INSTYTUT ŁĄCZNOŚCI
WARSZAWA-MIEDZESZYN

BIULETYN INFORMACYJNY

1 (206)
1982

MINISTERSTWO ŁĄCZNOŚCI

BIULETYN INFORMACYJNY

ROK 22 WARSZAWA 1982 NR 1/206/

INSTYTUT ŁĄCZNOŚCI

Branżowy Ośrodek
Informacji Naukowej, Technicznej i Ekonomicznej

Redakcja Biuletynu Informacyjnego

Redaktor Naczelny - prof. mgr inż. Lesław Kędzierski
Z-ca Redaktora Naczelnego - doc. dr inż. Krystyn Plewko

Redaktorzy działów:
doc. mgr inż. Władysław Cetner, doc. mgr inż. Adam Moniuszko

Adres Redakcji:
Instytut Łączności
Branżowy Ośrodek
Informacji Naukowej, Technicznej i Ekonomicznej
Warszawa - Miedzeszyn, ul. Szachowa 1

Redaktor: mgr K. Juszkiewicz Montaż tekstu: B. Drabik

Dział Wydawniczy Instytutu Łączności
Format B5. Nakład 625. Wpłynęło do
Działu Wydawniczego 25.III.1982 r.
Druk ukończono w maju 1982 r.

Andrzej Hildebrandt

JĘZYKI PROGRAMOWANEGO STEROWANIA, WEDŁUG ZALECEŃ CCITT

S P I S T R E Ś C I

Andrzej Hildebrandt

JĘZYKI PROGRAMOWANEGO STEROWANIA, WEDŁUG ZALECEŃ CCITT

1. WPROWADZENIE

Niniejsze opracowanie dotyczy trzech języków programowanego sterowania /ang. stored program control languages lub SPC languages/, które zostały opracowane przez CCITT i przyjęte w wielu instytucjach jako standardy. Są to następujące języki:

- język SDL - język wymagań i opisu /ang. Specification and Description Language/, który jest przeznaczony do opisu wymagań funkcjonalnych urządzeń telekomutacyjnych, opisu działania tych urządzeń, a także do opisu wewnętrznych procesów logicznych w nich zachodzących;
- język CHILL /CCITT Higher Level Language/, który jest językiem programowania wysokiego poziomu o właściwościach odpowiadających specyficznemu zastosowaniu, jakim jest tworzenie różnorodnego oprogramowania występującego w urządzeniach telekomutacyjnych;
- język MML /Man - Machine Language/, który jest językiem dialogowym przeznaczonym do komunikowania się obsługi z urządzeniami telekomutacyjnymi przy ich użytkowaniu i utrzymaniu a także przy instalowaniu tych urządzeń.

Wymienione języki mają trzy istotne cechy, na które warto zwrócić uwagę:

- są one niezależne od sprzętu, co znaczy, że ich stosowanie nie jest ograniczone ani przez technologię w jakiej wykonywany jest sprzęt /np. stopień integracji elementów/, ani przez architekturę sprzętu /np. sposób rozwiązania sterowania, liczbę i typy użytych procesorów i sposób współpracy między nimi/;
- są one funkcjonalnie bogate, co umożliwia różnorodność ich zastosowań; mogą one być stosowane w urządzeniach telefonicznych, telegraficznych i transmisji danych, zarówno w urządzeniach wąsko wyspecjalizowanych jak i w przyszłych urządzeniach systemów zintegrowanych; do określonego zastosowania dobiera się zwykle podzbiór języka;

- ich poziom jest odpowiednio dobrany, a więc nie jest zbyt wysoki, co mogłoby ograniczać zakres ich zastosowań, szczególnie w stosunku do tych zastosowań w przyszłości, których nie da się obecnie dokładnie określić; nie jest także zbyt niski, co mogłoby zniweczyć korzyści, wynikające z ich stosowania czy standaryzacji.

Powyższe cechy sprawiają, że języki te są uniwersalne i że jest możliwe szerokie ich stosowanie.

Celem niniejszego opracowania jest zapoznanie czytelników z podstawowymi właściwościami tych języków, ze stanem ich wdrażania a także przedstawienie propozycji działań, jakie w tej sprawie powinny być w kraju prowadzone. Autor liczy na to, że czytelnicy powiększą grono osób przekonanych o niezbędności wprowadzenia tych języków do praktyki inżynierskiej i będą aktywnie działać w tym kierunku. W opracowaniu zawarte są tylko podstawowe informacje o językach programowanego sterowania CCITT. Nie mogą one stanowić podstawy do nabycia umiejętności posługiwania się nimi ani do rozpoczynania prac nad ich implementacją. Tych czytelników, których zainteresują głębiej te zagadnienia, odsyłam do literatury. Mniej zorientowanym czytelnikom polecam dla uzupełnienia książkę J. Miernika: Programowane sterowanie central telefonicznych [1].

2. GENEZA JĘZYKÓW PROGRAMOWANEGO STEROWANIA CCITT

Znaczna część "inteligencji" programowo sterowanych urządzeń komutacyjnych zawarta jest w ich oprogramowaniu. W miarę postępu w ich budowie udział ten staje się coraz większy. W ostatnich latach przyczynia się do tego szerokie wprowadzanie mikroprocesorów, których programy przejmują inteligencję zawartą poprzednio w sprzęcie, oraz wzrost złożoności funkcji wykonywanych przez urządzenia komutacyjne.

Prace nad językami programowanego sterowania podjęto w CCITT w okresie, gdy pojawił się tzw. "kryzys softwarowy". Kryzys ten powstał z sytuacji, gdy wymagano, aby oprogramowanie wykonywało coraz bardziej złożone funkcje. Powodowało to znaczne zwiększanie jego rozmiarów. Prymitywne sposoby wytwarzania oprogramowania oparte głównie na pomysłowości poszczególnych programistów, nie pozwalały na sprostanie wzrastającym wymaganiom. Oprogramowanie stawało się zawodne, kosztowne i trudno modyfikowalne. Jako reakcja na tę sytuację powstała, zapoczątkowana pracami E.W.

Dijkstry w r. 1972, inżynieria oprogramowania. Inżynieria oprogramowania wprowadziła wiele reguł, które celowo ograniczają w znacznym stopniu istniejącą poprzednio ogromną dowolność postępowania przy konstruowaniu oprogramowania. Trzeba bowiem pamiętać o tym, że gotowy do wykonania program składa się z elementów, którymi są rozkazy maszynowe i dane /liczby/. Elementy te dla realizacji określonej funkcji mogą być zestawiane na bardzo wiele różnych sposobów, gdyż poszczególne tak rozumiane elementy "pasują do siebie" znacznie lepiej niż jakiekolwiek elementy sprzętowe. Dziś, gdy inżynierii oprogramowania uczymy się z podręczników /wydawanych także w języku polskim/, w umysłach wielu ludzi pozostało jeszcze przeświadczenie, że obszerne oprogramowanie jest czymś niebezpiecznym i niepożądanym. Pewne zamieszanie w tych sprawach spowodowało bardzo intensywne ostatnio wprowadzanie mikroprocesorów, dla których oprogramowanie tworzy się często prymitywnymi metodami i nie zawsze zgodnie z wymogami inżynierii oprogramowania. Jest to sytuacja paradoksalna. Olbrzymi postęp w sprzęcie, który doprowadził do wprowadzenia mikroprocesorów, spowodował regres w wytwarzaniu oprogramowania.

Na tym tle można stwierdzić, że działalność Komisji XI CCITT mająca na celu wprowadzenie w rozpatrywanym obszarze pewnych standardów była wysoce pożądana. Prace nad trzema opisanymi językami trwają w CCITT w dalszym ciągu. Osiągnęły one jednak takie stadium, że możliwe stało się wdrażanie tych języków, co jest z kolei warunkiem koniecznego sprzężenia zwrotnego umożliwiającego poprawienie i uzupełnienie zaleceń.

Język SDL powstał jako rozwinięcie stosowanego przez firmy japońskie sposobu przedstawiania wymagań funkcjonalnych w postaci diagramów stanów i przejść /ang. state transition diagrams [2]/. Sposób ten wybrano jako podstawę do dalszych prac spośród kilku metod opisu stosowanych przez przodujące firmy.

Język CHILL został zaprojektowany od podstaw. Próby przyjęcia jako standardu CCITT jednego z istniejących języków programowania lub jego modyfikacji zakończyły się niepowodzeniem, gdy stwierdzono, że żaden z przebadanych kilkudziesięciu języków nie spełnia sformułowanych wymagań.

Język MML powstał jako rozwinięcie opracowanego dla francuskiego systemu E1 języka człowiek-maszyna [3], znanego w Polsce w zmodyfikowanej wersji jako język komunikacji człowiek-maszyna stosowany w centrach eksploatacji technicznej /CTI/ systemu E10.

Definicje wymienionych języków, a także opisy ułatwiające ich wdrażanie i użytkowanie znajdują się m.in. w zaleceniach umieszczonych w Żółtej Księdze CCITT. Lista tych zaleceń wraz z komentarzami o stopniu ich kompletności zawarta jest w załączniku do niniejszego opracowania. Poza Żółtą Księgą zalecenia te można znaleźć w dokumentach CCITT [4,5,6].

3. OBSZARY ZASTOSOWAŃ JĘZYKÓW PROGRAMOWANEGO STEROWANIA CCITT

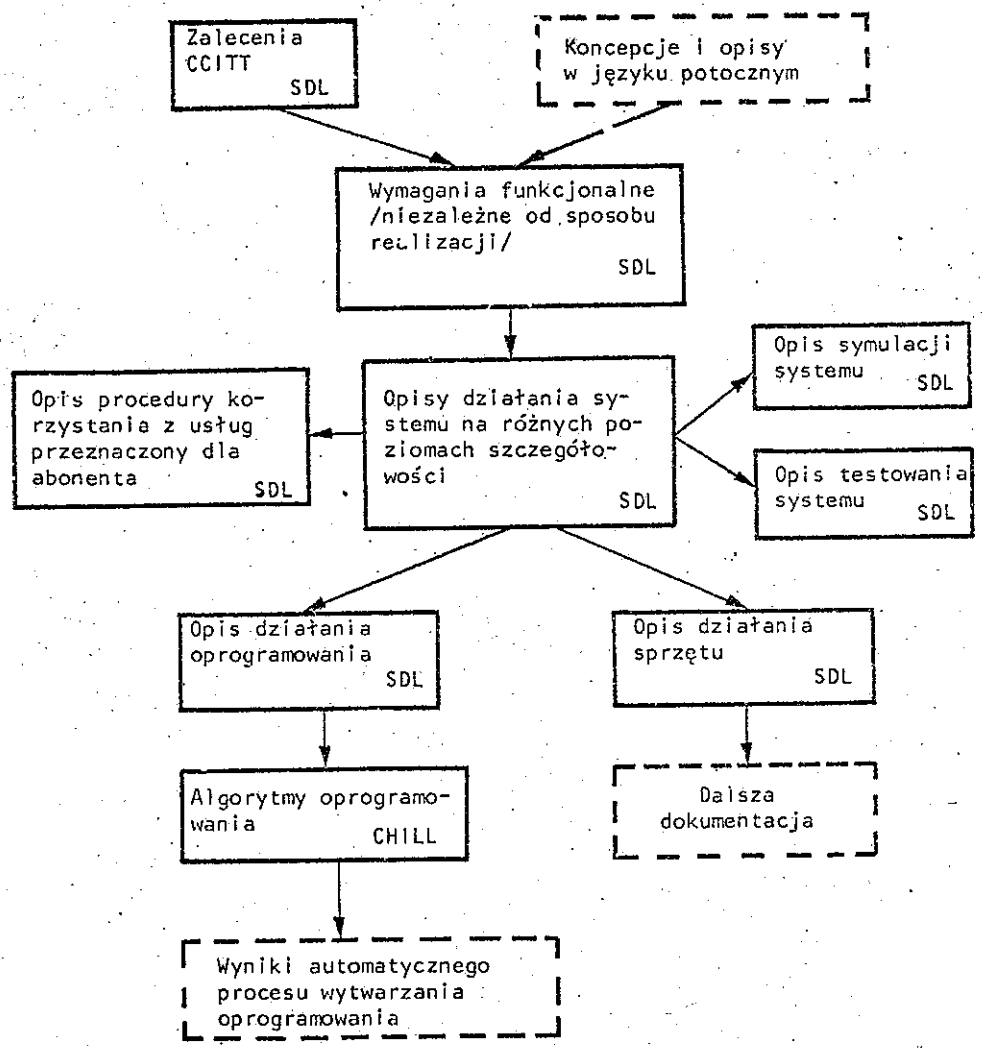
Języki programowanego sterowania CCITT mogą i powinny stanowić istotne elementy w procesach wytwarzania i eksploatacji urządzeń telekomutacyjnych. Oczywiście problemy mogłyby być rozwiązywane także przez tworzenie odpowiednich języków w skali lokalnej. Dlatego należy zwrócić uwagę na dwa aspekty sprawy:

- po pierwsze na to, że standaryzacja międzynarodowa języków znacznie ułatwia porozumiewanie się pomiędzy administracjami, konstruktorami i użytkownikami w różnych krajach;
- po drugie na to, że ponieważ języki CCITT są dobrze dostosowane do szerokiego wachlarza zastosowań, więc ich stosowanie ułatwia porozumienie się w sprawach dotyczących różnych rodzajów i typów urządzeń.

