ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1990
Seria: TRANSPORT z. 16 Nr kol. 1100

Stanisław KRAWIEC

MODELOWANIE ZŁOŻONYCH SYSTEMÓW DYSKRETNYCH ZDARZEŃ

Streszczenie. W artykule zaprezentowano zasoby symulacyjne języka LOGLAN, dla potrzeb realizacji symulacji Złożonych Systemów Dyskretnych Zdarzeń. Typ SIMULATION, zaimplementowany w LOGLANie jest językiem problemowym (dialektem), pozwalającym na pisanie programów symulujących systemy rzeczywiste, opisane w sposób formalny jako Złożone Systemy Dyskretnych Zdarzeń. W artykule zaprezentowano strukturę programu symulującego Złożony System Dyskretnych Zdarzeń, używając oznaczeń używanych przy opisie formalnym tych systemów.

1. Uwagi wstępne

Język LOGLAN zrealizowany został w 1977 r. przez zespół pracowników Instytutu Informatyki Uniwersytetu Warszawskiego, kierowany przez profesora Andrzeja Salwickiego. W roku 1977 powstała pierwsza wersja - LOGLAN-77, a następnie wersja LOGLAN-82 - zaimplementowana na komputer MERA-400 pracujący pod kontrolą systemu operacyjnego SOM-3M (IIUW). W chwili obecnej, biorąc pod uwagę szerokie rozpowszechnianie się w kraju komputerów z rodziny IBM PC,dokonano implementacji LOGLAN-u na ten właśnie typ komputera, rozszerzając jego możliwości między innymi o elementy grafiki komputerowej. W roku 1990 przedstawiono raport kolejnej wersji LOGLANU-u - LOGLAN'88.

Aktualnie istnieje już szereg publikacji poświęconych LOGLANO-wi. Można je podzielić na: dotyczące teorii programowania oraz dotyczące aplikacji. Zagadnieniom teoretycznym poświęcone są między innymi następujące prace: [1,7,10,11,13,15,17,18,21,22,23,25,27,28,29], zagadnieniom aplikacyjnym: [2,3,4,5,6,8,12,14,20,26,30,31,32,33].

Artykuł niniejszy ma na celu prezentację loglanowskich zasobów symulacyjnych dla potrzeb realizacji symulacji Złożonych Systemów Dyskretnych Zdarzeń oraz przedstawia strukturę programu symulacyjnego odpowiadającą opisowi formalnemu Złożonych Systemów Dyskretnych Zdarzeń, przedstawionych w artykułach [35,36].

2. Zasoby symulacyjne języka LOGLAN

Typ SIMULATION zaimplementowany w LOGLANie jest językiem problemowym (dialektem) pozwalającym na pisanie programów symulujących systemy rzeczywiste. W typie tym użyte zostały narzędzia, pozwalające na quasi-równoległe wykonywanie programu.

Podstawowym pojęciem symulacyjnym w LOGLAN-ie jest proces. Modelowany system jest reprezentowany w programie symulacyjnym jako zbiór współbieżnych i współdziałających procesów. Każdy proces jest klasą prefiksowaną współprogramem o nazwie SIMPROCESS, w związku z czym można stwierdzić, że procesy są reprezentowane w programie przez współprogramy (COROUTINE).

Każdy proces jest ciągiem uporządkowanych w czasie zdarzeń związanych z działaniem symulowanego obiektu, posiada własne zmienne (atrybuty) oraz zbiór instrukcji do wykonania. Wartość lokalnych atrybutów procesu określa jego stan. Wystąpienie zdarzenia reprezentowane jest przez zmianę stanu procesu, a zatem zmianę stanu całego systemu. Stany procesu mogą być następujące:

- stan tworzenia – po instrukcji NEW tworzony jest proces, który jest w stanie zawieszenia;
- stan wstrzymania – stan ten występuje wtedy, gdy wykonywany jest inny proces, a proces wstrzymany ma ustalony moment czasowy, w którym ma być wznowione wykonywanie jego instrukcji;
- stan aktywny – w stanie tym wykonywane są instrukcje procesu; przyjmuje się, że instrukcje te są wykonywane w określonym momencie czasowym, aż do chwili przekazania sterowania do innego procesu lub do zakończenia procesu;
- stan zawieszenia – stan ten występuje wtedy, gdy wykonywany jest inny proces, a rozpatrywany proces nie ma ustalonego momentu czasowego, w którym ma być wznowiony;
- stan zakończony – gdy wykonane zostały wszystkie instrukcje procesu;
- stan usunięcia – gdy wykonano instrukcję KILL.

