ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1989

Seria: TRANSPORT z.9 Nr kol. 952

Roman KONIECZNY

CHARAKTERYSTYKA ZASOBÓW JĘZYKA LOGLAN NA POTRZEBY MODELOWANIA SYSTEMÓW TRANSPORTOWYCH

CZĘŚĆ II: Prefiksowanie, procesy współbieżne, wyjątki ...

Streszczenie. Niniejszy artykuł dokonuje prezentacji języka LOGLAN pod kątem jego wykorzystania - jako narzędzia do modelowania systemów transportowych. Część II zawiera zagadnienia dla bardziej zaawansowanych użytkowników (tj. programistów systemowych, naukowców oprogramowujących duże modele symulacyjne... i in.) - potrafiących w pełni zdyskontować wyrafinowane walory tego języka. Przedstawiono mechanizm prefiksowania, tj. tekstowego składania modułów na zasadzie podobnej jak w języku SIMULA-67; podano przykłady prefiksowania, omówiono zagadnienia programowania w oparciu o procesy współbieżne, zestawiono nieformalne postulaty specyfikujące system procesów, dokonano również omówienia ważnego zagadnienia - jakim jest obsługa sytuacji wyjątkowych. Całość wsparto przykładem programu obsługi wyjątków. Przytoczono także udogodnienia dla symulacji dyskretnej (zdarzeniowej) oraz do operacji na tekstach.

Klasa SIMULATION zaimplementowana w LOGLAN-ie jest językiem problemowo-zorientowanym pozwalającym na pisanie programów symulujących systemy rzeczywiste. W klasie tej wprowadzono mechanizm pozwalający na osiągnięcie niedeterminizmu w programie symulacyjnym. W klasie SIMULATION użyte zostały narzędzia pozwalające na quasi-równoległe wykonywanie programu.

1. Prefiksowanie

Często przy budowie złożonych systemów oprogramowania pojawia się problem składania modułów, tj. łączenia wcześniej napisanych fragmentów kodu w większe całości. Składanie modułów może odbywać się na poziomie źródłowym (tekstowym) lub półskompilowanym (jak np. w językach asemblerowych). Składania tekstowego modułów można dokonać edytorem tekstu lub przy użyciu instrukcji INCLUDE (np. Turbo PASCAL). Prefiksowanie jest bardziej wyrafinowaną formą składania modułów - polegającą w pewnym sensie na dziedziczeniu przez moduł prefiksowany cech modułu prefiksującego.

Prefiksowanie jest dwuargumentową operacją działającą na modułach. Pierwszym jej argumentem jest moduł identyfikowany przez nazwę: klasa, współprogram lub proces (tzw. prefiks), drugim zaś dowolny moduł: blok, procedura, funkcja, klasa, współprogram lub proces. Wynikiem operacji prefiksowania jest moduł prefiksowany, którego typ zależy od typów obu argumentów.

Poniżej podano zestawienie dozwolonych w LOGLAN-ie operacji prefiksowania:

1	PREF A BLOCK - gdzie A jest klasą (UNIT A: CLASS), moduł prefiksowany jest blokiem,
2	UNIT P : A PROCEDURE - gdzie A jest klasą (UNIT A: CLASS), moduł P jest procedurą,
3	UNIT F : A FUNCTION - gdzie A jest klasą (UNIT A: CLASS), moduł F jest funkcją,
4	UNIT B : A CLASS - gdzie A jest klasą (UNIT A : CLASS), moduł B jest klasą typu B (podklasa A),
5	UNIT B : A CLASS - gdzie A jest współprogramem (UNIT A:COROUTINE), moduł B jest współprogramem typu B (podtypu A),
6	UNIT B : A CLASS - gdzie A jest procesem (UNIT A: PROCESS), moduł B jest procesem typu B (podtypu A)
7	UNIT C : A COROUTINE - gdzie A jest klasą (UNIT A: CLASS), moduł C jest współprogramem typu C (podtypu A),
8	UNIT C : A COROUTINE - gdzie A jest współprogramem (UNIT A: COROUTINE), moduł C jest współprogramem typu C (podtypu A),
9	UNIT C : A COROUTINE - gdzie A jest procesem (UNIT A: PROCESS), moduł C jest procesem typu C (podtypu A),
10	UNIT P : A PROCESS - gdzie A jest klasą (UNIT A: CLASS), moduł P jest procesem typu P (podtypu A),
11	UNIT P : A PROCESS - gdzie A jest współprogramem (UNIT A: COROUTINE), moduł P jest procesem typu P (podtypu A),
12	UNIT P : A PROCESS - gdzie A jest procesem (UNIT A: PROCESS), moduł A jest procesem typu P (podtypu A).