Na rysunkach 3.1 i 3.2 przedstawiono obszary zastosowań języków CCITT. Szczególnie szeroki jest obszar zastosowań języka SDL. Diagramy w języku SDL mogą być wykorzystywane przez administrację łączności do tworzenia wymagań funkcjonalnych dla producentów urządzeń, przez producentów urządzeń do tworzenia wielopoziomowej dokumentacji niezbędnej w procesie projektowania i wytwarzania urządzeń, przy czym stosuje się go z powodzeniem zarówno do opisu działania całego systemu, jak i do opisów sprzętu i opisów oprogramowania. W SDLu mogą być tworzone opisy korzystania z poszczególnych usług przeznaczone dla użytkowników, przy czym do użytkowników można tutaj zaliczyć oprócz ludzi także urządzenia transmisji danych. Automatyczna translacja z języka SDL na CHILL i odwrotnie ułatwia proces projektowania i wykonywania oprogramowania.

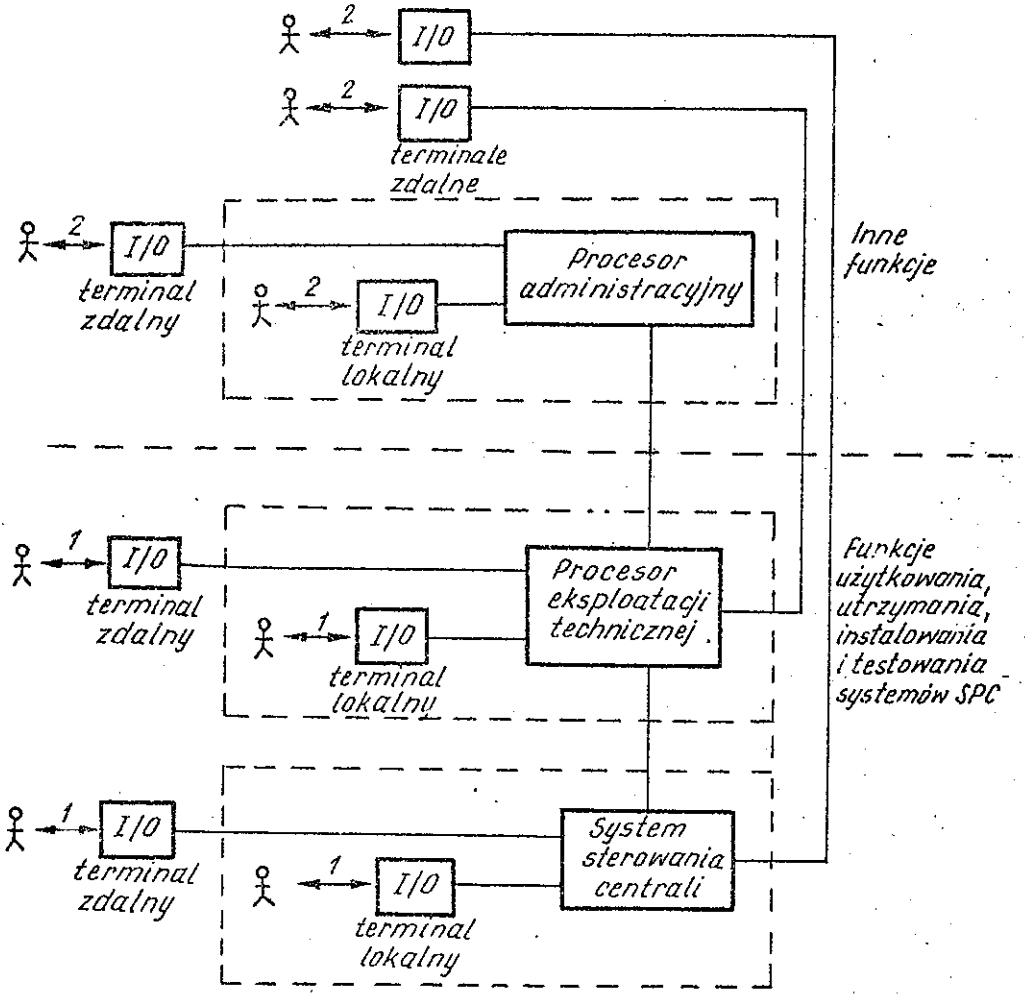
Obszar zastosowania języka CHILL ma ściślej wytyczone granice, gdyż jest to język programowania wysokiego poziomu, a więc służy do tworzenia programów. Używa się go zarówno do pisania programów operacyjnych, a więc tych, które sterują procesami przebiegającymi w urządzeniach w czasie ich eksploatacji, jak również w programach pomocniczych, które są potrzebne

w procesach wytwarzania, a więc np. w translatorach, symulatorach, czy testach.



Rys. 3.1. Obszary zastosowań języków SDL i CHILL

Język MML /zgodnie z rys. 3.2/ stosowany jest do prowadzenia dialogu przez personel producenta w procesie instalowania i testowania urządzeń, ale głównie przez personel eksploatacyjny, który przekazuje urządzeniom polecenia związane z użytkowaniem i utrzymaniem tych urządzeń, oraz sieci telekomunikacyjnej, a także odbiera od urządzeń informacje o ich aktual-



Rys. 3.2. Obszar zastosowań języka MML

1 - styki, na których zalecane jest stosowanie języka MML,
2 - styki, których zalecenia nie dotyczą

nym stanie, błędach, alarmach itp. Język ten stosowany jest także do komunikacji z systemem sterowania centrali oraz z systemami nadzorującymi poszczególne centrale czy ich grupy.

Standaryzacja języków SDL i CHILL ma duże znaczenie dla rozwoju technik zautomatyzowanego projektowania i produkowania oprogramowania. Automatyzacja polega tu na komputerowym lub wspomaganym przez komputer wytwarzaniu, przetwarzaniu i przechowywaniu różnego rodzaju dokumentów i pro-

gramów. Trzeba jednak pamiętać, że opracowanie systemu zautomatyzowanego projektowania jest pracochłonne oraz kosztowne i dlatego korzystne jest wczesne wprowadzenie międzynarodowych standardów, po to aby wielu użytkowników mogło korzystać z systemu wyprodukowanego przez jednego producenta.

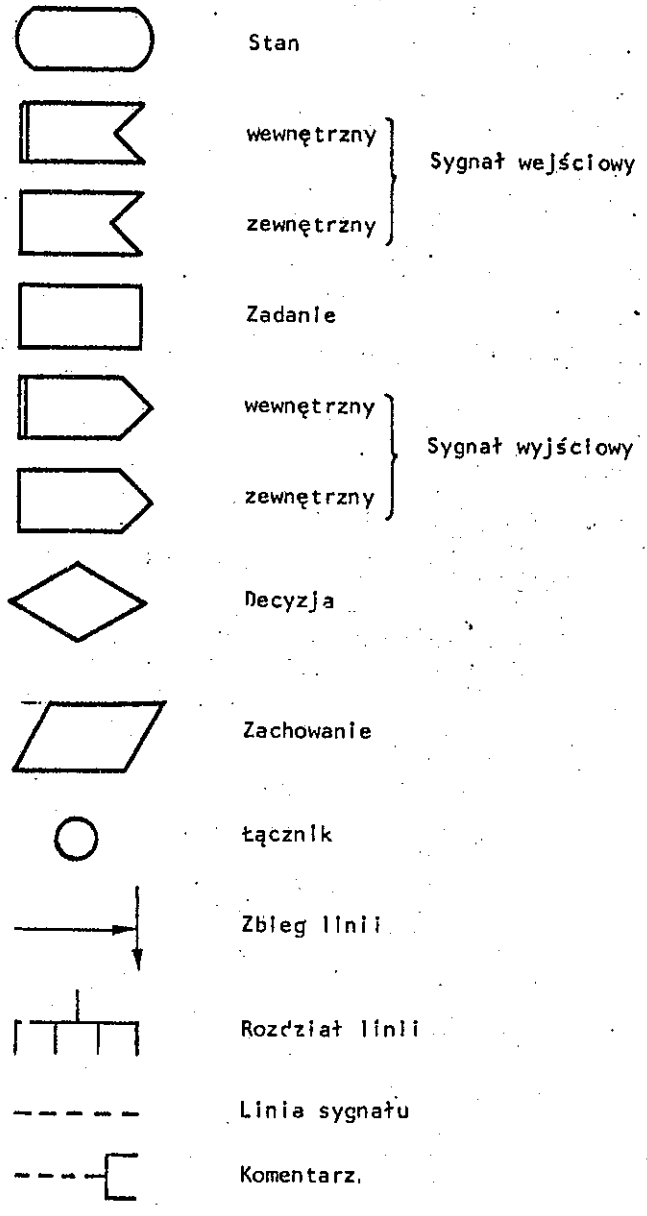
Dziś nie można wyobrazić sobie projektowania, wytwarzania i użytkowania urządzeń telekomunikacyjnych bez użycia odpowiednich języków. Dlaczego zatem nie stosować języków CCITT?

