanisms in CLU  
CACM, Vol. 20, pp. 564-576, Aug. 1977
- LI'78 Liskov, B. et al.:  
CLU Reference Manual  
Massachusetts Institute of Technology, Laboratory  
for Computer Science, Computation Structures  
Group Memo 161, July 1978
- WU'76 Wulf, W.A., London, R.L., Shaw, M.:  
An Introduction to the Construction and Verification  
of Alphard Programs  
IEEE Trans. Software Eng., Vol. SE-2, pp. 253-265,  
Dec. 1976
- ZE'78 Zelkowitz, M.V., Larsen, H.J.:  
Implementation of a Capability-Based Data Abstraction  
IEEE Trans. Software Eng., Vol. SE-4, pp. 56-64,  
Jan. 1978

Enterprise for Computing Application  
Büdapest, Pf. 146.  
H-1502  
Hungary





A Software for Computer System Performance  
Analysis - one more effort

T. Czachórski, M. Kowalówka, A. Wilk  
Poland

1. Introduction

Queueing network models have proven to be cost effective tools for evaluating computer systems. During the last two decades a variety of queueing models, including analytical /exact and approximate/, numerical and simulation ones, was developed and implemented. A practical use of these models involves generally a large amount of computations and the support of a computer is unavoidable, so every new conception is followed by its software counterpart. The quality of a model cannot be disputed in abstraction of its realization because even in the case of exact analytical models theoretically existing solution may be hardly obtainable in practice within reasonable time /cf. normalization constant calculation in product-form models/. In our opinion more effort was recently directed towards more effective computational algorithms for already existing models than towards creating conceptually new models.

Everybody who starts modelling real computer systems by means of queueing theory has to possess a software tool made more or less methodically at his disposal. Although numerous programs and program packages /reviewed in the next section/ were already developed and the descriptive literature of the subject is abounding, no code is obtainable /with - up to our knowledge - two exceptions: MARCA and SNAP, mentioned

later/. Thus we are obliged to start the work once again and build our own software.

We present here our project at the first stage of development. It is not decent to announce projects instead of realizations, we are perfectly aware of it. Nevertheless we think that possible reactions on this communiqué may influence our work and information that such a package will be available in not too far future may be of interest.

## 2. An overview of existing packages.

Making a brief survey of queueing network software one is obliged to come back to mid-sixties when RQA program [Wallace 66] was designed. RQA generated and solved Chapman-Kolmogorov equations /global balance equations/ for queueing networks which may be represented by a continuous-time Markov chain. Some other programs as MARCA [Stewart 76] and QSOLVE [Levy 77] were then constructed for the same purpose. As the steady - state solutions were sought, the question was reduced to solution of a system of linear algebraic equation and the numerical problems envisaged by the programs concerned dimensionality of the equations as well as operations on sparse matrices. The use of the programs is obviously restrained to relative simple networks whose number of states does not exceed few thousands.

New possibilities arose when the product-form-solution models were developed. Soon after Buzen's algorithms for computing steady-state probabilities in Gordon-Newell model /closed network with exponential servers and FIFO queues/ were published, the ASQ package [Keller 73] based upon these algorithms was developed; the extension of Jacksonian models due to Baskett, Chandy, Muntz and Palacios /BCMP model/ [Baskett et al. 75] resulted in several packages: QNET4 [Reiser 75], SNAP [Krzyszewski and Teunissen 77], PNET [Bruehl 78]. As for approximate methods, most popularity gained the iterative one, proposed by Chandy, Herzog and Woo /CHW model/ [Chandy et al. 75] which stimulated packages CADS [CADS 77] and IQNA [Reiser 78].

The discrete-event simulation has its own history and traditions. In this survey the programs QSIM [Gehearty 74] and APLOMB [Sauer 75] have to be mentioned. In addition to simulation they cope with the run length and confidence interval problems.

Multi-model packages, comprising programs for more than one model, opened a new generation of queueing software. The IBM's RESQ [Sauer et al. 77] is the union of QNET, APLOMB and IQNA programs, so it includes analytical, simulation and iterative modules. BEST/1 of BGS Systems [BEST 77] has /as one may guess knowing its functional scope/ an analytical part with BCMP model and the other with some approximate models. The French package QNAP [Merle et al. 78] includes BCMP model as well as iterative, diffusion approximation, Markovian and simulation modules. Recently the module based upon the mean value analysis was attached [Drix and Becker 81]

The input - output module is an important part of every package. In packages destined for user with little prior knowledge of queueing models the module covers majority /e.g. 90% in BEST/1 / of the whole code and is considerably smaller /e.g. 20% in PNET/ in university packages written for inner purposes. In the first case the interface module provides an interactive dialogue, in the latter case a model specification language is elaborated.

The language used to write a package is either Fortran /e.g. SNAP, QSIM, APLOMB, QNAP / or PL/I /e.g. RESQ, QSOLVE, IQNA/.

### 3. AMOK - the new package

Fig.1. presents the structure of our package.

Description of a considered model, written in a queueing network description language, is analysed by the program 'converter' and translated into a set of data available to resolution modules representing various queueing network modelling methods.

The description language distinguishes a certain number of objects, such as 'source', 'queue', 'server', 'resource', 'station' /i.e. server with queue/, each of them having its

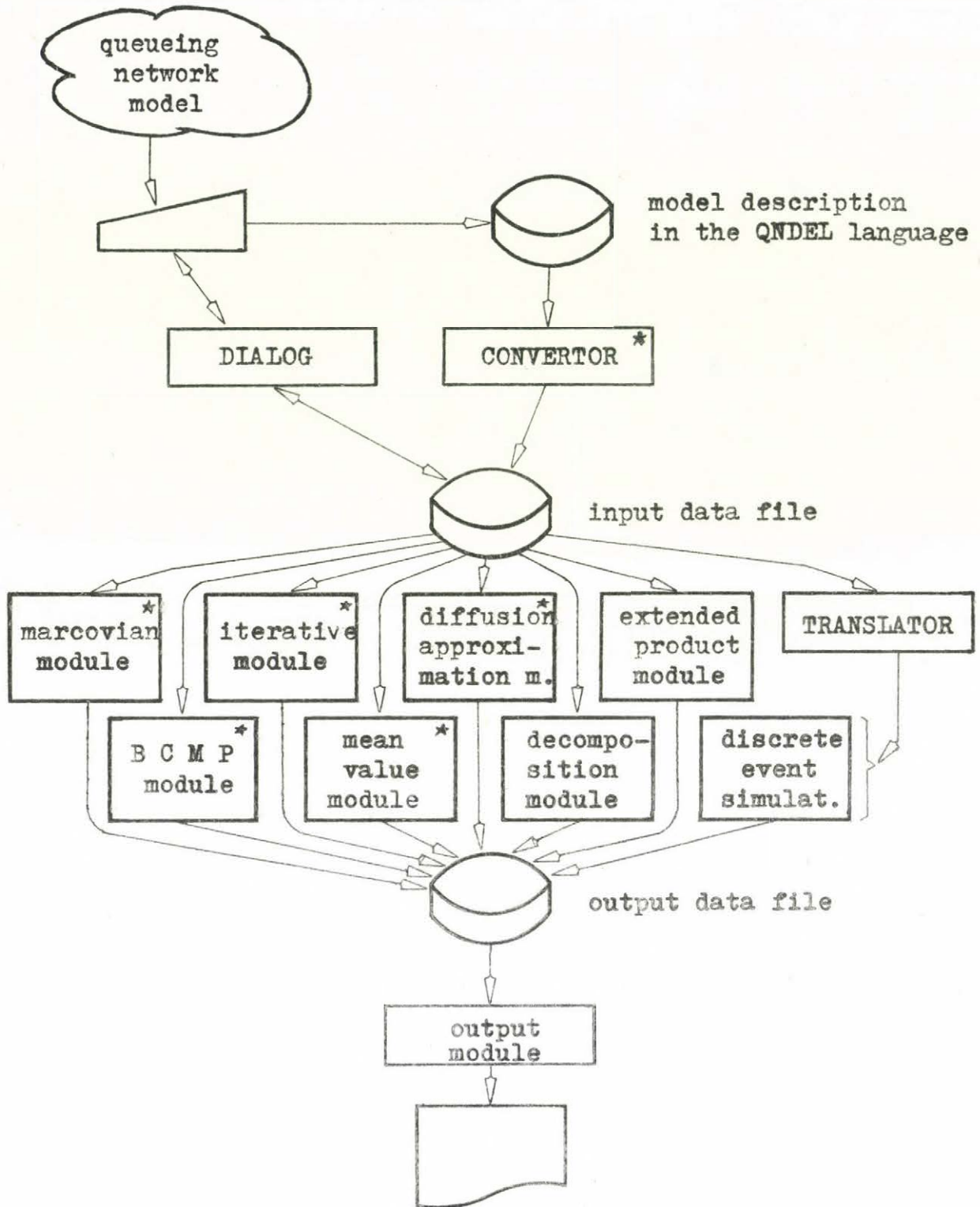


Fig. 1. The structure of AMOK.

\*/ modules implemented at the first stage of the project

parameters which express features of the object as well as the routing of customers among the objects.

Let us present the idea of the language using an example.

The structure of the network and its parameters are shown in Fig.2. There are terminals represented by infinite servers with Coxian and Erlangian service time distributions; CPU and disc are modelled by M/M/1 stations.

$b_i/x/$  denotes probability density function of service time at service station  $i$ ,  $i=1,\dots,4$ ;  $r_{ij}^l$  is routing probability between stations  $i$  and  $j$  within subchain  $l$ ,  $l=1,2$ , as two classes of customers are circulating.

This network has the following description in our language:

\*DESCRIPTION\*

/SERVER/ NAME = FIRST TERMINAL

SERVICE = COX [0.5,0.5,0.3,10]

TRANSIT = CPU : K1

/SERVER/ NAME = SECOND TERMINAL

SERVICE = ERL [10,8]

TRANSIT = CPU : K2

/STATION/NAME = CPU

SCHEDULING = FIFO

SERVICE = EXP [1.0]

TRANSIT (:K1) = [0.5]FIRST TERMINAL, [0.5] DISC

TRANSIT (:K2) = [0.3]SECOND TERMINAL, [0.7] DISC

/STATION/NAME = DISC

SCHEDULING = FIFO

SERVICE = EXP [1.5]

TRANSIT = CPU

\*END\*

Then the specifications for execution are matched.

When analytical models are concerned the language is similar to the realization which is in QNAP but has much more possibilities in the case of simulation models.

