STUDIA INFORMATICA 2000
Volume 21 Number 3 (41)

Maciej LUBIŃSKI
Politechnika Śląska, Instytut Informatyki

MODEL ŚRODOWISKA SYSTEMU PRZETWARZANIA DANYCH – TRANSLACJI JĘZYKÓW NATURALNYCH[1]

Streszczenie. Specyfika pracy badawczej nad mechaniczną translacją języka naturalnego stawia określone wymagania środowisku komputerowemu. W artykule zostały opisane właściwości, jakie doświadczalny system translacji powinien spełniać. Głównym problemem jest umożliwienie użytkownikowi wglądu i edycji wszystkich danych powstających w procesie translacji oraz odpowiednie sterowanie dostępem do nich. System taki powinna również charakteryzować łatwość rekonfiguracji etapów translacji.

MODEL OF THE DATA PROCESSING ENVIRONMENT – NATURAL LANGUAGE TRANSLATION

Summary. Research on mechanical translation of natural languages requires a particular computational environment. This paper describes what facilities an experimental translation system ought to have. The main problem is to allow the user to browse and edit all the data produced in translation process, and to control the access suitably. Also, the translation stages of such a system should be easily reconfigurable.

1. Wprowadzenie

Intensywne badania nad translacją języków naturalnych (JN) doprowadziły do wielu częściowych rozwiązań tego problemu. Żadne z dotychczas opracowanych nie jest na tyle

[1] Pracę wykonano częściowo w ramach projektu badawczego pt. *Translacja tekstów w języku polskim na język migowy* nr 8 T11C 007 17 finansowanego przez KBN.

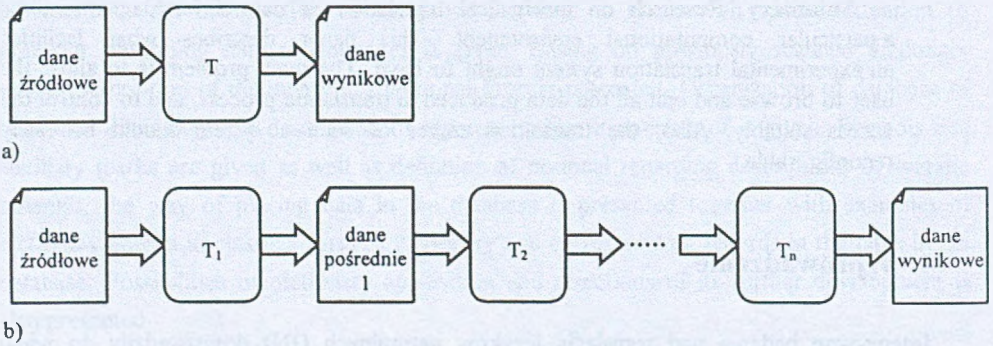
uniwersalne, by móc je powszechnie stosować niezależnie od języka wejściowego. Wszystkie są rozwijane i doskonalone.

W Zakładzie Oprogramowania Instytutu Informatyki prowadzone są prace badawcze nad systemem translacji tekstów napisanych w języku polskim na język migowy. Proces translacji jest wieloetapowy. Programy realizujące poszczególne etapy tłumaczenia są efektem przeprowadzonych badań i są cały czas udoskonalane. [9]

Niniejszy artykuł przybliży specyfikę pracy badawczej z doświadczalnym systemem translacji języka naturalnego (DSTJN). Zasygnalizowane zostaną możliwości przyśpieszenia działania systemu przez zrównoleglenie pracy poszczególnych podtranslatorów wchodzących w skład systemu translacji. Zostaną także przedstawione rozwiązania opisujące przebieg translacji w poszczególnych typach translatorów – od translatorów strumieniowych, o najprostszym schemacie działania, do translatorów złożonych, wielonurtowych o mieszanym przepływie danych.

2. Doświadczalne systemy translacji języków naturalnych

Struktura języka ma odwzorowanie w konstrukcji translatora. Nie każdy język umożliwia bezpośrednie, realizowane w jednej fazie, przeprowadzenie analizy i wygenerowanie przekładu. Ze względu na liczbę przebiegów możemy wyróżnić translatory jedno- (rys. 1. a.) i wieloprzebiegowe (rys. 1. b.). Translatory jednoprzebiegowe charakteryzują się dużą szybkością działania, jednak możliwość ich stosowania jest ograniczona.



Rys. 1. Translacja jedno- (a) i wieloprzebiegowa (b)
Fig. 1. Single (a) and multi (b) pass translation

Zastosowanie translacji wieloetapowej może być podyktowane:

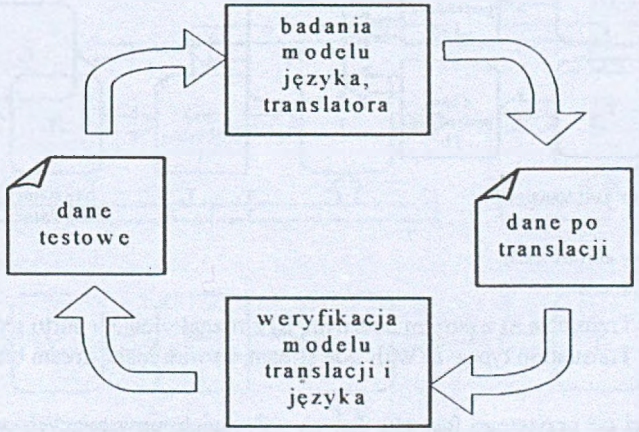
- technicznymi ograniczeniami wynikającymi z konstrukcji poprawnych fraz;

- względami praktycznymi, jak np.:
  - zastosowanie równoległego przetwarzania w celu przyśpieszenia translacji,
  - prowadzenie pracy zespołowej – samodzielne prace nad poszczególnymi podtranslatorami systemu (np. analizator morfologiczny, składniowy, semantyczny itd.),
  - prezentacja wyników pośrednich (np. w DSTJN).