4. CHARAKTERYSTYKA JĘZYKA SDL

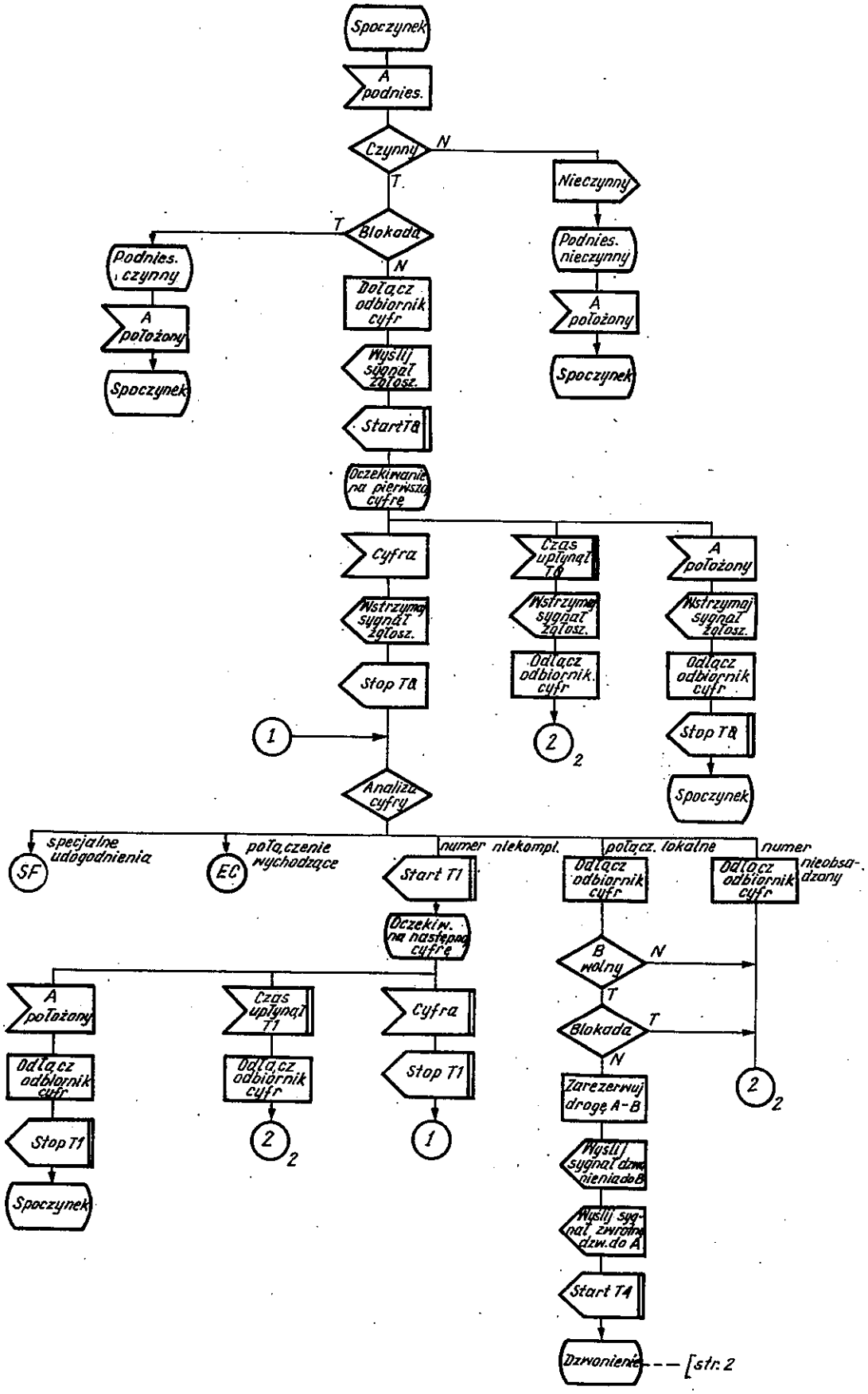
Właściwości języków programowanego sterowania CCITT opisywać będziemy posługując się przykładami ich zastosowań. Zdajemy sobie sprawę z niedoskonałości takiej metody, gdyż w wybranych przykładach nie można zawrzeć wszystkich istotnych czy interesujących możliwości języka. Niemniej jednak, taka metoda wydaje się najbardziej odpowiednia, biorąc pod uwagę zarówno obojętność niniejszego opracowania jak i cel, jakiemu ma ono służyć.

Istnieją dwie postaci języka SDL. Jedna z nich, to postać graficzna SDL-GR, /od ang. graphical form/. Posługuje się ona zestawem symboli przedstawionym na rys. 4.1. Druga postać - to postać alfanumeryczna SD-PR /od ang. programme-like form/, której elementami są znaki alfanumeryczne.

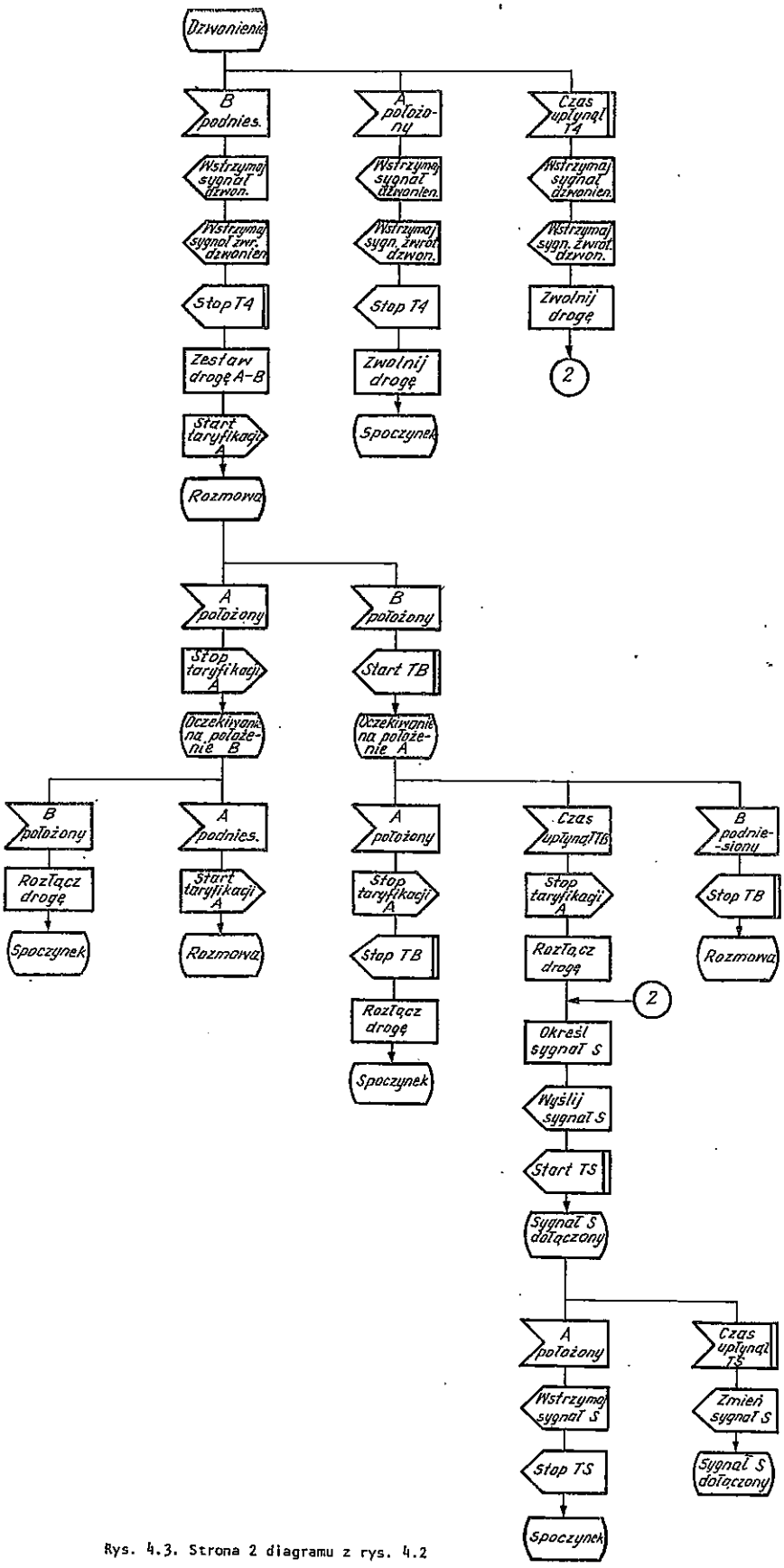
Właściwości języka SDL-GR przedstawimy posługując się diagramami na rys. 4.2 i 4.3, gdzie znajduje się opis zestawiania lokalnego połączenia telefonicznego. Opisany proces może znajdować się w jednym ze stanów np. "spoczynek", "dzwonienie", "oczekiwanie na pierwszą cyfrę". Przejście do innego stanu /w szczególnym przypadku do tego samego stanu/ może nastąpić w wyniku nadejścia sygnału wejściowego zewnętrznego, tj. pochodzącego od procesu znajdującego się w innym bloku funkcjonalnym /np. "cyfra", "A położony"/ lub wewnętrznego, tj. pochodzącego od procesu w tym samym bloku funkcjonalnym /np. "czas upłynął TØ"/. Sygnały mogą być zarówno typu sprzętowego /sygnał elektryczny/, jak i programowego /informacja w pamięci czy w rejestrze/. W trakcie przechodzenia procesu do nowego stanu mogą być wykonywane zadania /np. "odłącz odbiornik cyfr"/, mogą być wysyłane sygnały wyjściowe zewnętrzne, tj. skierowane do innego bloku funkcjonalnego /np. "wyślij sygnał dzwonienia do B"/ i wewnętrzne /np. "stop TØ" - zatrzymuj odliczanie czasu licznikiem TØ/. W trakcie przejścia mogą być również podejmowane decyzje, na podstawie których następuje wybór



Rys. 4.1. Zestaw symboli języka SDL-GR

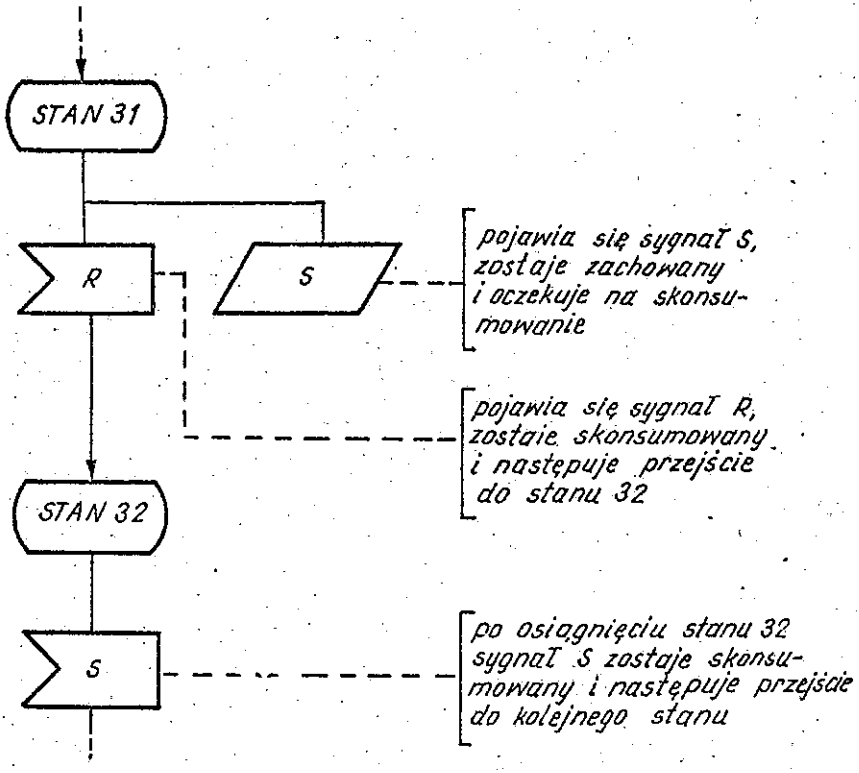


Rys. 4.2. Przykład zastosowania języka SDL-GR do opisu procesu zestawiania lokalnego połączenia telefonicznego. Strona 1 opisu



Rys. 4.3. Strona 2 diagramu z rys. 4.2

jednej z kilku ścieżek /np. decyzja "analiza cyfry", z której jest 5 wyjść/. Na rys. 4.4 pokazano zastosowanie symbolu zachowania /ang. save/, który nie był użyty w poprzednio omawianym przykładzie. Z symbolu tego nie ma wyj-

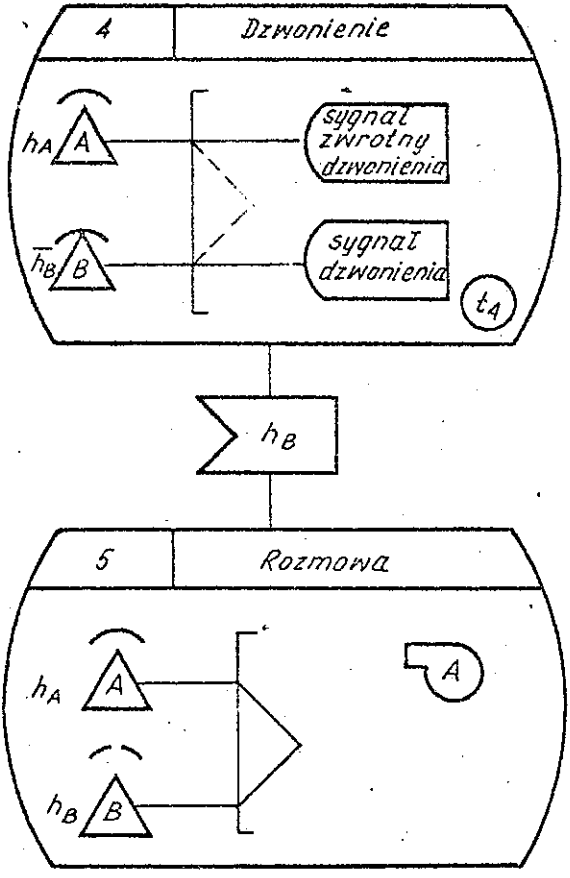


Rys. 4.4. Przykład ilustrujący użycie symbolu zachowania

ścia, a oznacza on, że sygnał wejściowy o nazwie umieszczonej wewnątrz symbolu zostaje dostrzeżony i zapamiętany, ale nie powoduje rozpoczęcia przebiegu przechodzenia do nowego stanu. Zapamiętany sygnał może być skonsumowany później, gdy proces napotka na swej drodze symbol sygnału wejściowego o tej samej nazwie. Zastosowanie symbolu zachowanie umożliwia rozwiązanie pewnych problemów związanych z opisywaniem sytuacji, gdy sygnały pojawiają się w nieznanej kolejności. Symbolu tego używa się również do opisu kolejek.