BCMP module implements well known and the most general of analytical product form solution models due to Baskett, Chandy, Muntz and Palacios [Baskett et al.75] with the job routing extension to the case of multiple chains as presented in

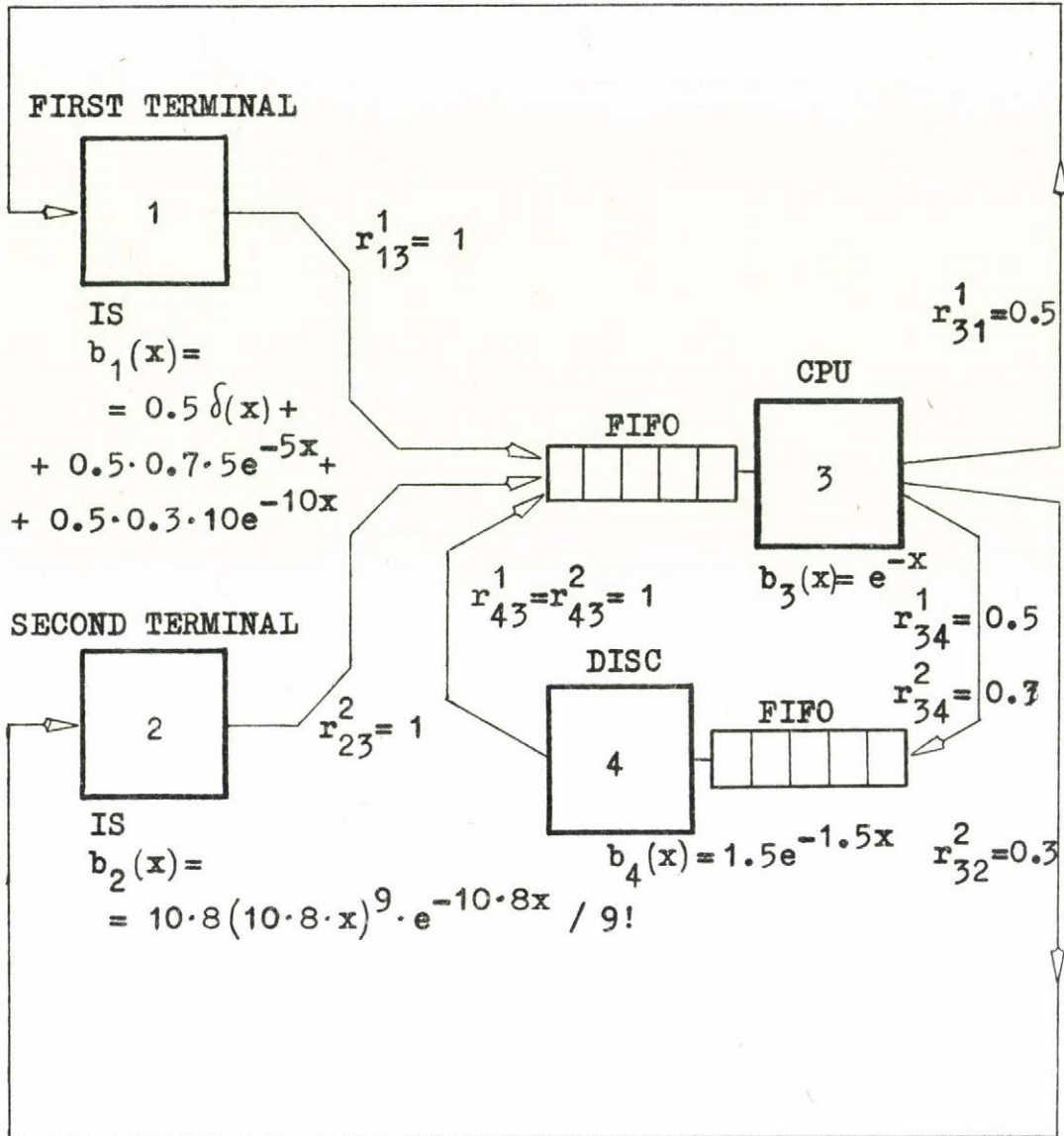


Fig. 2. An exemplary network.

[Reiser, Kobayashi 75]. The algorithms of the model make also use of [Merle 78], [Bruell 78], [Chandy, Sauer 80] and are based on the discrete convolution technique.

Mean value module is destined for the analysis of closed multi-chain queueing network and is based on computational algorithms developed in [Reiser, Lavenberg 80], [Chandy, Sauer 80].

The numerical approach is represented in our package by

Markovian module which solves the system of linear equations for the state occupancies of a closed queueing network that may be modelled by a continuous time Markov chain

$$\underline{R}^T \underline{P} = \underline{0} \quad \text{where } \underline{R} \text{ is the matrix of transition rates among states and } \underline{P} \text{ is the vector of the steady state probabilities.}$$

As  $\underline{R}$  is often sparse, the iterative methods are typically used to solve the equations. They do not change the content of the matrix which is therefore allowed to be remembered in a compact form [Stewart 78]. In AMOK however, a direct method was chosen /it is modified Jordan's elimination method with pivot element/; the use of a file handler for quick writing and retrieving record vectors makes this realization relatively efficient [Nałęcki 80].

The next three modules refer to approximation techniques which impose product form solutions in cases when it is not true. They are:

Extended product form /EPF/ module based on the scheme proposed by [Shum 76] where the joint distribution of a general queueing network is approximated by the product of  $M/G/1/N$  queues, the parameters of each being iterated until a set of thruputs satisfying flow balance is discovered.

Iterative module - implementing the device complement procedure proposed by [Chandy et al. 75] /CHW model/ with a modification [Marie 78] where  $\lambda(m)/C_k/1$  station serves as the equivalent server. The equivalent network is sought iteratively until conditions:

- sum of mean lengths of queues equal to number of customers in the network,

- Chang-Lavenberg theorem [Chang, Lavenberg 74] are satisfied.

Diffusion module is based on a diffusion approximation of GI/GI/1 station; the position of the diffusing particle corresponds to the number of customers present in the system. The version of instantaneous return process [Gelenbe 75] and of product form solution for the whole network [Gelenbe, Pujolle 75] were chosen. The extension to the model with priority scheduling will be provided [Czachórski 80].

The possibilities of decomposition of a network/to determine the cases where the above three models are more applicable/ will be checked by

Decomposition module, following criteria of [Courtois 77]. All cases non-tractable by the described modules will be treated by the Discrete event simulation module.

All the resolution modules are being written in standard Fortran and the package will be run on ODRA 1305 computer /equivalent of ICL 1900/ under George3 operating system. The modules are mutually independent so new ones, representing new methods or more efficient algorithms may be added. It is our intention to develop and maintain a program library gathering software representation of all outstanding methods for statistical modelling of computer systems.

Any type of cooperation in this field will be interesting for us.

#### REFERENCES

- F.Baskett, R.Muntz, M.Chandy, J.Palacios, "Open, Closed and Mixed Networks of Queues with Different Classes of Customers", Journal of the ACM, Vol. 22, No 2, pp.248-260 1975.
- "Best/1 Product Description", BGS Systems, BE 77-010-2, Lincoln, Massachusetts, 1977.
- S.Ch.Bruell, "On Single and Multiple class Queueing Network Models of Computer Systems", Ph.D.Thesis, Purdue University, 1978.



"Users Manual for the CADS Computer Analysis and Design Systems", Austin, Texas, 1977.

K.M.Chandy, U.Herzog, L.Woo, "Approximate Analysis of General Queueing Networks", IBM Journal of Research and Development, Vol. 19, No 1, pp.43-49, January, 1975.

K.M.Chandy, C.H.Sauer, "Computational Algorithms for Product Form Queueing Networks", Comm. of ACM, Vol. 23, No 10, pp.573-583, 1980.

A.Chang, S.S.Lavenberg, "Work rates in closed queueing networks with general independent servers", Operating Research, Vol. 22, pp.838-847, 1974.

P.J.Courtois, "Decomposability, Queueing and Computer System Applications", Academic Press, New York, 1977.

T.Czachórski, "A Multiqueue Approximate Computer Performance Model with Priority Scheduling and System Overhead", Podstawy Sterowania, Vol. 10, No 3, pp.223-240, 1980.

P.Drix, M.Becker, "Implantation dans QNAP d'algorithmes et d'heuristiques bases sur mean value analysis", Rapport de l'Institut de Programmation, Paris, 1981.

P.F.Gehearty, "QSIM - An Implementation of a language for analysis of queueing models", M.A.Thesis, Dept. of Comp. Sciences, University of Texas, Austin, Texas, 1974.

E.Gelenbe, "On approximate computer system models", Journal of the ACM, Vol. 22, No 2, pp.261-269, 1975.

E.Gelenbe, G.Pujolle, "The Behaviour of a Single Queue in a General Queueing Network", Acta Informatica, Vol. 7, Fasc. 2, pp.123-136, 1976.

T.W.Keller, "ASQ Manual", Dept. of Computer Sciences Report TR 27, University of Texas, Austin, Tx.1973.

A.Krzesinski, P.Teunissen, "Efficient Computational forms for the normalizing constant and the statistical measures of mixed multiclass queueing networks", Report RW 77-04, University of Stellenbosch, Dept. of Comp. Science, 1977.

A.I.Levy, "QSOLVE: A Queueing Network Solution System", TN-6, Computer Systems Research Group. University of Toronto, 1977

- R.Marie, "Modelisation par réseaux de files d'attente",  
Ph.D. Thesis, Université de Rennes, 1978.
- D.Merle, "Contribution à l'étude d'un analyseur de modèles  
à réseaux de files d'attente, algorithmes de calcul  
et applications", thèse de docteur ingénieur, l'Univer-  
sité National Polytechnique de Lorraine, 1978.
- D.Merle, D.Potier, M.Veran, "A tool for computer system per-  
formance analysis", in Performance of Computer Instal-  
lations", North Holland, Amsterdam, 1978.
- K.Nałęcki, "Numerical determination of the steady-state solu-  
tion in a system modelled by a Markov chain", Podstawy  
Sterowania, Vol. 10, No 3, 1980.
- M.Reiser, "QNET4 User's Guide", IBM Research Report RA-71,  
Yorktown Heights, New York, 1975.
- M.Reiser, H.Kobayashi, "Queueing Networks with Multiple Closed  
Chains Theory and Computational Algorithms", IBM J. of  
Research and Development, Vol. 19, No 3, May, 1975.
- M.Reiser, C.H.Sauer, "Queueing Network Models: Methods of  
Solution and their Program Implementation", in K.Mani  
Chandy, R.T.Yeh, /Edts./ "Current Trends in Programming  
Methodology, Vol. III Software Modeling", Prentice Hall,  
Englewood Cliffs, 1978.
- M.Reiser, S.S.Lavenberg, "Mean Value Analysis of Closed  
Multichain Queueing Network", J. of the ACM, Vol. 27,  
No 2, pp.313-322.
- C.H.Sauer, "Simulation Analysis of Generalized Queueing Net-  
works", Proceedings of 1975 Summer Computer Simulation  
Conference.
- C.H.Sauer, M.Reiser, E.A.MacNair, "RESQ - A Package for Solu-  
tion of Generalized Queueing Networks", Proceedings of  
1977 National Computer Conference, 1977.
- W.J.Stewart, "MARCA: Markov Chain Analyser", IRISA, Rapport  
No 45, Rennes, 1976.
- W.J.Stewart, "A comparison of numerical techniques in Markov  
modelling", CACM, Vol. 21, No 2, 1978.
- V.L.Wallace, R.S.Rosenberg, "Markovian Models and Numerical  
Analysis of Computer System Behavior", Proc.of Spring  
Joint Computer Conf. , 1966.

J.Zahorian, A.I.Levi, "An overview of the QSOLVE system",  
TR CSRG-83, Comp. System Research Group, University  
of Toronto, 1977.