Specyfika pracy nad translacją języków naturalnych, jak też budowa translatora i przebieg mechanicznego tłumaczenia JN stawia określone wymagania środowisku komputerowemu przeznaczonemu do prowadzenia badań [5]. Poniżej zostaną przedstawione główne właściwości DSTJN. Sprecyzowano je w trakcie badań nad analizatorem języka polskiego bazującym na systemie grup syntaktycznych (SGS) [9].

2.1. Charakterystyka Doświadczalnych Translatorów Języka Naturalnego

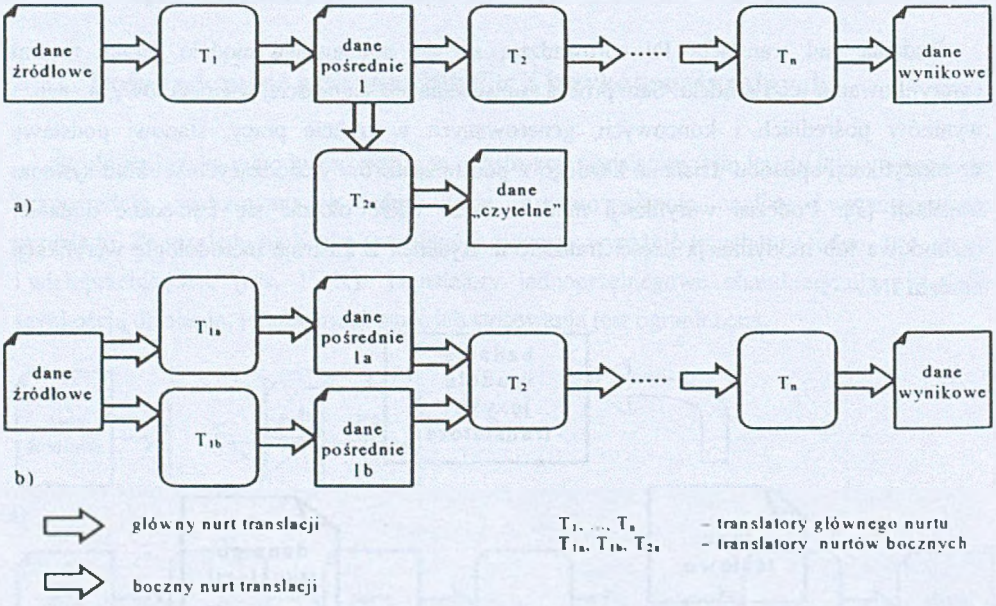
Badania nad translacją JN sprowadzają się do poddawania modelu języka testom i weryfikowaniu tego modelu. Sam proces tłumaczenia jest najczęściej wieloetapowy, a analiza wyników pośrednich i końcowych, generowanych w trakcie pracy, stanowi podstawę do modyfikacji sposobu działania każdego z podtranslatorów wchodzących w skład systemu translacji [7]. Podczas weryfikacji modelu może także okazać się konieczne dodanie, rozbudowa lub modyfikacja części translatora. Rysunek 2 ilustruje metodologię weryfikacji modelu JN.



Rys. 2. Proces konstruowania modelu translacji języka
Fig. 2. The process of constructing a language translation model

Działanie DSTJN sprowadza się do uruchamiania kolejnych programów lub procesów translatorów. Każdy z nich (analizator morfologiczny, składniowy, semantyczny itd.) przetwarza swoje dane wejściowe i generuje wyniki. Dane wyjściowe są podawane na wejście kolejnego podtranslatora. Proces translacji kończy się wraz z zakończeniem pracy ostatniego z podtranslatorów (po wygenerowaniu przekładu).

Ważną cechą środowiska sterującego przebiegiem translacji w DSTJN jest umożliwienie badaczowi edycji danych pośrednich – ręcznego ich wprowadzania – i uruchomienia procesu translacji od dowolnego etapu. Pozwala to na testowanie każdego z podtranslatorów z osobna lub ominięcie w procesie translacji elementów działających nieprawidłowo i badanie pozostałej części systemu. Podejście takie uniezależnia badania naukowców pracujących w zespole, w którym każdy z członków pracuje nad innym elementem translacji języka naturalnego (morfologia, składnia itd.), a co za tym idzie odpowiedzialnego za inny podtranslator [1].



Rys. 3. Translacja a) z nurtem bocznym, b) z rozgałęzieniem nurtu głównego
Fig. 3. Translation types: a) with side stream, b) with main stream branching

W zależności od przyjętego formatu danych pośrednich, powstających w trakcie procesu translacji, są one w mniejszym lub większym stopniu przyswajalne, czytelne dla człowieka. Niezbędne jest umożliwienie przetwarzania danych pośrednich do postaci „czytelnej". Sytuacja taka na tle procesu translacji jest odgałęzieniem od głównego jej nurtu. Rysunek 3.a. przedstawia przykład translacji z nurtem bocznym. Ogólnie, rozgałęzienie głównego nurtu