Oprócz przedstawionego poprzednio prostego symbolu stanu wprowadzony został /na razie tylko dla telefonii/ symbol stanu zawierający elementy pik-

tograficzne. Na rys. 4.5 przedstawiono fragment diagramu opisującego, ten sam co na rys. 4.2 i 4.3, proces obsługi połączenia lokalnego, ale z zastosowaniem elementów piktograficznych. W języku potocznym przedstawiony



Rys. 4.5. Odpowiednik fragmentu diagramu z rys.4.3 przedstawiony z użyciem symboli piktograficznych

na rys. 4.5 stan "dzwonienie" można opisać następująco. Numer stanu 4, nazwa stanu "dzwonienie". Abonent A o podniesionym mikrotelefonie dołączony jest do źródła sygnału zwrotnego dzwonienia. Abonent B o położonym mikrotelefonie dołączony jest do źródła sygnału dzwonienia. W polu komutacyjnym zarezerwowana jest droga połączeniowa pomiędzy A i B /linia przerywana/. Odmierzany jest czas t_4. W przykładzie pokazano tylko niektóre z istniejących elementów piktograficznych. Proponujemy czytelnikowi od-

gadnięcie opisu stanu "rozmowa" w tym przykładzie. Dla ułatwienia dodamy, że opłatę za rozmowę ponosi abonent A.

Opisanie stanów z użyciem elementów piktograficznych upraszcza sam diagram i powoduje jego lepszą czytelność. Jednak stosowanie tego sposobu jest ograniczone do tych tylko diagramów, które opisują procesy zestawiania połączeń i to na wyższym poziomie abstrakcji, a więc w praktycznej pracy z językiem SDL są to przypadki nieliczne.

Dokumentacja w języku SDL obejmuje następujące dokumenty podstawowe:

- diagramy procesów /opisane powyżej/;
- diagram dekompozycji, na którym jest podział systemu na bloki funkcjonalne na różnych poziomach abstrakcji;
- diagramy współdziałania bloków funkcjonalnych, gdzie pokazane są bloki funkcjonalne i sygnały przesyłane między nimi osobno dla każdego poziomu abstrakcji;
- listy sygnałów, zawierające wszystkie sygnały.

Oprócz tych dokumentów mogą być tworzone następujące dokumenty uzupełniające:

- diagram przeglądu stanów, który uwidacznia tylko stany i przejścia pomiędzy nimi;
- macierz sygnał-stan zawierającą informacje o tym, w jaki sposób proces reaguje na każdy z zespołu sygnałów, w zależności od stanu w jakim aktualnie się znajduje.

Zalecenia CCITT zawierają informacje o tym, jak te dokumenty powinny być tworzone.

Każdy proces, którego opis może być przedstawiony w języku SDL-GR, może być opisany również w języku alfanumerycznym SDL-PR. Przedstawiony na rys. 4.4 opis w języku SDL-PR jest odpowiednikiem diagramu umieszczonego na rys. 4.2 i 4.3. Dla oszczędności miejsca usunięto z opisu część środkową ełny opis zawiera 152 wiersze tekstu/. Opis bloku funkcjonalnego o nazwie OBSLUGAPOLACZENIA ograniczony jest słowami kluczowymi BLOCK i ENDBLOCK. Blok funkcjonalny zawiera tylko jeden proces o nazwie POLACZENIELOKALNE, którego opis ograniczony jest słowami kluczowymi PROCESS i ENDPROCESS. Opis procesu składa się z szeregu opisów stanów procesu ograniczonych słowami kluczowymi STATE i ENDSTATE. Opis stanu

zawiera faktycznie opis części diagramu zawierającego stan oraz wszystkie elementy znajdujące się na ścieżkach zaczynających się na tym stanie a kończących się na innych stanach. Ponieważ żaden element nie może być opisany dwukrotnie, więc opis ścieżki kończy się słowem kluczowym NEXTSTATE, jeśli opis ścieżki osiągnął następny stan lub słowem kluczowym JOIN, jeśli następuje zbieg opisywanej ścieżki ze ścieżką już opisaną. Wszystkim miejscom gdzie następuje zbieganie się ścieżek, muszą być zatem przypisane etykiety. W omawianym przykładzie zbiegowi ścieżek przed decyzją "analiza cyfry" na rys. 4.2 przypisana została etykieta ANALIZA, zaś przed zadaniem "określ sygnał S" na rys. 4.3 przypisana została etykieta SYGNAL. Stąd w opisie na rys. 4.6 występuje instrukcja JOIN SYGNAL.

Postać alfanumeryczna języka SDL została opracowana po to, aby ułatwić wprowadzanie do maszyny cyfrowej opisów procesów za pomocą prostych urządzeń wejściowych. Zestaw dwóch postaci języka SDL umożliwia tworzenie narzędzi do wspomaganego komputerowo projektowania a także do komputerowej archiwacji i produkcji dokumentacji. Współpraca człowieka z komputerem może przedstawiać się następująco. Na podstawie posiadanych informacji i własnych pomysłów człowiek tworzy /rysuje/ diagram w języku SDL-GR. Odpowiada to procesowi tłumaczenia informacji PT1 na rys. 4.7. Następnie człowiek dokonuje ręcznie tłumaczenia postaci graficznej na postać alfanumeryczną i wprowadza ją do maszyny /proces tłumaczenia PT2/. Od tej chwili może on żądać od maszyny wyprowadzenia tego diagramu w dowolnej postaci - - graficznej lub alfanumerycznej. Do uzyskania postaci graficznej konieczny jest proces tłumaczenia PT3 wykonywany przez maszynę[x/]. Człowiek może także w sposób interakcyjny powodować modyfikacje zapisanego w maszynie diagramu, posługując się np. jego postacią alfanumeryczną. Oprócz tego maszyna na życzenie tworzy inne rodzaje dokumentów przewidzianych przy pracy z językiem SDL a także udziela dodatkowych informacji, dotyczących na przykład poprawności formalnej diagramów, czy różnic występujących między diagramami. Po opracowaniu zadowalającej wersji diagramu można automatycznie przetworzyć go na program w języku CHILL /proces tłumaczenia PT4/, przy czym możliwe jest również przejście w przeciwnym kierunku /proces tłumaczenia PT5/. Proces tłumaczenia PT7 - to kompilacja, najlepiej opanowany spośród przedstawionych procesów.

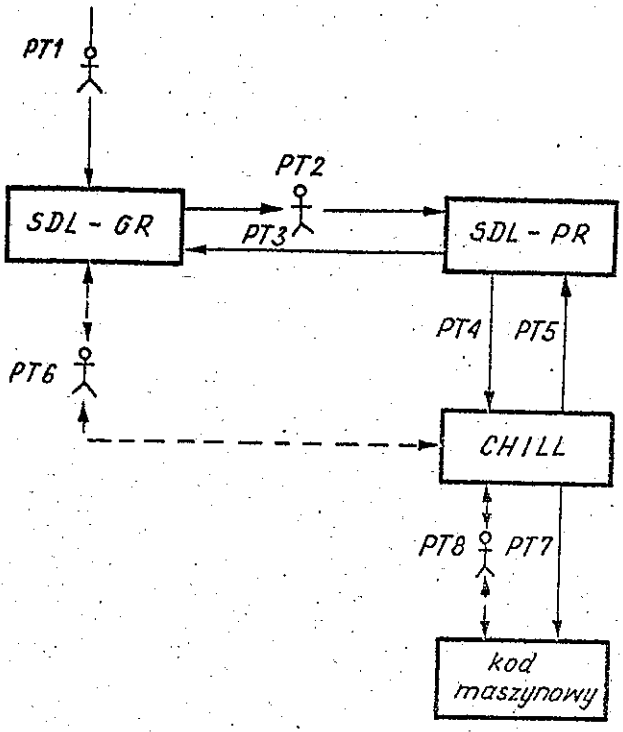
[x/] W praktycznych rozwiązaniach istnieje trzecia - wewnętrzna maszynowa postać zapisu.

```
BLOCK OBSLUGAPOLACZENIA;
PROCESS POLACZENIELOKALNE;
/* */
STATE 'SPOCZYNEK';
INPUT 'A PODNIESIONY';
   DECISION 'CZYNNY';
      (NIE): OUTPUT''NIECZYNNY';
             NEXTSTATE 'PODNIESIONY NIECZYNNY';
      (TAK): DECISION 'BLOKADA';
          (TAK): NEXTSTATE 'PODNIESIONY CZYNNY';
          (NIE): TASK 'DOLACZ ODBIORNIK CYFR';
                 OUTPUT 'WYSLIJ SYGNAL ZGLOSZENIA';
                 OUTPUT 'START T0' INTERNAL;
                 NEXTSTATE 'OCZEKIWANIE NA PIERWSZA CYFRE';
             ENDDECISION;
   ENDDECISION;
ENDSTATE 'SPOCZYNEK';
/* */
STATE 'OCZEKIWANIE NA PIERWSZA CYFRE';
INPUT 'CYFRA';
   OUTPUT 'WSTRZYMAJ SYGNAL ZGLOSZENIA';
   OUTPUT 'STOP T0' INTERNAL;
   ANALIZA:
      DECISION 'ANALIZA CYFRY';
         (NIEKOMPLETNY):
         OUTPUT 'START T1' INTERNAL;
         NEXTSTATE 'OCZEKIWANIE NA NASTEPNA CYFRE';
         (LOKALNE):
         TASK 'ODLACZ ODBIORNIK CYFR';
         DECISION 'B WOLNY';
            (NIE): JOIN SYGNAL;
            (TAK): DECISION 'BLOKADA';
                   (TAK): JOIN SYGNAL;
                   (NIE): TASK 'ZAREZERWUJ DROGE A-B';
                          OUTPUT 'WYSLIJ SYGNAL DZWONIENIA DO B';
                          OUTPUT 'WYSLIJ SYGNAL ZWROTNY DZWONIENIA DO A';
                          OUTPUT 'START T4' INTERNAL;
                          NEXTSTATE 'DZWONIENIE';
                   ENDDECISION;
         ENDDECISION;
         (NIEOBSADZONY):
         TASK 'ODLACZ ODBIORNIK CYFR';
         JOIN SYGNAL;
         (SPECJALNE):
         JOIN SF;
         (WYCHODZACE):
         JOIN EC;
      ENDDECISION;
-------------------------------------------------------------------
-------------------------------------------------------------------
/* */
ENDPROCESS POLACZENIELOKALNE;
ENDBLOCK OBSLUGAPOLACZENIA;
```

RYS.4.6. FRAGMENT ZAPISU W JEZYKU SDL-PR ROWNOWAZNEGO DIAGRAMOWI PRZEDSTAWIONEMU NA RYS.4.2. I 4.3.

```
 1   CIRCULAR_LIST:
 2   MODULE
 3      HANDLE_LIST:
 4      MODULE
 5         GRANT INSERT, REMOVE, NODE;
 6         NEWMODE NODE=STRUCT(PRED, SUC REF NODE, VALUE INT);
 7         DCL POOL ARRAY(1:1000)NODE;
 8         DCL HEAD NODE:=(: NULL,NULL,0 :);
 9         INSERT:
10         PROC(NEW NODE);
11            /* POMINIETE INSTRUKCJE DOLACZANIA */
12         END INSERT;
13         REMOVE:
14         PROC();
15            /* POMINIETE INSTRUKCJE USUWANIA */
16         END REMOVE;
17         INITIALIZE_LIST:
18         BEGIN
19            DCL LAST REF NODE:= ->HEAD;
20            DO FOR NEW IN POOL;
21               NEW.PRED ,:= LAST;
22               LAST->.SUC:= ->NEW;
23               LAST:= ->NEW;
24               NEW.VALUE,=0;
25            OD;
26            HEAD.PRED:=LAST;
27            LAST->.SUC:= ->HEAD;
28         END INITIALIZE_LIST;
29      END HANDLE_LIST;
30      DCL NODE_A NODE:=(: NULL,NULL,536 :);
31      REMOVE();
32      REMOVE();
33      INSERT (NODE_A);
34   END CIRCULAR_LIST;
```

RYS. 5.1. PRZYKLAD PROGRAMU W JEZYKU CHILL
NUMERACJA WIERSZY NIE JEST CZESCIA PROGRAMU



Rys. 4.7. Procesy tłumaczenia informacji przy stosowaniu dwóch postaci języka SDL

Jest to przykład najdalej idącej automatyzacji. O tworzeniu takich systemów informujemy w punkcie 7. Możliwe są różnorodne wersje pośrednie a także całkowicie ręczna praca z zastosowaniem języków SDL i CHILL, zawierająca dwukierunkowe tłumaczenia PT6 i PT8 oznaczone na rys. 3.3 liniami przerywanymi.