W celu pełniejszego zilustrowania mechanizmu prefiksowania podano poniżej przykład, w którym zarówno moduł prefiksujący, jak i prefiksowany są klasami:

```
moduł A  UNIT A: CLASS (<parametry formalne A>);
         <deklaracja A>;
         BEGIN
         <początkowe instrukcje A>;
         INNER
         <końcowe instrukcje A>;
         END A;
moduł B  UNIT B: A CLASS (<parametry formalne B>);
         <deklaracje B>;
         BEGIN
         <początkowe instrukcje B>;
         INNER
         <końcowe instrukcje B>;
         END B;
```

Klasa B prefiksowana klasą A ma nie tylko atrybuty wyspecyfikowane bezpośrednio w <deklaracji B>, lecz także dziedziczy je z klasy A. Oznacza to, iż deklarację klasy B należy odczytać następująco:

```
         UNIT B: CLASS (<parametry formalne A>,
                        <parametry formalne B>);
         <deklaracja A>;
         <deklaracja B>;
         BEGIN
          <instrukcje początkowe A>;
          <instrukcje początkowe B>;
         INNER
         <instrukcje końcowe B>;
         <instrukcje końcowe A>;
         END B.
```

UWAGI:

- słowo kluczowe INNER w module prefiksującym wskazuje miejsce na wstawienie instrukcji modułu prefiksowanego;
- słowo INNER może wystąpić co najwyżej raz w ciągu instrukcji modułu prefiksującego;
- jeśli INNER nie występuje, przyjmuje się, że zostało umieszczone przed końcowym END modułu;
- w module nie będącym prefiksem INNER jest instrukcją pustą.

Mechanizm prefiksowania umożliwia tworzenie w łatwy sposób rożnorodnych klas oraz podklas danych. Na przykład:

```
UNIT P_126: CLASS (NUMER_FABRYCZNY: INTEGER);
   VAR TYP_SILNIKA  :  INTEGER,
       KOLOR        :  INTEGER,
       WYPOSAŻENIE  :  INTEGER,
       MOC_SILNIKA  :  REAL;
   END P126;
```

```
UNIT P_126_ULEPSZONY:
     P_126  CLASS;
VAR BLACHA_NIERDZ : BOOLEAN,
    KLIMATYZACJA  : INTEGER,
    AKCESORIA_DOD : INTEGER
END P_126_ULEPSZONY:,
```

```
UNIT P_126_LUX : P_126_ULEPSZONY CLASS (SPEC_NR_EW: INTEGER);
   VAR  KOMPUTER_POKŁADOWY:  BOOLEAN,
        WYKRYWACZ_RADARU  :  BOOLEAN,
        DOD_NAPED_ODRZUT  :  BOOLEAN;
   END P 126 LUX;
```

... itd., itp.

Podstawowym zastosowaniem prefiksowania jest rozszerzanie typów danych o nowe atrybuty; użytkownik może tworzyć hierarchię typów poprzez ich stopniowe wzbogacanie (nie ma natomiast możliwości odwrotnej, tzn. stworzenie podklasy zubożonej np. UNIT P_126_DZIAD: P_126 CLASS ... (bez silnika)). Ponadto należy wspomnieć o ważnym ograniczeniu: klasa nie może się pojawiać w swoim ciągu prefiksowym.

Do chwilowej zmiany kwalifikacji (typu) danego obiektu - podobnie jak w SIMULI-67 - służy operator QUA.

Jeżeli na przykład zadeklarowano: VAR P126 : P_126, ..., i w pewnym miejscu programu wykonano: P126 := NEW P_126_LUX(NR); to dostęp poprzez kropkę np. do atrybutu KOMPUTER POKŁADOWY nie byłby możliwy, gdyż klasa P_126 takiego atrybutu nie posiada. Operator QUA daje możliwość chwilowej zmiany kwalifikacji obiektu, tzn. zapis P126.KOMPUTER POKŁADOWY jest w tej sytuacji nieprawidłowy; natomiast zapis P126 QUA P_126_LUX.KOMPUTER_POKŁADOWY jest prawidłowy.

Niektóre zastosowania hierarchii typów danych wymagają użycia dwóch predykatów: IS oraz IN. Zakładając dla przykładu, że zmienna referencyjna P wskazuje na obiekt typu P_126_LUX, wówczas:

P IS P_126_HIPER

zachodzi wtedy i tylko wtedy, gdy P_126_HIPER = P_126_LUX

P IN P_126_HIPER

zachodzi wtedy i tylko wtedy, gdy P_126_HIPER występuje w ciągu prefiksowym klasy P_126_LUX.

(W przypadku konieczności korzystania ze zmiennych, których wartościami mogą być obiekty różnych typów, wystarczy te typy prefiksować wspólną klasą i posłużyć się operatorami IS, IN oraz QUA).

