ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1989
Seria: TRANSPORT z. 13 Nr kol. 1034

Roman Konieczny

JĘZYK PROGRAMOWANIA LOGLAN JAKO NARZĘDZIE SYMULACJI SYSTEMU TRANSPORTOWEGO

Streszczenie. Artykuł niniejszy zawiera rozważania na temat możliwości wykorzystania języka LOGLAN do symulacji dyskretnej systemów transportowych. Istnienie klasy SIMULATION pozwala zaliczyć LOGLAN do grupy języków symulacyjnych, przeznaczonych do symulacji dyskretnych zdarzeń. Podstawowym pojęciem w klasie SIMULATION jest proces. Sterowanie procesami polega na planowaniu momentów czasowych, w których ma nastąpić uruchamianie bądź wznawianie określonych procesów.

W artykule omówiono instrukcje sterowania procesami oraz mechanizmy pomocnicze symulacji, jak: kolejki proste i kolejki priorytetowe. Przedstawiono również strukturę programu symulacyjnego oraz podano listing prostego programu symulacyjnego, sterującego przebiegiem pracy trzech abstrakcyjnych procesów.

Artykuł zawiera ponadto zestawienie porównawcze LOGLANu na tle innych języków symulacyjnych (SIMSCRIPT, GPSS, CSL, SIMON oraz SIMULA-67). Rozważono także - w formie przykładu - zagadnienie symulacji dużego systemu transportowego w języku LOGLAN.

1. Uwagi wstępne

Artykuł niniejszy zawiera rozważania na temat możliwości wykorzystania języka LOGLAN do symulacji dyskretnej systemów transportowych [5].

Istnienie klasy SIMULATION pozwala zaliczyć LOGLAN do grupy języków symulacyjnych, przeznaczonych do symulacji dyskretnej. Klasa ta jest dołączana do programu symulacyjnego na zasadzie łączenia tekstów. Program symulacyjny w języku **LOGLAN** - jest to blok prefiksowany klasą SIMULATION.

Podstawowym pojęciem w klasie SIMULATION jest proces. Każdy proces jest klasą prefiksowaną współprogramem o nazwie SIMPROCESS. Proces posiada własne zmienne (atrybuty) oraz zbiór instrukcji do wykonania. W zbiorze tym nie mogą jednak wystąpić instrukcje ATTACH i DETACH - charakterystyczne dla współprogramów ponieważ sterowanie procesami jest autonomiczną cechą klasy SIMULATION .

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.

2. Sterowanie procesami

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 .

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

Procedura 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

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

Wywołanie : CALL RUN(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 prze funkcję TIME .

Procedura 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
podaje proces aktywny .

3. Kolejki proste

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 .

4. Kolejki priorytetowe

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;
.............
.............
```

5. Struktura programu symulacyjnego

Typowy program symulacyjny posiada następującą strukturę :

```
PROGRAM NAZWA ;
( klasa SIMULATION ) włączana na zasadzie konkatenacji tekstów;
   BEGIN
   PREF SIMULATION BLOCK ;
   < lista deklaracji >
   ..................
   ..................
   UNIT ...          moduły - procesy programu symulacyjnego,
   UNIT ...          ( zdefiniowane przez użytkownika )
   ..................
   (* PROGRAM GŁÓWNY *)
   BEGIN  (* main NAZWA *)
   ........          instrukcje programu symulacyjnego
   ........
   END
   END NAZWA.
```

6. Przykład programu symulacyjnego

Poniżej podano listing prostego programu symulacyjnego, sterującego przebiegiem pracy trzech abstrakcyjnych procesów.

```
PROGRAM PSYM;
(* przykład symulacji - proste sterowanie procesami *)

BEGIN
PREF SIMULATION BLOCK;
VAR  A: PROCES_A, B: PROCES_B, C: PROCES_C;

  UNIT PROCES_A: SIMPROCESS CLASS;
    BEGIN
      writeln(TIME," Proces A/1 (start)");
```

```
      CALL HOLD(100);
      writeln(TIME,
      "   Proces A/2 (przekazanie sterowania do procesu C)");
      CALL RUN(C);
      writeln(TIME,
      "   Proces A/3 (ustawienie zawiadomienia dla procesu B)");
      CALL SCHEDULE(B,TIME+350);
      writeln(TIME,
      "   Proces A/4 (ustawienie zawiadomienia dla procesu C)");
      CALL SCHEDULE(C,TIME+400);
      writeln(TIME,"   Proces A/5 (wstrzymanie sie na czas 500)");
      CALL HOLD(500);
      writeln(TIME,"   Proces A/6 (zawieszenie dzialania)");
      CALL PASSIVATE;
      writeln(TIME,"   Proces A/7 (wznowienie dzialania)");
      CALL HOLD(220);
      writeln(TIME,
      "   Proces A/8 (przekazanie sterowania do procesu B)");
      CALL RUN(B);
      writeln(TIME,"   Proces A/9 (koniec)");
    END PROCES_A;

  UNIT PROCES_B: SIMPROCESS CLASS;
    BEGIN
      writeln(TIME,"   Proces B/1 (start)");
      CALL HOLD(150);
      writeln(TIME,
      "   Proces B/2 (wstrzymanie sie na czas 200)");
      CALL HOLD(200);
      writeln(TIME,"   Proces B/3 (zawieszenie dzialania)");
      CALL PASSIVATE;
      writeln(TIME,"   Proces B/4 (wznowienie dzialania)");
      CALL HOLD(200);
      writeln(TIME,"   Proces B/5 (koniec)");
    END PROCES_B;

  UNIT PROCES_C: SIMPROCESS CLASS;
    BEGIN
      writeln(TIME,"   Proces C/1 (start)");
      CALL HOLD(1500);
      writeln(TIME,
      "   Proces C/2 (przekazanie sterowania do procesu A)");
      CALL RUN(A);
      writeln(TIME,
      "   Proces C/3 (przekazanie sterowania do procesu B)");
      CALL RUN(B);
      writeln(TIME,
      "   Proces C/4 (ustawienie zawiadomien dla procesow A,B)");
      CALL SCHEDULE(A,TIME+300);
      CALL SCHEDULE(B,TIME+400);
       writeln(TIME,"   Proces C/5 (koniec)");
    END PROCES_C;