5. CHARAKTERYSTYKA JĘZYKA CHILL

CHILL jest językiem programowania wysokiego poziomu. Jest on językiem znacznie uboższym od takich języków uniwersalnych jak PL/1 czy ALGOL68, ale znacznie bogatszym od języków PASCAL czy LPA[x/], przy czym poziom CHILLa jest zbliżony do poziomu pozostałych wymienionych języków. Poziom

x/ Język programowania wysokiego poziomu o nazwie LPA jest używany do tworzenia oprogramowania centrów eksploatacji technicznej w systemie komutacji telefonicznej E10 [7].

taki został przyjęty świadomie, gdyż jest to najwyższy poziom, przy którym język jest jeszcze uniwersalny, a więc nadaje się do programowania wszystkich istniejących i przewidywanych w przyszłości systemów komutacyjnych. Dzięki temu zarazem, możliwości jego zastosowania nie ograniczają się do budowy programów komutacyjnych, a może on być z powodzeniem stosowany do innych celów, np. do programowania systemów operacyjnych.

Przed przystąpieniem do projektowania języka CHILL przeanalizowano specyficzne cechy programowo sterowanych urządzeń komutacyjnych. Poniżej omówiono te cechy, które miały największy wpływ na wybór właściwości języka.

1. Oprogramowanie systemów SPC jest duże i skomplikowane. Zatem język powinien umożliwiać łatwą segmentację programów, zapewnić możliwość opisu styków między segmentami, a także powinien umożliwiać budowę systemu o strukturze hierarchicznej.

2. Oprogramowanie systemów SPC zawiera wiele różnorodnych danych. Z tego względu język powinien umożliwiać opisywanie złożonych struktur danych i posiadać wiele mechanizmów dostępu do takich struktur.

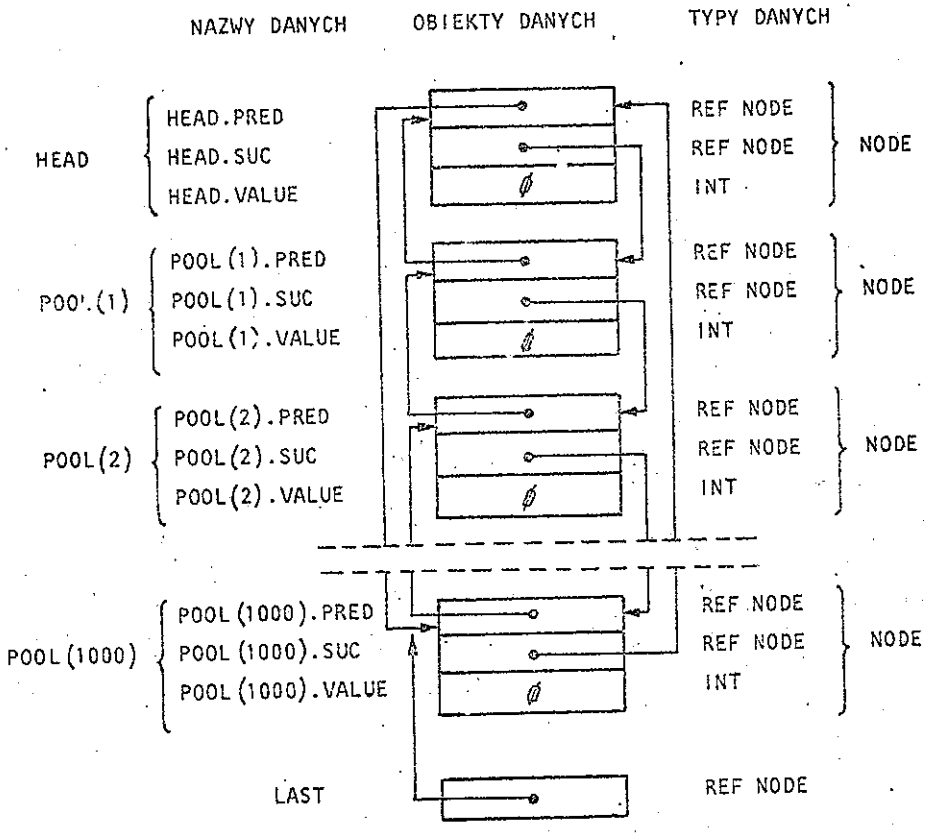
3. System SPC jest systemem czasu rzeczywistego, w którym istnieje wielka równoległość procesów. Dlatego język powinien ułatwiać programowanie procesów współbieżnych.

4. System SPC musi być szczególnie niezawodny. Dlatego język powinien być tak zaprojektowany, aby umożliwiać wszechstronne badanie poprawności programów już w czasie kompilacji.

5. System musi pracować efektywnie. Dlatego właściwości języka powinny być tak dobrane, aby ułatwić osiągnięcie efektywności generowanego kodu.

Niektóre właściwości języka CHILL przedstawiamy posługując się przykładowym programem /rys. 5.1/, zaczerpniętym z podręcznika tego języka [8]. Program ten związany jest z tworzeniem i modyfikowaniem dwukierunkowej listy cyklicznej - obiektu często spotykanego w oprogramowaniu komutacyjnym. Struktura dwukierunkowej listy cyklicznej tworzonej przez ten program przedstawiona jest na rys. 5.2. Pierwsze słowo każdego elementu listy wskazuje element poprzedni, drugie słowo wskazuje następny element listy, trzecie słowo zawiera daną przypisaną temu elementowi.

W programie tym, tak jak w każdym programie w języku CHILL, wyróżnić można trzy następujące części:

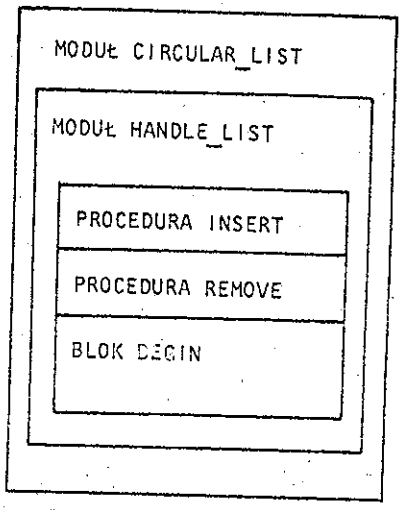


Rys. 5.2. Zespół danych opisanych programem z rys. 5.1

- opis danych,
- opis akcji, które przetwarzają dane,
- opis struktury programu.

Opis danych zawarty jest w wierszach 5, 6, 7, 8, 19 i 30, opis akcji w wierszach 11, 15, 20 ÷ 27 i 31 ÷ 33, a opis struktury programu w wierszach 1, 2, 3, 4, 9, 10, 12, 13, 14, 16, 17, 18, 28, 29 i 34. Dla uproszczenia pominięte zostały wszystkie instrukcje zawarte wewnątrz procedur INSERT i REMOVE.

Na rys. 5.3 przedstawiono strukturę programu tworzoną przez opis struk-



Rys. 5.3. Struktura programu przedstawionego na rys. 5.1

tury. Występują tu trzy spośród czterech istniejących w języku CHILL konstrukcji, którymi są: moduł, blok typu begin, procedura i proces. Przez stosowanie tych konstrukcji można uzyskać takie korzystne właściwości oprogramowania, jak:

- modularność - ułatwiającą konstruowanie i utrzymanie oprogramowania,
- czytelność i samodokumentacyjność oprogramowania,
- możliwość wpływania na rozmieszczenie programów w pamięci,
- izolowanie od siebie części programu i kontrolę komunikacji pomiędzy częściami.

Tak więc w przykładzie cała wewnętrzna budowa listy stworzonej w module HANDLE_LIST nie jest widoczna /dostępna/ spoza tego modułu. Jedyny możliwy sposób modyfikacji listy to użycie procedur o nazwach INSERT /dołącz/ i REMOVE /usuń/. Widoczność nazw tych procedur na zewnątrz modułu została uzyskana przez zastosowanie instrukcji nadania GRANT umieszczonej w wierszu 5. Nazwy HEAD i POOL nie są widoczne na zewnątrz modułu, zaś nazwy NEW i LAST nie są nawet widoczne na zewnątrz bloku begin o nazwie INITALIZE_LIST.

W języku CHILL każdy obiekt danych, nad którym program przeprowadza operacje musi być zadeklarowany. W deklaracji określa się m.in. typ obiektu /MODE/. W CHILLu istnieją takie typy, jak np.: BOOL - zmienna logiczna, CHAR - zmienna znakowa, INT - zmienna całkowita, ARRAY - tablica.

W wierszu 7 zadeklarowana jest właśnie tablica o nazwie POOL i elementach ponumerowanych od 1 do 1000. Typ każdego elementu jest NODE. Rzadziej spotykanym w innych językach programowania typem jest struktura /STRUCT/. Struktura składa się z obiektów /pól/, których typy mogą być różne, w przeciwieństwie do tablicy, która składa się z obiektów tego samego typu. W przykładzie występują np. struktury o nazwach HEAD, POOL/1/, POOL/2/ itd. Dostęp do pola struktury jest realizowany przez napisanie nazwy struktury, kropki i nazwy pola, np. NEW.PRED, HEAD.PRED.

Język CHILL umożliwia użytkownikowi tworzenie własnych typów danych. W wierszu 6 zdefiniowany jest nowy typ /NEWMODE/ o nazwie NODE, będący strukturą /STRUCT/ o trzech polach. Dwa z nich, to pola o nazwach PRED i SUC, które są typu odnośnik do obiektu NODE /REF NODE/, trzecie to pole o nazwie VALUE typu INT.

Korzyści wynikające z możliwości tworzenia własnych typów staną się oczywiste, gdy zwrócimy uwagę na to, że kompilator zawsze sprawdza, czy operacja przeprowadzana przez program nad obiektem danych jest dopuszczalna, to znaczy czy użyto obiektu właściwego typu.

W opisywanym przykładzie występują zmienne typu odnośnik, które służą do wskazywania obiektów danych /zawierają lokację tych obiektów/. W deklaracjach tych zmiennych umieszczone jest słowo REF. I tak, każde pole struktury o nazwie PRED i każde pole struktury o nazwie SUC wskazuje /o ile nie jest puste - wówczas zawiera NULL/ pewien element danych typu NODE. Zapisano to w deklaracji w wierszu 6 w postaci PRED, SUC REF NODE, co znaczy tyle samo co PRED REF NODE, SUC REF NODE. Inna zmienna typu odnośnik o nazwie LAST, zadeklarowana w wierszu 19 jest zmienną pomocniczą używaną tylko w procesie inicjalizacji listy i wskazuje kolejno wszystkie elementy listy /typu NODE/. Po zakończeniu inicjalizacji wskazuje element ostatni, tj. POOL/1000/. Użytkowanie zmiennych typu odnośnik możliwe jest dzięki istnieniu symbolu →. I tak, jeśli A jest obiektem danych, to → A oznacza jego lokację. Jeśli B jest zmienną typu odnośnik, to B → jest obiektem danych wskazywanym przez tę zmienną.