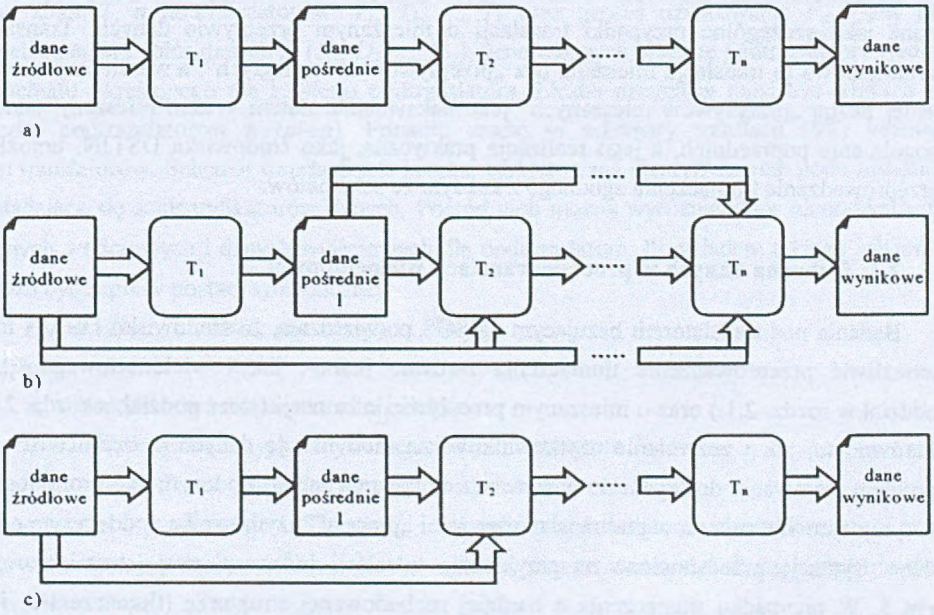
translacji wynika z faktu przetwarzania tych samych danych przez kilka podtranslatorów. Inny przykład rozgałęzionego nurtu translacji ilustruje rys. 3.b. W obu przypadkach naturalne jest zastosowanie współbieżnego przetwarzania w równoległych nurtach translacji celem jej przyśpieszenia (rys. 3.a. – równoległa praca T_2 i T_{2a}, rys. 3.b. równoległa praca T_{1a} i T_{1b}).

Każdy z etapów procesu tłumaczenia języka naturalnego jest najczęściej złożony i czasochłonny. Istotne jest zatem, by DSTJN umożliwiał badaczowi przerwanie tłumaczenia w dowolnym momencie (np. po stwierdzeniu niepoprawności wyników pośrednich). Powinien on także całkowicie oddawać użytkownikowi kontrolę nad danymi (źródłowymi, pośrednimi, końcowymi), które są danymi wejściowymi lub wyjściowymi zakończonych procesów podtranslacji.

2.2. Przepływ danych w procesie translacji

Translatory z jednym nurtem translacji możemy podzielić ze względu na przepływ danych. Przepływ może być:

- strumieniowy (szeregowy) – rys. 4.a.,
- równoległy – rys. 4.b.,
- mieszany – rys. 4.c.



Rys. 4. Modele przepływu danych w systemach translacji: a) strumieniowy, b) równoległy, c) mieszany

Fig. 4. Data flow models in translation systems: a) streamed, b) parallel, c) mixed

Praca wszystkich wyżej wymienionych systemów translacji polega na przekazywaniu „żetonu" zezwalającego na pracę kolejnym podtranslatorom. Zrównoleglenie procesu tłumaczenia z jednym nurtem jest trudne i wiąże się z zastosowaniem mechanizmów bezpośredniej komunikacji między kolejnymi podtranslatorami.

Każdy z podtranslatorów translatora strumieniowego (rys. 4.a.) korzysta z danych wynikowych wygenerowanych przez podtranslator pracujący bezpośrednio przed nim. Pierwszy translator korzysta z danych źródłowych. Wybrany podtranslator jest także generatorem danych wykorzystywanych przez podtranslator następny. Przy czym ostatni z podtranslatorów w strumieniu generuje dane wynikowe będące kompletnym przekładem.

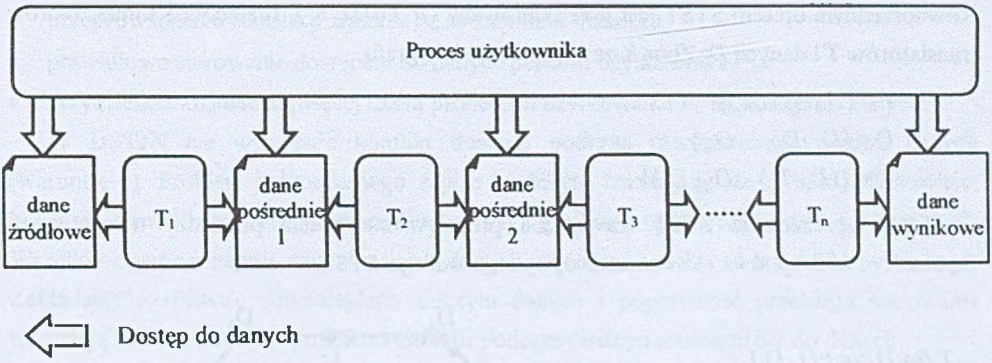
Translator z równoległym przepływem danych (rys. 4.b.), aż do wygenerowania wyniku końcowego translacji, przetwarza wszystkie dane pośrednie. Mianowicie, każdy z podtranslatorów potrzebuje do wygenerowania wyników wszystkich wcześniej wygenerowanych danych pośrednich i danych źródłowych.

W praktyce najczęściej spotykane są systemy mieszane. Można w nich wyszczególnić zarówno fragmenty realizujące translację strumieniową, jak i fragmenty, które odpowiadają translatorom o równoległym przepływie danych. Rysunek 4.c. jest przykładem takiego systemu translacji.

Zarówno systemy z przepływem strumieniowym, jak i równoległym mogą być potraktowane jako szczególne przypadki translacji o mieszanym przepływie danych. Translacja strumieniowa to translacja mieszana bez „przepływów równoległych", a w translacji równoległej liczba „przepływów mieszanych" jest maksymalna. Zatem system mieszany stanowi uogólnienie poprzednich, a jego realizacja praktyczna, jako środowiska DSTJN, umożliwi przeprowadzanie tłumaczenia zgodnego z każdym ze schematów.

2.3. Ochrona danych w procesie translacji wieloetapowej

Badania nad translatorem bazującym na SGS potwierdzają, że środowisko DSTJN musi umożliwić przeprowadzenie tłumaczenia zarówno jedno-, jak i wielonurtowego (patrz podział w rozdz. 2.1.) oraz o mieszanym przepływie informacji (patrz podział w rozdz. 2.2.). Zarówno to, jak i zezwolenie użytkownikowi na modyfikację danych pośrednich stwarza problem sterowania dostępem do nich w celu dokonania zapisu i odczytu. Kontrola dostępu dotyczy zarówno procesów podtranslatorów, jak i „procesu" użytkownika poddającego edycji dane. Sytuację przedstawiono na przykładzie translacji jednowątkowej, strumieniowej – rys. 5. W przypadku tłumaczenia o bardziej rozbudowanej strukturze (tłumaczenie wielowątkowe, mieszane) zwiększa się liczba jednoczesnych odwołań do tych samych danych.