BEGIN (* main PSYM *)
    WRITELN(TIME,
    "   Program glowny  PSYM  ... generacja procesow A,B,C ");
    A:=NEW PROCES_A;
    B:=NEW PROCES_B;
    C:=NEW PROCES_C;
    writeln(TIME,
    "   Program glowny ... przekazanie sterowania do procesu A");
    CALL RUN(A);
    writeln(TIME,
    "   Program glowny (wstrzymanie sie na czas symulacji 9999)");
    CALL HOLD(9999);
    writeln(TIME,"   Program glowny ... koniec");
```

```
    CALL ENDRUN;
 END
END PSYM.
```

Wykonanie programu PSYM ma następujący przebieg:

```
int psym
IIUW LOGLAN-82  Concurent Executor  Version 4.30
December 2, 1987
(C)Copyright  Institute of Informatics, University of Warsaw

   0.0000    Program glowny  PSYM  ... generacja procesow A,B,C
   0.0000    Program glowny ... przekazanie sterowania do procesu A
   0.0000    Proces A/1 (start)
   0.0000    Program glowny (wstrzymanie sie na czas symulacji 9999)
 100.0000    Proces A/2 (przekazanie sterowania do procesu C)
 100.0000    Proces C/1 (start)
 100.0000    Proces A/3 (ustawienie zawiadomienia dla procesu B)
 100.0000    Proces A/4 (ustawienie zawiadomienia dla procesu C)
 100.0000    Proces A/5 (wstrzymanie sie na czas 500)
 450.0000    Proces B/1 (start)
 500.0000    Proces C/2 (przekazanie sterowania do procesu A)
 500.0000    Proces A/6 (zawieszenie dzialania)
 500.0000    Proces C/3 (przekazanie sterowania do procesu B)
 500.0000    Proces B/2 (wstrzymanie sie na czas 200)
 500.0000    Proces C/4 (ustawienie zawiadomien dla procesow A,B)
 500.0000    Proces C/5 (koniec)
 800.0000    Proces A/7 (wznowienie dzialania)
 900.0000    Proces B/3 (zawieszenie dzialania)
1020.0000    Proces A/8 (przekazanie sterowania do procesu B)
1020.0000    Proces B/4 (wznowienie dzialania)
1020.0000    Proces A/9 (koniec)
1220.0000    Proces B/5 (koniec)
9999.0000    Program glowny ... koniec