Tworzenie hierarchii klas poprzez prefiksowanie w końcowym efekcie prowadzić może do powstania wielu różnych dialektów - wyspecjalizowanych języków problemowo-zorientowanych. Podstawą tej koncepcji jest pojęcie bloku prefiksowanego. Jego składnia jest następująca: PREF JEZYK_P (parametry aktualne) BLOCK

```
<deklaracje>
BEGIN
<instrukcje>
END.
```

Wykonanie instrukcji bloku prefiksowanego polega na wygenerowaniu obiektu klasy JEZYK_P i wykonaniu jego instrukcji. Instrukcje bloku zastępują słowa INNER z klasy JEZYK_P. Pojęcia zdefiniowane w klasie JEZYK_P są dostępne w instrukcjach bloku. Na blok można więc spojrzeć jak na abstrakcyjny algorytm, a na klasę prefiksującą jak na implementację abstrakcyjnych operacji tego algorytmu.

W celu zwiększenia bezpieczeństwa przy programowaniu z użyciem prefiksowania wprowadzono w LOGLAN-ie trzy możliwości ograniczania dostępu do atrybutów: CLOSE, HIDDEN i TAKEN. Jeżeli w danej klasie, np. K, wystąpi specyfikacja CLOSE $k_1, k_2, \ldots, k_n$ - oznacza to, że z zewnątrz klasy K dostęp przez kropkę do atrybutów $k_1 \ldots k_n$ jest niedozwolony. Specyfikacja HIDDEN występująca w klasie K oznacza, że atrybuty nie są dostępne ani przez kropkę, ani w podklasach klasy K. Natomiast TAKEN $k_1, \ldots, k_n$ oznacza że w danej klasie dostępne z prefiksu będą jedynie atrybuty $k_1 \ldots k_n$.

Innym istotnym pojęciem związanym z prefiksowaniem są procedury i funkcje wirtualne. Służą one do następujących celów:

- umożliwiają dostęp do atrybutów danej klasy z klasy występującej w jej ciągu prefiksowanym;
- umożliwiają redefiniowanie procedur i funkcji w obrębie wszystkich warstw prefiksowych modułu;
- umożliwiają tworzenie ogólnych algorytmów i systemów, których szczegóły prezentuje użytkownik na poziomie swoich programów.

Wewnątrz instrukcji bloku prefiksowanego mogą wystąpić inne bloki prefiksowane (wielopoziomowe prefiksowanie wprowadzone w LOGLAN-ie eliminuje ograniczenia SIMULI-67).

2. Procesy współbieżne

Język LOGLAN umożliwia implementację wielu systemów współbieżnych. Systemy te mogą się różnić sposobami komunikowania modułów bądź środkami synchronizacji. Można opisywać systemy o wspólnej pamięci, bądź systemy, w których moduły są związane bardzo luźno.

Prace nad LOGLAN-em przyniosły nowy matematyczny model MAX obliczeń współbieżnych - przedstawiony przez prof. Salwickiego w publikacji [10]. Model MAX opisuje efekty równoczesnego wykonania instrukcji różnych procesów i dostarcza narzędzi do semantycznej analizy zachowań systemów.

Nieformalne postulaty specyfikujące system procesów są następujące:

- system procesów i semaforów należy traktować jako strukturę składającą się ze zbioru i pewnych operacji na jego elementach, sam zbiór jest sumą dwóch rozłącznych podzbiorów: zbioru P procesów i zbioru S semaforów;

- predykatami systemu są: (aktywny?), (zawieszony?), (oczekujący?), (zakończony?), (zatrzymany?) - dla P, oraz (opuszczony?), (podniesiony?), (pustakolejka?) - dla S;
- operacjami systemu są: (new), (end procesu), (kill), (resume), (stop), (wait), (stopZ), (lock), (unlock), (pk - pierwszy z kolejki), (wk - wstaw do kolejki), (uk - usuń z kolejki), (father);
- podstawowe postulaty szczegółowe:

P1: Obiekt jest procesem, jeżeli jest obiektem programu (bloku głównego) albo gdy jest obiektem modułu typu PROCESS bądź gdy jest obiektem klasy prefiksowanej przez proces. Każdy proces jest także współprogramem.

P2: W początkowym stanie każdego obliczenia istnieje tylko jeden proces - obiekt programu i jest on w stanie aktywnym.

P3: Proces utworzony podczas wykonywania instrukcji
X := NEW nazwa_procesu (parametry aktualne) (@)
jest w stanie zawieszonym.
Jeżeli np. P jest aktywnym procesem, który wykonał instrukcję (@) - to proces P nazywany jest ojcem procesu X, (X jest synem P).

P4: Instrukcja RESUME(X) powoduje, że proces X przechodzi w stan aktywny, a jego instrukcje są wykonywane współbieżnie z instrukcjami wszystkich innych procesów aktywnych. (Dzieje się tak pod warunkiem, że proces X był w stanie zawieszonym: gdy jest on w stanie aktywnym, to instrukcja RESUME jest pusta).

P5: Po wykonaniu instrukcji STOP proces, który ją wykonał, zostaje zawieszony. Wykonywanie jego instrukcji jest wstrzymane.

