Yugoslav Journal of Operations Research
6 (1996), Number 2, 217-230

A FORMAL, OBJECT-ORIENTED APPROACH TO THE MACHINE INTERFERENCE MODEL

Ion FLOREA
Transylvania University of Brasov, Romania

Abstract. The paper presents a simulation model, based on Zeigler's formalism which describes the system dynamic behavior; also, using the principles of object-oriented simulation, model programming is achieved using the advanced concepts of Turbo Pascal programming.

Keywords: Machines interference model, simulation, discret event system specification, object oriented.

1. INTRODUCTION

A system of m machines is considered, operated by n workers, $m>n$. The machines run automatically yet, after certain periods of continous operation function occurs and worker intervention is required. The continous operation period of the machine and the time required by the worker to repair the machine are random values of random variables of the given distribution. In this situation two possibilities occur: there is at least one unoccupied, lazy worker, ready to intervene immediately in order to repair the respective machine or all workers are busy, in which case the machine is placed last in the waiting line. The whole waiting period in order to get repaired is named interference time. The problem of finding an optimum operation mode for the system arises, i.e. the optimum ratio of the worker's total time of "laziness" and the interference time. This problem is called the machine interference problem and it corresponds to a waiting model of the following form: $./.$/n;(m-n,.)(see [6],[7]). This problem has not been solved yet analytically in the must general case; the problem was solved only in the case when all machines have i.i.d. running time and the repair times are also i.i.d. random variables. "Experience" on this type of data is obtained by simulation and by changing several input parameters an optimum value is obtained. Work studies [4] the model Exp(λ)/Exp(μ)/n:(m-n;FIFO) and the quantities determined analytically for certain particular cases are compared to those obtained by simulation. Any simulation model may be considered as a time-related dynamic system, its components being described by three categories of variables, i.e.:

input entrance variables , state variables and output variables. The dynamic behaviour of a model with discrete events is described by the DEVS (Discrete Event System Specification) object, defined as follows: DEVS = $(I,S,O,t,\delta,\delta_{ex},\lambda)$, where: I is the "input" set , i.e. the carthesian product of the input variable value ranges ; O is the "output" set, i.e. the carthesian product of the output variable value ranges; S is the "states" set, i.e. the carthesian product of the state variables values ranges; $t:S\rightarrow R+\cup\{0\}$ is the time advance function, its value t(s), $s\in S$ representing the interval of time after which the system will switch into state $\delta(s)$, where δ has the meaning presented further on; $\delta:S(S$ is the transition function of the system when shifting from one state to another in the case of no entrance to into the system, i.e. if the system is in state s at moment T, then at moment $T+t(s)$ it will shift into state $\delta(s)$, if in the interval $[T,T+t(s)]$ no external event occurs; $\delta_{ex}:QxI\rightarrow Q$, where Q is the set of all type pairs (s,e), $s\in S$ and $e\in R$, $0\leq e\leq t(s)$, function δ_{ex} shows that if a model is in the state s at the moment T and input x occurs at moment $T+e$, then, after occurrence of the external event, the system state will be $\delta_{ex}(s,e,x)$; $\lambda:S\rightarrow Q$ is the output function, showing that if the system is in state s at moment T, then $\lambda(s)$ represents the information conveyed by the model at the output. (see [5]).

Observation 1.1. As the considered model is autonomous, i.e. it has input parameters (m, n, the type of random variables whose values are continuous operating and separation time respectively), but it has no input variables, i.e. $I=\Phi$, the system transition is described by function δ, function δ_{ex} having no signifiance.

2. THE STATE VARIABLES ARE:

(2.1) **List_Of_Operating_Machines** can be the empty set denoted Λ, or a list of type $(x_1,\tau_1)...(x_p,\tau_p)$, where x_i is the machine code, $x_i\in\{1,...,m\}$ and τ_i the time of continous operation of machine x_i from the respective moment on, $i=1,...p$. When machine x_i starts operation, the continous operation time τ_i will be generated as values of the random variable, using the Monte Carlo method.

(2.2) List_Of_Machines_In_Repair can be the void set, denoted Λ, or a list of the form: $(x_{p+1},\sigma_1,y_1)...(x_{p+q},\sigma_q,y_q)$ where x_{p+i} is the machine code, y_i is the workers code who is repairing the machine x_{p+i}, $x_{p+i}\in\{1,....,m\}$, $y_i\in\{1, ... , n\}$, σ_i is the time required for the repair, as the value of a random variable, generated by the Monte Carlo method; it can also be assumed that $\sigma_1\leq\sigma_2...\leq\sigma_q$.

(2.3) List_Of_Machines_Waiting_ For_Repair can be the void set noted Λ, or a queue of the form $x_{p+q+1}...x_{p+q+r}$ where x_{p+q+i}, $i=1,...,r$ is the code of the machine waiting for the repair; assuming that all workers are busy, x_{p+q+1} represents the first machine in the queue, x_{p+q+2} the next one a.s.o, and x_{p+q+r} the last one.

(2.4) List_Of_Workers_In_ Laziness an be the empty set noted Λ, or a queue of form $y_{q+1}...y_n$, where y_{q+i} is the code of worker, $i=1,..,n-q$; y_{q+1} is the first one in the queue a.s.o and y_n is the last one, showing that a damaged machine will be repaired by y_{q+1} and y_{q+2} will become the first one queuing; the worker having finished his repair job will be placed last in the queue.