End of LOGLAN-82 program execution
```

Symbole na wydruku: A/1..9 , B/1..5 i C/1..5 mogą utożsamiać stany poszczególnych procesów w danej chwili. W miejsce instrukcji WRITELN... mogą być umieszczone ciągi instrukcji, bloki, procedury... przygotowujące (lub obsługujące) zdarzenie (zmianę stanu procesu).

Program PSYM ilustruje typową konstrukcję programu symulacyjnego w LOGLANie, która następnie może być w różnorodny sposób rozbudowywana.

7. LOGLAN na tle innych języków symulacyjnych

W punkcie tym dokonano przeglądu wytypowanych języków programowania do symulacji dyskretnej . Ponieważ literatura dotycząca tego zagadnienia jest stosunkowo bogata - przegląd ten ma wyłącznie introdukcyjny charakter, ułatwiający wstępne porównanie możliwości LOGLANu z możliwościami innych języków. Punkt zawiera ponadto porównanie niektórych elementów klasy

SIMULATION języków LOGLAN i SIMULA-67, a także zwraca uwagę na pewne aspekty realizacyjne planowania zdarzeń w programach symulacyjnych.

7.1. Przegląd niektórych języków do symulacji dyskretnej

Język SIMSCRIPT

Literatura : [3], [7].

Uwagi ogólne :
- opracowany na początku lat 60-tych,
- początkowe wersje na bazie FORTRANu ,
- podział programu symulacyjnego na moduły,
- model konstruuje się z obiektów opisywanych za pomocą atrybutów,
- grupowanie obiektów w zbiorach, jeśli posiadają tę samą cechę,
- program symulacyjny składa się z :
  - części definicyjnej < PREAMBLE >,
  - części głównej < MAIN >,
  - części końcowej < REPORT >,
- aktualne wersje SIMSCRIPTu posiadają składnię zbliżoną do języka naturalnego,
- definicja obiektu ma następującą postać : EVERY<nazwa obiektu> HAS A<lista atrybutów>,
- obiekty uczestniczące w eksperymencie symulacyjnym dzieli się na dwie kategorie: obiekty stałe <PERMANENT> oraz oraz okresowe <TEMPORARY>,
- definicja zbioru ma postać: THE SYSTEM OWNS A < lista zbiorów >,
- wygodny zestaw działania na zbiorach,
- (brak krajowych danych na temat efektywności translatora).

Język GPSS (General Purpose Simulation System)

Literatura : [1], [3], [7].

Uwagi ogólne :
- opracowany w latach 60-tych głównie dla komputerów IBM 360,
- jest językiem o strukturze blokowej,
- opis blokowy modelu jest szczególnie przejrzysty i łatwy w interpretacji,
- funkcje wypełniane przez poszczególne bloki bliższe są realiów symulowanego procesu niż właściwego oprogramowania,
- zdarzenia są odwzorowane za pomocą przepływu zadań przez bloki, z których każdy wpływa na zmianę stanu określonego obiektu.

- połączenia między blokami wyznaczają sekwencje operacji i mogą być: zdeterminowane, przypadkowe, logiczne lub podejmowane na podstawie określonych reguł decyzyjnych,
- jedynymi obiektami czynnymi modelu są zadania < TRANSACTION >, tworzone za pomocą bloków GENERATE,
- stan lub własności każdego zadania charakteryzowane są za pomocą zbioru parametrów oraz priorytetu; priorytet stanowi dodatkowy czynnik porządkujący zadania,
- argumenty bloku GENERATE pozwalają określać:
  - rozkład statyczny odstępów czasowych między kolejnymi zadaniami,
  - priorytet zadania,
  - ograniczenie liczby generowanych zadań,
- zadania są obiektami okresowymi i pozostają w systemie do chwili wykonania operacji TERMINATE,
- funkcje operatora planowania zdarzeń spełnia blok ADVANCE - opóźniający przepływ zadań o liczbę jednostek czasu systemowego,
- język ten znany jest w kraju na komputerach IBM 360 i 370, były podejmowane próby przeniesienia go na m.c. ODRA serii 1300.

Język CSL (Control and Simulation Language)

Literatura : [4], [9].

Uwagi ogólne:
- opracowany w latach 60-tych głównie dla komputerów firmy ICL,
- język ten jest zbudowany na bazie FORTRANu i posiada fortranopodobną składnię,
- program symulacyjny może być mieszaniną instrukcji języka CSL oraz FORTRANu,
- struktura programu analogiczna jak w FORTRANie,
- podział obiektów na klasy zbiorów :
  CLASS <nazwa klasy> SET <lista nazw zbiorów danej klasy>,
- każdy obiekt może posiadać nieograniczoną liczbę atrubutów,
- charakterystycznym, wyróżnianym atrybutem obiektu jest atrybut czasu: CLASS TIME <nazwa klasy> SET..
- dostęp do atrybutu czasu możliwy jest poprzez zapis kropkowy np. T.OBIEKT.NR ,
- użycie słowa kluczowego ACTIVITIES powoduje do-

łączenie do programu standardowej procedury upływu czasu.
- wystąpienie danego zdarzenia określane jest na podstawie badania wartości atrybutu czasu obiektu.
- przyjmuje się, że zdarzenie wystąpiło gdy wartość atrybutu czasu obiektu jest równa lub mniejsza zero.
- obsługi zdarzeń odbywają się w blokach (modułach programowych), rozpoczynających się od słowa kluczowego BEGIN.
- przyjmuje się, że każde zdarzenie obsługiwane jest w osobnym bloku.
- nie wyróżnia się explicite priorytetów zdarzeń.
- język posiada bogaty zestaw instrukcji operacji na zbiorach oraz instrukcji pomocniczych jak : generatory liczb losowych dla różnych rozkładów, instrukcje notowania histogramów itd.,
- język CSL znany jest w kraju na komputerach ICL serii 1900, ODRA serii 1300 oraz MERA 400 w systemie CROOK-4.

Język SIMON

Literatura : [10].

Uwagi ogólne :
- opracowany w latach 60-tych dla komputerów dowolnego typu posiadających kompilator języka ALGOL.
- język ten jest zbiorem procedur napisanych w języku ALGOL (język bezkompilatorowy).
- rozumienie symulacji jak w języku CSL, tylko na bazie składni ALGOLu.
- rozbudowany zestaw procedur operacji na zbiorach.
- bogaty repertuar procedur pomocniczych.
- na treść programu symulacyjnego składają się :
  - zestaw procedur SIMONa.
  - tekst w języku ALGOL odwzorowujacy model symulowanego systemu (wywołania procedur SIMONa).
- w kraju język ten znany jest na komputerach ODRA serii 1300 oraz CYBER 72.
- w chwili obecnej język ten może być również zaimplementowany na komputery posiadające kompilatory języka PASCAL, MODULA-2 czy LOGLAN.

Język SIMULA

Literatura : [2], [6], [7], [8].

Uwagi ogólne :
- opracowany w latach 60-tych (druga połowa) dla komputerów IBM 360, UNIVAC 1100, PDP 10 i in.,
- język zbudowany na bazie ALGOLu.

- struktura programu analogiczna jak w ALGOLu,
- podstawowym pojęciem jest proces zdefiniowany jako uporządkowany ciąg zdarzeń; modelowany system reprezentowany jest przez zbiór współbieżnych, współdziałających procesów,
- strukturę statyczną procesu - lub inaczej mówiąc wzorzec działania obiektu realizującego proces - definiuje się za pomocą deklaracji:
  process class < nazwa klasy procesów >
  identycznej pod względem składni z deklaracją procedury,
- egzemplarze klasy procesów generuje się za pomocą operatora: new <nazwa klasy procesów>,
- oprócz typów standardowych SIMULA posiada również referencyjny typ zmiennych,
- mechanizm prefiksowania umożliwia rozbudowę modelu systemu "w górę", modelowanie struktur hierarchicznych itp.,