Rys. 5. Dostęp procesów do danych
Fig. 5. Access of processes to the data

3. Problem sterowania translacją i zarządzania dostępem do danych w zintegrowanym środowisku DSTJN

Problem prawidłowego przeprowadzenia translacji można przedstawić następująco. Dany jest zbiór T, n podtranslatorów: T_1, T_2, ..., T_n oraz proces użytkownika P_u. Dany jest m-elementowy zbiór danych D (D_1, D_2, ..., D_m), generowanych przez te podtranslatory według schematu określonego dla każdego podtranslatora. Liczba danych m musi być większa od liczby podtranslatorów n ($m>n$). Ponadto znane są schematy translacji (ST) każdego z n translatorów. Schemat translacji jest krotką, etykietowaną identyfikatorem podtranslatora, składającą się z identyfikatorów danych. Pośród nich można wyróżnić zbiór identyfikatorów danych wejściowych i danych wyjściowych dla podtranslatora. Przykładem takiego schematu może być zapis w postaci symbolicznej:

$T_{nr}(in : Dwe_1, ..., Dwe_i; out : Dwy_1, ..., Dwy_j)$,

gdzie:

T_{nr} - identyfikator podtranslatora o indeksie nr,

Dwe_i - identyfikator i-tych danych wejściowych translatora T_{nr},

Dwy_j - identyfikator j-tych danych wyjściowych translatora T_{nr}.

Interpretacja powyższego ST jest następująca: translator identyfikowany przez T_{nr} wykorzystując dane wejściowe Dwe_1, ..., Dwe_i przeprowadza translację i generuje dane wyjściowe Dwy_1, ..., Dwy_j.

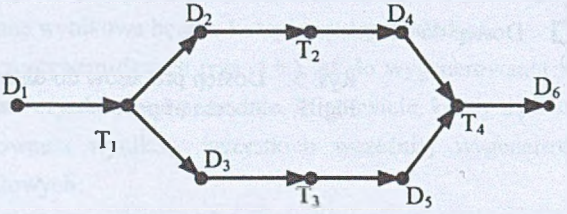
Zbiór ST, spełniający opisane w dalszej części rozdziału warunki, jest schematem translacji systemu translacji (STST). Opis symboliczny STST uwypukla związki translatorów z danymi.

Równorzędnym opisem STST jest graf skierowany $G(T \cup D, K)$, rozpięty na sumie zbiorów translatorów T i danych D. Zbór K zawiera krawędzie grafu.

$$T \in \{T_1, T_2, ..., T_n\}$$
$$D \in \{D_1, D_2, ..., D_m\}$$
$$K \in \{(D_i, T_j), ...(T_k, D_l)\}$$

Taka reprezentacja STST uwidacznia przepływ sterowania pomiędzy translatorami. Rysunek 6. ilustruje dwa równoważne opisy tego samego STST.

$T_1(in:D_1;out:D_2,D_3)$
$T_2(in:D_2;out:D_4)$
$T_3(in:D_3;out:D_5)$
$T_4(in:D_4,D_5;out:D_6)$



a) b)

Rys. 6. Równoważne opisy STST: a) prezentacja symboliczna, b) prezentacja graficzna opisu grafowego
Fig. 6. Equivalent STST descriptions: a) symbolic, b) graphic

Rozpatrzmy zbiór warunków:

1. Nie istnieją krawędzie pomiędzy dwoma elementami z tego samego zbioru (krawędzie grafu są skierowane od wierzchołków ze zbioru D do wierzchołków ze zbioru T lub od wierzchołków ze zbioru T do wierzchołków ze zbioru D).
2. Stopień wejściowy każdego wierzchołka ze zbioru T jest większy lub równy 1 (każdy translator posiada dane wejściowe).
3. Stopień wyjściowy każdego wierzchołka ze zbioru T jest większy lub równy 1 (każdy translator generuje dane wyjściowe).
4. Stopień wejściowy każdego wierzchołka ze zbioru D jest co najwyżej równy 1 (zabezpieczenie przed generowaniem tych samych danych przez więcej niż jeden translator).
5. Istnieje przynajmniej jeden wierzchołek ze zbioru D o stopniu wejściowym równym 0 (istnieją dane źródłowe).
6. Istnieje przynajmniej jeden wierzchołek ze zbioru D o stopniu wyjściowym równym 0 (istnieją dane wynikowe).
7. Graf jest acykliczny (proces translacji jest skończony).

Graf G, który spełnia zbór warunków od 1. do 7., jest STST.

Środowisko DSTJN powinno zapewnić:

- przeprowadzenie translacji zgodnie ze schematami translacji,
- prawidłowe sterowanie dostępem do danych procesu użytkownika P_u,
- przydzielić możliwie najwięcej czasu procesowi użytkownika P_u na dostęp do danych.

W DSTJN nie występuje konflikt dostępu podczas równoczesnego zapisu danych (warunek 4). Problem jednoczesnego zapisu i odczytu rozwiązuje sterowanie. Mianowicie, każdy z translatorów jest uruchamiany wtedy, gdy jego dane wejściowe są „gotowe". Wszelkie operacje zapisu danych są zakończone, więc jest możliwy bezkonfliktowy odczyt. Zakładając możliwość równoległego odczytu danych i poprawność przebiegu sterowania translacją, można pominąć problem konfliktu podczas dostępu translatorów do danych.

4. Zarządzanie przebiegiem translacji i dostępem do danych

Poniżej zostaną zaprezentowane rozwiązania problemu sterowania translacją i dostępu do danych dla wyszczególnionych (rozdz. 2.2.) typów translatorów. W ramach dalszej części przedstawione będzie rozwiązanie ogólne, możliwe do zastosowania dla dowolnego STST zarówno jedno –, jak i wielonurtowego z uwzględnieniem równoległej pracy podtranslatorów.