Obserevation 2.1. If the List_Of_Operating_Machines is Λ if $p=0$, the List_Of_Machines_In_ Repair is Λ if $q=0$, the List_Of_Machines_Waiting_For_ Repair is Λ if $r=0$ then $m=p+q+r$ and $n=q+r$.

3. OUTPUT VARIABLES

(3.1) Output variables are: Tu is a vector of m components , $Tu(1),...,Tu(m)$, where $Tu(i)$, represents the total time of effective operation of machine i , $i=1,...,m$.
(3.2) Tr is a vector of m components , $Tr(1),...,Tr(m)$, where $Tr(i)$, represents the total repair time of machine i, $i=1,...,m$.
(3.3) Ta is a vector of m components , $Ta(1),...,Ta(m)$, where $Ta(i)$, represents the total time of machine i waiting for repair , $i=1,...,m$.
(3.4) Tl is a vector of n components , $Tl(1),...,Tl(n)$, where $Tl(i)$, represents the total time of "laziness" of worker i, $i=1,...,n$.
(3.5) Trm is a vector of n components , $Trm(1),...,Trm(n)$, where $Trm(i)$, represents the total working time of worker i, $i=1,...,n$.
The defined vectors represent the image of function λ.

Observation 2.2. We have: $\sum_{i=1}^{m} Tr(i)=\sum_{i=1}^{n} Trm(i)$. (3.1)

Observation 2.3. If we note

$$TRT=\sum_{i=1}^{m}Tu(i);\ TST=\sum_{i=1}^{m}Tr(i);TWT=\sum_{i=1}^{m}Ta(i);TTID=\sum_{i=1}^{n}Tl(i);TTBS=\sum_{i=1}^{n}Trm(i);$$

$$EF=\frac{TRT}{TRT+TWT+TST};\ EM=\frac{TTBS}{TTBS+TTID};I=\frac{TWT}{TST} \quad (3.2)$$

then EF is the machine efficiency factor, EM is the worker efficiency factor, and I is the interference factor (see [4]).

4. DESCRIBING THE MODEL

Describing the model's behavior implies describing functions t,σ,λ of models object DEVS. We note: LOM the length of the List_Of_Operating_Machines (LOM=p) ; LMR the length of the List_Of_ Machines_In_Repair (LMR=q); LMWR the length of the queue List_Of_Machines_Waiting_For_ Repair (LMWR=r) ; LML the length of the queue List_Of_Workers_In_Laziness (LML=n-q)(see Remark 2.1). We can consider $2^4=16$ possibilities; it is easily confirmed that 11 of these cases are impossible:

LOM=0 and LMR=0 and LMWR=0 and LML=0;
LOM $\neq$0 and LMR=0 and LMWR=0 and LML=0;
LOM=0 and LMR $\neq$0 and LMWR=0 and LML=0;
LOM=0 and LMR=0 and LMWR $\neq$0 and LML=0;
LOM $\neq$0 and LMR=0 and LMWR $\neq$0 and LML=0;
LOM=0 and LMR=0 and LMWR=0 and LML $\neq$0;

LOM=0 and LMR ≠0 and LMWR=0 and LML≠0;
LOM=0 and LMR=0 and LMWR ≠0 and LML≠0;
LOM ≠0 and LMR=0 and LMWR ≠0 and LML≠0;
LOM=0 and LMR ≠0 and LMWR ≠0 and LML≠0;
LOM ≠0 and LMR ≠0 and LMWR ≠0 and LML≠0.

Thus the following 5 cases are left to be for studied:

LOM ≠0 and LMR ≠0 and LMWR=0 and LML=0;
LOM=0 and LMR ≠0 and LMWR ≠0 and LML=0;
LOM ≠0 and LMR ≠0 and LMWR ≠0 and LML=0;
LOM ≠0 and LMR=0 and LMWR=0 and LML ≠0;
LOM ≠0 and LMR ≠0 and LMWR=0 and LML ≠0.

Considering the system in state s, we note $s = \delta(s)$. Function $t(s)$ is expressed as follows:

$$t(s) = \begin{cases} \min(\tau_1, \sigma_1), & if\, LMW \neq 0, LMR \neq 0 \\ \tau_1 & , if\, LMW \neq 0, LMR = 0 \\ \sigma_1 & , if\, LMW = 0, LMR \neq 0 \end{cases} \qquad (4.1.)$$

In expressing function λ, operation := will be used, as in informatics; $a{:}=a+b$ meaning that a becomes the old a plus b.

Case 4.1. The system is in state: $s=((x_1,\tau_1)...(x_p,\tau_p);(x_{p+1},\sigma_1 y_1)...(x_{p+n},\sigma_n\ y_n)\ \Lambda;\Lambda)$, where $p=m\text{-}n$, i.e. $m\text{-}n$ machines are in operation and n are in repair, all workers are busy, there are no machines waiting for repair. The image of function λ changes as follows: $\mathrm{Tu}(x_i){:}= Tu(x_i)+t(s)$, $i=1,...,m\text{-}n$; $Tr(x_{m-n+i}){:}= Tr(x_{m-n+i})+t(s)$, $i=1,...,n$; $Trm(y_i){:}=Trm(y_i)+t(s)$, $i=1,...,n$.