---

Polish Academy of Sciences  
Complex Automation Dept.  
44-100 Gliwice, ul. Bałtycka 5  
Poland



## Data Base Performance in a Paging Environment

Marek Piwowarski - Poland

Abstract. Performance of the data base systems for real time applications can be defined as a steady-state expected number of page faults per one data base request. Using the concepts of buffer fault and terminal page reference it is possible to apply known models of paging memories to the analysis of such a data base. This enables the evaluation of the general expression for data base performance as a function of Buffer Replacement Algorithm and Search Strategy. Finally, the detailed explicit expressions of data base performance for LFU and LRU algorithms are evaluated.

### 1. Introduction

Performance of data bases for real time and process control system applications depends largely on the cost introduced by the I/O activities performed during the operation of a data base. It is so because applications programs utilizing the data base usually must satisfy some given time constraints i.e. maximal response time or minimal system throughput. Since the data base is usually stored in the auxiliary mass storage, transmissions between processor memory and storage constitute the main part of the data base I/O activity. When the physical data base is organized in pages such a transmission takes place during a page fault and frequency of page faults determines the I/O activity of the data base. Consequently, the expected number of page faults per data

base request can be chosen as an indicator of the data base performance.

During the run time the applications processes issue a string of record references /RRS/ to the Data Management System /DMS/. Each record reference /request/ is directed to a record stored in the data base storage, so DMS must bring it to the memory before processing. To do so DMS maintains the intermediate main memory buffer into which records are being brought from the data base. Since the buffer always contains some data, subsequent requests are serviced by first searching the buffer. If the referenced record is not found in the buffer the DMS must find and bring it from the data base. During this search process the page reference string /PRS/ is generated.

## 2. Data base buffer operations.

Let's assume that the data base consists of  $n$  pages containing  $d$  records and that the buffer maintained by the DMS has  $m$  page frames, /Fig. 1/.

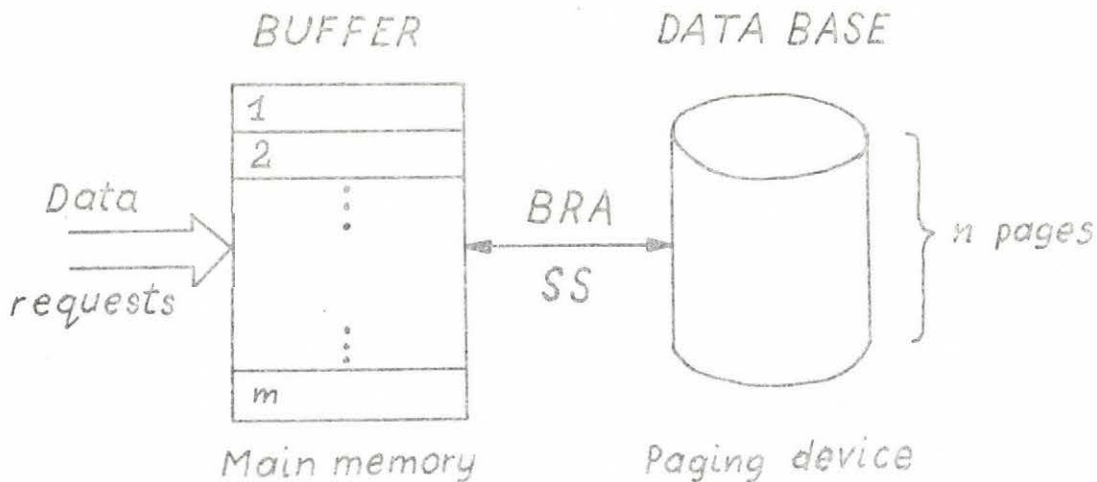


Fig. 1. Algorithms involved in the data base buffer operations.

When a request occurs the DMS first determines whether the

referenced record resides in the buffer. If the page containing the record is in the buffer, it is accessed by the requesting program. Otherwise the buffer fault occurs and two DNS mechanisms are being activated. The Search Strategy /SS/ generates the sequence of page references searching the data base for the requested record. If the page referenced in the search sequence is not in the buffer the page fault occurs and the buffer page has to be replaced by the referenced page. The corresponding algorithm is called the Buffer Replacement Algorithm /BRA/ and is responsible for choosing the buffer page for replacement. In case the buffer page chosen for replacement has been modified by the application program BRA must bring it back to the data base before releasing its buffer page frame.

### 3. The model.

Suppose that  $D = \{1, \dots, d\}$  is the set of database records,  $N = \{1, \dots, n\}$  is the set of data base pages and  $M = \{1, \dots, m\}$  is the set of buffer page frames,  $1 \leq m \leq n \leq d$ .

We will assume that records do not move within the data base /at least between data base reorganizations/. This can be described by the record placement mapping  $f: D \rightarrow N$  where  $f/j/=i$  means that record  $j$  resides on page  $i$ .

The model of application programs behaviour under which the data base performance will be evaluated is the known Independent Reference Model /IRM/ [1,2,3]. In this model the RRS is described by the sequence of independent identically distributed random variables

$$x_1, x_2, \dots, x_t, \dots \quad /1/$$

where  $t$  denotes the  $t$ -th record reference,  $t=1,2,\dots$ , with

$$P [x_t=i] = a_i \quad 1 \leq i \leq d \quad /2/$$

Let  $r_1, r_2, \dots, r_t$  denote the RRS. For each RRS we have the terminal page reference string /TPRS/ given by  $p_1, p_2, \dots, p_t$

where  $p_k = f/r_k$  ,  $1 \leq k \leq t$ .

It is easy to show that TPRS can be also modeled by the IRM in the form:

$$y_1, y_2, \dots, y_t, \dots \quad /3/$$

where  $t$  denotes the  $t$ -th terminal page reference with

$$P [y_t = i] = b_i = \sum_{k: f/k = i} a_k \quad 1 \leq i \leq n \quad /4/$$

Since the TPRS satisfies the IRM requirements we may apply the Aho, Denning and Ullmann model [1] of the BRA. Let  $r_t \in D$  be the  $t$ -th record request. BRA processes each RRS from initial state  $S_0$  by generating the sequence of configurations

$$\left\{ /S_t, q_t/ \right\}_{t=0}^T \text{ such that } /S_t, q_t/ = g /S_{t-1}, q_{t-1}, p_t/ \text{ where}$$

$p_t = f/r_t$  is the terminal page reference in moment  $t$ ,

$S_t \in N$  is a buffer state in moment  $t$ ,

$q_t \in Q$  is a control state in moment  $t$ ,

$g$  is the allocation map given by the BRA.

For such a description we have that buffer fault occurs iff  $p_t \notin S_{t-1}$  and that in order to minimize the long-run buffer-fault rate BRA must be a demand algorithm with respect to TPRS.

#### 4. Cost of algorithms.

Let  $p_t = f/r_t$  be a terminal page reference in moment  $t$ .

Unlike in the virtual memory systems where  $p_t$  is calculated in a single step by the address mapping hardware, DMS must find  $p_t$  according to the SS utilized. This results in a search page reference string /SPRS/ as below:

$$\text{SPRS}/r_t/ = \begin{cases} p_t^1, p_t^2, \dots, p_t^{n_t} = f/r_t/ & \text{buffer fault occurs} \\ \text{none} & \text{otherwise} \end{cases} \quad /5/$$

Note, that the SPRS is uniquely determined by the SS for a given record and constant for each record. Hence, the number



of search page references /length of the SPRS/  $n_t$  depends on both record being searched and the SS. Obviously, the number of search page faults  $m_t$  depends additionally on the BRA and satisfies the following inequality:

$$m_t/r_t, SS, BRA/ \leq n_t/r_t, SS/ \quad /6/$$

In a steady state the expected number of page faults performed during the search for terminal page  $i$  denoted by  $S_i/SS, BRA/$  is the following:

$$S_i/SS, BRA/ = E \left[ m_t/r_t, SS, BRA/ \right], f/r_t/=i \quad /7/$$

The number of page transmissions from the buffer to the data base /page removals/ is equal to 1 when the buffer page chosen for replacement has been modified and 0 otherwise.

Taking all this into account it is easy to show that the steady state expected number of page faults per data base request is given by the following expression:

$$C = \sum_{i=1}^n b_i \cdot X_i/BRA/ \cdot \left[ S_i/SS, BRA/ + R_i/BRA/ \right] \quad /8/$$

where  $X_i/BRA/$  is the probability of buffer fault due to the reference to page  $i$ ,  $R_i/BRA/$  is the removal probability for the buffer fault due to the terminal reference to page  $i$ . The above general expression for data base performance is of course subject to evaluation for each specific BRA and SS. Hereafter we present such an evaluation for two most often used BRAs: LFU /Least Frequently Used/ and LRU /Least Recently Used/ because Gelenbe has shown [2] that RR /Random Replacement/ and FIFO /First-In-First-Out/ algorithms have equal and the worst performances of all BRAs.

### 5. Evaluation of $X_i/BRA/$ .

Let's number the data base pages such that  $b_1 \gg b_2 \gg \dots \gg b_n$ .

Theorem 1.

$$X_i/LFU/ = \begin{cases} 0 & 1 \leq i \leq m-1 \\ 1 - b_i \cdot Y & m \leq i \leq n \end{cases} \quad /9/$$

where  $Y = 1 / \sum_{k=m}^n b_k$

Proof - in the Appendix.

Theorem 2.

$$X_i / \text{LRU} / = 1 - b_i \cdot W_i \quad /10/$$

where

$$W_i = \sum_{j=0}^{m-1} \sum_{k=0}^{m-1-j} (-1)^k \binom{n-1-j}{k} Q_{n-1,j}^i$$

where

$$Q_{n-1,m}^i = \sum_{1 \leq j_1 < \dots < j_m \leq n-1} \frac{1}{1-x_{j_1} \dots -x_{j_m}}$$

where  $X = \{x_i\} = \{b_1, b_2, \dots, b_{i-1}, b_{i+1}, \dots, b_n\}$

Proof- in the Appendix.

Corollary 1.

LFU is the optimal BRA with respect to the expected number of buffer faults.

Proof - in the Appendix.

### 6. Evaluation of the $S_i / \text{BRA, SS} /$ .

Exact evaluation of  $S_i / \text{BRA, SS} /$  though possible for any combination of BRA and SS is often quite laborious. For some cases it is very simple as in the below Example.

Example 1.

$$S_i / \text{LFU, sequential} / = \begin{cases} 0 & 1 \leq i \leq m-1 \\ 1 - b_m Y & i = m \\ i - m + 1 - b_m Y & m < i \leq n \end{cases} \quad /11/$$

where Y is given by /9/.

In especially difficult cases we may apply the expected number of search page references for page i,  $F_i / \text{SS} /$  instead of the expected number of search page faults  $S_i / \text{BRA, SS} /$  according to the below inequality :

$$\max_i [F_i / \text{SS} /] \geq F_i / \text{SS} / \geq S_i / \text{BRA, SS} / \quad , 1 \leq i \leq n \quad /12/$$

In this case we obtain the upper bound for the data base performance. It is especially advantageous because  $F_i/SS/$  has been widely investigated for various data structures and access methods.