Moment czasowy, w którym dany proces ma być wznowiony, nosi nazwę zawiadomienia. Każdy proces może mieć co najwyżej jedno zawiadomienie. Jeżeli proces nie posiada zawiadomienia, to jest to równoważne z tym, że jest on zawieszony.

Wszystkie zdarzenia (events) w symulowanym systemie są uporządkowane w czasie. Jest to osiągnięte przez umieszczenie znaczników zdarzeń (event

notices) w kolejce priorytetowej reprezentującej oś symulowanego czasu systemowego. Do planowania i porządkowania zdarzeń służą operacje: HOLD, SCHEDULE, CANCEL, PASSIVATE i RUN.

Ogólny schemat loglanowskiego typu SIMULATION jest następujący:

```
unit FIFO: class; ... ;
unit PRIORITYQUEUE: FIFO class ; ... ;
unit SIMULATION: PRIORITYQUEUE class ;
  ......
  var  MAINPR: MAINPROGRAM, ...
  unit SIMPROCESS:  FIFOEL coroutine ; ... ;
  unit MAINPROGRAM: SIMPROCESS class ; ... ;
  unit TIME:        function : REAL ; ... ;
  unit CURRENT:     function : SIMPROCESS ; ... ;
  unit SCHEDULE:    procedure (P: SIMPROCESS, T: REAL) ; ... ;
  unit HOLD:        procedure (T: REAL) ; ... ;
  unit PASSIVATE:   procedure ; ... ;
  unit RUN:         procedure (P: SIMPROCESS) ; ... ;
  unit CANCEL:      procedure (P: SIMPROCESS) ; ... ;
begin
......
end SIMULATION;
```

Program symulacyjny w języku LOGLAN jest blokiem prefiksowanym klasą SIMULATION.

3. Sterowanie procesami w LOGLAN-ie

Sterowanie to polega na planowaniu momentów czasowych, w których ma nastąpić uruchamianie bądź wznawianie określonych procesów, czyli tworzenie zawiadomień dla procesów. Do każdego procesu można się odwołać poprzez zmienną referencyjną, której wartość jest określona po wykonaniu instrukcji NEW .

Przykładowo niech P jest zmienną referencyjną wskazującą na proces A P:= NEW PROCES_A , wykorzystując funkcje pomocnicze klasy SIMULATION można uzyskać następujące informacje :

- czy proces jest zakończony lub zawieszony ?
  P.IDLE = TRUE (gdy tak);
- czy proces jest zakończony ?
  P.TERMINATED = TRUE (gdy tak);
- jaki jest moment czasowy , w którym proces ma zaplanowane zdarzenie ?
  CZAS_WZNOWIENIA := P.EVTIME ; (gdy proces był zakończony lub zawieszony, nastąpi sygnalizacja błędu - wysyłany jest sygnał

IDLEPROC).

W celu zaplanowania zdarzenia dla procesu P , czyli ustawienia zawiadomienia dla procesu P , należy użyć procedury SCHEDULE (CALL SCHEDULE (P,T)); procedura ta ustawia zawiadomienie dla procesu wskazywanego przez zmienną referencyjną P na moment czasowy T . Jeżeli proces ten posiadał już zawiadomienie , to jest ono usuwane. Proces P przed wywołaniem procedury SCHEDULE może być w stanie zawieszonym, czyli nie mieć zawiadomienia .Jeżeli chwila T jest mniejsza od czasu aktualnego, jako T przyjmuje się czas aktualny. Procedura SCHEDULE nie powoduje zmiany sterowania procesami , tzn. po procedurze tej wykonywane są instrukcje procesu, który ją wywołał, chyba że P jest procesem wywołującym procedurę SCHEDULE, czyli procesem aktywnym. W takim przypadku procedura SCHEDULE jest równoważna procedurze HOLD (T-TIME), gdzie TIME jest funkcją podającą aktualny czas symulacji .

```
unit SCHEDULE: procedure(P: SIMPROCESS, T: REAL);
   (* aktywacja procesu P w chwili T *)
   begin
      if T<TIME THEN T:= TIME fi;
      if P=CURRENT then
         call HOLD(T-TIME)
       else
         if P.IDLE and P.EVENTPOM=NONE then
            P.EVENT,P.EVENTPOM:= new EVENTNOTICE(RANDOM);
            P.EVENT.PROC:= P;
          else
            if P.IDLE then
               P.EVENT:= P.EVENTPOM;
               P.EVENT.PRIOR:=RANDOM;
             else
               P.EVENT.PRIOR:=RANDOM;
               call PQ.DELETE(P.EVENT)
            fi
         fi;
         P.EVENT.EVENTTIME:= T;
         call PQ.INSERT(P.EVENT);
      fi;
   end SCHEDULE;
```