In order to express s', two subcases will be studied:

Case 4.1.1. When $\tau_1 \leq \sigma_1$, the following event will be the damaging of machine x_1 and, as there are not "lazy" workers, machine x_1 will be placed last in the queue of machines waiting for repair, the system will shift into state:

$$s' = ((x'_1,\tau'_1)...(x'_{p-1},\tau'_{p-1});(x'_p,\sigma'_1,y_1)...(x'_{p+n-1},\sigma'_n,y_n);x'_{p+n};\Lambda) \text{ where:}$$
$$x'_1 = x_2,\ldots,x'_{p-1} = x_p, x'_p = x_{p+1},\ldots,x'_{p+n-1} = x_{p+n}, x'_{p+n} = x_1;$$
$$\tau'_1 = \tau_2 - t(s),\ldots,\tau'_{p-1} = \tau_p - t(s); \sigma'_1 = \sigma_1 - t(s),\ldots,\sigma'_n = \sigma_n - t(s).$$

Case 4.1.2. When $\tau_1 > \sigma_1$ the following event will be the returning to operating state of machine x_{p+1} and as there are no machines waiting for repair, worker y_1 will shift into laziness; machine x_{p+1} shifting into the state of operation implies random generation of time during continuous operation and its introducing into the list observing the increasing order of times; the system will shift into state:

$s' = ((x'_1, \tau'_1)\ldots(x'_{p+1}, \tau'_{p+1}); (x'_{p+2}, \sigma'_1, y'_1)\ldots(x'_{p+n}, \sigma'_{n-1}, y'_{n-1}); \Lambda; y_1)$; let τ be the time of continuous operation of machine x_{p+1}, and be k so that: $\tau_k - t(s) \le \tau \le \tau_{k+1} - t(s)$, then:

$$x'_1 = x_1, \ldots, x'_k = x_k, x'_{k+1} = x_{p+1}, x'_{k+2} = x_{k+1}, \ldots, x'_{p+1} = x_p; x'_{p+2} = x_{p+2}, \ldots, x'_{p+n} = x_{p+n};$$
$$r'_1 = r_1 - t(s), \ldots, r'_k = r_k - t(s); r'_{k+1} = r; r'_{k+2} = r_{k+1} - t(s), \ldots, r'_{p+1} = r_p - t(s);$$
$$\sigma'_1 = \sigma_2 - t(s), \ldots, \sigma'_{n-1} = \sigma_n - t(s).$$

Case 4.2. The system is in state: $s = (\Lambda; (x_1, \sigma_1, y_1)\ldots(x_n, \sigma_n, y_n); x_{n+1}\cdots x_m; \Delta)$, i.e. n machines are in repair requiring all n workers, the rest of m-n machines waiting for repair, while there are no "lazy" workers. The image of function λ changes as follows:

$Tr(x_i) := Tr(x_i) + t(s),\ i=1,\ldots,n;\quad Ta(x_{n+i}) := Ta(x_{n+i}) + t(s),\ i=1,\ldots,n;$
$Trm(y_i) := Trm(y_i) + t(s),\ i=1,\ldots,n.$

The following event will be machine x_1 shifting into the state of operation, simultaneously with machine x_{n+1} shifting into repair, wich will be introduced into the list of machines in repair, being repaired by worker y_1, who has become available; the system will shift into state: $s' = ((x'_1, \tau'_1); (x'_2, \sigma'_1, y'_1)\ldots(x'_{n+1}, \sigma'_n, y'_n); x'_{n+2}\ldots x'_m; \Lambda)$; be τ the time of continous operation of machine x_1 and σ the repair time of machine x_{n+1} and be k so that $\sigma_k - t(s) \le \sigma \le \sigma_{k+1} - t(s)$; then:

$$x'_1 = x_1, \ldots, x'_k = x_k, x'_{k+1} = x_{n+1}, x'_{k+2} = x_{k+1}, \ldots, x'_{n+1} = x_n, x'_{n+2} = x_{n+2}, \ldots, x'_m = x_m;$$
$$r'_1 = r; \sigma'_1 = \sigma_2 - t(s), \ldots, \sigma'_{k-1} = \sigma_k - t(s), \sigma'_k = \sigma, \sigma'_{k+1} = \sigma_{k+1} - t(s), \ldots, \sigma'_n = \sigma_n - t(s);$$
$$y'_1 = y_2, \ldots, y'_{k-1} = y_k, y'_k = y_1, y'_{k+1} = y_{k+1}, \ldots, y'_n = y_n.$$

Case 4.3. The system is in the state:

$s = ((x_1, \tau_1)\ldots(x_p, \tau_p); (x_{p+1}, \sigma_1, y_1)\ldots(x_{p+n}, \sigma_n, y_n); x_{p+n+1}\cdots x_m; \Lambda)$; $1 \le p \le m$-n-1;

i.e p machines are operating , n are in repair , m-n-p are waiting for repair and there are no "lazy" workers. The image of function λ changes as follows:

$Tu(x_i) := Tu(x_i) + t(s),\ i=1,\ldots,p;\qquad Tr(x_{p+i}) := Tr(x_{p+i}) + t(s),\ i=1,\ldots,n;$
$Ta(x_i) := Ta(x_i) + t(s),\ i=p+n+1,\ldots,m;\quad Trm(y_i) := Trm(y_i) + t(s),\ i=1,\ldots,n.$

In order to express s' two cases will be studied:

Case 4.3.1. When $\tau_1 \le \sigma_1$ the following event will be the damaging of machine x_1 , which will be placed last in the queue of machines waiting for repair; as there are no "lazy" workers available, the system will shift into state:

$s' = ((x'_1, \tau'_1)\ldots(x'_{p-1}, \tau'_{p-1}); (x'_p, \sigma'_1, y'_1)\ldots(x'_{p+n-1}, \sigma'_n, y'_n); x'_{p+n}\cdots x'_m; \Delta)$ where:

$$x'_1 = x_2, \ldots, x'_{p-1} = x_p, x'_p = x_{p+1}, \ldots, x'_{p+n-1} = x_{p+n}, x'_{p+n} = x_{p+n+1}, \ldots, x'_{m-1} = x_m, x'_m = x_1;$$
$$\tau'_1 = \tau_2 - t(s), \ldots, \tau'_{p-1} = \tau_p - t(s); \sigma'_1 = \sigma_1 - t(s), \ldots, \sigma'_n = \sigma_n - t(s); y'_1 = y_1, \ldots, y'_n = y_n.$$

Case 4.3.2. When $\tau_1 > \sigma_1$ the following event will be the shifting into operation of machine x_{p+1} and simultaneously the shifting into repair of machine x_{p+n+1} which will be repaired by worker y_1; the system will shift into state: $s' = ((x'_1, \tau'_1) \ldots (x'_{p+1}, \tau'_{p+1}); (x'_{p+2}, \sigma'_1, y'_1) \ldots (x'_{p+n+1}, \sigma'_n, y'_n); x'_{p+n+2} \ldots x'_m; \Delta)$; let τ be the time of continous operation of repaired machine x_{p+1} generated at random and σ the continuous time spent in repair by machine x_{p+n+1} and be k, l so that $\tau_k - t(s) \le \tau \le \tau_{k+1} - t(s)$, and $\sigma_l - t(s) \le \sigma \le \sigma_{l+1} - t(s)$; then:

$$x'_1 = x_1, \ldots, x'_k = x_k, x'_{k+1} = x_{p+1}, x'_{k+2} = x_{k+1}, \ldots, x'_{p+1} = x_p; x'_{p+2} = x_{p+2}, \ldots,$$
$$x'_l = x_l, x'_{l+1} = x_{p+n+1}, x'_{l+2} = x_{l+1}, \ldots, x'_{p+n+1} = x_{p+n}; x'_{p+n+2} = x_{p+n+2}, \ldots, x'_m = x_m;$$
$$\tau'_1 = \tau_1 - t(s), \ldots, \tau'_k = \tau_k - t(s), \tau'_{k+1} = \tau, \tau'_{k+2} = \tau_{k+1} - t(s), \ldots, \tau'_{p+1} = \tau_{p+1} - t(s)$$
$$\sigma'_1 = \sigma_2 - t(s), \ldots, \sigma'_{l-1} = \sigma_l - t(s), \sigma'_1 = \sigma, \sigma'_{l+1} = \sigma_{l+1} - t(s), \ldots, \sigma'_n = \sigma_n - t(s).$$

Case 4.4. The system is in the state:$s = ((x_1, \tau_1) \ldots (x_m, \tau_m); \Lambda; \Lambda; y_1 \ldots y_n)$, i.e. all m machines operate and all n workers are in laziness; the following event will be the damaging of machine x_1 and its shifting into repair, being repaired by worker y_1. The image of function λ changes as follows:

$Tu(x_i) := Tu(x_i) + t(s)$, $i=1,\ldots,m$; $Tl(y_i) := Tl(y_i) + t(s)$, $i=1,\ldots,n$.
The system will shift into state : $s' = ((x'_1, \tau'_1) \ldots (x'_{m-1}, \tau'_{m-1}); (x'_m, \sigma'_1, y'_1); \Delta; \Lambda; y'_2 \ldots y'_n)$;
where:

$$x'_1 = x_2, \ldots, x'_{m-1} = x_m, x'_m = x_1; \tau'_1 = \tau_2 - t(s), \ldots, \tau'_{m-1} = \tau_m - t(s)\, y'_1 = y_1, \ldots, y'_n = y_n\, \sigma'_1 = \ ;$$

σ being generate at random.

Case 4.5. The system is in the state:

$s = ((x_1, \tau_1) \ldots (x_p, \tau_p); (x_{p+1}, \sigma_1, y_1) \ldots (x_m, \sigma_{m-p}, y_{m-p}); \Lambda; y_{m-p+1} \ldots y_n)$; $1 \le p \le m-n-1$; i.e. p machines are operating , m-p machines are repaired by m-p workers, there are no machines are waiting for repair and n-m+p workers are in "laziness". The image of function λ changes as follows:

$Tu(x_i) := Tu(x_i) + t(s)$, $i=1,\ldots,p$; $Tr(x_i) := Tr(x_i) + t(s)$, $i=p+1,\ldots,m$;
$Tl(y_i) := Tl(y_i) + t(s)$, $i=m-p+1,\ldots,m$; $Trm(y_i) := Trm(y_i) + t(s)$, $i=1,\ldots,m-p$.