P6: Po wykonaniu instrukcji WAIT wykonywanie dalszych instrukcji procesu, który ją wykonał, jest zawieszone, proces ten jest w stanie oczekiwania - o ile chociaż jeden z istniejących procesów potomnych (synów) nie zakończył swoich obliczeń (w przeciwnym przypadku jest to instrukcja pusta, tzn. proces, który ją wykonał - kontynuuje swoje obliczenia).
Proces w stanie oczekiwania przechodzi w stan aktywny, gdy pewien z jego synów zakończy swoje akcje.

P7: Jeżeli proces aktywny wykonał wszystkie swoje instrukcje (tzn. osiągnął odpowiedni END) to przechodzi w stan zakończony. Proces zakończony nie może być uaktywniony ani przez instrukcję RESUME, ani przez operacje współprogramowe.
Zakończenie obliczeń pewnego procesu powoduje zmianę stanu procesu jego ojca (o ile nie był on w stanie oczekiwania - przechodzi w stan aktywny).

P8: Wykonanie instrukcji STOP(SEM) polega na łącznym, niepodzielnym wykonaniu kolejno instrukcji UNLOCK(SEM) i STOP, (gdzie SEM jest semaforem).

S1: O początkowych wartościach semafora zakłada się, że są one: podniesiony (SEM) i pustakolejka (SEM).

S2: Proces aktywny może wykonać instrukcję LOCK(SEM).
Wykonaniu tej instrukcji odpowiada:

```
IF podniesiony?(SEM) THEN opuść semafor(SEM)
                     ELSE wstaw proces do kolejki(SEM),
                          zatrzymaj proces FI.
```

UWAGA: Jeżeli dwa procesy zamierzają wykonać operację LOCK na tym samym semaforze SEM - to instrukcje te będą wykonane kolejno w niezdeterminowanej kolejności, ale w rozłącznych interwałach czasowych.

S3: Instrukcja UNLOCK(SEM), podobnie jak instrukcja LOCK, nie może być wykonywana równolegle z instrukcją LOCK(SEM) lub UNLOCK(SEM) innego procesu.
Znaczenie instrukcji UNLOCK jest następujące:

```
IF opuszczony?(SEM) THEN IF pustakolejka(SEM) THEN
                         p:= pierwszy z kolejki (SEM);
                            usuń p z kolejki (SEM);
                            RESUME(P)  FI
                     ELSE podnieść semafor(SEM) FI.
```

Powyższy system quasi-równoległych procesów został zaimplementowany w LOGLAN-ie (na podstawie współprogramów) jako klasa SYSPROC, będąca w istocie językiem problemowo-zorientowanym do programowania współbieżnego. Program prefiksowany klasą SYSPROC może być z łatwością uruchomiony w wieloprocesorowym systemie procesów i semaforów.

3. Obsługa sytuacji wyjątkowych

LOGLAN, podobnie jak wiele nowczesnych języków programowania, zawiera mechanizm obsługi sytuacji wyjątkowych, jakie mogą wystąpić w czasie wykonania programu. Zdarzeniem wyjątkowym (ang. exception) jest wystąpienie w czasie wykonywania programu sytuacji wymagającej specjalnej reakcji. Może to być błąd czasu wykonania (np. dzielenie przez zero, nadmiar lub próba dostępu do nieistniejącego obiektu), a także każda sytuacja, którą programista uzna za wyjątkową.

Wystąpienie wyjątku powoduje zawieszenie normalnych akcji programu i przejście do wykonywania modułu obsługi danego wyjątku (ang. handler). Rozpoznanie sytuacji wyjątkowej jest wyrażane przez wysłanie sygnału jest (automatycznie lub poprzez instrukcję RAISE). Wysłanie sygnału jest realizowane podobnie jak wywołanie procedury, lecz moduł obsługujący dany sygnał jest wyznaczany dynamicznie.

Specyfikacja sygnału służy do deklaracji sygnałów definiowanych przez programistę. Postać jej jest następująca:

SIGNAL nazwa_sygnału (parametry formalne)

Sygnały systemowe, odpowiadające błędom czasu wykonania, nie muszą być deklarowane.

Moduł obsługi sygnału jest ciągiem instrukcji, które mają być wykonane, gdy sygnał zostanie wysłany. Deklaracja modułu obsługi sygnału może wystąpić w części deklaracyjnej dowolnego modułu. Przykładowa (wariantowa) deklaracja modułu obsługi sygnałów ma następującą postać:

```
HANDLERS
  WHEN: sygnał_1: <ciąg instrukcji 1>; TERMINATE;
  WHEN: sygnał_2: <ciąg instrukcji 2>; RETURN;
  WHEN: sygnał_3: <ciąg instrukcji 3>; WIND;
  WHEN: ...
  WHEN: sygnał_n: CALL ENDRUN;
  OTHERS <ciąg instrukcji reagujących na "nieprzewidzianą" sytuację>;
         TERMINATE.
```

Modułem obsługi dla sygnałów wymienionych po słowie WHEN są występujące po dwukropku instrukcje. Instrukcje wymienione po słowie OTHERS obsługują wszystkie sygnały nie wymienione poprzednio.