Po omówieniu opisu struktury i opisu danych przykładowego programu omówimy opis akcji tego programu. Rozpatrzmy akcje w bloku begin o nazwie INITALIZE_LIST, gdzie inicjalizowana jest dwukierunkowa lista cykliczna przez stworzenie odpowiednich odniesień i wyzerowanie pól VALUE. Obraz tej listy bezpośrednio po inicjalizacji przedstawia rys. 5.2. Wiersze

20 i 25 zawierające słowa kluczowe DO i OD określają pętlę, dzięki czemu ciąg instrukcji w wierszach od 21 do 24 wykonywany jest wielokrotnie dla wszystkich elementów tablicy POOL /IN POOL/. Bieżącemu elementowi tablicy POOL przypisywana jest nazwa NEW /FOR NEW/. Po kompletnym wykonaniu pętli DO ustawione są wszystkie, z wyjątkiem dwóch, wartości zmiennych typu odnośnik. Zamknięcie listy w obu kierunkach następuje po wykonaniu instrukcji w wierszach 26 i 27. Instrukcja w wierszu 26 zamyka listę w kierunku wstecz, zaś w wierszu 27 w kierunku w przód. Po inicjalizacji lista może być użytkowana przez program. Przykłady modyfikowania tej listy to instrukcje 31 do 33, które usuwają dwa elementy tej listy /REMOVE/ i dołączają jeden, zadeklarowany w wierszu 30 element /INSERT/.

Proponujemy czytelnikom, aby na podstawie powyższych informacji prześledzili te szczegóły przedstawionego programu przykładowego, które nie zostały omówione w tekście.

Spośród nie pokazanych w przykładzie właściwości języka CHILL trzeba wymienić takie, jak:

- mechanizmy służące do programowania procesów współbieżnych,
- rozbudowane struktury sterujące takie, jak: IF... THEN ... ELSE ...FI, CASE ... OF ... ESAC, DO FOR ... OD, DO WHILE ... OD umożliwiające programowanie strukturalne, przy czym szczególnie wygodna jest instrukcja CASE pozwalająca na łatwe zapisanie tablicy decyzyjnej.

Reasumując, można stwierdzić, że właściwości języka CHILL czynią go wygodnym narzędziem służącym do konstruowania oprogramowania związanego z urządzeniami telekomutacyjnymi.

6. CHARAKTERYSTYKA JĘZYKA MML

W nowoczesnych systemach komutacyjnych ze sterowaniem programowanym główna część strumienia informacji przekazywanych pomiędzy urządzeniami a obsługującym je personelem ma postać ciągów znaków alfanumerycznych, które w kierunku do maszyny wysyłane są z klawiatury, a w kierunku do człowieka z ekranu monitora lub z urządzenia drukującego. Istniejące dawniej możliwości komunikowania się z urządzeniami przez obserwację, np. ruchomych jego elementów, czy nawet za pomocą słuchu są obecnie praktycznie niemożliwe. Również ręczna ingerencja personelu w pracę urządzeń jest

dzisiaj bardzo ograniczona. Z tych względów stało się bardzo pożądane opracowanie standardowego języka komunikacji człowiek-maszyna dla potrzeb telekomutacyjnych.

Opracowany w CCITT język MML przeznaczony jest do prowadzenia dialogu pomiędzy personelem a urządzeniami sterowanych programowo systemów komutacyjnych wszelkich typów i różnych zastosowań /telefonicznych, telegraficznych, teledacyjnych itp./. Uwzględniono dialog związany z wykonywaniem przez urządzenia następujących grup funkcji: funkcji użytkowania, funkcji utrzymania, funkcji związanych z instalowaniem urządzeń i związanych z ich testowaniem. Dialog może być prowadzony z poszczególnymi urządzeniami komutacyjnymi /centralami/ oraz z centrami eksploatacji dowolnego poziomu poprzez terminale zarówno lokalne, jak i zdalne. Język nie wprowadza w zasadzie ograniczeń na typ użytego terminala. Może to być zarówno prosty dalekopis, jak i każde inne urządzenie umożliwiające wprowadzanie i wyprowadzanie znaków alfabetu międzynarodowego nr 5 /np. alfanumeryczny monitor ekranowy z klawiaturą/.

Język MML w postaci przedstawionej w zaleceniach jest bardzo bogaty i dlatego normalne implementacje są podzbiorami tego języka. Przewiduje się określenie podstawowego podzbioru, który powinien być zawarty w każdej implementacji. Jednym z podzbiorów /z niewielkimi modyfikacjami/ języka MML jest, opracowany i wdrożony w Instytucie Łączności, język MMLS [9,10]. Posłużymy się przedstawionym na rys. 6.1 przykładem dialogu w języku MMLS dla opisu jego właściwości. Dialog zawiera trzy części: prolog /wiersze 1 ÷ 3/, sekwencję wprowadzania poleceń /wiersze 4 ÷ 21/ i epilog /wiersze 22 ÷ 24/. Dialog jest rozpoczynany przez operatora żądaniem dialogu, które polega na nadaniu znaku CTRL-A. Maszyna odpowiada znakiem <. W prologu operator wprowadza hasło, które go identyfikuje. Z hasłem związane są uprawnienia operatora do wprowadzania określonych poleceń. Znaki hasła wprowadzanego przez operatora nie są wyświetlane na ekranie - każdy znak hasła zastępowany jest znakiem # . Po rozpoznaniu hasła maszyna wyświetla datę i czas /wiersz 2/ oraz komunikat o przyjęciu operatora /wiersz 3/. Epilog zawiera wprowadzone przez operatora żądanie zakończenia dialogu /wiersz 22/, po którym maszyna wyświetla datę i czas /wiersz 23/ oraz komunikat o zakończeniu dialogu /wiersz 24/ i przechodzi w stan oczekiwania na następne żądanie dialogu.

W przykładzie operator wprowadził trzy polecenia. Rozpoczynające się

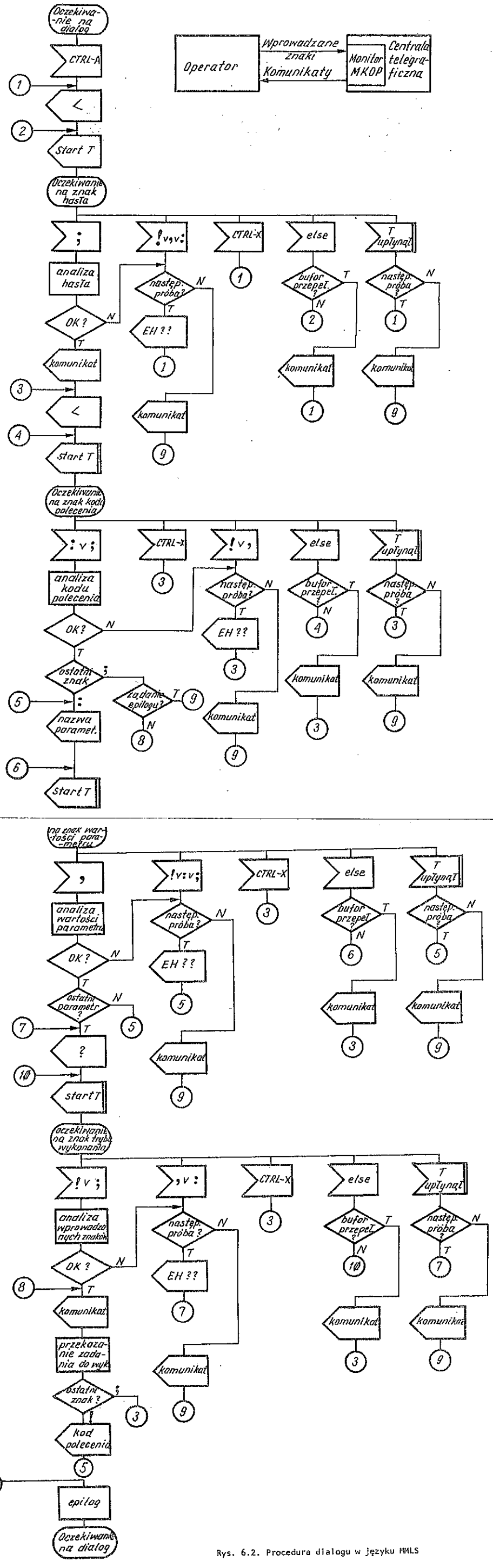
w wierszu 4 polecenie o kodzie W-LIST-SKR /wprowadź listę numerów skróconych/, w wierszu 10 to samo polecenie, ale z innymi wartościami parametrów i w wierszu 15 polecenie U-AB /usuń abonenta/. Wprowadzanie polecenia rozpoczyna się wprowadzeniem przez operatora kodu polecenia. Jeśli kod polecenia nie określa jednoznacznie zadania, jakie ma być wykonane przez maszynę, to muszą być wprowadzone parametry. W języku MMLS przyjęto, że nazwy parametrów są podpowiadane przez maszynę. Z poleceniem W-LIST-SKR związane są dwa parametry: NR KATALOGOWY i NR KAT-NR SKR, natomiast z poleceniem U-AB parametr NR KATALOGOWY. Po przyjęciu polecenia wraz z parametrami maszyna nadaje mu kolejny numer /modulo 10.000/ i wyświetla komunikat o przyjęciu polecenia /wiersze 6 i 7, 12 i 13, 16 i 17/. Przyjęcie polecenia świadczy o tym, że było ono z punktu widzenia formalnego poprawne. Po wykonaniu polecenia maszyna wyświetla komunikat o wykonaniu polecenia /wiersze 8 i 9, 18 i 19, 20 i 21/. Jeśli polecenie nie może być z jakichkolwiek powodów wykonane, to zostaje wyświetlony komunikat o niewykonaniu polecenia zawierający przyczynę niewykonania. Ponieważ wykonywanie polecenia może trwać dłuższy czas /w pewnych przypadkach nawet kilka godzin/, więc komunikat o wykonaniu polecenia może nie wystąpić bezpośrednio po komunikacie o jego przyjęciu. Dla identyfikacji polecenia, którego komunikat dotyczy służy w tym przypadku numer i nazwa polecenia.

Istnieją dwa tryby wprowadzania poleceń: tryb bez powtórzenia i tryb z powtórzeniem. Tryb z powtórzeniem realizowany jest przez maszynę na żądanie operatora wyrażone znakiem ! po żądaniu wprowadzenia trybu wykonania wyrażonym przez maszynę znakiem ? /wiersz 5/. W trybie z powtórzeniem po przyjęciu bieżącego polecenia maszyna przystępuje do przyjmowania polecenia o tej samej nazwie /wiersz 10/. W trybie bez powtórzenia wyrażonym znakiem ; /wiersz 11 i 15/ maszyna oczekiwać będzie na wprowadzenie nowego polecenia.

Jeśli maszyna wykryje błąd operatora, to wyświetla żądanie korekcji w postaci EH??, a operator ma możliwość powtórnego wprowadzenia ciągu znaków. I tak w wierszu 4 operator wprowadził nieprawidłową wartość parametru /zawiera literę/, w wierszu 10 zamiast przecinka, jako znaku rozdzielającego parametry, wprowadził średnik i w wierszu 15 zamiast znaku trybu wykonania, którym może być wykrzyknik lub średnik, wprowadził błędnie najpierw dwukropek, a następnie znaki WYK;. Operator ma również moż-

```
1    < #######
2      1981-05-22          20.17.53
3      OPERATOR PRZYJETY
4    < W-LIST-SKR: NR KATALOGOWY = 749A478, EH?? NR KATALOGOWY = 58587475,
5    NR KAT-NR SKR = 489365-H,?  !
6      1981-05-22          20.18.14
7      0006     W-LIST-SKR      PRZYJETE
8      1981-05-22          20.19.58
9      0006     W-LIST-SKR      WYKONANE
10   < W-LIST-SKR: NR KATALOGOWY = 125769; EH?? NR KATALOGOWY = 125769,
11   NR KAT-NR SKR = 42094-W,?  ;
12     1981-05-22          20.20.47
13     0007     W-LIST-SKR      PRZYJETE
14   < U-BA: EH??
15   < U-AB: NR KATALOGOWY = 852390645,?  : EH?? ? WYK; EH?? ?  ;
16     1981-05-22          20.21.32
17     0008     U-AB            PRZYJETE
18     1981-05-22          20.25.37
19     0007     W-LIST-SKR      WYKONANE
20     1981-05-22          20.27.04
21     0008     U-AB            WYKONANE
22   < FI;
23     1981-05-22          20.27.14
24     KONIEC DIALOGU
```

RYS.6.1. PRZYKLAD DIALOGU W JEZYKU MMLS
NUMERACJA WIERSZY I PODKRESLENIA NIE STANOWIA CZESCI DIALOGU. PODKRESLONO TE ZNAKI, KTORE SA WYSWIETLANE PRZEZ MASZYNE



Rys. 6.2. Procedura dialogu w języku MMLS

liwość korygowania swoich błędów zanim zostaną one wykryte przez maszynę. Nie jest to pokazane w przykładzie.