Rozwiązania zostaną podane przy użyciu instrukcji pseudokodu, zaproponowanego w dalszej części. Instrukcje pseudokodu pozwalają opisać zadania realizowane przez procesy.

4.1. Pseudokod opisu sterowania przebiegiem translacji

Każdy z procesów jest związany z jednym translatorem, a co za tym idzie z jednym schematem translacji tego translatora. Można wyróżnić dwa typy procesów: proces uruchamiany bezwarunkowo (przypadek 1) i warunkowo (przypadek 2) (pod warunkiem gotowości danych).

Przypadek 1: uruchamiany bezwarunkowo proces o identyfikatorze `id`

```
process id( id_zbioru_dancych, id_translatora,
   id_następnego_procesu )
begin
   //kod realizowany przez proces o identyfikatorze id
end
```

Przypadek 2: proces `id` uruchamiany warunkowo

```
process id( id_zbioru_dancych, id_translatora )
StartOnDataComplete
```

```
begin
   //kod procesu uruchamianego warunkowo
end
```

Parametrem każdego procesu jest `id_zbioru_danych`, który jest zbiorem zawierającym informacje na temat danych wejściowych i wyjściowych translatora związanego z bieżącym procesem. `Id_translatora` jest identyfikatorem translatora skojarzonego z procesem. `Id_następnego_procesu` to identyfikator procesu, który powinien być uruchomiony po zakończeniu translacji w bieżącym procesie. Jest on dostępny tylko dla procesów uruchamianych bezwarunkowo. W zbiorze danych (`id_zbioru_danych`) mogą znajdować się struktury składające się z trzech pól:

```
struct
   Data, Input, LockCount
end
```

Pole `Data` jest identyfikatorem danych wejściowych lub wyjściowych translatora. Pole `Input` może przyjmować wartości logiczne `true` lub `false`. Wartość `true` pola `Input` mówi, że dane `Data` są danymi wejściowymi dla translatora. Wartość `false` identyfikuje dane jako wyjściowe. Pole `LockCount` informuje o liczbie odwołań do danych w STST.

Uruchomienie procesu o identyfikatorze `id` (przypadek 1) odbywa się poprzez wywołanie funkcji `StartProces(id)`. W przypadku 2 uruchomienie procesu odbywa się automatycznie po skompletowaniu danych niezbędnych do przeprowadzenia skojarzonej z nim translacji. Gotowość danych sygnalizowana jest poprzez wywołanie funkcji `Complete(`D_i`)`, gdzie D_i jest identyfikatorem i-tych, gotowych danych. Na przykład dany jest ST:

T(in: D_1, D_2; out: ...)

Związany z nim jest proces `P`

```
process P( ... ) StartOnDataComplete
begin
   //kod procesu
end
```

który zostanie uruchomiony, jeżeli przez inne procesy zostaną wywołane funkcje `Complete(`D_1`)`, `Complete(`D_2`)` (w dowolnej kolejności).

Funkcja `Translate(`T_i`)` przeznaczona jest do uruchomienia i-tego translatora. Dba ona o przekazanie translatorowi odpowiednich danych. Ponadto są dostępne funkcje umożliwiające blokowanie – `Lock(`D_i`)` – i odblokowywanie – `Unlock(`D_i`)` – dostępu do danych dla procesu użytkownika. Z każdą z danych D_i (gdzie: $1 \leq i \leq m$) związany jest licznik wywołań tych funkcji. Zapewnia on, że zasób k-krotnie zablokowany musi zostać

k-krotnie odblokowany, by użytkownik mógł z niego korzystać. Dostępna jest także funkcja `Lock(Di, k)`. Jej wywołanie jest równoważne z k-krotnym wywołaniem funkcji `Lock(Di)`. Istnieje możliwość wywołania funkcji `IsFirst(Ti)` i `IsLast(Ti)`. Informują one, czy podtranslator T_i jest pierwszym czy ostatnim w ciągu translacji opisanym przez STST.

Przedstawiony język umożliwia bezpośredni przekład kodu procesów na język implementacji w systemie wielozadaniowym (np.: Unix, Windows). Proces uruchamiany bezwarunkowo lub warunkowo odpowiada zadaniu (ang. task) w takim systemie. W przypadku procesów warunkowych konieczne jest stworzenie odpowiedniego „zaplecza" zajmującego się rejestracją skompletowanych zasobów oraz odpowiednim uruchamianiem wątków – procesów warunkowych. Wszystkie podtranslatory mogą być programami bądź bibliotekami uruchamianymi z wnętrza zadań.

Zabezpieczenie zasobów przed modyfikacją ich przez użytkownika w trakcie translacji może być zrealizowane na wiele sposobów, np. przez wykonanie kopii pliku danych na potrzeby translacji lub poprzez uniemożliwienie modyfikacji danych w edytorze środowiska DSTJN.

4.2. Rozwiązanie problemu – translator równoległy

Rozważania nad translacją z równoległym przepływem danych (patrz rozdz. 2.2.) zostały celowo skrócone w tym rozdziale. Jest ona przypadkiem skrajnym, a rozwiązanie problemu zarządzania danymi i sterowania sprowadza się do:

- zablokowania dostępu do wszystkich danych, dla procesu użytkownika P_u, przed rozpoczęciem translacji,
- przekazywania sterowania kolejnym podtranslatorom T_1, ..., T_n w oparciu o zasadę żetonu,
- zezwolenie na dostęp do wszystkich danych, dla procesu użytkownika P_u, po zakończeniu translacji.