Sygnał może być wysłany przez wykonanie instrukcji RAISE (parametry aktualne).

Akcje, które mają być wykonane w odpowiedzi na wysłanie sygnału, mogą być różnorodne. Musi istnieć możliwość zakończenia programu (abort), zakończenie błędnego modułu, a także wznowienie programu w punkcie, w którym został przerwany. Aby to zapewnić, wprowadzono specjalne instrukcje, są to: RETURN, WIND i TERMINATE. Dwie ostatnie mogą wystąpić tylko w module obsługi sygnału.

Instrukcja RETURN, podobnie jak w procedurze, powoduje wznowienie modułu, który wysłał sygnał. Sterowanie jest przekazywane do instrukcji następującej po RAISE. Instrukcje TERMINATE i WIND powodują zakończenie wykonania modułu, który wysłał sygnał oraz tych, z których był propagowany. Dla przykładu można rozważyć taką sytuację:

```
.......  (łańcuch dynamiczny)
RAISE F;      moduł O_k  (wysyłający sygnał F)
............
 moduł O_i+1
............
HANDLERS
   WHEN F: ...  moduł O_i  (obsługi sygnału F)
........
 moduł O_1
```

Wykonanie instrukcji RETURN, WIND lub TERMINATE w module obsługi powoduje zakończenie obsługi sygnału i przekazanie sterowania do odpowiedniej instancji. W przypadku gdy wykonana jest instrukcja RETURN, sterowanie przekazane jest do instancji O_k, która zostaje wznowiona w miejscu przerwania. Wykonanie instrukcji WIND powoduje zakończenie wykonywania instancji $O_k, \ldots, O_{i+1}$. Instancja O_i zostaje wznowiona w miejscu wskazanym przez ślad powrotny instancji O_{i+1}. W efekcie wykonania instrukcji TERMINATE instancje $O_k \ldots O_i$ zostają zakończone. Sterowanie jest przekazane do instancji O_{i-1}, o ile taka istnieje. W przeciwnym wypadku proces (współprogram) O_1 zostaje zakończony przez wykonanie instrukcji END.

Czasami przed zakończeniem modułu spowodowanym wystąpieniem wyjątku (czyli przed wykonaniem instrukcji WIND lub TERMINATE) należy wykonać pewne akcje "porządkowe", np. zamknąć pliki, zwolnić pamięć, wysłać dodatkowe komunikaty na ekran, itp. W LOGLAN-ie wprowadzono w tym celu instrukcję LASTWILL.

Sygnały systemowe, odpowiadające błędom czasu wykonania, są wysyłane automatycznie. Nie mają parametrów i nie są deklarowane przez użytkownika (chociaż można utworzyć moduły obsługujące je, w modułach takich niedozwolone jest użycie instrukcji RETURN).

Dla IBM PC sygnały systemowe są następujące:

ACCERROR - próba dostępu do nieistniejącego obiektu,
CONERROR - przekroczenie zakresu tablicy lub niezgodność typów,
LOGERROR - błędy związane z przekazywaniem sterowania,
MEMERROR - przepełnienie (brak) pamięci,
NUMERROR - błąd numeryczny (np. dzielenie przez zero, nadmiar),
TYPERROR - niezgodność typów w instrukcji przypisania lub przy transmisji parametrów,
SYSERROR - błędy związane z komunikacją z systemem operacyjnym, błędy w instrukcjach wejścia/wyjścia, za dużo otwartych plików itp.