Do sterowania procesami służą również procedury HOLD, PASSIVATE, RUN i CANCEL.

Procedura HOLD

```
unit HOLD: procedure(T: REAL);-- wstrzymaj
   begin
      call PQ.DELETE(CURRENT.EVENT);
      CURRENT.EVENT.PRIOR:=RANDOM;
      if T<0 THEN T:=0; fi;
      CURRENT.EVENT.EVENTTIME:=TIME+T;
      call PQ.INSERT(CURRENT.EVENT);
      call CHOICEPROCESS;
   end HOLD;
```

Wywołanie : CALL HOLD(dt)

Służy do wstrzymywania procesu aktywnego, tj. takiego, w którym wywołano tę procedurę i ustawienia zawiadomienia dla tego procesu na moment czasowy równy TIME+dt . Jeżeli dt jest ujemne, to przyjmuje się dt=0. Następnie uruchamiany jest proces o najbliższym zawiadomieniu. Jeżeli jest kilka procesów o tej samej wartości zawiadomienia to proces do aktywacji wybierany jest losowo ze zbioru procesów o tej samej wartości zawiadomienia . Jeżeli brak zawiadomień, to uruchamiany (wznawiany) jest proces MAINPR, czyli proces główny. Proces główny można rozumieć jako zasadniczy moduł symulacyjny (najbardziej zewnętrzny blok programu symulacyjnego), z którego są uruchamiane procesy "potomne".

Procedura PASSIVATE

```
unit PASSIVATE: PROCEDURE;
   begin
     call PQ.DELETE(CURRENT.EVENT);
     CURRENT.EVENT:=NONE;
     call CHOICEPROCESS
   end PASSIVATE;
```

Wywołanie : CALL PASSIVATE

Służy do przerwania procesu aktywnego , nie ustawiając mu następnego zawiadomienia, czyli zawiesza ten proces. Po przerwaniu danego procesu uruchamiany jest inny proces o najbliższym zawiadomieniu, analogicznie jak w procedurze HOLD .

Procedura RUN

```
unit RUN: procedure(P: SIMPROCESS);
   begin
      CURRENT.EVENT.PRIOR:=RANDOM;
      if not P.IDLE then
         P.EVENT.PRIOR:=0;
         P.EVENT.EVENTTIME:=TIME;
         call PQ.CORRECT(P.EVENT,FALSE)
       else
         if P.EVENTPOM=NONE then
            P.EVENT,P.EVENTPOM:=new EVENTNOTICE(0);
            P.EVENT.EVENTTIME:=TIME;
            P.EVENT.PROC:=P;
            call PQ.INSERT(P.EVENT)
          else
            P.EVENT:=P.EVENTPOM;
            P.EVENT.PRIOR:=0;
            P.EVENT.EVENTTIME:=TIME;
            P.EVENT.PROC:=P;
            call PQ.INSERT(P.EVENT);
         fi
      fi;
      call CHOICEPROCESS;
   end RUN;
```

Wywołanie : CALL RUNC P)
Służy do przerwania bieżącego procesu aktywnego (proces ten nadal posiada zawiadomienie, które miał w chwili aktywacji) i uruchomienia lub wznowienia procesu P . Jeżeli proces P miał już zawiadomienie , to zawiadomienie to jest uaktualniane na moment czasowy określony przez funkcję TIME .