Opis procedury dialogu prowadzonego w języku MMLS przedstawiony jest na rys. 6.2. Sam opis wykonany jest w języku SDL i może być traktowany jako przykład zastosowania tego języka.

W językach MML i MMLS istnieją dość różnorodne możliwości wyrażania wartości parametrów. /Wartość parametru reprezentowana jest przez ciąg znaków umieszczonych po prawej stronie znaku równości za nazwę parametru/. Najmniejszym elementem wartości parametru jest jednostka informacji, którą może być na przykład liczba lub identyfikator. Prosty argument parametru jest zbudowany z jednej jednostki informacji. Z kilku jednostek informacji buduje się złożone argumenty parametru. Złożony z trzech jednostek informacji argument parametru może mieć na przykład postać 6-12-81 i może oznaczać datę. Zarówno argumenty proste, jak i argumenty złożone mogą być grupowane tworząc w ten sposób wartości parametru. Grupa argumentów prostych może mieć postać: 3&7&12. Jest to skrócony zapis grupy zawierającej argumenty 3, 7, 8...11, 12. Grupy argumentów złożonych mogą mieć, np. postaci:

23-12&-14&-16
23-12&&-59
120-7&123-7&12-8

Pierwszy przykład, to skrócony zapis trzech argumentów złożonych: 23-12, 23-14 i 23-16. Drugi przykład, to skrocony zapis argumentów 23-12, 23-13,...23-58 i 23-59, zaś trzeci przykład to trzy następujące argumenty złożone: 120-7, 123-7 i 12-8.

Ten pozornie skomplikowany sposób przedstawiania wartości parametru okazuje się w wielu przypadkach bardzo wygodny. Jeśli na przykład dla określenia obiektu /np. łącza/ nie wystarcza jedna jednostka informacji, to może być użyty argument złożony. W przypadkach, gdy polecenie ma być wykonane dla pewnego zbioru obiektów, np. grupy abonentów, może być używane grupowanie argumentów, dzięki czemu unikamy konieczności wielokrotnego powtarzania tego samego polecenia.

Język MMLS jest dość ograniczonym podzbiorem języka MML. Spośród właściwości języka MML, które nie zostały wykorzystane w języku MMLS, można wymienić np. formatowany sposób pracy. Różni się on od przedstawionego w

przykładzie tym, że maszyna nie zadaje kolejnych pytań, na które odpowiada operator, ale wyświetla formularz, który operator wypełnia w dowolnej kolejności. Sposób ten, nieco wygodniejszy dla operatora, może być jednak stosowany tylko wtedy, gdy urządzeniem wejścia/wyjścia jest monitor ekranowy. Inny jeszcze sposób pracy przewidziany w języku MML to sposób pracy z wyborem /menu mode/. Polega on na tym, że maszyna przedstawia operatorowi zestaw możliwości /menu/. Zadaniem operatora jest tylko dokonanie wyboru. Ten sposób pracy wygodny jest dla słabo wyszkolonych operatorów, dlatego w języku MML przewiduje się wybór tego trybu pracy na życzenie operatora.

Zbiór nazw poleceń w języku MMLS został określony dla konkretnego zastosowania, jakim jest centrala telegraficzno-teleinformatyczna ECTT i zawiera 69 kodów poleceń. W CCITT trwają prace nad określeniem zbioru poleceń dla wszystkich możliwych zastosowań. W tym celu dąży się do określenia zbioru tych funkcji realizowanych przez urządzenia telekomutacyjne, do których powinien istnieć dostęp poprzez dialog w języku MML. Następnie w wyniku analizy informacji potrzebnych do wykonania każdej funkcji określona będzie postać informacji wejściowych i informacji wyjściowych. W prace te zaangażowane są, oprócz Komisji XI, następujące Komisje CCITT: II-eksploatacja i jakość usługowa w telefonii, IV - transmisja i utrzymanie sieci, VII - publiczna sieć danych i XVIII - zasady utrzymania sieci zintegrowanej.

7. WDROŻENIA JĘZYKÓW PROGRAMOWANEGO STEROWANIA CCITT

Pierwsze wdrożenia języków CCITT pojawiły się przed zakończeniem prac nad definiowaniem tych języków, w momencie gdy stan ich zaawansowania już na to pozwalał. Przewidywano bowiem, że próby wdrożeń wytworzą korzystne sprzężenia zwrotne do zespołów opracowywujących języki i sytuacja taka rzeczywiście wystąpiła. Wykryto wiele usterek i niejednoznaczności w zaleceniach. Stwierdzono konieczność bardzo precyzyjnego definiowania języków. W związku z tym prowadzone są prace nad sformalizowaną /matematyczną/ definicją języka CHILL. Także dla SDLa zostanie stworzony formalny /matematyczny/ model. Modele formalne okazały się niezbędne dla konstruktorów narzędzi takich, jak: translatory czy komputerowe systemy archiwacji dokumentów. Równolegle istniejące modele nieformalne są łatwiej przy-

swajalne przez użytkowników języków.

Spośród trzech omawianych języków najłatwiejszy do szybkiego początkowego wdrożenia jest język SDL, gdyż nie są tu potrzebne żadne inwestycje ani narzędzia /poza może prostym szablonem do rysowania symboli/. Niemniej jednak, aby dobrze wykorzystać zawarte w tym języku możliwości buduje się wyrafinowane narzędzia komputerowe.

Prawdopodobnie najwcześniejsze zastosowania języka SDL pojawiły się w samym CCITT. Diagramy w SDLu spotykamy na przykład w zaleceniach dotyczących systemów sygnalizacji. Coraz częściej schematy rysowane w SDLu zobaczyć można w czasopismach o tematyce telekomunikacyjnej.

Wprowadzony w Zakładzie Teleinformatyki Instytutu Łączności do opisów dialogu człowiek-maszyna /por. rys. 6.2/, a następnie stosowany przy projektowaniu oprogramowania teledacyjnego w centrali telegraficzno-teleinformatycznej ECTT, język SDL okazał się skutecznym i wygodnym narzędziem. Po przełamaniu pewnych początkowych oporów pracowników przyzwyczajonych do innych metod opisu wytworzyła się taka sytuacja, że pracownicy próbują go stosować z powodzeniem w innych dość odległych zastosowaniach, np. do tworzenia harmonogramów prac.

W jednym z dokumentów CCITT ze stycznia 1981 r. [11] znajduje się zestawienie informacji o aktualnych i planowanych sposobach użytkowania języka SDL przez różne instytucje. Z zestawienia wynika, że SDL jest stosowany w co najmniej 17 instytucjach, w tym w takich firmach, jak Siemens, Philips czy Ericsson. W instytucjach włoskich CSELT, ITALTEL i SIP oraz w holenderskim PTT język SDL został przyjęty jako standard. W firmie Bell Laboratories istniejący tam wspomagany komputerowo system projektowania zostanie zmodyfikowany w ten sposób, aby jego językiem wejściowym był pewien podzbiór CHILLa /nie SDL-PR!/, a językami wyjściowymi SDL-GR dla diagramów stanów i przejść, a CHILL dla modułów programowych. Także inne organizacje informują o tworzeniu podobnych systemów.

Użytkownicy wykryli wiele niedoskonałości języka, które powinny zostać usunięte. Wymienić tu można: brak koncepcji i reguł dekompozycji opisywanego systemu, brak możliwości przedstawiania danych /niedoskonałość zaobserwowana także,w It/, brak odpowiednika makroinstrukcji. Słusznie także postuluje się, aby język SDL-PR był podzbiorem języka CHILL. Można przypuszczać, że po usunięciu niedoskonałości języka SDL, a także po wprowadzeniu formalnego modelu znikną ostatnie przeszkody na drodze do jego szerokiego rozpowszechniania.

Wdrożenie dowolnego języka programowania wysokiego poziomu, a więc także CHILLa, może być wykonywane w dwóch etapach. Etap pierwszy może zawierać stosowanie języka do opisu algorytmów w sytuacji, gdy nie posiadamy translatora. Translacja przeprowadzana jest wówczas ręcznie lub też język służy wyłącznie jako język dokumentacyjny. W etapie drugim rozpoczyna się stosowanie translatora ewentualnie wcześniej opracowanego. Na takie dwuetapowe wdrożenie zdecydowano się właśnie w Instytucie Łączności. Taką drogą postępuje się również w węgierskim instytucie łączności PKI.

Zasadniczą przeszkodą we wdrożeniach języków wysokiego poziomu była do niedawna znaczna pracochłonność opracowania translatora. Raz opracowany translator dawał program wynikowy dla jednej tylko maszyny i mógł pracować tylko na jednej maszynie. Postęp w dziedzinie konstruowania translatorów, jaki dokonał się ostatnio na świecie, wyeliminował wiele trudności. Istniejące obecnie generatory translatorów /metatranslatory/ są w stanie wytworzyć automatycznie translator dla dowolnego języka źródłowego i dowolnego języka wynikowego. Oczywiście generator taki wymaga, aby przedstawiony mu został odpowiednio sformalizowany opis języków źródłowego i wynikowego. Opis taki dla języka PASCAL-a na przykład mogą wykonać 3 osoby w ciągu ok. miesiąca. Jeśli porównamy pracę automatycznie wygenerowanego translatora, z translatorem opracowanym ręcznie przez wykwalifikowany zespół, to okaże się, że translator wytworzony automatycznie realizuje lepiej analizę składniową, nieco gorzej analizę semantyczną, zaś generator kodu jest porównywalny w obu przypadkach.

Metatranslator jest w trakcie opracowywania w Instytucie Maszyn Matematycznych w Warszawie. Można sądzić, że w niedługim czasie będzie on wykorzystany dla prac nad drugim etapem wdrożenia języka CHILL w Instytucie Łączności.

Istotna wada języków wysokiego poziomu polegała do niedawna na tym, że generowany przez translator kod wynikowy był niedostatecznie efektywny pod względem czasowym i przestrzennym. Oznacza to, że generowany kod wynikowy zajmował więcej pamięci i pracował wolniej niż kod powstający z napisanego ręcznie przez doświadczonych programistów programu w języku symbolicznym. Doświadczenia z wdrożenia języka CHILL w firmie Philips w Holandii wskazują, że translator wyposażony w prostą optymalizację kodu wynikowego produkuje kod o nadmiarze nie przekraczającym 15% zarówno w

dziedzinę czasu, jak i zajętości pamięci. Autorzy opracowania [12] twierdzą, że te dobre efekty są możliwe dzięki temu, że CHILL ma wiele właściwości, które są z natury efektywne przy wdrożeniu.

Nowe podejście do problemu języków wysokiego poziomu polega na zmniejszaniu dystansu pomiędzy językiem programu a kodem maszynowym. Osiągane to jest poprzez budowanie procesorów wyspecjalizowanych względem danego języka wysokiego poziomu. Nie eliminuje to całkowicie potrzeby translacji, ale ją znacznie upraszcza. Jako przykład służyć tu może zbudowany w CNET we Francji model sterowania centralą telefoniczną. Jest on programowany w CHILLu [13].

W ramach CCITT istnieje zespół grupujący przedstawicieli instytucji wdrażających CHILLa, tzw. "Implementors Forum". Według informacji posiadanych przez ten zespół, w maju 1981 roku /por. dokument CCITT [14], str. 133/ istniało 9 zakończonych implementacji CHILLa, a 8 dalszych było w trakcie realizacji. Ukończyły implementację tego języka firmy Siemens i Philips. W grudniu 1980 r. administracje 10 krajów zachodnioeuropejskich wyraziły pogląd, że nie są zainteresowane uzgadnianiem żadnego innego poza CHILLem języka wysokiego poziomu do zastosowań telekomunikacyjnych.

Spośród istniejących wdrożeń języka MML zwrócimy uwagę na dwa: zrealizowane w Zakładzie Teleinformatyki Instytutu Łączności i zrealizowane w firmie Philips.