7. Evaluation of the  $R_i/BRA/$ .

Each request may either change a record's contents /modification/ or leave it unchanged /retrieval/. Assuming that the probability of retrieval is constant, equal for all requests in the RRS and denoted by  $P_r$ , we have :

Theorem 3.

$$R_i/LFU/ = 1 - P_r \cdot Y \sum_{\substack{j=1 \\ j \neq i}}^n \frac{b_j / 1 - b_j \cdot Y/}{1 - b_i \cdot Y - b_j \cdot Y \cdot P_r} \quad /13/$$

where  $Y$  is again given by /9/.

Proof - in the Appendix.

Theorem 4.

$$R_i/LRU/ \approx 1 - P_r \sum_{\substack{j=1 \\ j \neq i}}^n \frac{X_j/LRU/}{1 + X_j/LRU/ \cdot P_r - P_r} \quad /14/$$

where  $X_j/LRU/$  is given by /10/.

Proof - in the Appendix.

8. Conclusions.

Results of the above analysis show the impact of the BRA and the buffer size on the data base performance. This provides better support especially at the physical data base design stage giving at least the upper bound of the data base performance. In case when the DMS buffer is itself in the virtual memory /such as in the IBM's IMS/ the above approach yields input information for appropriate models /e.g. Lang, Wood and Fernandez [6] or Sherman and Brice [7] models/.

The area of practical implementations will of course depend on the extensive simulation support needed for verification of the above approach.

THE APPENDIX

Proof of Theorem 1.

In steady-state LRU keeps  $m-1$  page frames constantly occupied with  $m-1$  most frequently accessed pages. Hence, the buffer fault probability due to the reference to pages  $1, \dots, m-1$  is equal to 0. Reference to page  $i$ ,  $m \leq i \leq n$ , causes the buffer fault iff the last reference to page  $j$ ,  $m \leq j \leq n$ , was such that  $j \neq i$ . Probability of such an event is equal to

$$1 - b_i / \sum_{k=m}^n b_k .$$

Proof of Theorem 2.

Hereafter we will apply the Knuth's approach [4] to the analysis of the selforganizing files. Instead of saying that page  $i$   $m$ -th recently used we will say that page  $i$  is in position  $m$  in the buffer.

Let  $f_m / x_1, x_2, \dots, x_m$  be the sum of all distinct ordered products  $x_{i_1}, x_{i_2}, \dots, x_{i_k}$  such that  $1 \leq i_1 \leq \dots \leq i_k \leq m$  where each of  $x_1, x_2, \dots, x_m$  appears in every term.

Hence,  $f_{m-1} / x_{i_1}, x_{i_2}, \dots, x_{i_{m-1}} \cdot b_i$  are probabilities of possible sequences of requests leaving page  $i$  in position  $m$ .

Setting

$$p_{nm}^i = \sum_{1 \leq j_1 < j_2 < \dots < j_m \leq n} f_m / x_{j_1}, \dots, x_{j_m} / \quad /A1/$$

Knuth shows that page  $i$  will be in position  $k$  in the buffer with probability  $b_i \cdot p_{n-1, k-1}^i$  where

$$X = \{ b_1, b_2, \dots, b_{i-1}, b_{i+1}, \dots, b_n \} . \quad /A2/$$

Again Knuth proves that

$$P_{nm}^i = Q_{nm}^i - \binom{n-m+1}{1} \cdot Q_{n,m-1}^i + \dots + (-1)^m \binom{n-m+m}{m} \cdot Q_{n0}^i \quad /A3/$$

where by convention  $P_{n0}^i = Q_{n0}^i = 1$ ,  $i, n = 1, 2, \dots$

and where

$$Q_{nm}^i = \sum_{1 \leq j_1 < \dots < j_m \leq n} \frac{1}{1 - x_{j_1} - \dots - x_{j_m}} \quad /A4/$$

where  $X$  is given by /A2/.

Probability of page fault due to the reference to page  $i$  is equal to  $1 - \text{Probability that page } i \text{ is in the buffer.}$

Hence,

$$X_i = 1 - \sum_{j=1}^m b_i \cdot P_{n-1, j-1}^i \quad /A5/$$

Substituting /A3/ to /A4/ and reversing the order of summation finally yields /10/.

Proof of the Corollary 1.

The steady-state probability of the buffer fault is :

$$F/B/ = \sum_{i=1}^n b_i \cdot X_i /BRA/ \quad /A6/$$

after substituting /9/ to /A6/ we have :

$$F/B/LFU = \sum_{i=m}^n b_i \cdot /1 - b_i/Y / = \frac{\left(\sum_{i=m}^n b_i\right)^2 - \sum_{i=m}^n (b_i)^2}{\sum_{i=m}^n b_i}$$

which is equal to the expression obtained by King [5] and Gelenbe [2] for the algorithm proved to be optimal in [1].

Proof of Theorem 3.

Suppose that buffer fault in moment  $t$  was caused by the refe-

rence to page  $i$ . Thus, if the page to be replaced is page  $j$   $m \leq j \leq n$ ,  $j \neq i$ , its conditional probability of reference in moment  $t-1$  is equal to

$$\frac{b_j}{1/Y - b_i} \quad /A7/$$

Therefore, probability that page  $j$  was exactly  $k$  times retrieved before replacement is :

$$\left( \frac{b_j \cdot Y \cdot P_r}{1 - b_i \cdot Y} \right)^k \cdot /1 - b_j \cdot Y/ \quad /A8/$$

Hence, probability that page to be replaced leaves buffer unmodified during the buffer fault due to page  $i$  is :

$$1 - R_i = \sum_{\substack{j=m \\ j \neq i}}^n \sum_{k=1}^{\infty} /b_j \frac{Y \cdot P_r}{1 - b_i \cdot Y} /^k \cdot /1 - b_j \cdot Y/ \quad /A9/$$

Finally, we obtain /13/ after the evaluation of the infinite geometric series.

Proof of Theorem 4.

Suppose, as before, that buffer fault is due to page  $i$  and that page to be replaced is some page  $j \neq i$ . For  $n \gg m$  we may write that page  $j$  leaves the buffer after exactly  $k$  successive retrievals with approximate probability  $X_j / LRU / \cdot /1 - X_j /^{k-1} \cdot P_r^k$ . Hence, probability that page to be replaced leaves buffer unmodified after  $k$  successive retrievals is approximately

$$\sum_{\substack{j=1 \\ j \neq i}}^n X_j / LRU / \cdot /1 - X_j /^{k-1} \cdot P_r^k \quad /A10/$$

Finally, probability that page to be replaced leaves buffer unmodified may be estimated as :

$$1 - R_i / LRU / = \sum_{k=1}^{\infty} \sum_{\substack{j=1 \\ j \neq i}}^n X_j /^{k-1} \cdot P_r^k =$$

$$= P_r \sum_{\substack{j=1 \\ j \neq i}}^n \frac{X_j / \text{LRU}}{1 - P_r + P_r \cdot X_j / \text{LRU}} \quad /A11/$$

#### REFERENCES

- [1] Aho, A.V., P. Denning, J.D. Ullmann, Principles of optimal page replacement, J. Assoc. Comp. Mach., v.18, 1971
- [2] Gelenbe, E., A unified approach to the evaluation of a class of replacement algorithms, IEEE Trans. Comput., v.22, 1973.
- [3] Aven, O., L. Boguslavsky, Y. Kogan, Some results on distribution-free analysis of paging algorithms, IEEE Trans. Comput., v.25, 1976.
- [4] Knuth, D.E., The art of computer programming /Volume 3 : Sorting and Searching/, Addison-Wesley Publishing Co. Reading, Mass. 1973.
- [5] King, W.F., Analysis of paging algorithms, Proc. IFIP Congress /Ljubljana, Yugoslavia/, 1971.
- [6] Lang, T., C. Wood, E.B. Fernandez, Database buffer paging in virtual storage systems, ACM Trans. on Database Systems, v.2, 1977.
- [7] Sherman, S.W., R.S. Brice, Performance of a database manager in a virtual memory system, ACM Trans. on Database Systems, v.1, n.4, 1976.

Marek Piwowarski  
Polish Academy of Sciences  
Dept. of Complex Control Systems  
Bałtycka 5  
44-100 GLIWICE, Poland





Performance Evaluation of Computing System  
Subject to Failures

Andrzej Duda, Poland

**Abstract.** The paper presents an analysis of program performance in the computing system subject to failures. Program checkpointing is considered as a method to reduce the overhead due to restarts after failures. The distribution and the expectation of the elapsed time are derived for programs with and without checkpointing. The optimum interval between checkpoints is given. A comparison between performance of a program with and without checkpointing is provided and shows when the use of checkpointing is beneficial.

## 1. Introduction

Continuous and reliable operation of computing systems depends not only on reliable hardware but also on the operating system and software mechanisms which enable error tolerance [DENN 76], [RAND 75]. It is interesting to study the performance of computing systems that use recovery techniques to cope with failures. Traditional approach to performance evaluation [FERR 78] often does not take into account the impact that system reliability has on the system performance. The performance measures such as throughput, elapsed time, response time reflect the efficiency of a system for various workloads. However the system performance may be degraded due to failures and also such techniques as dynamic testing, error detection, recovery decrease the computation capacity of the system. It is important from the user viewpoint to characterize

both performance and reliability of computing systems. Several attempts have been made to develop the reliability and performance characteristics of computing systems [BEAU 78], [MEYE 80]. We want to deal with a measure that gives the information about the execution time of a program in the computing system subject to failures.

A program may not terminate its execution because of failures which can force the program to be restarted. The model of the computing system considering such performance measures as apparent capacity and expected elapsed time required to correctly execute a given program was proposed by Castillo and Siewiorek [CAST 80]. This model assumes two types of errors fatal and nonfatal. After fatal errors the program is restarted. The time to the program termination depends also on the system workload. Statistics of the workload gathered from a real system are used to derive the distribution of the apparent capacity and the expected elapsed time. The analysed scheme of the system's operation necessitates starting a fresh run after the fatal error from the beginning of the program. The work that was done is repeated and therefore effectively wasted.

We analyse a program checkpointing as a method to reduce the overhead due to restarts. Checkpointing allows the program to be resumed from the earlier point in its execution and not from the beginning. Brock [BROC 79] has analysed and evaluated the expected execution time of programs with and without checkpointing. In this paper we derive the distribution and the expectation of the elapsed time assuming program checkpointing. We consider the optimum interval between checkpoints. Finally a comparison between performance of the program with checkpointing and the program without checkpointing is presented.

## 2. Measures of system performance

Let us define the elapsed time  $T_E$  as the time required to correctly execute a program in a given computing system. The minimum execution time  $T_{Min}$  is the time needed to the program execution in a single-programming system when no failures are present. Assuming that the program is executed in the multi-programming, multiuser system and it may stop because of system

failures, the elapsed  $T_E$  depends on  
a/ frequency and duration of failures,  
b/ workload of the system,  
c/ minimum execution time  $T_{Min}$ .