Procedura CANCEL

```
unit CANCEL: procedure(P: SIMPROCESS);
    begin
      if P= CURRENT then call PASSIVATE else
         call PQ.DELETE(P.EVENT);
         P.EVENT:=NONE;
      fi;
    end CANCEL;
```

Wywołanie : CALL CANCEL (P) .
Powoduje likwidację zawiadomienia dla procesu P , czyli zawieszenie tego procesu. Jeżeli proces P nie miał zawiadomienia , to CALL CANCEL (P) jest instrukcją pustą .

Funkcja CURRENT

```
unit CURRENT: function: SIMPROCESS;
    begin
      RESULT := CURR;
    end;
```

Funkcja ta wskazuje pierwszy (bieżący) proces na osi czasu.

4. Kolejki proste i priorytetowe

Współprogram SIMPROCESS , którym prefiksuje się klasę użytkownika, aby była procesem , jest z kolei prefiksowany klasą FIFOEL, co umożliwia grupowanie procesów w kolejki proste. Kolejka, a ściślej zmienna referencyjna wskazująca na tę kolejkę , musi być typu FIFO.

```
Np.  VAR KOLEJKA: FIFO;
     BEGIN
          KOLEJKA:= NEW FIFO;    <---
          ..................
     END
```

Bezpośrednio po instrukcji wskazanej strzałką kolejka o nazwie KOLEJKA jest pusta . Można później wykonać następujące czynności:

- wstawić obiekt (proces) do kolejki:

  CALL P.INTO(KOLEJKA); gdzie P jest zmienną referencyjną

wskazującą na ten obiekt ;

- usunąć pierwszy obiekt z kolejki :

CALL KOLEJKA.OUTFIRST .

Ponadto można korzystać z następujących funkcji :

- wskazanie pierwszego obiektu w kolejce, a ściślej podanie referencji do tego obiektu:

PA := KOLEJKA.FIRST ; teraz poprzez PA można uzyskać dostęp do atrybutów obiektu ;

- sprawdzenie , czy kolejka jest pusta :

KOLEJKA.EMPTY = TRUE (gdy tak);

- podanie liczby obiektów (elementów) w kolejce :

N := KOLEJKA.CARDINAL .

Kolejki priorytetowe są mechanizmem pomocniczym klasy SIMULATION stosowanym do tworzenia zawiadomień dla procesów . Elementy kolejki są wybierane według atrybutu typu REAL, poczynając od najmniejszego do największego . W klasie SIMULATION atrybutem tym jest moment czasowy zawiadomienia . Element kolejki musi być typu ELEM, zaś sama kolejka typu QUEUHEAD.

Operacje możliwe do wykonania dla kolejek priorytetowych, przy założeniu

```
VAR X,R: ELEM,
         Q: QUEUHEAD;
```

są następujące :

X := Q.MIN; X wskazuje na najmniejszy element kolejki Q.

CALL Q.INSERT(R); wstawienie elementu R do kolejki Q.

CALL Q.DELETE(R); usunięcie elementu R z kolejki Q (praktycznie służy do usuwania elementu najmniejszego).

Elementy kolejki można powoływać w następujący sposób :

```
VAR Y : OBIEKT;
UNIT OBIEKT : ELEM CLASS;
  ..........
  ..........
```