- wszystkie niezbędne do prowadzenia symulacji środki programowe zabezpieczają klasy systemowe SIMSET i SIMULATION,
- fragment programu, w którym, wykorzystywane są zmienne i procedury z klas SIMSET i SIMULATION musi być prefiksowany nazwą klasy SIMULATION i może stanowić część wiekszego programu zawierającego m.in. procedury obróbki uzyskanych wyników, optymalizacji itp.,
- wszystkie obiekty, których programy modelują czynności trwające pewien okres , muszą zostać zadeklarowane w specjalny sposób; deklaracja klasy takich obiektów musi być prefiksowana klasą PROCESS; w terminologii związanej z językiem SIMULA 67 obiekty prefiksowane klasą PROCESS noszą nazwę procesów,
- wadą języka, która się przejawia w modelowaniu typowych systemów, jest stosunkowo duża pracochłonność związana z budową modelu w porównaniu z takimi językami jak GPSS i jemu podobnymi specjalizowanymi systemami symulacyjnymi; dodatkową zaletą SIMULI w porównaniu ze specjalizowanymi systemami, jest możliwość tworzenia bardzo dużych modeli (makromodeli), jak również modelowania systemów mieszanych (dyskretno-ciągłych),
- największa możliwość jaką oferuje język SIMULA,

wiąże się z budową zbioru procedur, tworzących specjalizowany język, przydatny do modelowania niektórych klas systemów transportowych, jak np. ruch pociągów w sieci kolejowej, stacje rozrządowe, ruch pojazdów w sieci ulic itd..

- język SIMULA znany jest w kraju głównie na komputerach IBM 360 i 370, jednakże zdania na temat efektywności jego translatora są podzielone.

7.2. Porównanie niektórych elementów klasy SIMULATION w języku SIMULA oraz LOGLAN

Język LOGLAN-82 jest unowocześnioną i rozszerzoną wersją języka SIMULA-67. Dla klasy SIMULATION w obu językach występują pewne różnice. Niektóre z nich podano poniżej :

a) POBRANIE DOWOLNEGO ELEMENTU Z KOLEJKI

W języku SIMULA kolejki są implementowane w postaci list dwukierunkowych, co pozwala na proste operacje pobrania, wstawiania lub usunięcia elementu z kolejki - bez potrzeby przeglądania całej kolejki. W języku LOGLAN kolejki proste są implementowane w postaci listy jednokierunkowej. Poza tym elementy pomocnicze w kolejkach prostych są niedostępne dla użytkownika z uwagi na stosowanie mechanizmów ochrony (HIDDEN). Aby "wydobyć" element ze środka kolejki należy stosować metodę przesiewania - polegającą na pobieraniu kolejnych elementów z kolejki i umieszczaniu ich w kolejce roboczej z jednoczesnym kasowaniem tych elementów w kolejce przeszukiwanej. Końcową operacją jest przepisanie elementów kolejki roboczej do kolejki przeszukiwanej.

b) PROCEDURY PLANUJĄCE KOLEJNOŚĆ ZDARZEŃ

W obu językach procedury te są identyczne. W języku SIMULA dochodzi procedura WAIT (S), która może być użyta w LOGLANIE przez zastosowanie dwóch instrukcji: CALL INTO (S); CALL PASSIVATE .

c) INSTRUKCJE AKTYWACJI I REAKTYWACJI PROCESÓW

W SIMULI zastosowano siedem instrukcji aktywacji i reaktywacji procesów. Instrukcje reaktywacji mają postać identyczną jak aktywacji, z tym, że słowo ACTIVATE jest poprzedzone przedrostkiem "RE".

W LOGLANIE nie rozróżnia się czy uruchomiany jest zawieszony proces, czy też przekazywane jest sterowanie do innego niezawieszonego procesu. A zatem nie ma podziału na instrukcje aktywacji i reaktywacji.

Poniżej podano loglanowskie odpowiedniki instrukcji aktywacji i reaktywacji procesów w odniesieniu do języka SIMULA :

S I M U L A	L O G L A N
ACTIVATE X	CALL RUN(X)
ACTIVATE X AT T	CALL SCHEDULE(X,T)
ACTIVATE X DELAY T	CALL SCHEDULE(X,MAX(TIME,TIME+T))
ACTIVATE X AT PRIOR ACTIVATE X DELAY PRIOR	brak odpowiednika z uwagi na losowy charakter wyboru procesu uruchamianego w czasie T lub TIME+T
ACTIVATE X BEFORE Y ACTIVATE X AFTER Y	instrukcje te można zasymulować w LOGLANIE znając czasy zawiadomienia procesów X i Y , i zmieniając je o mały przyrost czasu.

7.3. Walory LOGLANu

Walory języka LOGLAN zaprezentowane zostały szczegółowo m.in. w pracy [8]. Oceniając ten język na tle innych języków programowania przeznaczonych do symulacji dyskretnej - stwierdzić należy, iż posiada on zasoby równoważne (a nawet przewyższające) zasoby innych specjalizowanych systemów symulacyjnych. Zasoby te są wystarczające do budowy modeli dużych systemów transportowych. Ponadto, język ten zapewnia przejrzystość tekstu źródłowego programu, ponieważ wymusza modularny i strukturalny styl programowania.

Już samo stwierdzenie, że jest on unowocześnioną wersją języka SIMULA-67 - wystarcza aby postawić go na czele listy języków symulacyjnych... Jednocześnie pamiętać również należy, że jest to uniwersalny język wysokiego poziomu, mogący być wykorzystany do różnorodnych zastosowań, a nie tylko do symulacji dyskretnej.

7.4. Uwagi uzupełniające na temat konstrukcji programu symulacyjnego

Program symulacyjny może być tworzony w języku uniwersalnym, jak np. PASCAL, MODULA-2, ADA, LOGLAN lub w języku specjalizowanym (pod kątem symulacji dyskretnej lub ciągłej), np. GPSS, CSL, SIMSCRIPT lub CSMP, MIMIC, CEMMA... itd.

Zasoby danego języka rzutują w poważnej mierze na sposób konstrukcji programu, styl w jakim jest on napisany, a także na jego efektywność.

Przyjmując, że program symulacyjny tworzony będzie w języku uniwersalnym - należy rozważyć następujące aspekty:

- czy będzie preferowane podejście "stałotablicowe" (tj. alokacja statyczna), czy obiektowe (tj. alokacja dynamiczna);
- czy podstawowym pojęciem w budowie modelu będzie zdarzenie prowadzące do zmiany stanu obiektu (podejście CSL-podobne) lub podstawowym pojęciem będzie proces rozumiany jako ciąg zdarzeń (podejście SIMULO-podobne);
- czy z programistycznego punktu widzenia preferowane będą moduły typu PROCEDURE, COROUTINE czy PROCESS...
- jak będzie rozwiązane zagadnienie wyboru zdarzeń (tj. oś czasu w programie symulacyjnym) ...

Język LOGLAN umożliwia realizację różnych podejść ("filozofii") symulacyjnych. Podejścia te można pogrupować następująco:

1) podejście bazujące na klasie SIMULATION (SIMULO-podobne),
2) podejście bazujące na klasie SYSPROC (procesy quasi-równoległe),
3) podejście bazujące głównie na współprogramach (MODULO-podobne),
4) podejście bazujące głównie na procedurach (SIMONO-podobne),
5) podejście bazujące na tablicach statycznych i blokach obsługi zdarzeń (CSL-podobne).

Bazując na klasie SIMULATION - użytkownik nie musi tworzyć własnej procedury postępu czasu, ma ponadto udostępniony mechanizm synchronizacji procesów. W pozostałych przypadkach mechanizm taki musi być stworzony oddzielnie.

Zakładając, że preferowane będzie podejście 3), można przyjąć że każdy obiekt aktywny (czynny) symulowanego systemu reprezentowany będzie poprzez współprogram, a jego stany poprzez stopień zaawansowania (wykonania) współprogramu. Struktura takiego modułu, naśladującego działanie obiektu może być następująca:

