ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1999
Seria: INFORMATYKA z. 36 Nr kol. 1414

Marcin SKOWRONEK
Politechnika Śląska, Instytut Informatyki

RÓWNOLEGŁA REALIZACJA EKSPERYMENTÓW MODELOWANIA W SIECI KOMPUTEROWEJ

Streszczenie. W artykule przedstawiono koncepcję równoległej realizacji sekwencji eksperymentów modelowania w sieci komputerowej. Do realizacji programu *Wykonawcy* użyto bezpośrednio narzędzi programowych wykorzystywanych do budowy modeli cyfrowych układów dynamicznych. Umieszczono wyniki wykonanych badań oceny efektywności opracowanych algorytmów równoległej realizacji sekwencji eksperymentów.

PARALLEL EXECUTION OF MODELLING EXPERIMENTS USING COMPUTER NETWORK

Summary. The idea of parallel processing of the sequence of experiments (occurring in the modelling tasks) in the network environment is presented in this paper. *The Executor* programs, that were built with restrictions resulting from modelling means. The results of the experiments connected with the efficiency estimate of the algorithms are also included.

1. Wprowadzenie

Koncepcję wykorzystania sieci NetWare firmy Novell do równoległej realizacji sekwencji eksperymentów przedstawiono w pracach [1] i [2]. Istotą tej koncepcji jest to, że pamięć dyskowa serwera sieci, udostępniona stacjom roboczym, może być również wykorzystana do celów komunikacji między procesami realizowanymi na poszczególnych stacjach. W algorytmach przedstawionych w pracach [1] i [2] do realizacji tej komunikacji wykorzystywane są operacje plikowe:

a) sprawdzenia istnienia pliku o określonej nazwie,

b) przemianowania lub usunięcia pliku o określonej nazwie,

c) odczytania zawartości wskazanego pliku,

d) zapisu informacji do wskazanego pliku.

Wymienione tu operacje, dostępne w uniwersalnych językach programowania, nie zawsze są dostępne w środkach programowych stosowanych do budowy modeli cyfrowych układów dynamicznych. Przykładem może tu być program SIMNON [3], w którym niedostępne są operacje a) i b). Pewną niedogodnością jest tu również ustalony format plików wejściowych. Kilkusekundowa prezentacja informacji o twórcach programu dyskwalifikuje go również jako model wsadowy uruchamiany z poziomu niezależnego programu komunikacji (przypadek 2 z pracy [1]). Program SIMNON nie umożliwia również wywołania potomnego procesu w makropoleceniu, tak więc zastosowanie oddzielnego programu dla obsługi funkcji komunikacji (przypadku 3 z pracy [1]) jest tu również niemożliwe.

Innym przykładem jest środowisko MATLAB-SIMULINK, w którym brak bezpośredniej operacji przemianowania pliku. Można tu zastępczo użyć polecenia *rename*, lecz z uwagi na pracę w środowisku Windows realizacja takiego polecenia jest czasochłonna. Ta sama przyczyna dyskwalifikuje również rozwiązanie polegające na uruchomieniu modelu wsadowego opracowanego w środowisku MATLAB-SIMULINK z niezależnego programu komunikacji oraz metodę opisaną jako przypadek 3 w pracy [1].

Łatwość budowy modeli cyfrowych za pomocą tych środków programowych i ich dostępność były motywem do modyfikacji algorytmów koordynacji równoległej realizacji sekwencji eksperymentów przedstawionych w pracach [1] i [2] i ich prezentacji w tym opracowaniu.

2. Koordynacja równoległej realizacji sekwencji eksperymentów

Przyjmuje się, że zadanie wykonania sekwencji M eksperymentów sterowane będzie przez proces zwany *Nadawcą*. Eksperymenty wykonywane będą przez procesy zwane *Wykonawcami*, których liczba N jest znana *Nadawcy*. Proces *Nadawcy* i procesy *Wykonawców* realizowane są na różnych stacjach sieci. Przed rozpoczęciem sekwencji eksperymentów są uruchomione procesy *Wykonawców*, które oczekują na komunikat od *Nadawcy*. Każdy z procesów *Wykonawców* zainicjowany jest z innego katalogu bieżącego. Przyjęcie takiego założenia wynika stąd, że modele zbudowane za pomocą dostępnych środków modelowania często korzystają z plików roboczych o ustalonych nazwach. Inicjowanie procesów *Wykonawców* w oddzielnych katalogach jest tu prostym sposobem uniknięcia kolizji dostępu do plików roboczych modelu. Przyjęto, że procesy *Wykonawców* inicjowane są w katalogach

o nazwach W1, W2, ..., W*N* podrzędnych w stosunku do katalogu, w którym zainicjowano proces *Nadawcy*.