3. Struktura programu symulacyjnego Złożonych Systemów Dyskretnych Zdarzeń

Poniżej przedstawiono strukturę typowego programu wykorzystującego zasoby symulacyjne języka LOGLAN i przedstawiającego Złożony System Dyskretnych Zdarzeń. Używane oznaczenia starano się maksymalnie przybliżyć do opisu formalnego Złożonych Systemów Dyskretnych Zdarzeń, przedstawionego w [35,36].

```
PROGRAM ZŁOŻONYSYSTEMDYSKRETNYCHZDARZEŃ
(klasa SIMULATION)  (* włączona na zasadzie konkatenacji tekstów *)
BEGIN
PREF SIMULATION BLOCK;
...
UNIT ELEMENT_AKTYWNY_A: SIMPROCESS_CLASS(a:INTEGER);
CONST parametr_1 :=          ;
      parametr_2 :=          ;
      :
      parametr_lpa :=        ;

VAR   zmienna_1 :=           ;  (* STAN_ELEMENTU *)
  (*(*zmienna_2 :=           ;  ATRUBUT_CZASU*)*)
      zmienna_3 :=           ;
      :
      zmienna_lza :=         ;

BEGIN  (* MAIN ELEMENT_AKTYWNY_A *)
...
(* kolejne zdarzenie *)  (* przykładowe akcje *)
zmienna_1:=       ;
IF parametr_2=100 then zmienna_7=zmienna_6 else zmienna_7=zmienna_lza FI;
CALL SCHEDULE(ELEMENT_B(5),CZAS_OPÓŹNIENIA);
CALL RUN(ELEMENT_C(2));
ELEMENT_D.zmienna_3=ELEMENT_D.zmienna_3+1;
...
CALL HOLD(OKRES_CZASU_DO_ZMIANY_STANU);
(* kolejne zdarzenie *)
...
END ELEMENT_AKTYWNY_A;
...
UNIT ELEMENT_AKTYWNY_G: SIMPROCESS_CLASS(g:INTEGER);
CONST parametr_1 :=          ;
      parametr_2 :=          ;
      :
      parametr_lpg :=        ;
VAR   zmienna_1 :=           ;  (* STAN_ELEMENTU *)
  (*(*zmienna_2 :=           ;  ATRUBUT_CZASU*)*)
      zmienna_3 :=           ;
      :
      zmienna_lzg :=         ;
```

```
BEGIN (* MAIN ELEMENT_AKTYWNY_G
 ...
 (* kolejne n-te zdarzenie *)
 DO (* początek pętli nieskończonej *)
  IF (* warunek_aktywacji = TRUE *)
   THEN (* akcje zdarzenia *)
        ...
        (* przykładowe akcje na elementach podległych *)
        ...
        CALL RUN(ELEMENT_A(2));
        ELEMENT_A(3).zmienna_4:=ELEMENT_A(3).zmienna_4+100;
        ...
        (* przykładowe akcje na swoich zmiennych *)
        ...
        zmienna_1:= (* nowa wartość zmiennej stanu *)
        zmienna_3:= zmienna_3-50;
        IF zmienna_4 <= 100 THEN zmienna_4=zmienna_4+10 FI;
        ...
        EXIT; (* wyjście z pętli DO-OD po realizacji zdarzenia *)
   ELSE
        CALL_PASSIVATE; (* zawieszenie akcji simprocessu, czyli nadanie
                           zmiennej zmienna_2 wartości ∞*)
        REPEAT; (* ponowne sprawdzenie warunku aktywacji WARUNEK_AKTYWACJI,
                   po aktywacji tego procesu przez inny proces tego programu
                   *)
  FI;
 OD; (* koniec pętli nieskończonej *)
 CALL HOLD (OKRES_CZASU_DO_ZMIANY_STANU); (* nadanie zmiennej zmienna_2
                                             wartości OKRES_CZASU_DO_ZMIANY_
                                             _STANU *)
 (* kolejne n+1 zdarzenie *)
 ...
 (* kolejne n+k zdarzenie *)
 IF (* INNY_WARUNEK_AKTYWACJI=TRUE *)
  THEN (* akcje zdarzenia *)
     ...
     (* przykładowe akcje na elementach podległych *)
     IF (* warunek_1 *) THEN CALL RUN(ELEMENT_F(20)) FI;
     IF (* warunek_2 *)
      THEN CALL RUN(ELEMENT_F(21))