```
UNIT OBIEKT1: COROUTINE (lista parametrów formalnych);
  VAR < lista atrybutów >;

  BEGIN
    RETURN;
      < ciąg instrukcji dotyczących stanu A
        ... lub przejścia do stanu A > ;      (* zdarzenie A *)

      < ciąg instrukcji wyznaczający czas
        przejścia w stan B >;

      < oddanie sterowania >;

      < ciąg instrukcji dotyczących stanu B
        ... lub przejścia do stanu B >;       (* zdarzenie B *)

      < ciąg instrukcji wyznaczający czas
        przejścia w stan C >;

      < oddanie sterowania >;

      ..... (* itd. *)

  END OBIEKT1;
```

Przedstawiona sekwencyjna struktura może być bardziej rozbudowana (posiadać szereg rozgałęzień i nawrotów).

Współprogram może także zawierać procedury (obsługi) poszczególnych zdarzeń, np.:

```
UNIT OBIEKT2: COROUTINE ( lista parametrów formalnych );
  VAR < lista atrybutów >;
  UNIT ZDARZENIE1: PROCEDURE ( lista parametrów formalnych );
    VAR ...
    BEGIN
    ......
    END ZDARZENIE1;
  UNIT ZDARZENIE2: PROCEDURE ( lista ... );
    VAR ...
    BEGIN
    ......
    END ZDARZENIE2;
  ..................
  BEGIN (* main OBIEKT2 *)
    RETURN;
    CALL ZDARZENIE1(...);
    <wyznaczenie odstępu czasu>;
    <oddanie sterowania>;
    CALL ZDARZENIE2(...);
    <wyznaczenie odstępu czasu>;
    <oddanie sterowania>;
    IF war THEN CALL ZDARZENIE3(...); ...
           ELSE CALL ZDARZENIE4(...); ...
    ..............
  END OBIEKT2;
```

W przypadku preferowania podejścia 4) (albo w sytuacji gdy "narzucony" jest język programowania - np. Turbo PASCAL) współprogram można w pewnym sensie zasymulować, używając wyłącznie procedur.

Np:

```
UNIT OBIEKT: PROCEDURE (typ_zdarzenia, lista parametrów);
  UNIT ZDARZENIE1: PROCEDURE (...);
    BEGIN ... END;
  UNIT ZDARZENIE2: PROCEDURE (...);
    BEGIN ... END;
  ..............................
  BEGIN (* main OBIEKT *)
    CASE TYP_ZDARZENIA
      WHEN 1: ... CALL ZDARZENIE1(...); ...
      WHEN 2: ... CALL ZDARZENIE2(...); ...
      ......  ..............................
      OTHERWISE  CALL ZDARZENIE_X(...); ...
  END OBIEKT;
```

Z każdą procedurą OBIEKT... powinien być związany rekord (klasa bez instrukcji), w którym byłyby pamiętane wartości chwilowe atrybutów obiektu (stany obiektu).

Przy podejściach 2-4) bardzo dokładnie musi być zaprojektowana procedura wyboru zdarzeń. Dodatkowo - powinna być kontrolowana poprawność następstwa zdarzeń (stanów), (tzn. np. dla obiektu POCIĄG po zdarzeniu

JAZDA_WYBIEGIEM nie może wystąpić ROZRUCH tylko HAMOWANIE lub JAZDA_ZE_STALĄ_PREDKOŚCIĄ...)

W przypadku podejść 2-5) należy opracować procedurę postępu czasu. Procedura ta może być zrealizowana poprzez działanie na tablicy przechowującej wartości atrybutów czasu wszystkich obiektów (sposób CSL-owski) - polegające na ciągłym uaktualnianiu zawartości tej tablicy. Uaktualnianie polega na odnalezieniu obiektu o najmniejszej nieujemnej wartości atrybutu czasu, a następnie odjęciu tej wartości (odcinka czasowego) od wartości atrybutu czasu wszystkich obiektów. Z kolei w każdym bloku obsługi zdarzeń badane są wartości atrybutu czasu obiektów danej klasy. W przypadku odnalezienia obiektu o zerowej lub ujemnej wartości atrybutu czasu - uruchamiana jest obsługa odpowiadającego tej sytuacji zdarzenia. Sposób ten jest kosztowny i dlatego wprowadzane są usprawnienia polegające na budowaniu specjalnych tablic stanu obiektów, stosów lub list (jedno- lub dwukierunkowych) celem efektywniejszego uporządkowania znaczników zdarzeń na osi czasu.

W jezyku LOGLAN oś czasu jest kolejką priorytetową, korygowaną przez procedurę CORRECT. W celu uruchomienia procesów, na początku symulacji proces główny - po wygenerowaniu procesów potomnych - może od razu ustawić im wstępne wartości zawiadomień (znaczników czasu) instrukcją CALL SCHEDULE (proces, znacznik_czasu), po czym "karuzela zdarzeń" w programie symulacyjnym "rozkręca się" sama.

Istnienie klasy SIMULATION jest dużym atutem języka LOGLAN. Przedstawione powyżej uwagi sygnalizują jedynie problemy, jakie należałoby rozwiązać, gdyby tej klasy nie było.

W tym miejscu należy także zwrócić uwagę na istnienie możliwości stworzenia na bazie loglanowskiej klasy SIMULATION oraz innych zasobów tego języka - symulacyjnego języka problemowo-zorientowanego do modelowania systemów transportowych (np. ruch pociągów). Istnieją trzy sposoby realizacji takiego języka:

1) Język ten byłby zbiorem procedur zestawionych w postaci bloku. Program symulacyjny miałby wówczas następującą strukturę:

```
PROGRAM NAZWA;
PREF JEZYK_PROBLEMOWY BLOCK;
<ciąg instrukcji języka problemowego
 lub mieszanina instrukcji języka problemowego
 i LOGLANu >
END.
```

2) Translator tego języka byłby prekompilatorem, tłumaczącym tekst napisany w języku problemowym (o składni GPSS-podobnej, SIMSCRIPT-podobnej lub zbliżonej do języka naturalnego) na tekst w języku LOGLAN.

3) Translator tego języka byłby interpreterem, tzn. tekst programu w języku problemowym mógłby być traktowany od razu jako kod pośredni (lub przetworzony wcześniej na taki kod) i każda ins-

trukcja tego kodu byłaby realizowana poprzez wywołanie odpowiedniej procedury loglanowskiej.

8. Zagadnienie symulacji dużego systemu transportowego w języku LOGLAN

W punkcie tym poprzez duży system transportowy rozumie się taki system, którego liczba elementów oraz relacji międzyelementowych dąży do nieskończoności, a jedynym ograniczeniem - co do wielkości jego modelu - są zasoby dostępnego sprzętu komputerowego.

Rozmiary symulowanego systemu trudno określić metodami analitycznymi (z uwagi na ciągłą rozbudowę translatorów języka LOGLAN dla mikrokomputera IBM PC XT/AT) - dlatego przyjęto empiryczny sposób postępowania - na przykładzie programu REJONY.

Program ten symuluje pracę LR odosobnionych rejonów sieci kolejowej, zawierających po LS szlaków wyposażonych w samoczynną blokadę liniową (SBL) dla LSEM semaforów (wymiarowanie pesymistyczne). Struktura programu jest następująca:

- podstawowym modułem SIMPROCESS REJON, w części głównej programu generowane jest zadeklarowane LR rejonów,
- każdy wygenerowany moduł REJON generuje z kolei zadeklarowane LS simprocesów typu SZLAK,
- na każdym szlaku generowane jest zadeklarowane LSEM semaforów,
- SIMPROCESS SZLAK składa się z następujących modułów: funkcji losującej RANDREAL, klasy SEMAFOR, simprocesu POCIĄG, simprocesu STACJA_POCZATKOWA, simprocesu STACJA_KONCOWA oraz simprocesu GENERATOR,
- na szlaku realizowany jest ruch jednokierunkowy; dla uproszczenia przyjęto, że czas przejazdu pociągu na każdym odstępie (między kolejnymi semaforami) generowany jest losowo z przedziału od 3 do 5 minut, zaś odstępy czasowe zgłoszeń pociągów na poszczególne szlaki wynoszą również od 3 do 5 minut;
  simprocess GENERATOR generuje obiekty typu POCIAG w określonych funkcją RANDREAL odstępach czasu; po wygenerowaniu każdego obiektu uruchamiany jest simprocess STACJA_POCZATKOWA; simprocesy POCIAG wstawiane są do kolejki DO_WJAZDU przed uruchomieniem simprocesu STACJA_POCZATKOWA;
  simprocess STACJA_POCZATKOWA wysyła pierwszy z kolejki DO_WJAZDU pociąg na szlak; wysłanie pociągu następuje, gdy semafor nr 1 jest podniesiony (STAN = TRUE);
  simprocess POCIAG symuluje ruch pociągu po szlaku; czas przejazdu generowany jest funkcją RANDREAL; po przejechaniu odstępu (instrukcja CALL HOLD(CZAS_JAZDY)), następuje próba wjazdu na następny odstęp; jeżeli odstęp jest wolny, to semafor odstępowy ustawiany jest na STÓJ (FALSE), a semafor poprzedzają-

cy na WOLNA DROGA; jeżeli pociąg wjechał na ostatni odstęp, to wywoływany jest simprocess STACJA_KONCOWA, który usuwa obiekt typu POCIAG z kolejki USUWANIE.

Poniżej podano listing programu REJONY :

```
PROGRAM REJONY;
BEGIN
PREF SIMULATION BLOCK;
VAR  LR:    INTEGER, (* liczba rejonow              *)
     TR:    ARRAYOF REJON,
     LS:    INTEGER, (* liczba szlakow w rejonie    *)
     LSEM:  INTEGER, (* liczba semaforow na szlaku *)
     NP:    INTEGER, (* biezacy numer pociagu       *)
     I:     INTEGER,
     CZAS_SYMULACJI: REAL;

UNIT REJON: SIMPROCESS CLASS (NUMER_REJONU: INTEGER);
  VAR  TS:                ARRAYOF SZLAK,
       NR:                INTEGER,
       I:                 INTEGER;

  UNIT SZLAK: SIMPROCESS CLASS(NUMER_SZLAKU: INTEGER);
    VAR DO_WJAZDU,USUWANIE: FIFO,
        SEM:                ARRAYOF SEMAFOR,
        ST_POCZ:            STACJA_POCZATKOWA,
        ST_KON:             STACJA_KONCOWA,
        GEN:                GENERATOR,
        I,N,NS:             INTEGER;

    UNIT RANDREAL: FUNCTION(A,B: REAL): REAL;
      BEGIN
        RESULT:=(B-A)*RANDOM+A;
      END RANDREAL;

    UNIT SEMAFOR: CLASS;
      VAR STAN: BOOLEAN, X: INTEGER;
      END SEMAFOR;

    UNIT POCIAG: SIMPROCESS CLASS (NUMER_POC: INTEGER);
      VAR  POZYCJA,NUMER: INTEGER,
           CZAS_JAZDY,PRZYJAZD,CZEKANIE: REAL;
      BEGIN
        POZYCJA:=1;
        NUMER:=NUMER_POC;
        WRITELN(TIME:7:2,"   POCIAG NR ",NUMER:2," WJECHAL NA SZLAK ",
         NS:2," REJON ",NR:2);
        DO
          SEM(POZYCJA).STAN:=FALSE;
          CALL RUN(ST_POCZ);
          CZAS_JAZDY:=RANDREAL(3,5);
          CALL HOLD(CZAS_JAZDY);
          PRZYJAZD:=TIME;
          IF POZYCJA=N THEN EXIT FI;
          CZEKANIE:=0;
          DO
            IF SEM(POZYCJA+1).STAN
              THEN
                SEM(POZYCJA).STAN:=TRUE;
                POZYCJA:=POZYCJA+1;
                IF CZEKANIE>0
                  THEN
                    WRITELN(TIME:7:2,"   POCIAG NR ",NUMER:2," CZEKAL ",
```

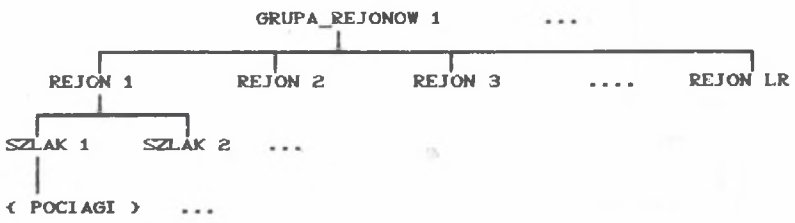
```
                    " SZLAK  ",NS:2," REJON ",NR:2,
                    " POD SEM. ",POZYCJA+1:2," /",CZEKANIE:5:1);
                FI;
                EXIT
              ELSE
                CZEKANIE:=TIME-PRZYJAZD;
                CALL HOLD(0.5);
            FI
          OD
        OD;
        SEM(POZYCJA).STAN:=TRUE;
        CALL INTO(USUWANIE);
        CALL SCHEDULE(ST_KON,TIME+0.5);
        WRITELN(TIME:7:2,"   POCIAG NR ",NUMER:2," WYJECHAL ZE SZLAKU ",
        NS:2," REJON ",NR:2);
      END POCIAG;