Przyjmuje się, że program procesu *Nadawcy* opracowany jest za pomocą uniwersalnego języka algorytmicznego, w którym dostępne są wszystkie operacje plikowe. Elementarne operacje w procesie *Nadawcy* związane z wykonaniem pojedynczego eksperymentu to:

1. Dla każdego eksperymentu w sekwencji:
   - przygotuj plik z danymi,
   - wywołaj procedurę wyślij_odczytaj_komunikat.
2. Dopóki nie odczytano wszystkich wyników:
   - wywołaj procedurę wyślij_odczytaj_komunikat.

Procedura wyślij_odczytaj_komunikat składa się z dwóch faz:

a) fazy sprawdzania, czy istnieje wolny *Wykonawca* i wysłania komunikatu do *Wykonawcy*,

b) fazy sprawdzania, czy istnieją komunikaty odpowiedzi od *Wykonawców* i ich odczytu.

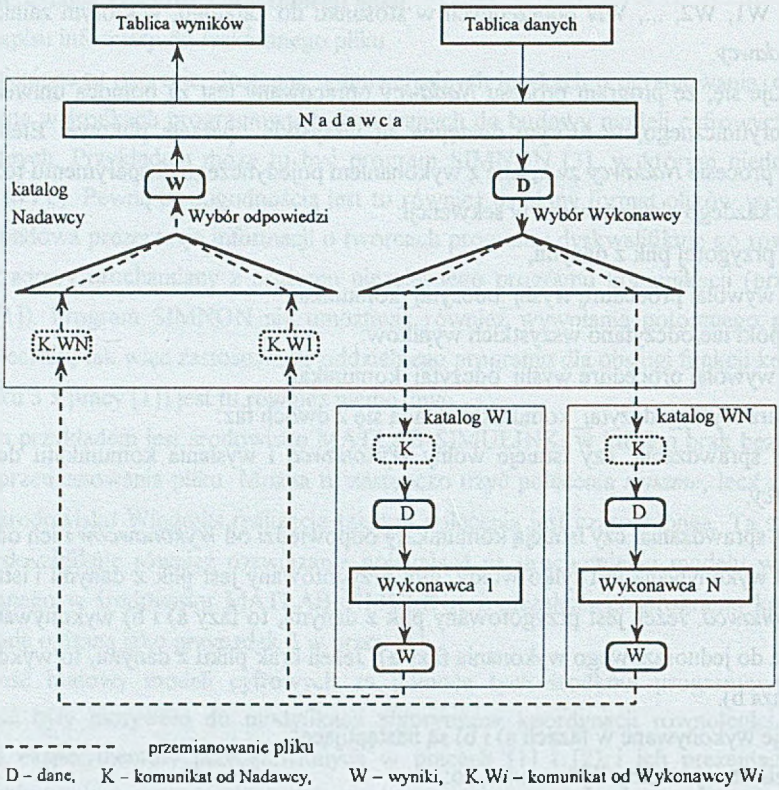
Faza a) wykonywana jest tylko wtedy, gdy przygotowany jest plik z danymi i istnieje nie zajęty *Wykonawca*. Jeżeli jest przygotowany plik z danymi, to fazy a) i b) wykonywane są na przemian aż do jednorazowego wykonania fazy a). Jeżeli brak pliku z danymi, to wykonywana jest tylko faza b).

Operacje wykonywane w fazach a) i b) są następujące:

a) jeżeli istnieje wolny *Wykonawca*, to:
   1) przemianuj plik z danymi na komunikat do określonego *Wykonawcy*,
   2) zaznacz wysłanie komunikatu,
   3) zaznacz zajęcie *Wykonawcy*,

b) dopóki istnieją komunikaty od *Wykonawców*, to:
   1) przemianuj plik odpowiedzi,
   2) zaznacz zwolnienie *Wykonawcy*,
   3) zaznacz odebranie odpowiedzi,
   4) odczytaj wyniki eksperymentu,
   5) usuń plik wyników.