      ELSE CALL SCHEDULE(ELEMENT_F(21),OKRES_CZASU
     FI;
     ELEMENT_M.zmienna_3:=ELEMENT_N.zmienna_4+5.2;
  ELSE CALL HOLD (* OKRES_CZASU_DO_PRÓBY_PONOWNEJ_AKTYWACJI *)
```

```
  ...
END ELEMENT_AKTYWNY_G;
UNIT ELEMENT_AKTYWNY_ZEWNĘTRZNY_H: SIMPROCESS_CLASS(h:INTEGER);
CONST parametr_1 :=          ;
      parametr_2 :=          ;
       :
      parametr_1ph :=        ;

VAR   zmienna_1 :=           ;
  (*(*zmienna_2 :=           ;*)*)
       :
      zmienna_1zh :=         ;

BEGIN  (* MAIN ELEMENT_AKTYWNY_ZEWNĘTRZNY_H *)
...
(* W ELEMENTACH TEGO TYPU MOŻNA UWZGLĘDNIĆ ODDZIAŁYWANIE CZŁOWIEKA NA MODEL
*)
...
END ELEMENT_AKTYWNY_ZEWNĘTRZNY_H;
:
UNIT ELEMENT_AKTYWNY_ZEWNĘTRZNY_K: SIMPROCESS_CLASS(k:INTEGER);
CONST parametr_1 :=          ;
      parametr_2 :=          ;
       :
      parametr_1pk :=        ;

VAR   zmienna_1 :=           ;
  (*(*zmienna_2 :=           ;*)*)
       :
      zmienna_1zk :=         ;
...
BEGIN  (* MAIN ELEMENT_AKTYWNY_ZEWNĘTRZNY_K *)

END ELEMENT_AKTYWNY_ZEWNĘTRZNY_K
UNIT ELEMENT_PASYWNY_L: CLASS(l:INTEGER);
CONST parametr_1 :=          ;
      parametr_2 :=          ;
       :
      parametr_1pl :=        ;
```

```
VAR   zmienna_1 : =          ; (* STAN_ELEMENTU *)
      zmienna_2 : =          ;
      :
      zmienna_1zl : =        ;
...
BEGIN  (* MAIN ELEMENT_PASYWNY_L *)
...
(* EWENTUALNE AKCJE WEWNĄTRZ TEGO ELEMENTU *)
...
END ELEMENT_PASYWNY_L
:
UNIT ELEMENT_PASYWNY_R: CLASS(r:INTEGER);
CONST parametr_1 : =         ;
      parametr_2 : =         ;
      :
      parametr_1pr : =       ;

VAR   zmienna_1 : =          ; (* STAN_ELEMENTU *)
      zmienna_2 : =          ;
      :
      zmienna_1zr : =        ;
...
BEGIN  (* MAIN ELEMENT_PASYWNY_R *)
...
(* EWENTUALNE AKCJE WEWNĄTRZ TEGO ELEMENTU *)
...
END ELEMENT_PASYWNY_R

(*                          PROGRAM GŁÓWNY
        (*  tworzenie zmiennych referencyjnych do elementów  *)

VAR  ELEMENT_A :       ARRAYOF ELEMENT_AKTYWNY_A;
     ...
     ELEMENT_G :       ARRAYOF ELEMENT_AKTYWNY_G;
     ELEMENT_H :       ARRAYOF ELEMENT_AKTYWNY_ZEWNĘTRZNY_H;
     ...
     ELEMENT_K :       ARRAYOF ELEMENT_AKTYWNY_ZEWNĘTRZNY_K;
     ELEMENT_L :       ARRAYOF ELEMENT_PASYWNY_L;
     ...
     ELEMENT_R :       ARRAYOF ELEMENT_PASYWNY_R;
     ...
```

```
    PARAMETR_S :      ARRAYOF  (* TYP PARAMETRU *)
    ...
    PARAMETR_Z :      ARRAYOF  (* TYP PARAMETRU *)
    ...

BEGIN  (* PROGRAM GŁÓWNY *)
 (* generacja obiektów *)
 ...
 (* przykładowa generacja obiektów reprezentujących ELEMENT_AKTYWNY_G *)
 ARRAY ELEMENT_G DIM(1:g);
 FOR zm_pomocnicza:=1 TO g DO
                          ELEMENT_G(zm_pomocnicza)=NEW ELEMENT_
                          _AKTYWNY_G(zm_pomocnicza);
                          ...
                          (* nadanie warunków początkowych zmiennym i
                          parametrom tego obiektu *)
                          ...
                         OD;
 ...
 (* ustalenie parametrów ogólnych symulacji *)
 ... .
 CALL HOLD(OKRES_CZASU_DO_ZAKOŃCZENIA_SYMULACJI);
END; (* PROGRAM GŁÓWNY *)
END ZŁOŻONYSYSTEMDYSKRETNYCHZDARZEŃ.
```