The apparent capacity [CAST 80]

$$Ac = \frac{T_{Min}}{T_E} \quad /2.1/$$

determines what is the fraction of the total system capacity that the user receives. Since  $T_E$  is a random variable whose distribution depends on the statistics of the system reliability, the statistics of the workload, and the time  $T_{Min}$ , the distribution of the apparent capacity can be calculated. In order to make our model simpler we consider only the influence of the system reliability and the execution time on the elapsed time. The influence of the workload we take into account implicitly through the execution time  $T$ . Then we assume that  $T$  is the time of continued system operation required in the given multiprogramming environment /the system shared with other users/ to execute the program. Below we consider the distribution and the expectation of the elapsed time in order to compare quantitatively the performance of the program with and without checkpointing and to obtain the parameters that optimize the expected elapsed time.

### 3. Program without checkpointing

The execution of a program in the computing system is illustrated in Figure 1.

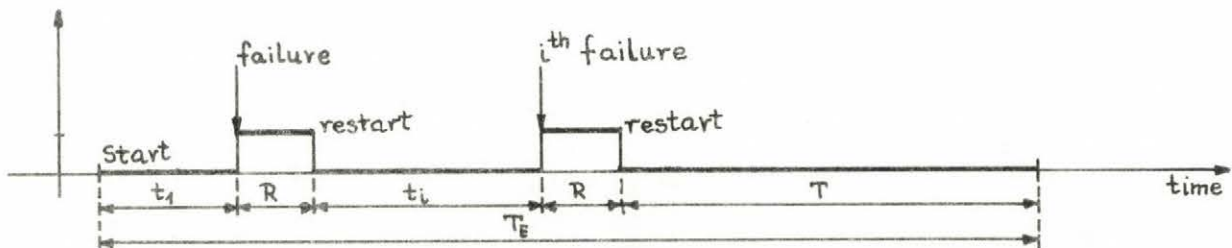


Figure 1. Execution of a program without checkpointing

The program requires the time  $T$  to its execution. The  $i$ -th failure is detected after the time  $t_i$  since the last restart.

The program can be restarted after the time  $R$  necessary to reestablish the normal operation of the system. The program terminates after the time  $T$  of continued operation.

In order to derive the distribution and the expectation of  $T_E$  we adopt the following assumptions:

1. The instants of the failures' occurrences form a homogeneous Poisson process of parameter  $\gamma$ .
2. Failures do not occur when the system is recovering from a failure.
3. We regard the instants of the failures' detection as the instants of their occurrences.
4.  $R$ ,  $T$ ,  $\gamma$  are constants and do not change during program execution.

Under these assumptions the sequence of intervals  $t_1, t_2, \dots, t_i, \dots$  forms a renewal process i.e. the periods of time between a restart and the next failure form a sequence of independent, identically distributed random variables.

The probability density function of  $T_E$  can be written as

$$f_{T_E}(\alpha) = \sum_{k=0}^{\infty} P(n=k) f_{T_E|n=k}(\alpha), \quad /3.1/$$

where

$$P(n=k) = e^{-\gamma T} (1 - e^{-\gamma T})^k \quad /3.2/$$

is the probability of  $k$  occurrences of failures during the execution time  $T$  and  $f_{T_E|n=k}$  is the p.d.f. /probability density function/ of  $T_E$  given that the program is restarted  $k$  times.

We have

$$T_E = T + t + kR, \quad t = \sum_{i=1}^k t_i \quad /3.3/$$

The interval  $t_i$  is the time between a restart and the next failure provided the next failure occurs before the time  $T$ . So, the distribution  $f_{t_i}$  is a truncated exponential distribution

$$f_{t_i}(\beta) = \frac{1}{1 - e^{-\gamma T}} \gamma e^{-\gamma \beta} [1(\beta) - 1(\beta - T)], \quad /3.4/$$

where  $1(\beta)$  is the unit step function.

Since  $t$  is the summation of  $k$  independent, identically distributed random variables  $t_i$ , the Laplace transform of its distribution can be expressed as

$$\bar{f}_{t|n=k}(s) = \left\{ \frac{1}{1 - e^{-\gamma T}} \cdot \frac{\gamma}{\gamma + s} [1 - e^{-T(\gamma + s)}] \right\}^k \quad /3.5/$$

that is

$$\bar{f}_{t_i|n=k}(s) = \left( \frac{1}{1-e^{-\gamma T}} \cdot \frac{\gamma}{\gamma+s} \right)^k \sum_{l=0}^k \binom{k}{l} (-1)^l e^{-lT(\gamma+s)} \quad /3.6/$$

Then, the Laplace transform of the p.d.f.  $f_{T_E|n=k}$  is given by

$$\bar{f}_{T_E|n=k} = e^{-(T+kR)s} \left( \frac{1}{1-e^{-\gamma T}} \cdot \frac{\gamma}{\gamma+s} \right)^k \sum_{l=0}^k \binom{k}{l} (-1)^l e^{-lT(\gamma+s)} \quad /3.7/$$

The transform of  $f_{T_E}$  can be written as

$$\bar{f}_{T_E}(s) = \sum_{k=0}^{\infty} e^{\gamma T} e^{-(T+kR)s} \left( \frac{\gamma}{\gamma+s} \right)^k \sum_{l=0}^k \binom{k}{l} (-1)^l e^{-lT(\gamma+s)} \quad /3.8/$$

After computing the inverse Laplace transform we obtain

$$f_{T_E}(\alpha) = e^{-\gamma T} \sum_{k=0}^{\infty} \gamma^k \sum_{l=0}^k \binom{k}{l} (-1)^l \frac{[\alpha - T(l+1) - kR]^{k-1}}{(k-1)!} e^{-\gamma[\alpha - T(l+1) - kR]} \mathbb{1}[\alpha - T(l+1) - kR] \quad /3.9/$$

In order to compute the expected elapsed time we derive first the expected value of  $t_i$  through the differentiation of the Laplace transform  $\bar{f}_{t_i}(s)$  and setting  $s \rightarrow 0$

$$E(t_i) = \frac{1}{\gamma} - \frac{T e^{-\gamma T}}{1 - e^{-\gamma T}} \quad /3.10/$$

So, we have

$$E(T_E|n=k) = T + k \left( R + \frac{1}{\gamma} - \frac{T e^{-\gamma T}}{1 - e^{-\gamma T}} \right) \quad /3.11/$$

and the following expression can be obtain

$$E(T_E) = \sum_{k=0}^{\infty} e^{-\gamma T} (1 - e^{-\gamma T})^k \left[ T + k \left( R + \frac{1}{\gamma} - \frac{T e^{-\gamma T}}{1 - e^{-\gamma T}} \right) \right] \quad /3.12/$$

Finally the summation yields

$$E(T_E) = \left( R + \frac{1}{\gamma} \right) (e^{\gamma T} - 1) \quad /3.13/$$

This expression shows that the expected elapsed time depends strongly on the failure rate and may attain large values. Note that  $E(T_E)$  is an exponential function of the execution time  $T$ .

#### 4. Program with checkpointing

Checkpointing consists in writing an appropriate information to checkpoint file /usually stored in the auxiliary memory/ at some instants during the run of a program that allows its execution to be resumed not from the beginning. The creation of the checkpoint includes the following information:

- a/ program store image or data areas if the program area is not altered,
- b/ contents of relevant registers,
- c/ current positions reached in the files accessed by the program.

When the system is repaired after a failure, the state of the

program is reestablished by:

- a/ reconstitution of the program store image,
- b/ resetting the relevant registers,
- c/ repositionning the files,
- d/ resuming the execution of the program.

The execution of a program with checkpointing is presented in Figure 2.

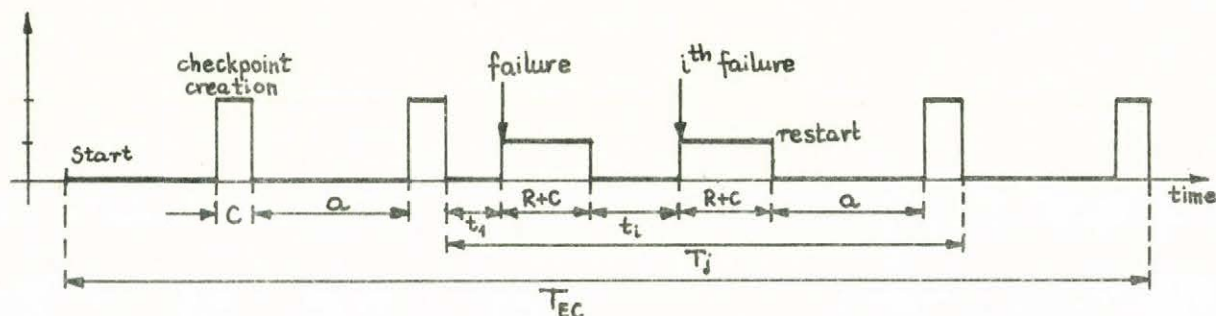


Figure 2. Execution of a program with checkpointing

The checkpoint is created after the constant time  $a$  of continued system operation. The  $i$ -th failure occurs after time  $t_i$  since the resuming of the program from the last checkpoint. The system is down during the time  $R$ . The checkpoint creation as well as the program resuming last the time  $C$ .  $T_{EC}$  denotes the the elapsed time of program with checkpointing.

We adopt the following assumptions:

1. The instants of the failures occurrences form a homogeneous Poisson process of parameter  $\lambda$ .
2. Failures do not occur when the system is recovering from a failure and when it is creating a checkpoint.
3. We regard the instants of the failures detection as the instants of their occurrences.
4.  $R, T, C, \lambda, a$  are constants and do not change during program execution.

5.  $N$  checkpoints are created during the time  $T$  i.e.  $T = N \cdot a$

Under these assumptions the sequence of intervals  $t_1, t_2, \dots, t_i, \dots$  and also the sequence of total times between the ends of consecutive checkpoints  $T_1, T_2, \dots, T_j, \dots$  form renewal processes. We begin with calculation of the p.d.f.  $f_{T_j}$

$$f_{T_j}(\alpha) = \sum_{k=0}^{\infty} P(n=k) f_{T_j|n=k}(\alpha)$$

/4.1/

where

$$P(n=k) = e^{-\gamma a} (1 - e^{-\gamma a})^k \quad /4.2/$$

is the probability of k occurrences of failures during the interval a and  $f_{T_j|n=k}$  is the p.d.f. of  $T_j$  provided the program is resumed k times from the last checkpoint.