W rozwiązaniu tym przyjęto, że komunikaty do *Wykonawców* kierowane są do ich katalogów bieżących, natomiast *Wykonawcy* kierują odpowiedzi do katalogu, z którego uruchomiono proces *Nadawcy*, przy czym nazwa pliku stanowiącego komunikat odpowiedzi wskazuje *Wykonawcę* (rys. 1). Pliki komunikatów zawierają tu dane eksperymentu (komunikat od *Nadawcy*) i wynik eksperymentu (komunikat od *Wykonawcy*).



Rys. 1. Komunikacja między *Nadawcą* i *Wykonawcami*
Fig. 1. Communication between *Sender* and *Executors*

2.1. Algorytm *Wykonawcy* dla uniwersalnego języka algorytmicznego

Zastosowanie uniwersalnego języka algorytmicznego umożliwia użycie dowolnej operacji plikowej. Algorytm *Wykonawcy* może tu zawierać następujące operacje:

a) czekaj na komunikat od *Nadawcy* (do *Wykonawcy*),
b) przemianuj plik komunikatu od *Nadawcy* na plik *rob*,
c) odczytaj dane z pliku *rob*,
d) wykonaj eksperyment,
e) zapisz wyniki eksperymentu do pliku wyników (pliku *rob*),
f) przemianuj plik wyników na komunikat od *Wykonawcy* (do *Nadawcy*).

Operacje te wykonywane są cyklicznie do chwili usunięcia procesu *Wykonawcy*. Należy tu zwrócić uwagę, że poprawne wykonanie operacji przemianowania pliku wymaga, by w katalogu docelowym nie istniał plik o nazwie określonej w operacji przemianowania.

2.2. Algorytm *Wykonawcy* dla programu SIMNON

Repertuar poleceń udostępniających operacje plikowe w programie SIMNON [3] jest bardzo skromny. Dostępne operacje to:

a) zapis parametrów lub warunków początkowych wybranego bloku do pliku (polecenie *save*),

b) odczyt zawartości pliku (polecenie *get*),

c) utworzenie pliku pustego (polecenie *write*),

d) zapis informacji do pliku (polecenie *write*).

Polecenia *save* i *write* tworzą pliki w katalogu bieżącym. Polecenie *save* niszczy starą zawartość pliku. Polecenie *write* może dopisywać informacje do istniejącego pliku lub przed zapisaniem zniszczyć starą zawartość. Polecenie *get* przeznaczone jest do odczytywania plików o ustalonej strukturze (pliki utworzone poleceniem *save*). Dodatkową cechą polecenia *get* jest to, że wskazany plik poszukiwany jest w następujących katalogach:

a) katalogu bieżącym (katalogu *Wykonawcy*),

b) katalogu wskazanym przez zmienną środowiskową 'SIMNON',

c) katalogu zdefiniowanym jako logiczny dysk przeszukiwany (polecenie MAP S*i*:=.....).

Katalogi b) i c) będziemy nazywali dalej katalogami macierzystymi.

Ostatnia właściwość polecenia *get* pozwala zastąpić operację „czekaj na komunikat od *Nadawcy*" operacją

<u>powtarzai</u> czytaj plik *kdow.t* <u>aż</u> a[m1]≠0.

Plik *kdow.t* z zerową wartością parametru *a* z pomocniczego bloku *m1* jest umieszczony w katalogu macierzystym, co gwarantuje poprawność wykonania operacji *get*. *Nadawca* przygotowuje taki plik z niezerową wartością parametru *a* i umieszcza go w katalogu *Wykonawcy*. Przykładowy ciąg operacji algorytmu *Wykonawcy* może być następujący:

a) <u>powtarzai</u> czytaj plik *kdow.t* <u>aż</u> a[m1]≠0,

b) wyślij komunikat od *Wykonawcy* (plik *kodw.t*),

c) <u>powtarzaj</u> czytaj plik *kdow1.t* <u>aż</u> a[m1]≠0,

d) odczytaj dane,

e) wykonaj eksperyment,

f) utwórz plik wyników,

g) wyślij komunikat od *Wykonawcy* (plik *kodw.t*).

Operacje te wykonywane są cyklicznie do chwili usunięcia procesu *Wykonawcy*. Operacja c) jest tu niezbędna, by możliwe było bezkonfliktowe usunięcie pliku *kdow.t* z katalogu bieżącego. Operacja usunięcia (przemianowania) pliku musi być wykonana przez *Nadawcę*. Plik komunikatu od *Wykonawcy* (*kodw.t*) musi być plikiem pustym. *Nadawca* poszukuje pliku komunikatów i pliku wyników w katalogu *Wykonawcy*.