    UNIT STACJA_POCZATKOWA: SIMPROCESS CLASS;
      VAR  POC: POCIAG;
      BEGIN
        DO
          POC:=DO_WJAZDU.FIRST;
          IF POC=NONE THEN CALL PASSIVATE; REPEAT FI;
          CALL DO_WJAZDU.OUTFIRST;
          (* czekanie na otwarcie semafora wjazdowego na szlak *)
          DO
            IF SEM(1).STAN THEN CALL RUN(POC); EXIT
                           ELSE CALL PASSIVATE   FI
          OD;
        OD
      END STACJA_POCZATKOWA;

   UNIT STACJA_KONCOWA: SIMPROCESS CLASS;
     VAR  POC: POCIAG;
     BEGIN
       DO
         POC:=USUWANIE.FIRST;
         IF POC=NONE THEN CALL PASSIVATE; REPEAT FI;
         CALL USUWANIE.OUTFIRST;
         KILL(POC)
       OD
     END STACJA_KONCOWA;

  UNIT GENERATOR: SIMPROCESS CLASS;
    VAR  POC:     POCIAG,
         ODSTEP: REAL;
    BEGIN
      DO
        NP:=NP+1;
        ODSTEP:=RANDREAL(3,5);
        POC:=NEW POCIAG(NP);
        CALL POC.INTO(DO_WJAZDU);
        CALL RUN(ST_POCZ);
        CALL HOLD(ODSTEP)
      OD
    END GENERATOR;


  BEGIN (*  main SZLAK  *)

    NS:=NUMER_SZLAKU;
    N:=LSEM;
    CALL RANSET(44.77);
    ARRAY SEM DIM(1:N);
    FOR I:=1 TO N       DO
      SEM(I):=NEW SEMAFOR;
```

```
      SEMCI).STAN:=TRUE OD;
    GEN:=NEW GENERATOR;
    ST_POCZ:=NEW STACJA_POCZATKOWA;
    ST_KON :=NEW STACJA_KONCOWA;
    DO_WJAZDU:=NEW FIFO;
    USUWANIE :=NEW FIFO;
    CALL RUNCGEN);
  END SZLAK;

BEGIN (* -- main REJON -- *)

  NR:=NUMER_REJONU;
  ARRAY TS DIMC1:LS);
  FOR I:=1 TO LS
    DO TSCI):= NEW SZLAKCI); CALL SCHEDULECTSCI),TIME+0.01) OD;
  END REJON;


BEGIN (* ---------------------- main REJONY ---------------------------- *)