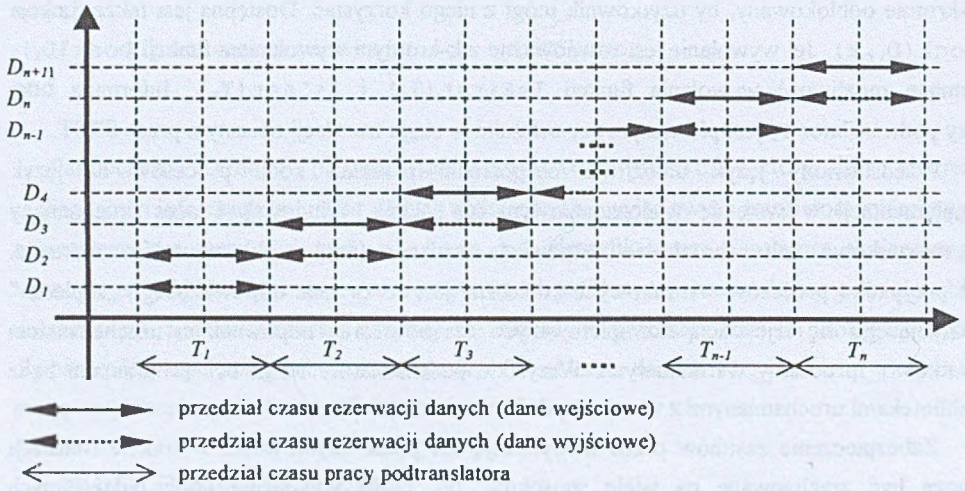
Rozwiązanie takie jest rozwiązaniem optymalnym pod względem czasu, jaki użytkownik ma na edycję danych. Jednocześnie przypadek ten, z punktu widzenia użytkownika, jest najmniej korzystny, ponieważ użytkownik nie ma możliwości modyfikacji żadnych danych od startu aż do zakończenia translacji.

4.3. Rozwiązanie problemu – translator strumieniowy

Schemat działania każdego z podtranslatorów w translacji strumieniowej, jednonurtowej można wyrazić następującą formułą:

$T_i(\ in: D_i;\ out: D_{i+1}\ )$,

gdzie i jest indeksem podtranslatora w strumieniu.



Rys. 7. Rezerwacja danych przez podtranslatory w translatorze strumieniowym
Fig. 7. Data reservation by subtranslators in streamed translator

Translator Ti w oparciu o dane D_i generuje wyniki i zapisuje do danych identyfikowanych jako D_{i+1}. Wykres na rysunku 7 przedstawia rezerwację (w czasie) danych, systemu translacji skonstruowanego w oparciu o powyższy schemat. Działanie każdego z procesów Ti (gdzie $1 < i < n$) sprowadza się do następujących operacji:

- zablokowania dostępu (dla P_u) do danych wynikowych D_{i+1},
- wykonania właściwej translacji,
- powołania procesu T_{i+1},
- zwolnienia (na rzecz P_u) danych źródłowych D_i.

Uruchomienie procesu, identyfikowanego przez $i+1$, po zakończeniu pracy i-tego translatora gwarantuje, że nie dojdzie do konfliktu zapis-odczyt, jaki mógłby mieć miejsce w trakcie równoległej pracy translatorów o identyfikatorach T_i oraz T_{i+1}.

Inaczej niż P_i będą wyglądały procesy P_1 i P_n. Proces P_1 oprócz wykonywania czynności, takich jak i-ty proces, powinien, przed realizacją właściwej translacji (T_1), rezerwować dane źródłowe D_1. Żaden z pozostałych procesów takiej czynności nie wykonuje, gdyż jego dane źródłowe są zablokowane przed dostępem P_u przez proces poprzedni. Podobnie ma się sytuacja z danymi wynikowymi D_{n+1} translatora T_n. Mianowicie, muszą one zostać zwolnione w procesie n-tym po wykonaniu n-tej translacji właściwej. Ponadto n-ty proces translatora nie uruchamia kolejnego procesu translacji. Poniżej został przedstawiony pseudokod realizujący powyższe zadania.

```
process P( DataSet, T, PNext )      //proces translatora
begin
  if IsFirst( T ) then
    for each D in DataSet do
      Lock( D.Data )                //rezerwacja danych wej. i wyj.
  else
    for each D in DataSet do
      if D.Input = false then
        Lock( D.Data )              //rezerwacja danych wyj.

  Translate( T )                    //translacja właściwa

  if not IsLast( T ) then
    StartProcess( PNext )           //uruchomienie kolejnego procesu
  if IsLast( T ) then
    for each D in DataSet do
      Unlock( D.Data )    //odblokowanie danych wej. i wyj.
  else
    for each D in DataSet do
      if D.Input = true then
        UnLock( D.Data )            //odblokowanie danych wej.
end
```

Modyfikacja schematu translacji (dodanie lub usunięcie podtranslatora) nie pociąga zmian w kodzie realizującym procesy translacji. Dodanie lub usunięcie i-tego procesu wiąże się jedynie z modyfikacjami wartości parametrów przekazywanych do procesów P_{i-1} oraz P_{i+1}. Istotną zaletą zarówno rozwiązania przedstawionego wyżej, jak i rozwiązań zaprezentowanych w dalszej części artykułu jest możliwość zastosowania automatycznego generowania parametrów procesów na podstawie schematów translacji. Bazując na przedstawionych (za pomocą pseudokodu) zasadach sterowania, możliwa jest też bezpośrednia interpretacja schematów.

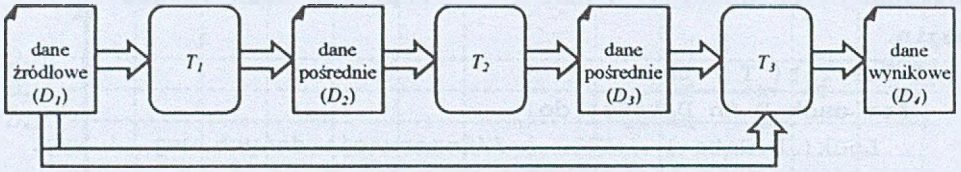
4.4. Rozwiązanie problemu – translator mieszany

Załóżmy przebieg translacji zgodny z poniższym schematem (rys. 8.).