Poniżej przedstawiono przykładowe makropolecenie będące implementacją algorytmu *Wykonawcy* w programie SIMNON:

```
macro r4
"realizacja procesu Wykonawcy
label start
"oczekiwanie na pierwszy komunikat
get kdow
disp a[m1]/a
if a. eq 0 goto start
"zakończenie procesu Wykonawcy
if a. gt 1.5 goto koniec
write(dk kodw ff)
label start1
"oczekiwanie na drugi komunikat
get kdow1
disp a[m1]/a
if a. eq 0 goto start1
"odczyt danych
get dane
"przekazanie danych do modelu
disp k[m2]/k
disp tr[m2]/tr
par k[sys]:k.
par tr[sys]:tr.
"wykonanie eksperymentu
simu 0 1000 0
"zapis wynikow i komunikatu
disp qq/q
write (dk wyniki)
write (dk wyniki) q.
write (dk kodw ff)
goto start
label koniec
write (dk kodw ff)
end
```

W przedstawionym makropoleceniu umieszczono również polecenia umożliwiające zakończenie procesu *Wykonawcy*.

2.3. Algorytm *Wykonawcy* dla środowiska MATLAB-SIMULINK

W środowisku MATLAB dostępne są następujące funkcje do realizacji operacji plikowych:

a) badanie istnienia pliku o wskazanej nazwie (funkcje *fopen* lub *exist*),

b) usuwanie pliku o wskazanej nazwie (funkcja *delete*),

c) odczyt pliku o ustalonej strukturze (funkcja *load*),

d) utworzenie pliku o wskazanej nazwie i we wskazanym katalogu (funkcje *fopen*, *fprintf*, *save*),

e) utworzenie pliku o pustej zawartości (funkcja *save*).

W bogatym zestawie funkcji MATLAB'a brak operacji przemianowania pliku. W tym celu można wykorzystać polecenie *rename* systemu operacyjnego, lecz jego realizacja w systemie Windows każdorazowo wyświetla okno, w którym wyświetlane są komunikaty o realizacji polecenia, co powoduje dodatkowe wydłużenie czasu realizacji eksperymentu.

Operacje przemianowania pliku w algorytmie *Wykonawcy* opisanym w punkcie 2.1 mogą być zastąpione operacją usunięcia pliku komunikatu po wcześniejszym odczytaniu danych (operacje b i c) oraz wykorzystania oddzielnego pliku wyników i pustego pliku komunikatu od *Wykonawcy* (operacje e i f). Przykładowy ciąg operacji algorytm *Wykonawcy* może być następujący:

a) czekaj na komunikat od *Nadawcy* (do *Wykonawcy*),

b) odczytaj dane z pliku,

c) usuń plik komunikatu do *Wykonawcy*,

d) wykonaj eksperyment,

e) zapisz wyniki eksperymentu do pliku wyników,

f) utwórz (pusty) plik komunikatu od *Wykonawcy* (do *Nadawcy*).

Operacje te wykonywane są cyklicznie do chwili usunięcia procesu. *Wykonawca* umieszcza plik wyników i komunikatu od *Wykonawcy* w katalogu *Nadawcy*. Usuwanie tych plików wykonywane jest przez *Nadawcę*. Z uwagi na przyjęty w MATLAB'ie sposób poszukiwania pliku, nie tylko w katalogu bieżącym, lecz również w wielu własnych katalogach systemowych, ważne jest, by w funkcji *fopen* wykorzystywanej do realizacji operacji "czekaj na komunikat do *Wykonawcy*" nazwa pliku komunikatu była poprzedzona *ścieżką dostępu*, co ogranicza poszukiwanie do wskazanego katalogu *Wykonawcy*.