We can write

$$T_j = a + k(R+C) + t + C \quad t = \sum_{i=1}^k t_i \quad /4.3/$$

The p.d.f.  $f_{t_i}$  is a truncated exponential distribution

$$f_{t_i}(\beta) = \frac{1}{1 - e^{-\gamma a}} \gamma e^{-\gamma \beta} [1(\beta) - 1(\beta - a)] \quad /4.4/$$

where  $1(\beta)$  is the unit step function. The Laplace transform of the p.d.f.  $f_{t|n=k}$  can be written as

$$\bar{f}_{t|n=k}(s) = \left\{ \frac{1}{1 - e^{-\gamma a}} \frac{\gamma}{\gamma + s} [1 + e^{-a(\gamma + s)}] \right\}^k \quad /4.5/$$

because t is the sum of k independent, identically distributed random variables  $t_i$ . Then we have

$$\bar{f}_{t|n=k}(s) = \left( \frac{1}{1 - e^{-\gamma a}} \frac{\gamma}{\gamma + s} \right)^k \sum_{l=0}^k \binom{k}{l} (-1)^l e^{-la(\gamma + s)} \quad /4.6/$$

The Laplace transform of the p.d.f.  $f_{T_j|n=k}$  is given by

$$f_{T_j|n=k}(s) = e^{-[a+C+k(R+C)]s} \left( \frac{1}{1 - e^{-\gamma a}} \frac{\gamma}{\gamma + s} \right)^k \sum_{l=0}^k \binom{k}{l} (-1)^l e^{-la(\gamma + s)} \quad /4.7/$$

So, the transform of  $f_{T_j}$  can be expressed as

$$\bar{f}_{T_j}(s) = \sum_{k=0}^{\infty} e^{-\gamma a} e^{-[a+C+k(R+C)]s} \left( \frac{\gamma}{\gamma + s} \right)^k \sum_{l=0}^k \binom{k}{l} (-1)^l e^{-la(\gamma + s)} \quad /4.8/$$

We obtain the p.d.f. of  $T_j$  after computing the inverse Laplace transform

$$f_{T_j}(x) = e^{-\gamma a} \sum_{k=0}^{\infty} \gamma^k \sum_{l=0}^k \binom{k}{l} (-1)^l \frac{[\alpha - a(l+1) - C - k(R+C)]^{k-1}}{(k-1)!} e^{-\gamma[\alpha - a(l+1) - C - k(R+C)]} \cdot [1[\alpha - a(l+1) - C - k(R+C)]] \quad /4.9/$$

Since  $T_{EC}$  is the sum of N identically, independent random variables  $T_j$ , the Laplace transform of the p.d.f.  $f_{T_{EC}}$  is

$$\bar{f}_{T_{EC}}(s) = \left[ \sum_{k=0}^{\infty} e^{-\gamma a} e^{-[a+C+k(R+C)]s} \left( \frac{\gamma}{\gamma + s} \right)^k \sum_{l=0}^k \binom{k}{l} (-1)^l e^{-la(\gamma + s)} \right]^N \quad /4.10/$$

and we obtain the distribution of the elapsed time  $T_{EC}$

$$f_{T_{EC}}(x) = [f_{T_j}(x)]^{N(*)} \quad /4.11/$$

where  $N = \frac{T}{a}$  and  $N(*)$  denotes N-fold convolution.

Let us consider the expected elapsed time  $E(T_{EC})$ . We calculate first the expectation of  $t_i$

$$E(t_i) = \frac{1}{\gamma} - \frac{a e^{-\gamma a}}{1 - e^{-\gamma a}} \quad /4.12/$$

Then we have

$$E(T_j | n=k) = a + C + k(R+C + \frac{1}{\gamma} - \frac{a e^{-\gamma a}}{1 - e^{-\gamma a}}) \quad /4.13/$$

The expectation of the total time between checkpoints  $T_j$  is the following

$$E(T_j) = \sum_{k=0}^{\infty} e^{-\gamma a} (1 - e^{-\gamma a}) [a + C + k(R+C + \frac{1}{\gamma} - \frac{a e^{-\gamma a}}{1 - e^{-\gamma a}})] \quad /4.14/$$

that is

$$E(T_j) = C + (R+C + \frac{1}{\gamma})(e^{\gamma a} - 1). \quad /4.15/$$

This expression is similar to Eq./3.13/ where  $T$  is replaced by the checkpoint interval  $a$ . It follows that the mean total time between the ends of consecutive checkpoints depends on the value of  $a$  and does not depend on the execution time  $T$ .

Finally we obtain

$$E(T_{EC}) = \frac{T}{a} [C + (R+C + \frac{1}{\gamma})(e^{\gamma a} - 1)]. \quad /4.16/$$

Note that  $E(T_{EC})$  is a linear function of the execution time  $T$ . In order to minimize the expected elapsed time we can compute the optimal checkpoint interval  $\hat{a}$  that gives  $\frac{d}{da} [E(T_{EC})] = 0$ .

This leads to the equation

$$e^{\gamma \hat{a}} (\gamma \hat{a} - 1) = \frac{\gamma C}{1 + \gamma(R+C)} - 1 \quad /4.17/$$

The approximate solution yields

$$\hat{a} \approx \sqrt{\frac{2C}{\gamma[1 + \gamma(R+C)]}} \quad /4.18/$$

### 5. Comparisons and results

From the user viewpoint it is very important to know how unreliability of the computing system increases the execution time of a program and what is the effect of program checkpointing. It follows from previous sections that the expected elapsed time of the program without checkpointing increases as an exponential function of the execution time  $T$ , whereas the use of checkpointing causes the linear increase. The relative increase of the expected elapsed time for two cases - program without checkpointing and program with checkpointing is compared in Figure 3.

It is seen from curves in Figure 3 that checkpointing is not beneficial for small values of  $T$ . But for long times  $T$  the use of checkpointing is advantageous.



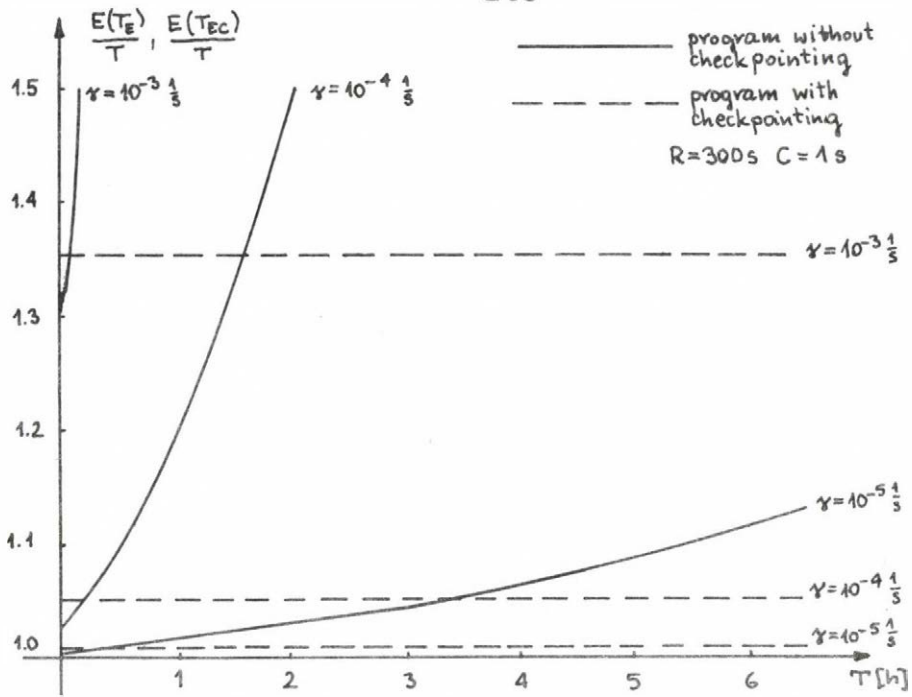


Figure 3. Relative increase of expected elapsed time

The influence of the failure rate  $\gamma$  on the expected elapsed time is illustrated in Figure 4.

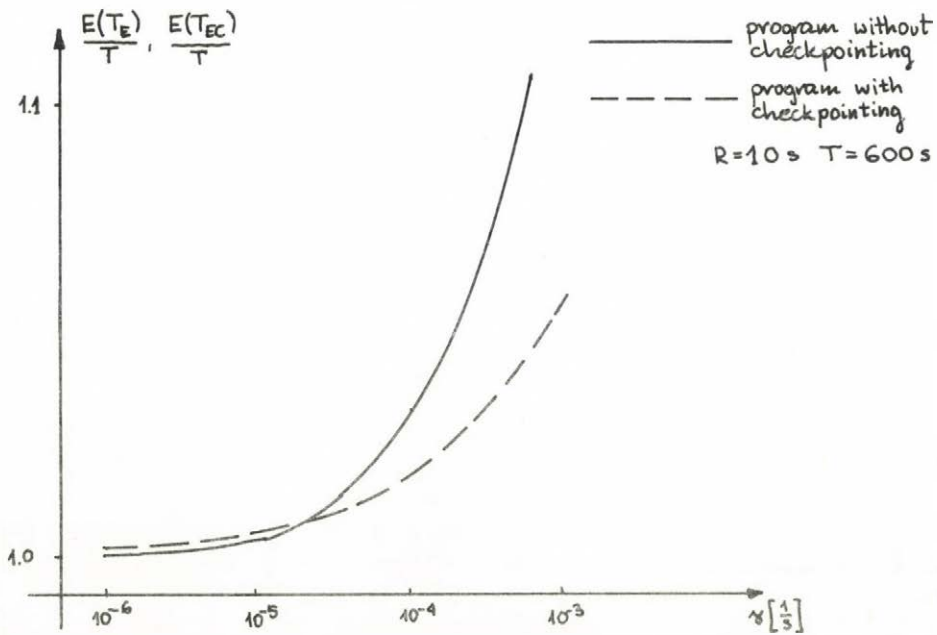


Figure 4. Influence of the failure rate on the expected elapsed time

It is also seen there exist values of the failure rate  $\gamma$  for which the use of checkpointing is not beneficial. In general both values of  $\gamma$  and  $T$  decide whether  $E(T_E)$  is greater than  $E(T_{EC})$ : the greater value of  $T$ , the smaller  $\gamma$  for which the use of checkpointing is not beneficial.

It is interesting to study the relative importance of the failure rate and the down time  $R$ . The notion of availability defined as

$$Av = \frac{\frac{1}{\gamma}}{\frac{1}{\gamma} + R} = \frac{1}{1 + \gamma R} \quad /5.1/$$

determines the fraction of the time the system is up. Figure 5 presents the increase of the expected elapsed time for constant values of availability.

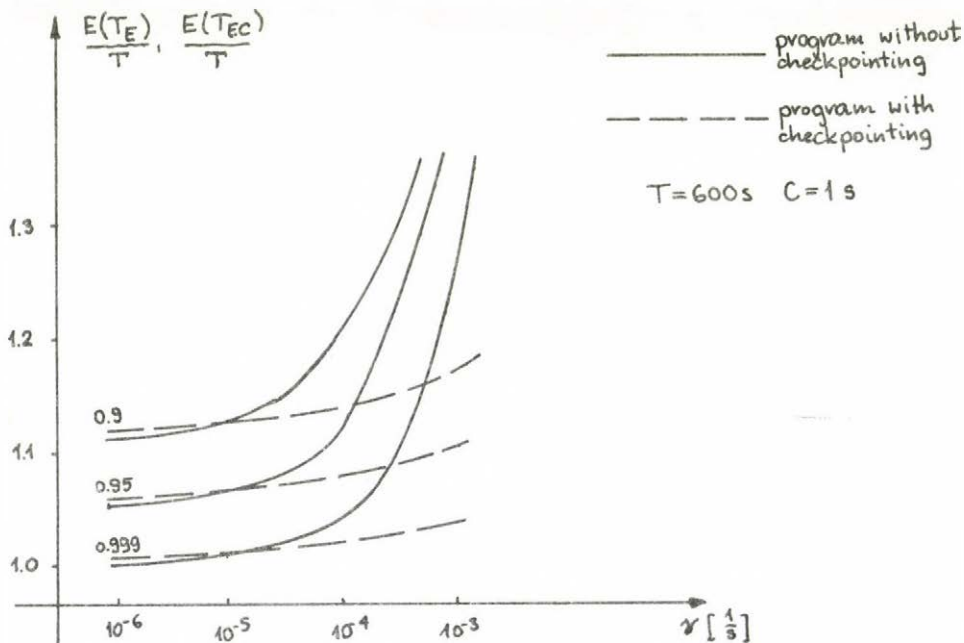


Figure 5. Increase of the expected elapsed time for the constant values of the availability

The failure rate for the program without checkpointing is the dominant factor. That means [CAST 80] that no matter how short the down time period is made, the expected elapsed time will still be very long. In the case of checkpointing the relative importance of the failure rate is compensated and the expected elapsed time rises slightly according to the failure rate.