Poniżej podano przykład programu reagującego na sytuacje wyjątkowe: Przykładowe wykonanie programu WYJĄTKI jest następujące:

```
PROGRAM WYJĄTKI;
VAR    NC,NO,NZ,IC,IZ:   INTEGER,
    C: ARRAYOF CZYTACZ_PLIKU,
    Z: ARRAYOF ZAPEŁNIACZ_TABLICY;
SIGNAL  START,KONIEC;
UNIT  CZYTACZ_PLIKU:  COROUTINE(NUMER: INTEGER);
   VAR ZNAK, ZNAK_WYJATK:   CHAR, AKTYWNY: BOOLEAN, F: FILE;
   SIGNAL JEST_ZNAK_WYJ;
   HENDLERS
      WHEN JEST_ZNAK_WYJ:
```

```
      WRITELN("Czytacz pliku numer", NUMER,
              "sygnał JEST_ZNAK_WYJ odebrany");
      RETURN;
   WHEN SYSERROR: WRITELN("Czytacz pliku numer", NUMER);
                  WRITELN("Blad SYSERROR"); WIND;
   WHEN ACCERROR: WRITELN("Czytacz pliku numer", NUMER);
                  WRITELN("Blad ACCERROR"); WIND;
   OTHERS WRITELN("Czytacz pliku numer", NUMER, "nie wiem co jest");
                  TERMINATE;
END HANDLERS;
BEGIN
   AKTYWNY:=TRUE;  WRITELN;
   WRITELN("Czytacz numer", NUMER ,"pliku ANE.GRF");
   WRITE("Co ma być znakiem wyjątkowym?");
   READLN(ZNAK_WYJATK);
   OPEN(F,TEXT,"ANE.GRF");  CALL RESET(F);
   RETURN;
   WHILE NOT EOF(F)
      DO
        READ(F,ZNAK);
        IF ZNAK=ZNAK_WYJATK THEN
           WRITELN("Czytacz pliku numer", NUMER, "sygnał wysłany");
           RAISE JEST_ZNAK_WYJ  FI;
      OD;
   KILL(F);
   AKTYWNY:=FALSE;  DETACH;
LASTWILL
   WRITELN("Ostatnia wola czytacza pliku nurmer",NUMER);
END CZYTACZ PLIKU;
UNIT ZAPELNIACZ_TABLICY: COROUTINE(NUMER: INTEGER);
   CONST G=500;
   VAR TABLICA: ARRAYOF INTEGER, AKTYWNY:BOOLEAN,
       I,INDEKS_WYJATK,INDEKS_GORNY:  INTEGER;
   SIGNAL JEST_INDEKS_WYJ;
   HANDLERS
      WHEN JEST_INDEKS_WYJ:
      WRITLEN("Zapełniacz tablicy numer",NUMER,
              "sygnal JEST_INDEKS_WYJ odeb.");
      RETURN;
   WHEN CONERROR: WRITELN("Zapelniacz tablicy numer", NUMER);
                  WRITELN("Blad CONERROR"); WIND;
   WHEN TYPERROR: WRITELN("Zapelniacz tablicy numer",NUMER);
                  WRITELN("Blad TYPERROR"): TERMINATE;
```

```
   OTHERS WRITELN("Zapelniacz tablicy numer",NUMER ,
                  " nie wiem co jest...");
         TERMINATE;
END HANDLERS;
BEGIN
   AKTYWNY:=TRUE;  WRITELN;
   WRITELN("Zapełniacz tablicy [1:500] numer", NUMER);
   WRITE("Podaj indeks wyjątkowy          :");  READLN(INDEKS_WYJATK);
   WRITE("Podaj roboczy górny indeks      :");  READLN(INDEKS_GORNY);
   ARRAY TABLICA DIM(1:G);
   RETURN;
   FOR I:=1 TO INDEKS_GORNY
      DO
         TABLICA(I):=1313;
         IF I=INDEKS WYJATK THEN
            WRITELN("Zapełniacz tablicy numer", NUMER, "sygnał wysłany");
            RAISE JEST_INDEKS_WYJ; FI;
      OD;
   KILL(TABLICA);
   AKTYWNY:=FALSE;
   DETACH;
LASTWILL
   WRITELN("Ostatnia wola zapełniacza tablicy numer", NUMER);
END ZAPELNIACZ_TABLICY;
HANDLERS
   WHEN START:  WRITELN;
        WRITELN("Modul glowny... sygnal START odebrany"); RETURN;
   WHEN KONIEC: WRITELN;
        WRITELN("Modul glowny... sygnal KONIEC odebrany"); RETURN;
   WHEN ACCERROR:  WRITELN("Modul glowny...  ACCERROR");  TERMINATE;
   WHEN CONERROR:  WRITELN("Modul glowny...  CONERROR");  TERMINATE;
   WHEN LOGERROR:  WRITELN("Modul glowny...  LOGERROR");  TERMINATE;
   WHEN MEMERROR:  WRITELN("Modul glowny...  MEMERROR");  TERMINATE;
   WHEN TYPERROR:  WRITELN("Modul glowny...  TYPERROR");  TERMINATE;
   WHEN SYSERROR:  WRITELN("Modul glowny...  SYSERROR");  TERMINATE;
   OTHERS
        WRITELN("Modul glowny... jakis nieproszony sygnal");
END HANDLERS;
BEGIN (* main WYJATKI *)
   WRITELN; WRITELN("Program reagowania na wyjątki"): WRITELN;
   WRITE("Liczba czytaczy pliku  ANE.GRF   =   ");  READLN(NC);
   WRITE("Liczba zapelniaczy tablicy ...   =   ");  READLN(NZ);
```

```
   RAISE START; ARRAY C DIM(1:NC);   ARRAY Z DIM(1:NZ);
   FOR IC:=1 TO NC DO C(IC):=NEW CZYTACZ_PLIKU(IC)   OD;
   FOR IZ:=1 TO NZ DO Z(IZ):=NEW ZAPELNIACZ_TABLICY(IZ) OD;
   IC,IZ:=0;
DO
   IC:=IC+1; IF  IC<=NC AND C(IC).AKTYWNY THEN ATTACH(C(IC)) FI;
   IZ:=IZ+1; IF  IZ<=NZ AND Z(IZ).AKTYWNY THEN ATTACH(Z(IZ)) FI;
   IF  IC>NC AND IZ>NZ THEN EXIT FI;
OD;
   WRITELN("Koniec pracy programu WYJATKI");
   RAISE KONIEC; CALL ENDRUN;
LASTWILL
   WRITELN("Ostatnia wola modulu glownego...");
END WYJATKI.
```