Wszystkie atrybuty czasu elemnetów aktywnych (element_2 w treści programu), występujące w opisie formalnym Złożonego Systemu Dyskretnych Zdarzeń, w strukturze programu zaznaczone są tylko symbolicznie, jako komentarz ((* ...*)*), bowiem operacje na tych właśnie atrybutach są istotą klasy SIMULATION i realizowane są w sposób automatyczny za pomocą instrukcji HOLD, RUN, PASSIVATE itd.

4. Uwagi końcowe

Przykłady programów symulacyjnych w języku LOGLAN, bazujących na zasobach klasy SIMULATION zawierają między innymi publikacje: [3], [4], [8], [23], [25], [27].

Na bazie przedstawionej w artykule metodologii zrealizowany został także model symulacyjny ruchu pociągów dla potrzeb regulacji ruchu.

LITERATURA

[1] BARTOL W.: Programowanie za pomocą współprogramów. Informatyka nr 6 / 1983.
[2] KOŁODZIEJSKA H.: Implementacja operacji na tekstach w języku LOGLAN. Sprawozdania Instytutu Informatyki Uniwersytetu Warszawskiego, Warszawa 1984.
[3] KONIECZNY R. (praca zbiorowa): Zastosowanie języka LOGLAN do modelowania dużych systemów transportowych na przykładzie modelu ruchu pociągów - praca naukowo-badawcza NB-277/RT/87 wykonana w ramach programu RI.P.09 *"Rozwój języków, metod i podstaw formalnych oprogramowania"*. Instytut Transportu Politechniki Śląskiej, Katowice 1987.
[4] KONIECZNY R.: Język programowania LOGLAN jako narzędzie modelowania systemów transportowych. Zagadnienia Transportu PAN, Warszawa nr 3/4 1986/87.
[5] KONIECZNY R.: Przegląd zasobów języka LOGLAN dla potrzeb modelowania systemów transportowych. Automatyka Kolejowa, nr 7, 9 / 1988.
[6] KONIECZNY R.: Charakterystyka zasobów języka LOGLAN dla potrzeb modelowania systemów transportowych. Zeszyt Naukowy Politechniki Śląskiej, s. Transport, nr 9, Gliwice 1989.
[7] KONIECZNY R.: Specyfikacja formalna w projektowaniu programów komputerowych - *niniejszy zeszyt*.
[8] KONIECZNY R.: Język programowania LOGLAN jako narzędzie symulacji systemu transportowego - *niniejszy zeszyt*.
[9] KONIECZNY R.: Zasoby symulacyjne języka SIMULA-67 - *niniejszy zeszyt*.
[10] KRECZMAR A., SALWICKI A.: Język programowania LOGLAN. Informatyka nr 7,8 / 1982, 1 / 1983.
[11] KRECZMAR A.: Język programowania LOGLAN 82. Materiały II Konferencji Użytkowników Minikomputera MERA-400, Gdańsk 1984.
[12] KRECZMAR A.: Dokumentacja dla minikomputera MERA-400 - Język programowania LOGLAN-82 - Podstawowe konstrukcje i cechy charakterystyczne języka - IIUW, 1984 .
[13] KRECZMAR A.: Języki obiektowe - Informatyka nr 1-2 / 1988
[14] KOSTON H.: Generatory rozkładów pseudolosowych w LOGLANIE - praca magisterska, Instytut Informatyki Uniwersytetu Warszawskiego, 1987 (opiekun naukowy: prof. A. Salwicki).
[15] MULDNER T.: Pewne uwagi o dwóch nowych językach programowania wysokiego poziomu LOGLAN i ADA. Biuletyn Techniczny MERA nr 11-12 / 1980.
[16] OKTABA H., RATAJCZAK W.: SIMULA 67. WNT, Warszawa 1980.
[17] OKTABA H.: Klasy w LOGLANIE. Informatyka nr 5 / 1983.
[18] OKTABA H.: Prefiksowanie klasami w LOGLANIE. Informatyka nr 6/1983.
[19] PERKOWSKI P.: Technika symulacji cyfrowej. WNT, Warszawa 1980.
[20] SALWICKI A.: LOGLAN - narzędzie produkcji oprogramowania. Materiały II Konferencji Użytkowników Minikomputera MERA- 400, Gdańsk 1984.
[21] SALWICKI A.: Metodologia programowania w LOGLANIE. Materiały III Konferencji Użytkowników Minikomputera MERA-400 , Gdańsk 1985.
[22] SZAŁAS A., SZCZEPAŃSKA-WASERSZTRUM D.: Exception handling in parallel computations. Sprawozdania Instytutu Informatyki Uniwersytetu Warszawskiego, Warszawa 1984.
[23] SZCZEPAŃSKA D.: Narzędzia symulacyjne - art. poz. [25].
[24] WINKOWSKI J.: Programowanie symulacji procesów . WNT, Warszawa 1974.
[25] *Język programowania LOGLAN 82. Materiały Jesiennej Szkoły PTI - Serock, 1985.*
[26] *LOGLAN - USER'S GUIDE (version JANUARY '88) IIUW, Warsaw. (Suplement dla IBM-PC).*
[27] *Materiały: International Summer School of the Programming Language LOGLAN. ZABORÓW, POLAND September, 5-10.1983, IIUW.*
[28] *Report on the programming language LOGLAN. Praca zbiorowa, Instytut Informatyki UW 1982.*
[29] *Report on the LOGLAN 82 programming language. PWN Warszawa - Łódź, 1984.*
[30] Konieczny R.: Zasoby symulacyjne języka LOGLAN. Zeszyt Naukowy Politechniki Śląskiej, s.Transport, nr 13 Gliwice 1989.