In order to express s' two cases will be studied:

Case 4.5.1.When $\tau_1 \le \sigma_1$ the following event will be machine x_1 shifting into repair, being repaired by worker y_{m-p+1} , the first in the queue of "lazy" workers; the system will shift into state:

$s' = ((x'_1, \tau'_1)\ldots(x'_{p-1}, \tau'_{p-1}); (x'_p, \sigma'_1, y'_1)\ldots(x'_m, \sigma'_{m-p+1}, y'_{m-p+1}); \Delta; y'_{m-p+2}\cdots y'_n)$ let σ be the time of repair of machine x_1 and k so that: $\sigma_k - t(s) \leq \sigma \leq \sigma_{k+1} - t(s)$; then:

$$x'_1 = x_2, \ldots, x'_{p-1} = x_p, x'_p = x_{p+1}, \ldots, x'_{p+k} = x_{p+k+1}, x'_{p+k+1} = x_1, x'_{p+k+2} = x_{p+k+2}, \ldots, x'_m = x_m;$$
$$y'_1 = y_1, \ldots, y'_{p+k} = y_{p+k}, y'_{p+k+1} = y_{m-p+1}, y'_{p+k+2} = y_{k+1}, \ldots, y'_{m-p+1} = y_{m-p};$$
$$\tau'_1 = \tau_2 - t(s), \ldots, \tau'_{p-1} = \tau_p - t(s);$$
$$\sigma'_1 = \sigma_1 - t(s), \ldots, \sigma'_k = \sigma_k - t(s), \sigma'_{k+1} = \sigma, \sigma'_{k+2} = \sigma_{k+1} - t(s), \ldots, \sigma'_{m-p+1} = \sigma_{m-p} - t(s).$$

Case 4.5.2. When $\tau_1 > \sigma_1$ the following event will be machine x_{p+1} shifting into operation, worker y_1 will be placed last in the queue of "lazy" workers; the system will shift into state:

$s' = ((x'_1, \tau'_1)\ldots(x'_{p+1}, \tau'_{p+1}); (x'_{p+2}, \sigma'_1, y'_1)\ldots(x'_m, \sigma'_{m-p-1}, y'_{m-p-1}); \Delta; y'_{m-p}\cdots y'_n)$; let τ be the time of continous operation of machine x_{p+1} and be k so that $\tau_k - t(s) \leq \tau \leq \tau_{k+1} - t(s)$; then:

$$x'_1 = x_1, \ldots, x'_k = x_k, x'_{k+1} = x_{p+1}, x'_{k+2} = x_{k+1}\ldots, x'_{p+1} = x_p; x'_{p+2} = x_{p+2}, \ldots, x'_m = x_m;$$
$$\tau'_1 = \tau_1 - t(s), \ldots, \tau'_k = \tau_k - t(s), \tau'_{k+1} = \tau, \tau'_{k+2} = \tau_{k+1} - t(s), \ldots, \tau'_{p+1} = \tau_p - t(s)$$
$$\sigma'_1 = \sigma_2 - t(s), \ldots, \sigma'_{m-p-1} = \sigma_{m-p} - t(s).$$
$$y'_1 = y_2, \ldots, y'_{m-p-1} = y_{m-p}; y'_{m-p} = y_{m-p+1}, \ldots, y'_{n-1} = y_n, y'_n = y_1.$$

Observations:

4.1. An event occuring in a system is equivalent to the transition of the system from one state into another; this is achieved using the "rule of the minimum time", i.e. an event will occur after a period of time equal to min $\{\tau_1, \sigma_1\}$ at the respective moment.

4.2. If $\tau_1 = \sigma_1$ a rule has to be set up for the determination of the following event; in the paper it was established that if $\tau_1 = \sigma_1$, then the following event will be the first of the machines on the operation list shifting into repair.

4.3. The initial state of the system consists of: all machines operate and all workers are in "laziness" (see case 4) and the vectors representing the image of the output function are all zero.

5. OBJECT-ORIENTED SIMULATION OF THE MODEL

The object-oriented simulation represents a reflection of OOP (Object-Oriented Programming) in the simulation of models with discrete events. It is thus possible to make use of the advantages offered by OOP in simulation (see [1],[2]). The object-oriented simulation of a model is in fact its decomposition into several submodels; the description of their behaviour uses methods grouped according to the objects corresponding to each submodel and emphasizing the connections between the component submodels. A possibile presentation of the digraph model associated to the considered simulation problem (see [5]), in this case is:

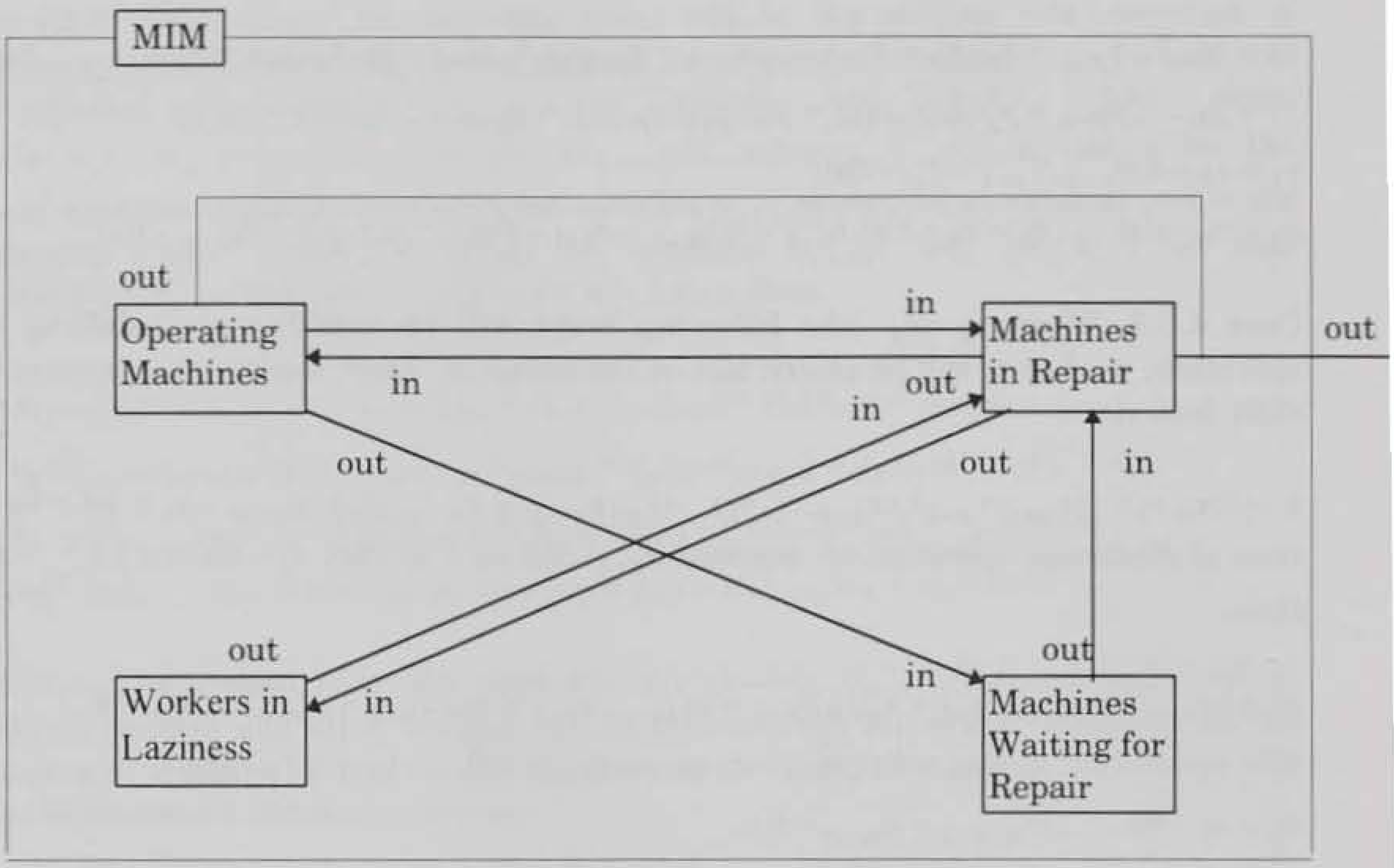

Figure 1: A possibile presentation of the digraph model

The correspondence between the submodels and the objects described by them is: "Operating Machines" correspond to M_Op, "Machines in Repair" to M_Rep, "Workers in Laziness" to W_Lazy and "Machines Waiting for Repair" to M_Waiting_Rep. The links betwen submodels are achieved through a seventh object called the coordinator. Object M_Op manages the dynamically alotted list of operating machines; each node of the list comprises the machine code, the time of continous operation and a pointer to the following element, as presented in section 2. The program frame describing the data and methods of this object is :

```
Type M_Op=Object
        First:...;                          Tu:array[1..dim_max] of real;
        Procedure Init(m:word);             Procedure Ins_Ord(Nod:...);
        Procedure Act_Time(t:real);         Procedure Act_Tu(t:real);
        Procedure Delete;                   Function Lg_List:Word;
       Function Total_Tu(m:word):real;  Function The_First:...
end;
```

where: First is a pointer to the first element of the list; *Tu* is the machines' operating times vector; Init method initializes the list of machines in operation with the empty list and *Tu* by zero; Ins_Ord method implements a new node into the list, in rising order of time; Delete method suppressess the first node of the list; Act_Time method adjusts the time of continous operation of the respective machine by subtracting the advance time (minimum time); Lg_List method supplies the number of nodes (machines) on the list; Act_Tu method actualizes the operation time of the machines. The description of the M_Rep object which manages the list of machines in repair is almost identical to that of object M_Op; the difference lies in the fact that each node of the list comprises an additional field comprising the code of the work repairing the respective machine; also, the object processes vectors *tr* and *trm* which have the same signifiance as in the formal description of the model. The sequence presenting the data and methods of this object is:

```
Type M_Rep=Object
        First:...;                              Tr:array[1..dim_max] of real;
        Trm:array[1..dim_max] of real;          Procedure Init(m:word);
        Procedure Ins_Ord(Nod:...);             Procedure Act_Time(t:real);
        Procedure Act_Tr(t:real);               Procedure Act_Trm(t:real);
        Procedure Delete;                       Function Lg_List:Word;
        Function Total_Tr(m:word):real;         Total_Trm(m:word):real;
        Function The_First:...                  end;
```

The W_Lazy object manages the dynamically alotted list of "lazy" workers, based on the queue principle, i.e. it will process the first and last fields, pointers to the first and last nodes of the queue; introduction into the queue will take place after the element shown by the last pointer, and deletion will be performed at the other end. Also, it will change vector tl, the signifiance of which has been presented in section 3. The sequence describing this object is:

```
Type W_Lazy=Object
        First,Last:...;
        Tl:array[1..dim_max] of real;
        Procedure Init(m:word);         Procedure Delete;
        Procedure Implement(....);              Procedure Act_Tlazy(t:real);
        Function Lg_qeue:word;                  Function The_First:...;
        Function Total_tl(m:word):real;         end;
```