Mechanizm zaproponowany w LOGLAN-ie pozwala obsługiwać w sposób elegancki i przejrzysty zarówno wyjątkowe zdarzenia występujące w czasie wykonania programu, jak i błędy czasu wykonania. Zwiększa także czytelność programów, pozwalając oddzielić akcje normalne i typowe od wyjątkowych. Mechanizm ten można wykorzystać również (w pewnych sytuacjach) do "planowej", a nie tylko "wyjątkowej" komunikacji między modułami. Oczywiście, obsługa sytuacji wyjątkowych jest bardziej skomplikowana, a zatem wymagająca większej ostrożności, niż użycie konwencjonalnych narzędzi (np. procedur).

4. Udogodnienia dla symulacji dyskretnej

Klasa SIMULATION zaimplementowana w LOGLAN-ie jest językiem problemowo-zorientowanym (dialektem) pozwalającym na pisanie programów symulujących systemy rzeczywiste. Może być wykorzystana do symulacji systemów dyskretnych, tzn. takich, w których stany zachodzą w sposób nieciągły, w wyróżnionach chwilach czasu. W klasie tej wprowadzono mechanizm pozwalający na osiągnięcie niedeterminizmu w programie symulacyjnym. Dzięki temu powtórzenie tego samego eksperymentu daje dodatkowe informacje oraz zbliża model symulacyjny do rzeczywistego systemu równoległego. W klasie SIMULATION użyte zostały narzędzia pozwalające na quasi-równoległe wykonywanie programu (tzn. współprogramy oraz operacje ATTACH i DETACH). Procesy są reprezentowane przez współprogramy.

Modelowany system jest reprezentowany przez zbiór współbieżnych i współdziałających procesów. Proces jest ciągiem uporządkowanych w czasie zdarzeń związanych z działaniem symulowanego obiektu. Wartość lokalnych atrybutów procesu określa jego stan.

Wystąpienie zdarzenia jest reprezentowane przez zmianę stanu procesu, a zatem zmianę stanu systemu. Realizacja zdarzenia polega na wykonaniu ciągu instrukcji opisujących je.

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. Program symulacyjny jest blokiem posiadającym prefiks SIMULATION.

5. Udogodnienia dla operacji na tekstach

W celu ułatwienia operacji na tekstach opracowano klasę LOGTEXT. Zasadnicze rodzaje operacji tekstowych zawartych w tej klasie są następujące:

1) generacja obiektów tekstowych:
- konwersja ciągu znakowego na obiekt tekstowy,
- wczytanie tekstu o danej długości z urządzenia wejściowego,
- utworzenie tekstu o danej długości, zainicjalizowanego spacjami,
- tekst pusty (notext);

2) wypisanie wartości tekstu,

3) zapis/odczyt pojedynczego znaku tekstu,

4) utworzenie podtekstów i znajdowanie tekstu głównego,

5) dostarczenie informacji o wartościach atrybutów tekstu oraz ustawienie wskaźnika bieżącej pozycji,

6) przypisanie wartości tekstów

7) konkatencja tekstów oraz zastąpienie jednego podtekstu przez inny,

8) sprawdzenie, czy dany znak występuje w tekście,

9) leksykograficzne porównanie wartości tekstów.

Dokładny opis klasy LOGTEXT zawiera publikacja [2].

6. Uwagi końcowe

Język LOGLAN zaimplementowany został w kraju dla minikomputera MERA-400, pracującego pod kontrolą zmodyfikowanego (przez Instytut Informatyki Uniwersytetu Warszawskiego) systemu operacyjnego SOM-3; oraz dla mikrokomputera IBM-PC XT/AT. W wersji dla IBM-PC LOGLAN nie posiada kompilatora; wykonywanie programu odbywa się poprzez interpretację kodu pośredniego (tzw. L-kodu) przez program napisany języku MS-PASCAL.

Aktualnie trwają prace nad kompilatorem LOGLAN-u dla IBM-PC (będzie on opracowany w języku C - co w przyszłości być może pozwoli przenieść LOGLAN również pod system operacyjny CROOK-4 na MERZE-400).

Za granicą LOGLAN był zaimplementowany (interpreter) m.in. na komputer VAX.

Z uwagi na niezaprzeczalne walory funkcjonalne przewiduje się szeroki wzrost zastosowań tego języka w ciągu najbliższych lat - szczególnie na sprzęcie typu IBM-PC.

LITERATURA

[1] Iszkowski W., Maniecki M.: Programowanie współbieżne, WNT, Warszawa 1982.

[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 RP.I.09 - Instytut Transportu Politechniki Śląskiej, Katowice 1987.

[4] Kreczmar A., Salwicki A.: Język programowania LOGLAN. Informatyka 1982, nr 7,8, 1983, nr 1.