[31] Konieczny R.: Przykłady rozwiązania problemów symulacyjnych w języku LOGLAN. Zeszyt Naukowy Politechniki Śląskiej, s.Transport, nr 13 Gliwice 1989.
[32] Konieczny R.: Zagadnienia rozszerzenia loglanowskich zasobów symulacyjnych. Zeszyt Naukowy Politechniki Śląskiej, s.Transport, nr 13 Gliwice 1989.
[33] Konieczny R.: Język programowania LOGLAN jako narzędzie opisu modelu systemu transportowego. Zeszyt Naukowy Politechniki Śląskiej, s.Transport, nr 13 Gliwice 1989.
[34] Kreczmar A., Salwicki A., Warpechowski M.: LOGLAN'88 - Report on the programming language - Lecture Notes in Computer Science 414 - Springer-Verlag.
[35] Krawiec S.: Wybrane zagadnienia opisu Systemów Dyskretnych Zdarzeń - niniejszy zeszyt.
[36] Krawiec S.: Złożone Systemy Dyskretnych Zdarzeń - niniejszy zeszyt.

MODELING OF COMPLEX DISCRETE EVENTS SYSTEMS

Summary

Simulations resources of LOGLAN for application of Complex Discrete Events Systems simulation have been presented in the paper. SIMULATION type implemented in LOGLAN is a problem language (a dialect) which enables to design programmes which simulate real systems described formally as Complex Discrete Events Systems. Program structure which simulates complex Discrete Events Systems has been presented. Markings used in formal description of these systems have been applied.

MODELLIERUNG VON KOMPLEXSYSTEMEN MIT DYSKRETEN EREIGNISSEN

Zusammenfassung

Im aufsatz wurden Simulationsvorräte der Programmiersprache LOGLAN für die Zwecke der Realisierung der Simulation von Komplexsystemen mit Dyskreten Eregnissen vorgestellt. Der Typ SIMULATION, der in LOGLAN implementiert wurde, stellt eine problemorienterte Programmiersprache (Dialekt) dar, die das Schreiben der reale Systeme simulierenden Programme erlaubt. Die Systeme werden formal als Komplexsysteme mit Dyskreten Ereignissen beschrieben. Im Aufsatz wurde Struktur eines Simulationsprogramms vorgestellt, welches die Komplexsysteme mit Dyskreten Ereignissen simuliert und welches die Bezeichnungen nutzt, die bei der formalen Beschreibung dieser Systeme verwendet werden.

МОДЕЛИРОВАНИЕ СЛОЖНЫХ СИСТЕМ ДИСКРЕТНЫХ СОБЫТИИ

РЕЗЮМЕ

В статьи представлено симуляционные, возможности языка LOGLAN для реализации моделирования Слохных Систем Дискретных Событии. Тип SIMULATION использованный в LOGLAN-е является проблемным языком позволяющим на писание программ моделирующих действительные системы представлены формально как Слохные Системы Дискретных Событии. В статьи показано структуру программы модели Слохной Системы Дискретных Событии с использованием обозначении применяемых при формальной описании этих Систем.