$T_1(in : D_1; out : D_2)$

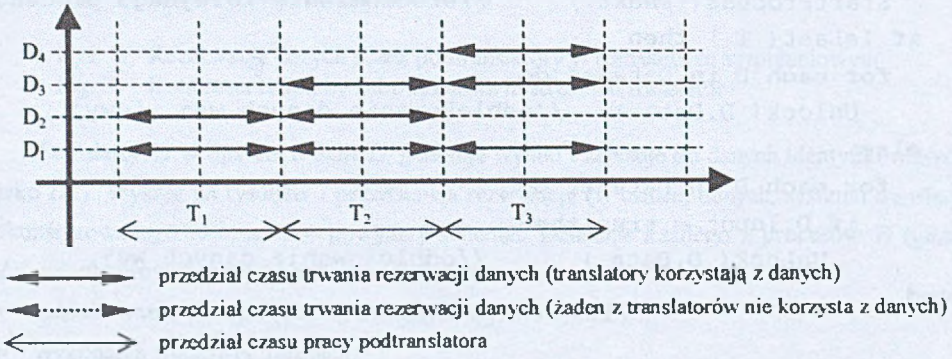
$T_2(in : D_2; out : D_3)$

$T_3(in : D_1, D_3, out : D_4)$



Rys. 8. Przykład translacji o mieszanym przepływie danych
Fig. 8. Example of mixed data flow translation

W przykładzie tym dwa podtranslatory T_1 i T_3 używają danych D_1 jako swoich danych źródłowych wykorzystywanych do przeprowadzenia translacji. Użytkownik systemu translacji będzie widział taką sytuację jako brak dostępu do danych D_1 od momentu rozpoczęcia pracy procesu z translatorem T_1 do zakończenia procesu z translatorem T_3. Rysunek 9 przedstawia przebieg czasowy blokady danych.



Rys. 9. Rezerwacja danych dla przykładowego translatora mieszanego – przebiegi czasowe
Fig. 9. Data reservation for an example mixed translator – time diagram

Problem sterowania dostępem do danych D_1 można rozwiązać blokując dostęp do nich przed uruchomieniem translatora T_1 i zwalniając po zakończeniu pracy translatora T_3. Dla takiego podejścia automatyczna generacja kodu procesów lub interpretacja schematów mogłyby się okazać niemożliwe do zrealizowania. Przyczyną tego jest wystąpienie sytuacji, w której nie jest znana kolejność zakończenia się procesów korzystających z tych samych danych wejściowych.

Ze względu na możliwość ewentualnej automatyzacji lepszym rozwiązaniem jest wykorzystanie instrukcji blokującej i zwalniającej dane (`Lock(D_i,c)`, `Unlock(D_i)`), które rejestrują liczbę operacji sterujących dostępem do danych (rozdz. 4.1.). Kod implementujący procesy przykładowego translatora został podany poniżej.

```
process P( DataSet, T, PNext )
begin
  if IsFirst( T ) then
    for each D in DataSet do
      Lock(D.Data,D.LockCount )//rezerwacja danych wej. i wyj.
  else
    for each D in DataSet do
      if D.Input = false then
        Lock( D.Data, D.LockCount )//rezerwacja danych wyj.

  Translate( T )                    //translacja właściwa

  if not IsLast( T ) then
    StartProcess( PNext )       //uruchomienie kolejnego procesu
  for each D in DataSet do
    Unlock( D.Data )        //odblokowanie danych wej. i wyj.
end
```

Podobnie jak w translatorze strumieniowym tak i w przypadku translatora o mieszanym przepływie danych należy wskazać schemat lub schematy startowe. Proces, który go realizuje, jest wykonany jako pierwszy. Zastosowanie liczników i związanej z nimi techniki zapewnia prawidłowe zwolnienie wszystkich zasobów. Dzięki temu zwalnianie dostępu do danych jest realizowane w taki sam sposób przez wszystkie procesy.

4.5. Rozwiązanie ogólne

Chcąc rozszerzyć system translacji o schematy wielonurtowe, należy rozpatrzyć dwie nieprzeanalizowane w poprzednich rozdziałach sytuacje:

a) tworzenie nowego nurtu translacji (rys. 10.a.),

b) schodzenie się dwóch nurtów translacji w jeden (rys. 10.b.)

$T_k(\ in : ...;\ out : D_i\ )$ (a)

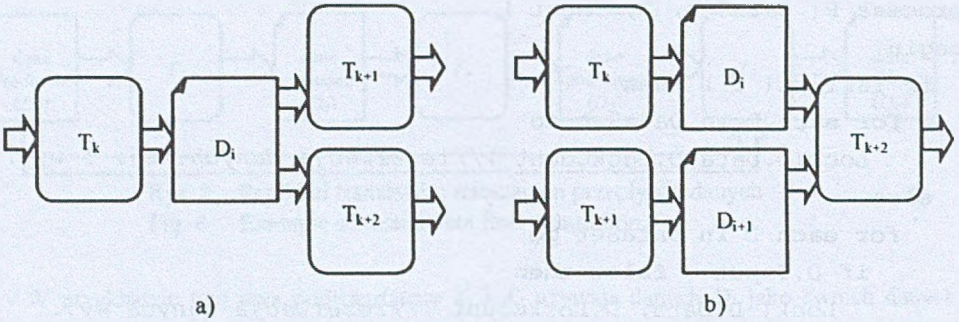
$T_{k+1}(\ in : D_i;\ out: ...\ )$

$T_{k+2}(\ in : D_i;\ out: ...\ )$

$T_k(\ in : ...;\ out: D_i\ )$ (b)

$T_{k+1}(\ in : ...;\ out: D_{i+1}\ )$

$T_{k+2}(\ in : D_i, D_{i+1}\ ;\ out : ...\ )$



Rys. 10. Rozwidlenie nurtu translacji (a) i złożenie nurtów translacji (b)
Fig. 10. Translation stream branching (a) and assembly (b)