[5] Kreczmar A.: Język programowania LOGLAN 82. Materiały II Konferencji Użytkowników Minikomputera MERA-400, Gdańsk 1984.

[6] Report on the Programming Language LOGLAN. Praca zbiorowa. Wyd. Instytut Informatyki WU, Warszawa 1982.

[7] Materiały: International Summer School of the Programming Language LOGLAN. ZABORÓW, POLAND September, 5-10.1983. IIUW.

[8] Report on the LOGLAN 82 Programming Language.PWN, Warszawa - Łódź 1984.

[9] Kreczmar A.: Dokumentacja dla minikomputera MERA-400 - Język programowania LOGLAN-82 - Podstawowe konstrukcje i cechy charakterystyczne języka. IIUW, Warszawa 1984.

[10] Język programowania LOGLAN 82 - Materiały Jesiennej Szkoły PTI, Serock 1985.

[11] LOGLAN - USER'S GUIDE (version November'86) IIUW, Warsaw. (Suplement dla IBM PC).

[12] Müldner T.: Pewne uwagi o dwóch nowych językach programowania wysokiego poziomu LOGLAN i ADA. Biuletyn Techniczny MERA 1980, nr 11-12.

[13] Oktaba H.: Klasy w LOGLAN-ie. "Informatyka" 1983, nr 5.

[14] Oktaba H.: Prefikowanie klasami w LOGLAN-ie. "Informatyka" 1983, nr 6.

[15] Salwicki A.: LOGLAN - narzędzie produkcji oprogramowania. Materiały II Konferencji Użytkowników Minikomputera MERA-400, Gdańsk 1984.

[16] Salwicki A.: Metodologia programowania w LOGLAN-ie. Materiały III Konferencji Użytkowników Minikomputera MERA-400, Gdańsk 1985.

[17] Szałas A., Szczepańska-Wasersztrum D.: Exception Handling in Parallel Computations. Sprawozdania Instytutu Informatyki Uniwersytetu Warszawskiego, Warszawa 1984.

Recenzent: Doc. dr hab. inż. Krzysztof Chwesiuk

Wpłynęło do Redakcji 19.11.1987 r.

ХАРАКТЕРИСТИКА ВОЗМОЖНОСТЕЙ ЯЗЫКА ЛОГЛАН
ДЛЯ ПОТРЕБНОСТЕЙ МОДЕЛИРОВАНИЯ
ТРАНСПОРТНЫХ СИСТЕМ
ЧАСТЬ II: Префиксирование, взаимодействие процессы, исколючения ...

Р е з ю м е

В настоящей работе представлен язык программирования ЛОГЛАН с точки зрения его использования как инструмента моделирования транспортных систем. Часть II содержит вопросы для более опытных потребителей (т.е. системных программистов, исследователей программирующих большие имитационные модели и.т.п.), умеющих вполне дисконтировать уточненные положительные качества этого языка. Представлен механизм префиксации т.е. текстового складывания модулей по принципу похожему как в языке СИМУЛА-67, даны примеры префиксации, оговорены вопросы программировании на базе параллельных процессов, сопоставлены неформальные постулаты специфицирующие систему процессов, оговорен также очень важный вопрос, каким является обслуживание непредвиденных ситуаций. Дан пример программы обслуживания исключений. Приведен ряд удобств облегчающих дискретную а также удобных для операций на текстах.

Клласс СИМУЛЕЙШН имплементирован в ЛОГЛАН являтся языком проблемно-ориентированным позваляющим писать имитационные программы реальных систем. В этом классе введен механизм позваляющий достигнуть недетерминизм в имитационной программе. В классе СИМУЛЕЙШН употреблены инструменты программирования, позваляющие на почти параллельное выполнение программы.

CHARACTERISTIC OF THE LOGLAN LANGUAGE
RESERVES FOR NEEDS OF MODELLING TRANSPORT
TRANSPORT SYSTEMS
PART II: Prefixing, concurrency processes, exceptions ...

S u m m a r y

The present article describes the LOGLAN language from the view - point of its using as an instrument for modelling the transport systems. Part II comprises problems for more advanced users (i.e. system programmers, scientists programming big simulation models ... etc.) being able to turn sophisticated qualities of this language to profit entirely. Mechanism of prefixing, i.e. text setting the modules by a similar principle as in the SIMULA-67 language has been presented, prefixing examples have been given, problems of programming based on concurrency processes have been discussed, informal postulates specifying the system of processes have been listed and important problem e.g. handling exceptional situations has been discussed too. The whole has been supported by the example of the program of the exception handling.

Some facilities discrete (event) simulation and for text operations have been mentioned.

The SIMULATION class implemented in LOGLAN is a problem - oriented language allowing to write the programs which simulate the real systems. In this class a mechanism has been introduced which allows to reach indeterminism in the simulation program. There have been instruments used in the SIMULATION class that allowed quasi - parallel performance of the program.