## 6. Conclusions

The presented analysis of the program performance provides quantitative results about how the elapsed time increases due to system failures. We have considered checkpointing as a method to reduce an overhead caused by restarts of a program in the computing system subject to failures. It follows from the analysis that the advantage of checkpointing depends mainly on the execution time  $T$  and the failure rate  $\gamma$ . Long programs or execution of programs in unreliable computing systems require the use of checkpointing. Let us remember that in our investigation the execution time  $T$  is the time to execute a given program in a multiprogramming, multiuser system i.e. a short program of the 10 minutes execution time in a single-programming system may require the execution time  $T$  increased several times. By means of the time  $T$  we have implicitly taken into account the workload of the system.

The equation /4.17/ for the optimal checkpoint interval is more accurate version of the "square root law" which has been often investigated [YOUN 74], [CHAN 75], [LOHM 77], [GELE 79], [BROC 79]. We must point out some assumptions adopted in the analysis. We assumed that the failure rate and the system workload are constants. However it is not realistic assumption for long programs. We considered that the failure rate is independent of the workload. It has been reported BUTN 80 that increased utilization of the system results in a degradation in reliability. In our analysis of checkpointing we have not taken into account other factors than the processor's time. The complete model must consider store occupancy and peripheral activity increased by the use of checkpointing. All these assumptions made our analysis simpler and allowed to obtain quantitative results.

### References

- BEAU 78 : Beaudry, M.D. "Performance-Related Reliability Measures for Computing Systems", IEEE Trans. on Computers, Vol.C-27, No.6, June 1978
- BROC 79 : Brock, A. "An analysis of Checkpointing", ICL Technical Journal, Vol.1, No.3, November 1979
- BUTN 80 : Butner, S.E. Iyer R.K. "A Statistical Study of Reliability and System Load at SLAC", Proc. 10<sup>th</sup> Int. Conf. on FTC, 1980
- CAST 80 : Castillo, X. Siewiorek D.P. "A Performance-Reliability Model for Computing Systems", Proc. 10<sup>th</sup> Int. Conf. on FTC, 1980
- CHAN 75 : Chandy, K.M. "A Survey of Analytic Models of Roll-back and Recovery Strategies", Computer, Vol.8, No.2, 1975
- DENN 76 : Denning, P.J. "Fault Tolerant Operating Systems", ACM Comp. Surveys, Vol.8, No.4, December 1976
- FERR 75 : Ferrari, D. "Computer Systems Performance Evaluation", Prentice-Hall, Englewood Cliffs, 1975
- GELE 79 : Gelenbe, E. "On the Optimum Checkpoint Interval", Journal of the ACM, Vol.26, No.2, April 1979
- LOHM 77 : Lohman, G.M. Muckstadt, J.A. "Optimal Policy for Batch Operations: Backup, Checkpointing, Reorganization and Updating", ACM Trans Database Systems, Vol.2, No.3, September 1977
- MEYE 80 : Meyer, J.F. "On Evaluating the Performability of Degradable Computing Systems", IEEE Trans. on Computers, Vol.C-29, No.8, August 1980
- RAND 75 : Randell, B.R. "System Structure for Software Fault Tolerance", IEEE Trans. on Software Engineering, Vol.SE-1, No.2, June 1975
- YOUN 74 : Young, J.W. "A First-Order Approximation to the Optimum Checkpoint Interval", Communications of the ACM, Vol.17, No.6, 1974

Andrzej Duda : Polish Academy of Sciences  
Dept. of Complex Control Systems  
5, Baltycka  
44-100 Gliwice, Poland

AUTOMATIC PROGRAMMING SYSTEM DEVELOPMENT ON USER LEVEL

Kerékfy, Pál      Ruda, Mihály

Hungary

1.0 Introduction

Software development, beside the analysis of theoretical problems, concentrates mainly on producing new programming languages, problem oriented languages and complete, closed systems of programs. In this way a bulk of languages, systems and theories overwhelms the user. There is a lot of valuable theoretical results and software based on them that can be utilized properly for certain purposes. However it seems that the efficiency of the usage of the programs was sometimes not considered to be important while creating them. Perhaps, a qualitative change similar to the appearance of the high-level languages is indispensable. (Over and above creating "ready-made" programs and packages, producing software tools for wide-ranging applications is indispensable.) The primary aspect of software development must be the efficiency of usage - there is no other aspect, indeed. Of course, efficiency depends on many factors. Therefore the complexity of the process of software development, application and maintenance must be taken into consideration, for this aspect see e.g. [4]. First, some of these factors will be looked over without claiming for completeness. Then usefulness of a program generating method of the authors will be summarized.

## 2.0 Some practical experiences

First, let some experiences be mentioned that turned the authors' attention to the problems of software engineering described in section 4.0. The authors and their collaborators have performed a lot of tasks of descriptive and mathematical statistics. Though there is a plenty of statistical program packages available (such as the well-known SPSS and BMDP), a significant proportion of the applications claims for extensions of these tools. (This need can be caused by e.g. special data structure of the input, unusual output requirements, an enormous mass of data.) It has been noticed e.g. in a time series analysis program package (see [1]) and in BMDP. Concerning this latter one, attention is drawn to a common aspiration observed frequently in software development. While designing a program or a program package, far from taking into account later development of the system, the designer often aspires to prevent the end-user from modifying the system. It is an extremely important requirement in a system utilized simultaneously by more than one user (such as operating systems). But it is extremely disadvantageous in a system that each user disposes of a separate copy of.

In computer processing of the data of the Hungarian Hospital Morbidity Study the matter of program manipulation and optimization became particularly important. This involves detailed statistical processing of the data of several hundred thousands of inpatients every year [2]. A characteristic of this system, like of many other "living" systems, is the continual change of the demands. The changes can affect almost every part of the system. E.g. having new data collection, preparation and coding methods initiated the structure of the input data file can be altered. The most important change affecting the system is just brought about by the utilization of the information system itself. The demand of information can be changed, it affects both the form and the contents of the information required.

In the beginning, the insufficient computer resources (a small and overloaded CDC 3300) gave rise to special difficulties. Thus, we had no other choice but to apply procedures that guaranteed both the optimal utilization of the configuration and fulfilling the needs of the continually changing demands [3].

Later we have tried to make use of our experiences in general.

### 3.0 Some remarks on software development

Nowadays the term "software crisis" is being used frequently, see e.g. [5]. The authors just mention some experiences that can support the usage of this term. Attention will be focused on the efficiency of usage. Here, efficiency is not limited to efficient usage of computer resources but it includes efficient usage of human efforts during all the tasks connected to computer usage (e.g. software development, programming, debugging, etc.). Now, let some aspects of the crisis be mentioned.

a) It must be realized that the demand of generality is sometimes inconsistent with the convenient usage (the optimum of all variables cannot ever be reached in a multivariate system). Systems providing the user with the possibility of the most convenient usage are ususally rigid, they serve a predefined purpose and can hardly be developed further.

b) As there is a plenty of specialized software products available, choosing the most suitable one is a complicated task. To determine if a program really meets all the requirements then implement it and learn its usage needs hard efforts of the user.

c) The matter of conscious usage. Undiscerning use of a program product is a great convenience and can be accomplished without special expertness while well-considered and fruitful use of a complicated system makes great demands on

the end-user's power. Note that it is equally advantageous in computer applications and in the scope of technical development to utilize the applicable means given their full knowledge and the utilization is not based merely on the "instructions for use".

d) Systems constructed from elementary "building blocks" (as e.g. macro languages, preprocessors, problem-oriented languages, subroutine libraries) get the user to apply them consciously, nevertheless they need much programming. Applying a complicated, closed system and having no good grasp of its logical structure one can easily make conceptual mistakes while, however, elementary programming may cause a lot of elementary errors.

e) Having a system applied the user may want to modify or develop it and to make its results attainable for other systems.

Summing up what has been said it can be stated that closed systems providing the end-user with convenient usage usually can be applied in a limited scope of problems only. In a field requiring dynamic user activity (examples can be planning or statistical information retrieval procedures) it is a vain hope that a "prefabricated" system can be applied successfully. The situations and demands changing continually and the new tasks evolving during the process require tools supporting software development in general.

The authors submit for consideration a method supporting software development and usage of software products as well, therefore having applications in users' tasks conforming to varying conditions.

In the light of the above considerations the requirements of efficiency can be stated as:

A system or method is efficient if

a) It is not so general that much power is to be devoted to learn its usage or the operation occupies too many computer resources,



- b) its structure is clear and easy to survey for the end-user,
- c) the results obtained are reliable,
- d) the operation does not need much elementary programming,
- e) it is open to other systems, can be developed or modified.

In the following the authors show how they attempted to fulfil the above programme. The proposal is based on a simple program generator procedure called GENERA [6] that was first applied in statistical processing of large data bases, see [3,7]. The first aim of generating was to create programs that are very efficient on general conditions. The meaning of efficiency is not limited to efficient computer usage but it includes flexibility and convenient usage as well. System GENERA which is going to be described here serves this purpose.

#### 4.0 The generator system GENERA

GENERA supports processing of problem-oriented directives embedded in a host language. Macro and subroutine libraries can be utilized and programs or jobs are generated. Besides, it assists the system programmers in creating generator procedures and in adding them to the system.

In the following, the basic ideas are summed up. The most expedient way to describe the great variety of elementary tasks is an elementary way: use of a high-level language such as e.g. FORTRAN, PL/1, ALGOL, etc. Typical tasks that can be specified in advance and demand much work are to be solved by prefabricated procedures. Subroutine libraries, macro libraries and problem oriented languages serve this purpose. However, the organization is the end-user's task and it can be a rather complex work. It is the main point supported by GENERA.