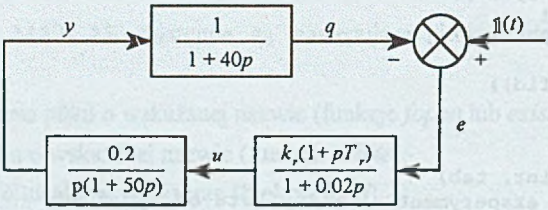
Poniżej przedstawiono przykładowe funkcje będące implementacją algorytmu *Wykonawcy* w środowisku MATLAB-SIMULINK:

```
function dane=czekaj
% czekaj - oczekiwanie na komunikat (plik dane) od Nadawcy
% dane   - tablica odczytanych danych
global DANE;
  dane=[];
  fid=fopen(DANE,'r');
  while fid==-1
    fid=fopen(DANE,'r');
  end;
  stat=fclose(fid);
  load dane;
  delete DANE;

function pisz(inr, tab)
%zapis wyników eksperymentu i komunikatu od Wykonawcy
%inr  - nr eksperymentu
%tab  - tablica wynikow
global KODW WYNIKI PUSTE
fid=fopen(WYNIKI, 'w');
fprintf(fid, '%g  %g', inr, tab);
fclose(fid);
% zapis pustego pliku komunikatu od Wykonawcy
eval(['save ', KODW, ' PUSTE']);

function f=wskaz2
% wyznaczanie wartości wskaźnika jakości
% przekazywaną jako pierwsza składowa wektora stanu
global MODEL KR TR TMAX Y0 OPTIONS
[t,x,y]=rk45(MODEL,TMAX,Y0,OPTIONS);
f=x(length(t),1);

function wykmatl(katw)
% procedura wykonawcy
% katw   - katalog bieżący Wykonawcy
global KODW WYNIKI PUSTE DANE
global KR TR TMAX Y0 OPTIONS MODEL
KR=0.02;TR=0;TMAX=1000;Y0=[0;0;0;0;0];
OPTIONS=[1e-3 1e-3 TMAX 0 0 0]; MODEL='ureg2';
% nazwy wykorzystywanych plików
```

```
% KODW    - zmienna zawierająca nazwę pliku komunikatu od Wykonawcy
% WYNIKI  - zmienna zawierająca nazwę pliku wynikowego
% DANE    - zmienna zawierająca nazwę pliku komunikatu do Wykonawcy
KODW=['..\kodw.' katw]; WYNIKI=['..\wyniki.' katw];
DANE=['..\' katw '\dane'];
dane=[0 0 0];
while dane(1)>=0
  dane=czekaj;
  if dane(1)>0
    KR=dane(2);
    TR=dane(3);
    f=wskaz2;
    dane(1) %wyświetlanie numeru wykonywanego eksperymentu
    pisz(dane(1),f);
  end
end
```

3. Wyniki badań

W celu oceny przyśpieszeń jakie daje równoległa realizacja sekwencji eksperymentów wykonano zestaw badań dla modelu układu dynamicznego o schemacie blokowym przedstawionym na rys. 2.



Rys. 2. Schemat blokowy układu dynamicznego
Fig. 2. The block diagram of the dynamical system

Celem modelowania było wyznaczenie charakterystyki

$$Q(k_r, T_r) = \int_0^{1000} e^2 dt \tag{1}$$

dla k_r = 0.02, 0.04, ..., 0.32 (16 wartości) i T_r = 0, 20, ..., 100 (6 wartości).

Rozpatrzono dwa przypadki implementacji algorytmu *Wykonawcy*:

a) wykorzystując program SIMNON,

b) wykorzystując środowisko MATLAB-SIMULINK.

Do badań użyto czterech stacji z procesorem Pentium MMX 166 MHz oraz jednej stacji z procesorem Pentium 133 MHz. Średnie czasy szeregowego wykonania sekwencji eksperymentów na tych stacjach przedstawiono w tabeli 1.

Tabela 1

Średnie czasy szeregowego wykonania sekwencji eksperymentów w [s]

Procesor	SIMNON	MATLAB-SIMULINK
Pentium MMX 166 MHz	138	208
Pentium 133 MHz	199	300

Podczas równoległej realizacji sekwencji eksperymentów proces *Nadawcy* uruchomiony był na stacji z procesorem Pentium 133 MHz, natomiast procesy *Wykonawców* uruchomiane były na stacjach z procesorem Pentium MMX 166 MHz. Uzyskane średnie czasy równoległego wykonania sekwencji eksperymentów oraz uzyskane przyśpieszenia przedstawiono w tabeli 2.