The methods perform the actions suggested by their names. The list of machines waiting for repair, organized on the same principle as the list of "lazy" workers is processed by the M_Waiting_Rep object, which also updates vector ta (see section 3). The methods, whose names suggest the performed action, and the processed data are presented in the following description:

```
Type M_Waiting_Rep=Object
        First,Last:...;                 Ta:array[1..dim_max] of real;
        Procedure Init(m:word);         Procedure Delete;
        Procedure Implement(...);       Procedure Act_Twait_rep(t:real);
        Function Lg_queue:word;         Function The_First:...
        Function Total_ta(m:word):real;         end;
```

The coordinator object uses instances mo of M_Op type , mr of M_Rep type, *wl* of W_Lazy type and mwr of M_Waitig_Rep type; it is introduced by the following sequence:

```
type coordinator=Object
        Procedure InitSist;
        Function GenVarAl(...): real;
        Procedure SelectEv (var tip_ev: word ;var t_min : real);
        Procedure Transition(m,n:integer);
        Procedure TeoEfFact(.....)    end;
```

The Initsist method reads the model input parameters (number of machines, number of workers, the exponential variable parameters which generate the continuous operation and repair time, number of simulated events) and initializes the system, i.e. all´machines are operating, all workers are in "laziness", the list of machine in repair and the queue of machines waiting for repair are empty. Applying the Monte Carlo method, the GenVarAl method generates random variables, their values being the operation and repair times. The SelectEv method determines the type of the following event according to the list lengths and the times of the first operating of the first machine in repair. The sequence of Turbo Pascal language instructions required for this is:

```
Procedure coordonator.SelectEv;
        var P:...; r:...; q:...;t:...;
begin
        p:=mo.The_First; q:=mr.The_First; r:=ml.The_First; t:=mwr.The_First;
        if (mo.Lg_List<>0) and (mr.Lg_List<>0) and (ml.Lg_Queue=0) and
        (mwr.lg_Queue=0) and (p^.Time_Op<=q.Time_Rep)
        {case 4.1.1} then
                begin
                 tip_ev:=1;t_min:=p^.Time_Op
                end;
        if (mo.Lg_List<>0) and (mr.Lg_List<>0) and (ml.Lg_Queue=0) and
                        (mwr.lg_queue=0) and (p^.Time_Op>q^.Time_rep)
      {case 4.1.2} then
              begin
               tip_ev:=2;t_min:=q^.Timp_rep
              end;
     if (mo.Lg_List=0) and (mr.Lg_List<>0) and (ml.Lg_queue=0) and
                        (mwr.lg_queue<>0)
    {case 4.2} then
              begin
               tip_ev:=3;t_min:=q^.Timp_rep
              end;
        if (mo.Lg_List<>0) and (mr.Lg_List<>0) and (mwr.Lg_queue <>0) and
        (ml.lg_queue=0)   and (p^.Time_Op <=q^.Time_rep)
              {case 4.3.1}then
              begin
               tip_ev:=4; t_min:=p^.Timp_Func
              end;
  if (mo.Lg_List<>0) and (mr.Lg_List<>0) and (mwr.Lg_queue<>0) and
        (ml.lg_queue=0)  and (p^.Time_Op>q^.Time_rep)
  {case 4.3.2} then
```

```
            begin
             tip_ev:=5;    t_min:=q^.Timp_rep
            end;
 if (mo.Lg_List<>0) and (mr.Lg_List=0) and (mwr.lg_queue=0) and
        (ml.Lg_queue<>0)
    {case 4.4} then
            begin
             tip_ev:=6; t_min:=p^.Time_Op
            end;
  if (mo.Lg_List<>0) and (mr.Lg_List<>0)  and (mwr.Lg_queue=0) and
        (ml.lg_queue<>0)  and (p^.Time_Op<=q^.Time_rep)
      {case 4.5.1} then
            begin
             tip_ev:=7;
             t_min:=p^.Time_Op
            end;
  if (mo.Lg_List<>0) and (mr.Lg_List<>0)       and (ml.Lg_queue<>0) and
        (mwr.Lg_queue=0) and (p^.Time_Op>q^.Time_rep)
     {case 4.5.2}then
             begin
                  tip_ev:=8;       t_min:=q^.Time_rep
                          end;
```