  WRITELNC"=================================================================");
  WRITELNC"  S Y M U L A C J A   GRUPY ODOSOBNIONYCH REJONOW               ");
  WRITELNC"                     ZAWIERAJACYCH SZLAKI WYPOSAZONE w  S B L ");
  WRITELNC"=================================================================");
  WRITEC" LICZBA REJONOW:                 "); READLNCLR);
  WRITEC" LICZBA SZLAKOW W REJONIE:       "); READLNCLS);
  WRITEC" LICZBA SEMAFOROW NA SZLAKU:     "); READLNCLSEM);
  WRITEC" CZAS SYMULACJI:                 "); READLNCCZAS_SYMULACJI);
  WRITELN;
  ARRAY TR DIMC1:LR);
  FOR I:=1 TO LR
    DO TRCI):=NEW REJONCI); CALL SCHEDULECTRCI),TIME+0.01) OD;
  CALL HOLDCCZAS_SYMULACJI);
 END
END REJONY.
```

Program REJONY umożliwia orientacyjne oszacowanie maksymalnych rozmiarów symulowanego systemu, dając jednocześnie pogląd na temat czasów wykonania przebiegów symulacyjnych przez interpreter LOGLANu. Deklarując różne kombinacje parametrów wejściowych można empirycznie określić "kres górny" liczby obiektów typu POCIAG, pracujących równocześnie w trybie quasi-równoległym.

Program ten jest jednocześnie przykładem użycia w LOGLANie techniki bottom-up, tj. tworzenia hierarchii typów - idąc "od dołu" - poprzez zagnieżdżanie modułów. Podstawowym obiektem jest tutaj SZLAK, na bazie szlaku tworzy się wieloszlakowy obiekt REJON, na bazie rejonu może być utworzony wielorejonowy obiekt GRUPA_REJONOW, itd.

Struktura programu REJONY tworzy następującą hierarchię:



Przykład wykonania programu REJONY podaje praca [5] .

Posługując się programem REJONY można badać (dla danej wersji implementacji LOGLANu):

- maksymalną (możliwą do symulacji) liczbę semaforów i pociągów na 1 szlaku,
- maksymalną liczbę szlaków dla 1 rejonu,
- maksymalną liczbę rejonów.

Oczywiście, uzyskiwane wyniki dają tylko pewien orientacyjny pogląd na temat maksymalnych rozmiarów symulowanego systemu oraz czasu wykonania przebiegów symulacyjnych, ponieważ zajętość pamięci operacyjnej oraz czasy wykonania zależą od złożoności obiektów występujących w programie.

9. Uwagi końcowe

Ogólnie - na podstawie wstępnych badań przeprowadzonych w Instytucie Transportu P.Śl. - można stwierdzić, że przedstawiony wyżej testowy program REJONY umożliwia wygenerowanie i równoczesne działanie kilkudziesięciu obiektów typu POCIĄG dla kilkunastu szlaków. Aktualnie przebiegi symulacyjne mogą być realizowane przy użyciu dwóch wersji interpretera LOGLANu dla IBM PC (INT i HINT). Wersja pierwsza umożliwia szybsze wykonanie programu, wersja druga umożliwia wygenerowanie większej liczby obiektów.

Na zakończenie podkreślić należy, że wszystkie ograniczenia dotyczące rozmiarów symulowanego systemu oraz czasów wykonania - podyktowane są specyfiką nie języka LOGLAN, ale zastosowanego komputera (a raczej mikrokomputera). Z praktycznego punktu widzenia sam język LOGLAN nie nakłada istotnych ograniczeń co do liczby typowych obiektów - i może być użyty do symulacji dużych systemów transportowych.

LITERATURA

[1] CHWESIUK K., SOKOŁOWSKI J.: Specjalizowany język do symulacji procesów dyskretnych GPSS/360 w modelowaniu procesów transportowych. Zagadnienia Transportu, nr 1-2 Wyd. PAN, Warszawa .

[2] DAHL O.J., MYHRHAUG B., NYGAARD K.: Simula-67 Common Base Language. Norwegian Computing Center, Oslo 1970 .

[3] GORDON G.: Symulacja systemów. WNT, Warszawa 1974 .

[4] KONDRATOWICZ L.: Modelowanie symulacyjne systemów. WNT, Warszawa 1978 .

[5] 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 . Instytut Transportu Politechniki

Śląskiej, Katowice 1987
[6] OKTABA H., RATAJCZAK W.: Simula 67. WNT, Warszawa 1980 .
[7] PERKOWSKI P.: Technika symulacji cyfrowej . WNT, Warszawa, 1980.
[8] ROMANIUK J.: Zastosowanie języka SIMULA 67 do modelowania systemów transportowych. Zagadnienia Transportu nr 3-4 , Wyd. PAN, Warszawa 1981
[9] WAJS W.: CSL - język do symulacji zdarzeń. Wyd. AGH, Kraków 1979 .
[10] WAJS W.: SIMON - uniwersalny język do modelowania zdarzeń. Wyd. AGH, Kraków 1980 .

LOGLAN PROGRAMMING LANGUAGE AS AN INSTRUMENT FOR SIMULATING TRANSPORT SYSTEM

Summary

The present paper includes consideration on the possibility of using LOGLAN language for discrete simulation of transport systems.

The existence of the SIMULATION class allows to number LOGLAN among simulation languages appropriated for discrete events simulation. A basic notion in the SIMULATION class is "process".

Process control consist in planning the moments in which starting or restarting specified process is to occur.

Process control instruction and simulation auxiliary mechanisms such as: simple narrow-gauge railways and priority narrow-gauge railways have been discussed in the paper.

The structure of a simulation program has also been presented and listing of simple simulation program controlling the course of three abstract processes has been given. Apart from this, the paper comprises a comparative sheet of LOGLAN on the background of other simulation languages (SIMSCRIPT, GPSS, CSL, SIMON and SIMULA-67).

Also the problem of a large transport system simulation in LOGLAN has been considered in the form of an example.

DIE PROGRAMMIERSPRACHE LOGLAN ALS WERZEUNG ZUR SIMULATION DES TRANSPORTSYSTEMS

Zusammenfassung

Vorliegender Aufsatz beinhaltet Betrachtungen zum Thema: Möglichkeiten der Ausnutzung der Sprache LOGLAN zur diskreten Simulation von Transportsystemen.

Das Vorhandensein der klasse SIMULATION in der Sprache LOGLAN erlaubt, LOGLAN zu einer Gruppe, von Simulationssprachen zuzuzählen, die für

Simulation diskreter Ereignisse bestimmt sind. In der Klasse SIMULATION bildet Prozeß einen Grundbegriff.

Die Prozeßsteuerung beruht auf der Planung von Zeitmomenten, in welchen Inbetriebnahme bzw. Wiederaufnahne bestimmter Prozessen erfolgen soll.

In dem Aufsatz wurden Instruktionen zur Prozeßsteuerung und Hifsmechnismen der Simulation, wie: einfache Schlangen und Schlangen mit Prioritäten besprochen.

Es wurden auch Struktur eines Simulationsprogramms vorgestellt und Listing eines einfachen Simulationsprogramms angegeben, das drei Abstraktionsprozesse steuert.

Der Aufsatz beinhalted darüber hinaus eine Vergleichszusammenstellung zwischen LOGLAN und anderen Simulationssprachen (SIMSCRIPT, GPSS, CLS, SIMON sowie SIMULA-67).

Es wurde auch - in Form eines Beispiels - das Simulationsproblem eines großen Transportsysteme in der Sprache LOGLAN betrachtet.

ЯЗЫК ПРОГРАММИРОВАНИЯ LOGLAN КАК СРЕДСТВО СИМУЛЯЦИИ ТРАНСПОРТНОЙ СИСТЕМЫ

Резюме

В статье ведутся пассуждения по теме возможности использования языка LOGLAN для симуляции дискретных транспортных систем. Наличие класса SIMULATION дает возможность зачесть LOGLAN к группе симуляционных языков, предназначенных для симуляции дискретных событый.

Основным понятием в классе SIMULATION является процесс. Управление процессами состоит в планировании временных моментов, в которых наступает пуск или возобновление определенных процессов.

В статье дана инструкция управления процессами а также вспомагательные механизмы симуляции: простые и приоритетные очереди. Дана также структура симуляционной программы а также листинг простой симуляционной программы, управляющей работой трех абстрактных процессов.

Статья состаит также из сравнительного описания LOGLANа на фоне других симуляционных языков (SIMSCRIPT, GPSS, CSL, SIMON и SIMULA-67).

В виде примера рассмотрен вопрос симуляции большой транспортной системы на языке LOGLAN.