Tabela 2

Średnie czasy i przyśpieszenie równoległego wykonania sekwencji eksperymentów

Liczba wykonawców j	1	2	3	4
Algorytm *Wykonawcy* przygotowano w programie SIMNON				
Czas realizacji w [s] t_{rj}	144,3	75,2	49,2	37,3
Przyśpieszenie p_j	0,96	1,83	2,80	3,70
Algorytm *Wykonawcy* przygotowano w środowisku MATLAB-SIMULINK				
Czas realizacji w [s] t_{rj}	211,1	105,7	70,5	53,1
Przyśpieszenie p_j	0,99	1,97	2,95	3,92

Przyśpieszenie określono według wzoru:

$$p_j = \frac{t_{śr}}{t_{rj}} \tag{2}$$

gdzie:

p_j – przyśpieszenie równoległej realizacji sekwencji eksperymentów przez j *Wykonawców*,

$t_{śr}$ – czas szeregowej realizacji sekwencji eksperymentów na stacjach z procesorem Pentium MMX 166 MHz,

t_{rj} – czas równoległej realizacji sekwencji eksperymentów przez j *Wykonawców*.

Porównując wyniki szeregowego wykonania sekwencji eksperymentów i wyniki realizacji sekwencji eksperymentów z jednym wykonawcą można określić czas niezbędny na przygotowanie i transmisję danych i komunikatów między procesem *Nadawcy* a procesem *Wykonawcy*. W zależności od implementacji algorytmu *Wykonawcy* wynosi on:

6,3 [s] – algorytm opracowany z użyciem programu SIMNON,

3,1 [s] – algorytm opracowany w środowisku MATLAB-SIMULINK.

Dłuższy czas transmisji w przypadku implementacji algorytmu za pomocą programu SIMNON wynika stąd, że w rozwiązaniu tym wysyłana jest większa liczba komunikatów oraz wszystkie komunikaty i pliki usuwane są przez program *Nadawcy*.

4. Uwagi końcowe

Przedstawione wyniki badań potwierdzają zasadność wykorzystania sieci komputerowej w zadaniach, w których występuje potrzeba wykonywania sekwencji eksperymentów modelowania. Efektywność takiego rozwiązania jest tym większa, im krótszy jest czas transmisji komunikatów i danych w stosunku do czasu wykonania samych eksperymentów. Sytuacja taka występuje często podczas modelowania układów typu "stiff". W tym przypadku, z uwagi na znaczne różnice wartości własnych układu, zachodzi konieczność stosowania małego kroku całkowania, co wydłuża czas trwania eksperymentu.

W przedstawionych badaniach wynikiem eksperymentu jest pojedyncza liczba. W zadaniach modelowania, w których wynikiem eksperymentu są przebiegi sygnałów, w wielu przypadkach wystarczy je zapamiętać w oddzielnych pikach na stacjach roboczych *Wykonawców* bez potrzeba ich bezpośredniej transmisji do procesu *Nadawcy* w trakcie wykonywania sekwencji eksperymentów. Dla tego typu zadań również można zastosować przedstawioną metodę równoległej realizacji sekwencji eksperymentów.

Przedstawione rozwiązanie pokazuje również, że implementacja algorytmu *Wykonawcy* może być wykonana za pomocą typowych narzędzi programowych stosowanych do budowy modeli cyfrowych układów dynamicznych. Nawet skromny zestaw operacji plikowych dostępnych w programie SIMNON wystarczył do implementacji algorytmu *Wykonawcy*.

LITERATURA

1. Skowronek M.: Wykorzystanie sieci NetWare do równoległej realizacji zadań modelowania. ZN Pol. Śl. s. Informatyka, z. 27, Gliwice 1994.
2. Skowronek M.: Równoległa realizacja eksperymentów z wykorzystaniem pakietu DarP. ZN Pol. Śl. s. Informatyka, z. 31, Gliwice 1996.
3. Elmqvist H., Åström K. J., Schönthal T.: Simnon User's Guide for MS-DOS Computers. Department of Automatic Control Lund Institute of Technology, Lund 1986.
4. MATLAB User's Guide, The Math Works, Inc., 1994.
5. SIMULINK User's Guide, The Math Works, Inc., 1994.

Recenzent: Dr hab. inż. Stanisław Wołek, Prof. Pol. Rzeszowskiej

Wpłynęło do Redakcji 31 marca 1999 r.

Abstract

The modelling tasks require a lot of time because a great number of experiments is necessary. One of ways to limit this time is parallel execution of the experiments in distributed system like NetWare network. In the paper some ideas of the parallel execution of modelling experiments are discussed with respect on the feature of the modelling means used for model building. In the paper was also presented the communication rule between processes that work together basing on the files, which names are known to the processes, the examples of the algorithms and their implementation using programming means such as SIMNON or MATLAB-SIMULINK. The results of examples of applications illustrate the speed up which can be expected.