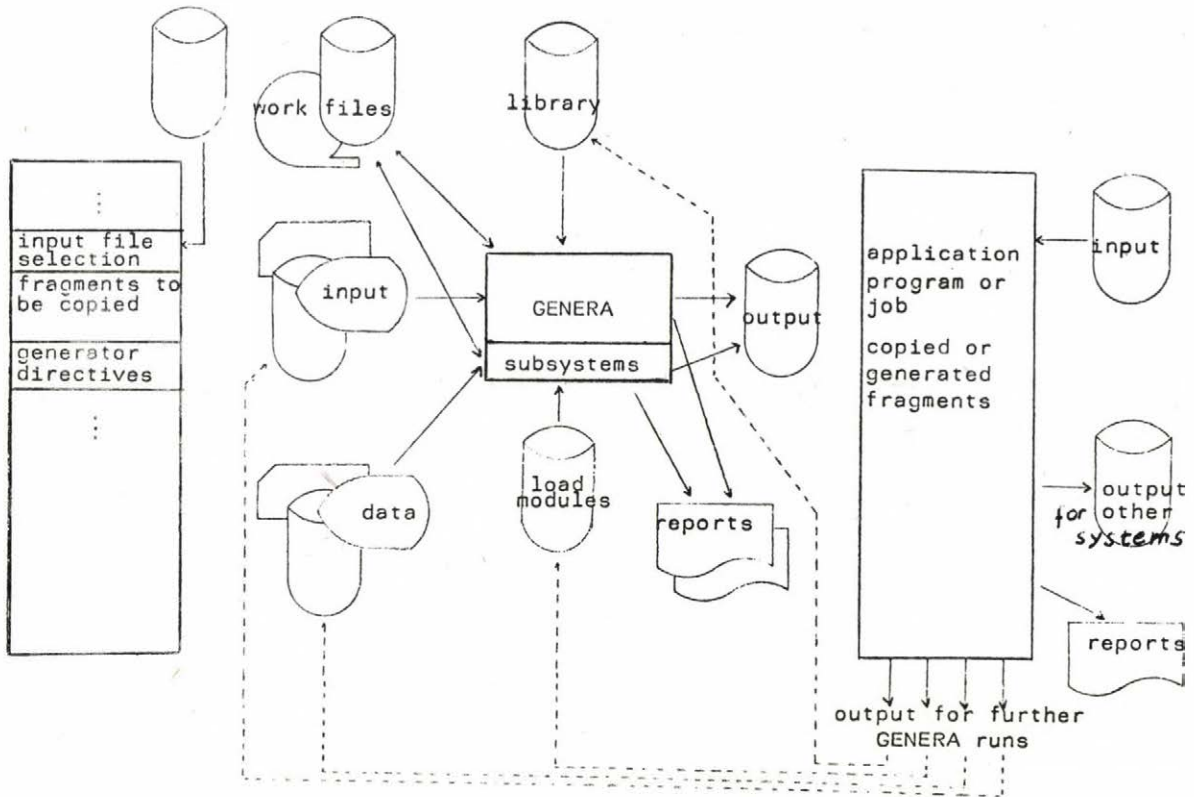
Beside preprocessing, generator procedures can aspire to produce optimal solutions of the problem. To find the optimal solution is especially important when a large and complex system is to operate in conversational mode. GENERA, also, gives the system programmer assistance in producing the prefabricated procedures. This is accomplished partly by providing standard procedures (e.g. for error handling, listing, etc.) and partly by specifying a uniform framework for the prefabricated procedures.

GENERA is a framework to build generator programs. An extended file handling facility and a preliminary syntax analyzer (preprocessor) is established in it. The input processed is a program written in a host language (such as PL/1 or FORTRAN) and contains directives that drive the generator system. The requests expressed by the directives are fulfilled by installation-defined subsystems. This method provides the user with full control of the system since the programs are built from elementary parts (host language statements) and generator directives that can cover a large set of typical tasks. The environment (e.g. JCL statements, external program calls, etc.) required to execute the generated program is produced by the system as defined at installation time or by the user. As the set of directives can be widened any time by installing new subsystems, and the user can code any host language statements, generated programs can be made consistent with any other program product and the set of generated programs can be extended.

The systems built up using GENERA are of clear structure, they get the user to use them consciously and can be connected to other systems easily. As a consequence they give solutions to the above problems and, besides, provide means for generating efficient programs. The efficiency is emphasized since modest demands to computer resources (CPU time, memory, etc.) is important in conversational mode of processing. The generated programs are of clear and well arranged structure and they exist in the moment of their

utilization only. This latter feature facilitates the maintenance of the system.

The reader can be referred again to the summary paper of Lehman [4]. Having GENERA applied, the efficiency of the whole process of program development and utilization is improved significantly. The main features are: accommodation to the varying conditions, a uniform framework for program development, simple system management. Among those types of programs defined in the above referenced paper of Lehman, the system GENERA has the closest relations to the E-programs.



The figure illustrates input and output files together with the data transfer in a generator system based on GENERA. The left hand side shows structure of the input file that consists of program fragments to be copied to the output

without any modification and generator directives driving the subsystems together with file selection statements defining parts of the input file stored on external media. The right hand side exhibits structure of the output file that usually contains a program or a job including input and output file definitions needed for execution. The generated program can interact with any other system or with the generator system based on GENERA as well.

During the process of building the input file or generating, the designer can make use of procedures and data created by a previous run. Results of generating (the output file) or those of the generated program or job can be used in a subsequent step.

Finally, an example of the applications of GENERA is shown. It is an input file of SIS79 (Statistical Information System '79) developed for medical statistics.

#OPTION

```
$PARAM LIST="ERROR",SYSTEM="FORTRAN"$
      INTEGER AGE,HYEAR,BYEAR,SEX,PROFS,
      *CODE,MAINCD,SUBCD,ERROR(1)
1    CONTINUE
```

#LECTOR

```
$PARAM FC=5,END=500,ERR=600$
$DESCR PATIENT
      1 NAME 30X
      1 BYEAR I4
      1 SEX I1
      1 COUNTY 2X
      1 PROFS I4
      1 HDATE
      2 HYEAR I4
      2 HMONTH 2X
      2 HDAY 2X
      1 CODE I4
```

```
      2 MAINCD I3
      2 SUBCD I1
$
      AGE = HYEAR - BYEAR
#GRAPH
  $PARAM GRAPH="AGE CODING",DATANA="AGE",
  NEWDAT="CDAGE",SACKNO=1,LEVELS=1,
  UPPBOU=100$
      IF ( NUMERR.NE.0 ) GOTO 100
#GRAPH
  $PARAM GRAPH="CONTROL",DATANA="CDAGE",
  "SEX","MAINCD","SUBCD",SACKNO=34,LEVELS=
  1,10,15,8,UPPBOU=10,10*10,896,788,
  10*999,618,528,496,2*7,6*9,
  LOWBOU(2)= 10*1,LOWBOU(15)=125$
      IF ( NUMERR.NE.0 ) GOTO 101
      GOTO 1
100   WRITE (6,10) ERROR(1)
10    FORMAT (' ERROR IN AGE:',I4)
      GOTO 1
101   WRITE (6,11) ERROR(1)
11    FORMAT (' ERROR:',I4)
      GOTO 1
600   WRITE (6,12)
12    FORMAT (' READ ERROR')
      GOTO 1
500   STOP
      END
```

The program contains FORTRAN statements and SIS79 directives intermixed. There is a directive #OPTION containing declarative statements for GENERA. In this example output listing will contain error messages only and a FORTRAN program will be generated (LIST="ERROR", SYSTEM="FORTRAN"). The meaning of the set \$PARAM after the directive #LECTOR goes without saying. The set DESCR gives format for reading

a record named PATIENT (COBOL-style level numbers and FORTRAN format items are used). The subsystem LECTOR generates a fast input procedure in FORTRAN. The directive #GRAPH is described in [7]. It is a method to describe and evaluate multivariate functions. The first #GRAPH directive of the example generates procedure named "AGE CODING". Argument of the function is variable AGE while the function value is assigned to variable CDAGE. The parameters SACKNO and LEVELS indicate the structure of the graph that is a single table of values in this example. The table gives function values for independent variables 1 - 100 (UPPBOU=100). The procedure generated by the second directive is named CONTROL. It has four independent variables CDAGE, SEX, MAINCD and SUBCD. SACKNO=34 means that the graph contains 34 vertices (elementary tables, "sacks"). There are 1, 10, 15, 8 vertices on the levels corresponding to the independent variables, respectively. The minima and maxima of the independent variables are stored in arrays LOWBOU and UPPBOU. The parameters NUMMER and ERROR are the names of variables for error processing.

#### 5.0 References

1. M. Ruda, L. Szeidl, G. Tusnády, "Time series analysis on CDC 3300", Progress in Statistics, ed. J. Gani et al., North-Holland, 1974, Vol. 2, pp. 661-665.
2. M. Csukás, L. Greff, A. Krámlí, M. Ruda, "An Approach to the Hospital Morbidity Data System Development in Hungary", Colloques IRIA, Tome 1, Informatique Médicale, IRIA, 1975, pp. 381-390. (paper presented at the Symposium on Medical Data Processing, Toulouse, 1975.)
3. A. Krámlí, M. Ruda, M. Csukás, M. Galambos, "Large Sample Size Statistical Information System for HWB", Data Analysis and Informatics, ed. E. Diday, North-Holland, 1980, pp. 457-462.

4. M. M. Lehman, "Programs, Life Cycles, and Laws of Software Evolution", Proceedings of the IEEE, Vol. 68, 1980, No. 9, pp. 1060-1076.
5. J. Foisseau, R. Jacquart, M. Lemaitre, M. Lemoine, J.C. Vignat, G. Zanon, "Program Development With or Without Coding", Software World, Vol. 12, No. 1, 1981, pp. 9-12.
6. P. Kerékfy, "GENERA - A Program Generator System", Progress in Cybernetics and Systems Research, Vol. 11., Hemisphere, Washington (to appear). (paper presented at the Fifth European Meeting on Cybernetics and Systems Research (EMCSR'80), Vienna, 1980.)
7. P. Kerékfy, A. Krámli, M. Ruda, "SIS79/GENERA Statistical Information System", Progress in Cybernetics and Systems Research, Vol. 11., Hemisphere, Washington (to appear). (paper presented at the Fifth European Meeting on Cybernetics and Systems Research (EMCSR'80), Vienna, 1980.)
8. A. Krámli, P. Lukács, M. Ruda, "Probabilistic Approach to the Performance Evaluation of Computer Systems", Third Hungarian Computer Science Conference, Budapest, 1981, Invited Papers, pp. 51-63.

Authors' address:

Computer and Automation Institute  
Hungarian Academy of Sciences  
Budapest P. O. Box 63, H-1502

A TANULMÁNYOK SOROZATBAN 1982-BAN MEGJELENTEK:

- 130/1982 Barabás Miklós - Tőkés Szabolcs: A lérez printer képképzés hibái és optikai korrekciójuk
- 131/1982 RG-II/KNVVT "Szisztemü upravlenija bazani dannüh i informacionnue szisztemü"  
Szbornik naučno-iszszledovatel'szkih rabot rabocsej gruppü RG-II KNVVT, Budapest, 1979.  
T o m I.
- 132/1982 RG-II/KNVVT T o m I I.
- 133/1982 RG-II KNVVT T o m I I I.
- 134/1982 Knuth Előd - Rónyai Lajos: Az SDLA/SET adatbázis lekérdező nyelv alapjai (orosz nyelvü)
- 135/1982 Néhány feladat a tervezés-automatizálás területéről. Örmény-magyar közös cikkgyűjtemény
- 136/1982 Somló János: Forgácsoló megmunkálások folyamatának optimalizálási és irányítási problémái
- 137/1982 KGST I-15.1. Szakbizottság 1979. és 80. évi előadásai
- 138/1982 Kovács László: Számítógép-hálózati protokollok formális specifikálása és verifikálása