The method also determines the minimum time after which the following event occurs, i.e. the transition of the system from one state into another. The system transition from one state into another is performed by the Transition Method , which is presented further on:

```
Procedure Coordinator. Transition(m,n:integer);
var     p:...; r:..; q:...; t_min:...;  nod:...;        nod1:...; nod2:...;
begin
        SelectEv(tip_ev,t_min);
        p:=mo.The_First;          q:=mr. The_First;
        r:=ml. The_First;         t:=mwr. The_First;
        case tip_ev of
        1: begin
           mo.Act_Tu(t_min);  mr.Act_Tr(t_min);   mr.Act_Trm(t_min);
              with mo do
                 begin
        cod_m1:=p^.Cod_M;       mwr.Implement(cod_m1);Delete;
        Act_Time(t_min);mr.Act_Time(t_min);
   end            end;
2: begin
   mo.Act_Tu(t_min);   mr.Act_Tr(t_min);   mr.Act_Trm(t_min);
    with mr do
     begin
      mo.Act_Time(t_min); nod2.Cod_M1:=q^.Cod_m;
      nod2.Time_op1:=GenVarAl(lam); mo.Ins_Ord(Nod2);
     cod_m1:=q^.Cod_W;  ml.Implement (cod_m1);  Delete; Act_Time(t_min);
    end end;
 3:begin
   mr.Act_Tr(t_min);     mr.Act_Trm(t_min);     mwr.Act_Ta(t_min);
   with mr do
    begin
  nod2.Cod_M1:=q^.Cod_m; nod2.Timp_op1:=GenVarAl(lam); mo.Ins_Ord(Nod2);
        nod.Cod_W:=q^.Cod_W; nod.Time_rep:=GenVarAl(niu);
      nod.Cod_m:= mwr.The_First^. CodM_wait; mwr.Delete;
     Delete; Act_Time(t_min);Ins_Ord(Nod);
    end end;
 4:begin
  mf.Act_Tu(t_min); mr.Act_Tr(t_min); mr.Act_Trm(t_min);mwr.Act_Ta(t_min);
  with mo do
    begin
      cod_m1:=p^.Cod_M;mwr.Implement(cod_m1);Delete;
      Act_Time(t_min); mr.Act_Time(t_min);
    end   end;
 5:begin
  mo.Act_Tu(t_min); mr.Act_Tr(t_min); mr.Act_Trm(t_min); mwr.Act_Ta(t_min);
   with mr do
    begin
      nod2.Cod_M1:=q^.Cod_m; nod2.Timp_op1:=GenVarAl(lam);
      mo.Act_Time(t_min); mo.Ins_Ord(Nod2);
      nod.Cod_m:=mwr.The_First^.CodwaitREp;nod.Cod_W:=q^.Cod_W;
 nod.Time_rep:=GenVarAl(niu);Delete; mwr.Delete; Act_Time(t_min);ins_ord(nod);
    end   end;
6:begin
     mo.Act_Tu(t_min);ml.Act_Tl(t_min);nod.Cod_m:=p^.Cod_M; mo.Delete;
     nod.Time_rep:=GenVarAl(niu); nod.Cod_W:=ml.The_First^.Lcod_W;
     ml.Delete;mo.Act_Time(t_min); mr.Ins_Ord(Nod);
   end;
7:begin
  mo.Act_Tu(t_min);mr.Act_Tr(t_min); mr.Act_Trm(t_min); ml.Act_Tl(t_min);
   nod.Cod_m:=p^.Cod_M; mo.Delete; nod.Time_rep:=GenVarAl(niu);
  nod.Cod_W:=ml.The_First^.Lcod_munc;  ml.Delete;
  mo.Act_Time(t_min);  mr.Act_Time(t_min);  mr.Ins_Ord(Nod);
 end;
8:begin
   mo.Act_Tu(t_min); mr.Act_Tr(t_min);  mr.Act_Trm(t_min); ml.Act_Tl(t_min);
   with mr do
   begin
 mo.Act_Time(t_min); nod2.Cod_M1:=q^.Cod_m; nod2.Timp_op1:=GenVarAl(lam);
      mo.Ins_Ord(Nod2);cod_w1:=mr.The_First^.Cod_W;  ml.implement(cod_w1);
      Delete;  Act_Time(t_min);
    end
  end;  end;  end;
```

6. PRACTICAL CONSIDERATIONS

In [4] the theoretical inference factor is established for $m=4$, $n=1$ and $\lambda=1$. With my simulation program, I obtained values approximately equal to the theoretical values. In the following table, for $\lambda=1$ and for different values of m and n, the results obtained using my program are presented, considering $\mu=2.0$ (slow service), $\mu=5.0$ (moderate service) and $\mu=10.0$ (fast service).

For $m=10, n=2$

niu	I	EF	EM
2.0	1.95932	0.41554	0.98749
5.0	0.50911	0.77843	0.73412
10.0	0.14008	0.90018	0.43779

For $m=10, n=3$

niu	I	EF	EM
2.0	0.71552	0.52240	0.92799
5.0	0.10162	0.81812	0.55033
10.0	0.02006	0.90727	0.30302

For $m=10, n=4$

niu	I	EF	EM
2.0	0.16303	0.64534	0.76237
5.0	0.01276	0.83317	0.41185
10.0	0.00261	0.90716	0.23150

Thus, I can find the best value of efficiency factor I, EM, EF (see section 2).

REFERENCES

[1] Beaumariage, T., and Mize, J.H., "Object oriented modeling; concepts and ongoing research", *Proceedings of the 1991 SCS Western Simulation Multiconference*, La Jolla, California.

[2] Doyle, R.J., "Object - oriented simulation programming", in: *Proceedings of the 1991 SCS Western Simulation Multiconference,* La Jolla, California.

[3] Garzia, R.F., and Garzia, M.R., *Network Modeling, Simulation, and Analysis,* AT&T Bell Laboratories, Columbus, Ohio, 1990.

[4] Shammas, N.C., *Turbo Pascal 6 Object Oriented Programming,* SAMS, 1991.

[5] Lee, A.M., *Teoria asteptarii cu aplicatii* (Applied Queing Theory), Editura Tehnica, Bucuresti, 1976.

[6] Vaduva, I., *Modele de simulare cu calculatorul*(Computer Simulation Models), Editura Tehnica, Bucuresti, 1977

[7] Vaduva I., a.o *Simularea proceselor economice* (Simulation of the Economic Processes). Editura Tehnica ;Bucuresti, 1983

[8] Zeigler, B.P., *Theory of Modelling and Simulation,* John Wiley & Sons, New York, 1976.

[9] Zeigler, B.P., *Object -Oriented Simulation with Modular Models,* Academic Press, San Diego, 1990.