Sytuację pierwszą (a) można zaimplementować jak poniżej.

```
process P( DataSet, T ) StartOnDataComplete
begin
  if IsFirst( T ) then
    for each D in DataSet do
      Lock(D.Data,D.LockCount)//rezerwacja danych wej. i wyj.
  else
    for each D in DataSet do
      if D.Input = false then
        Lock(D.Data,D.LockCount)    //rezerwacja danych wyj.
  Translate( T )                    //translacja właściwa
  for each D in DataSet do
    if not D.Input then
      Complete( PNext )          //uruchomienie kolejnego procesu
  for each D in DataSet do
    Unlock( D.Data )          //odblokowanie danych wej. i wyj.
end
```

Ze schematu translacji b. (rys. 10.b.) wynika, że translator T_{k+1} może być uruchomiony tylko po zakończeniu translacji przez procesy T_k i T_{k+1}. Bez wiedzy o kolejności zakończenia T_k i T_{k+1}, nie można rozwiązać problemu sterowania za pomocą procesów uruchamianych bezwarunkowo. Zastosowanie procesów uruchamianych warunkowo umożliwia implementację schematu b. Zatem powyższy kod pozwala zrealizować także ten przypadek.

Zaprezentowany pseudokod sterowania przebiegiem translacji umożliwia opisanie translacji najszerszego typu, tj. translacji wielonurtowej, o mieszanym przepływie danych.

Pozwala także na opis wszystkich rodzajów translatorów omówionych w rozdziałach 2.1. i 2.2. Kod zapisany w tym języku można wprost przetłumaczyć na język programowania. Powinien on zostać uzupełniony o „zaplecze" umożliwiające uruchamianie procesów warunkowych oraz realizujące przekazywanie do nich parametrów. Można też, mając na uwadze mechanizmy zaimplementowane w pseudokodzie, konstruować interpretatory STST.

5. Podsumowanie

Podstawę do sformułowania opisanych zasad stanowiły badania nad stworzonym środowiskiem DSTJN, przeprowadzającym trójstopniową analizę morfologiczną. Środowisko to nie jest jednak uniwersalne. Schematy translacji opisujące pracę DSTJN są zaszyte w jego kodzie, co uniemożliwia ich zmianę bez ingerencji w tekst źródłowy.

Niniejszy artykuł może stanowić podstawę do stworzenia konfigurowalnego środowiska dla systemu translacji JN. Powinno ono zawierać interpretator schematów translacji, który będzie pracował w oparciu o przedstawione rozwiązania. Przyczyni się to do ułatwienia i przyśpieszenia prowadzonych aktualnie badań nad translacją języków naturalnych, bazującą na systemie grup syntaktycznych. Dzięki możliwości interpretacji STST i stworzeniu systemu bibliotek podtranslatorów umożliwi łatwą i szybką rekonfigurację systemu translacji. Pozwoli to także na prowadzenie badań porównawczych wielu DSTJN, bez konieczności zmiany środowiska programowego.

Moduły czy też biblioteki z poszczególnymi podtranslatorami będą mogły być napisane w różnych językach programowania. Niezależnie od tego będą one współpracowały ze sobą i z modułem interfejsu użytkownika środowiska DSTJN. Zatem artykuł ten może stanowić podstawę opracowania zasad i mechanizmów komunikacji między modułami, standardu interfejsów programowych, jak i elementów interfejsu użytkownika aplikacji środowiska DSTJN.

LITERATURA

1. Raport badań BK-202/Rau2/98, Wizualizacja algorytmów i procesów obliczeniowych; automatyczna analiza składniowa języka polskiego, Politechnika Śląska, Instytut Informatyki.
2. Bolc L., Mykowiecka A.: Podstawy przetwarzania języka naturalnego. Wybrane metody formalnego zapisu składni, AOWRM, Warszawa 1992.

3. Bolc L., Cichy M., Różańska L.: Przetwarzanie języka naturalnego. WNT, Warszawa 1982.
4. Gries D.: Konstrukcja translatorów dla maszyn cyfrowych. PWN, Warszawa 1984.
5. Jassem K.: POLENG – a Machine Translation System Based on an Electronic Dictionary. Poznań 1998.
6. Klein M.: Przewodnik po bibliotekach DLL i sposobach zarządzania pamięcią. Intersoftland, Warszawa 1994.
7. Pius ten Hacken: Science and Technology in Machine Translation, 10th ESSiLLaI, 17-28 August 1998 in Saarbrucken. s. 4-16.
8. Suszczańska N.: Zautomatyzowane stanowisko robocze lingwisty. Materiały IV Krajowej Konferencji KOWBAN 97, Świeradów Zdrój 1997. s. 45-53.
9. Suszczańska N., Szmal P., Szczepankowski B.: Koncepcja programu tłumaczenia komputerowego z języka polskiego na język migowy. Katowice 1998. s. 63-72.

Recenzent: Dr hab. Zygmunt Vetulani Prof. UAM

Wpłynęło do Redakcji 12 września 2000 r.

Abstract

In the Software Division at the Department of Computer Science of Silesian Institute of Technology a research on automatic translation of Polish into gesture language is led. The programs performing consecutive stages of the translation are a result of research conducted so far, and are still being improved [9].

Research over natural language translation consist in creating a model of a language and verifiying it in practice. The translation process is usually multistage, and the modifications of the subtranslators are based on an analysis of intermediate and final results, generated while working [7].

An important feature of an environment that orders the translation process in an experimental system of NL translation is to let the researcher edit the intermediate results and execute translation process from any stage. It allows for autonomous testing of the subtranslators and bypassing the anomalously working stages of the translation process to test the rest of the system. This way the researches working in the team are independent. Each of them works on another element of natural language translation (morphologic, syntactic, semantic, etc.), and is responsible for a different subtranslator [1].

The present paper describes the specific of research work on an experimental system of natural language translation. The possibilities of speeding the system up by parallel operation of subtranslators are investigated. The flow of translation in different types of translators is described: from unistream translators with the simplest translators to complex multistream translators with mixed data flow.