Wdrożenie zrealizowane w Instytucie Łączności przeznaczone jest dla opracowywanej w Oddziale Gdańskim IŁ elektronicznej centrali telegraficzno-teleinformatycznej ECTT. Przy opracowywaniu podzbioru języka MML o nazwie MMLS /MML Simplified/ przyjęto następujące założenia:

- język ma zapewniać komunikację człowiek-maszyna dla konkretnego zastosowania, jakim jest centrala ECTT;
- język ma być możliwie prosty.

W czasie opracowywania języka okazało się, że pomimo powyższych założeń, które są założeniami mocno upraszczającymi, nie udaje się stworzyć języka będącego po prostu podzbiorem języka MML. Niezbędne okazały się pewne, niewielkie zresztą modyfikacje wybranego podzbioru.

Dalsze niewielkie modyfikacje wprowadzono dla podwyższenia czytelności zapisu w tym języku. W trakcie pracy znaleziono pewne niedoskonałości

zaleceń CCITT. Uwagi i propozycje zmian w zaleceniach zostały przekazane do CCITT w formie kontrybucji [15]. Rozstrzygnięcia wymagało też wiele kwestii wynikających z faktu, że prace nad zaleceniami opisującymi język MML nie były jeszcze zakończone. W tej sytuacji stworzony został własny sposób tworzenia kodów poleceń i nazw parametrów.

Oprogramowanie realizujące dialog w języku MMLS pracuje na maszynie SM3, której procesor steruje również pracą centrali ECTT.

Dla tej samej centrali ECTT istnieje ponadto wdrożenie języka MML opracowane przez Instytut Informatyki Politechniki Gdańskiej. Wybrany podzbiór języka jest dużo uboższy od opracowanego w Instytucie Łączności.

W literaturze istnieje dość dużo informacji o wdrożeniu które zostało zrealizowane przez firmę Philips dla systemu central telefonicznych PRX/D [16]. Wybrano tu podzbiór bogatszy niż we wdrożeniu Instytutu Łączności. Możliwy jest dialog zarówno w trybie sekwencyjnym, jak i formatowanym. Podpowiadanie w trakcie wprowadzania bloku parametrów następuje tylko na życzenie operatora.

Porównując język opracowany w Instytucie Łączności z językiem firmy Philips można stwierdzić, że występuje między nimi wiele różnic. Część z tych różnic wynika z dowolności, jaką implementatorom języków pozostawiają zalecenia /na przykład dowolność wyboru podzbioru języka/. Pozostałe różnice związane są z faktem, że w czasie opracowywania języków zalecenia były niekompletne lub niedopracowane. Przyczyna ta objawiła się szczególnie w języku firmy Philips, gdyż powstał on wcześniej.

Według informacji z maja 1981 r. /dok. CCITT [14], str. 141/ język MML został wdrożony przez administracje 8 krajów, a także przez 13 instytucji naukowych i przemysłowych w różnych krajach.

Aby bardziej rozpowszechnić języki programowanego sterowania CCITT, Komisja XI podjęła akcję propagowania tych języków. Przewiduje się inicjowanie, popieranie i prowadzenie różnego rodzaju działalności służącej temu celowi. Jest wydawany biuletyn dla użytkowników CHILLa, planuje się zorganizowanie konferencji użytkowników tego języka.

8. PROBLEMY DO ROZWIĄZANIA

Jak wspomnieliśmy już poprzednio, prace nad językami programowanego sterowania trwają w CCITT w dalszym ciągu. Na okres studiów 1981-84 w

ramach Komisji XI przewidziano trzy następujące zagadnienia dotyczące języków [17]:

- zagadnienie 7/XI - udoskonalanie i rozszerzanie zaleceń dotyczących języka SDL,
- zagadnienie 8/XI - aspekty utrzymaniowe, szkoleniowe, zgodności z zaleceniami i otoczenia języka CHILL,
- zagadnienie 9/XI - udoskonalanie i rozszerzanie zaleceń dotyczących języka MML.

W rámach zagadnienia 7/XI spodziewane jest uzyskanie odpowiedzi na takie m.in. pytania:

- jak przedstawić w języku SDL wspólne procedury, dane i inne środki wymiany informacji,
- jaka powinna być ostateczna postać języka SDL-PR,
- jak rozwiązać problemy dekompozycji i poziomów abstrakcji specyfikacji i opisów,
- jakie dodatkowe wskazówki dla użytkowników powinny być opracowane.

W zakresie zagadnienia 8/XI przewiduje się pracę nad następującymi problemami:

- nad pielęgnacją Zalecenia Z.200 i związanych z nim dokumentów,
- nad szkoleniem użytkowników języka CHILL,
- nad sposobami zabezpieczenia, aby użytkowanie CHILL'a było zgodne z Zaleceniem Z.200,
- nad otoczeniem w którym CHILL jest stosowany,
- nad lansowaniem CHILLa w tych zastosowaniach gdzie stwierdzono jego przydatność.

W ramach zagadnienia 9/XI przewiduje się dalsze studiowanie następujących problemów:

- ulepszania i uzupełniania składni języka wejściowego i wyjściowego i procedur dialogu,
- standardów dotyczących wymiany informacji pomiędzy terminalami a cen-

tralami oraz pomiędzy centralami a innymi ośrodkami, takimi jak np. CET /centrum eksploatacji technicznej/,

- zdefiniowania podstawowego podzbioru języka MML,
- opracowania poradnika implementatora i podręcznika użytkownika,
- opracowania wskazówek do tworzenia poleceń i komunikatów,
- opracowania zestawu zalecanych poleceń i komunikatów.

Sądzimy, że aktywny udział Instytutu Łączności w uzyskiwaniu odpowiedzi na powyższe pytania jest możliwy i pożądany.

Można już w tej chwili stwierdzić, że szerokie wprowadzenie w kraju języków programowanego sterowania CCITT jest pożądane. Środki służące do osiągnięcia tego celu można w skrócie przedstawić następująco:

1. Należy uczestniczyć w pracach CCITT, dzięki czemu można będzie dysponować pełnymi i aktualnymi informacjami.

2. Należy prowadzić akcję popularyzacji języków CCITT poprzez publikowanie artykułów w czasopismach telekomunikacyjnych, wygłaszanie odczytów, przekazywanie informacji na konferencjach itp.

3. Należy dbać o to, aby studenci kierunków telekomunikacyjnych wyższych uczelni byli w dostatecznym stopniu zapoznawani z właściwościami i zaletami tych języków oraz z ich użytkowaniem.

4. Należy zadbać o to, aby we wszystkich prowadzonych w Instytucie Łączności pracach związanych z urządzeniami telekomutacyjnymi języki te były w pełni wykorzystywane. W tym celu należy m.in. opracować translator /czy też rodzinę translatorów/ języka CHILL oraz prowadzić prace nad wspomaganym komputerowo systemem projektowania opartym na języku SDL.

5. Należy doprowadzić do przyjęcia języków CCITT jako standardów, początkowo w Instytucie Łączności, a następnie rozszerzyć ten obszar na wszystkie instytucje prowadzące prace w tym zakresie w kraju.

9. PODSUMOWANIE

Na podstawie prowadzonych prac oraz studiów literatury przedmiotu można dziś stwierdzić, że powstała przed kilku laty inicjatywa, aby opracować w CCITT standardowe języki programowanego sterowania, była celowa i

pożyteczna. Przyniosła ona w wyniku zalecenia na trzy języki, a mianowicie SDL, CHILL i MML. Języki te w chwili obecnej mają szereg wdrożeń, dalsze są w trakcie realizacji, a znaczenie języków stale wzrasta. Dzięki odpowiedniemu dobraniu poziomów tych języków nie są one językami nadającymi się tylko do wąsko wyspecjalizowanych zastosowań. Ze względu na ich sprawdzoną w praktyce przydatność powinny one zostać szerzej rozpropagowane w kraju, a ich stosowaniu powinna towarzyszyć współpraca z CCITT prowadząca do doskonalenia określających je zaleceń.

WYKAZ LITERATURY

1. Miernik J.: Programowane sterowanie central telefonicznych. WKŁ, Warszawa, 1980.
2. Kawashima H., Futami K., Kano S.: Functional specification of call processing by state transition diagram. IEEE Trans. 1971, Vol. COM-19, No 5, s. 581-587.
3. Spécification générales des relations homme-machine du système E1. Dokumentacja SOCOTEL. Styczeń 1974.
4. Proposed Revised and Expanded Recommendations for the CCITT Specification and Description Language /SDL/. Dokument CCITT AP VII-No. 20-E. June 1980.
5. Proposed Recommendation for the CCITT High Level Programming Language /CHILL/ /Recommendation Z.200/. Dokument CCITT AP VII-No.21-E. June 1980.
6. Proposed New and Revised Recommendations for the CCITT Man-Machine Language /MML/. Dokument CCITT AP VII-No.22-E. June 1980.
7. Podręcznik programowania w języku LPA. Instytut łączności, Warszawa 1977.
8. Introduction to CHILL /3rd edition/. Dokument CCITT COM XI-No. 342-E. October 1979.
9. Język człowiek-maszyna MMLS i monitor komunikacji operatorskiej MKOP. Instytut łączności. Warszawa 1981.

10. Hildebrandt A., Porada M., Pentman H.: Język - człowiek-maszyna dla węzła komutacyjnego. II Konferencja Naukowa Systemy i Sieci Telekomunikacyjne. Warszawa - październik 1981, s. 242-247.

11. Report on the Stockholm meeting: 2-5 December 1980. Dokument CCITT COM XI-No 5-E. January 1981.

12. Bourgonjon R.H., Breeus C.: Implementation Experiences with the CCITT High Level Language CHILL. International Switching Symposium 1979. Paris - France, s. 1151-1155.

13. Denis G., Langlois C.: Présentation d'une machine-language téléphonique orientée vers le language CHILL /language évolué du CCITT/. L'Echo des Recherches 1981 No 103.

14. Report on the meeting held in Geneva from 6 to 16 April 1981. Dokument CCITT COM XI-No. R1-E. May 1981.

15. Remarks and proposals on the CCITT Series Z.300 Recommendations for a Man-Machine Language - MML. Contribution to the CCITT Study Group XI concerns question 9/XI. Instytut Łączności 1981.

16. Steenhuisen A.C.: Man-machine Language for Digital PRX Telephone System. Philips Telecomm. Rev. 1980, Vol. 38, No 4, s. 149-167.

17. Questions allocated to Study Group XI for the period 1981-1984. Dokument CCITT COM XI-No. 1-E. December 1980.

Załącznik

Lista zaleceń CCITT, dotyczących języków programowanego sterowania /Seria Z/

Język SDL

Symbol zalecenia	Nazwa zalecenia	Stan zaawansowania
Z.101	Ogólne wyjaśnienia dotyczące języka SDL	możliwe modyfikacje
Z.102	Symbole i reguły	możliwe modyfikacje
Z.103	Stosowanie elementów piktograficznych wewnątrz symboli stanów	możliwe modyfikacje
Z.104	Semantyka	możliwe modyfikacje
Z.105	Postać alfanumeryczna języka SDL /SDL-PR/	w trakcie opracowywania
Aneks A	Przykłady zastosowania języka SDL	możliwe modyfikacje
Aneks B	Słownik terminów języka SDL	możliwe modyfikacje
Aneks C	Poradnik użytkownika języka SDL	przewidziane rozszerzenia

Język CHILL

Symbol zalecenia	Nazwa zalecenia	Stan zaawansowania
Z.200	Język programowania wysokiego poziomu /CHILL/	przewidziane rozszerzenie zawartości

Język MML

Symbol zalecenia	Nazwa zalecenia	Stan zaawansowania
Z.311	Wprowadzenie	możliwe modyfikacje
Z.312	Układ przedstawiania informacji	możliwe modyfikacje
Z.313	Metajęzyk do opisu składni i procedur	możliwe modyfikacje
Z.314	Zestaw znaków i elementy podstawowe	możliwe modyfikacje
Z.315	Składnia języka wejściowego /poleceń/	możliwe modyfikacje
Z.316	Składnia języka wyjściowego	możliwe modyfikacje
Z.317	Dialog człowiek-maszyna	możliwe modyfikacje
Z.318	Lista funkcji	przewidziane rozszerzenia
Z.341	Słownik terminów	możliwe modyfikacje