INSTYTUT ŁĄCZNOŚCI
WARSZAWA-MIEDZESZYN

PROBLEMY ŁĄCZNOŚCI

133
1975

MINISTERSTWO ŁĄCZNOŚCI

PROBLE(I)Y ŁĄCZNOŚCI

ROK 15 WARSZAWA 1975 NR 133

INSTYTUT ŁĄCZNOŚCI
Branżowy Ośrodek
Informacji Naukowo-Technicznej i Ekonomicznej

Redakcja Problemów Łączności

Redaktor Naczelny - mgr inż. Jerzy Rutkowski
Redaktorzy działów:
mgr inż. Władysław Cetner, mgr inż. Adam Moniuszko,
mgr inż. Józef Możejko

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

NA PRAWACH RĘKOPISU

Redaktor: J. Borkowska Montaż tekstu: B. Drabik

Dział Wydawniczy Instytutu Łączności
Format B5. Nakład 665. Wpłynęło do
Działu Wydawniczego 28.02.1975 r.
Druk ukończono w maju 1975 r.

P R O B L E M Y Ł Ą C Z N O Ś C I

Andrzej Hildebrandt

POSTĘP W DZIEDZINIE JĘZYKÓW PROGRAMOWANIA DLA URZĄDZEŃ TELEKOMUTACYJNYCH

SPIS TREŚCI

Ryszard Kowalik

DIAGNOSTYKA STEROWANYCH PROGRAMOWO URZĄDZEŃ TELEKOMUTACYJNYCH

Opracowania zawarte w niniejszym zeszycie Problemów łączności zapoczątkowują cykl z dziedziny komutacji elektronicznej.

Wprowadzenie do komutacji elektroniki oraz tak istotnego elementu, jakim jest obecnie oprogramowanie spowodowało skok jakościowy w tej dziedzinie. Z tego względu wydaje się celowe zapoznanie szerszego ogółu ze stanem wiedzy i najnowszymi osiągnięciami komutacji elektronicznej, tym bardziej że nie jest to dziedzina wyizolowana, lecz wiąże się coraz ściślej z innymi dziedzinami, takimi jak np. transmisja danych, czy ogólnie biorąc informatyka.

Zakład Telekomutacji IŁ prowadzi cały szereg prac z komutacji elektronicznej, co jest związane z zakupieniem licencji na system E-10. Prace te, prowadzone częściowo we współpracy z francuskim instytutem CNET, z pewnością zainteresują szersze grono specjalistów.

Andrzej Hildebrandt

621.395.345
681.3.04

POSTĘP W DZIEDZINIE JĘZYKÓW PROGRAMOWANIA DLA URZĄDZEŃ TELEKOMUTACYJNYCH

1. WSTĘP

Sterowanie programowane /stored program control - SPC/ jest podstawową cechą nowoczesnych systemów komutacyjnych. Wiele firm w różnych krajach produkuje systemy ze sterowaniem programowanym i instaluje je na całym świecie. W Polsce, po zakupieniu licencji francuskiej na system CITEDIS, przewiduje się w 1980 r. produkcję roczną tego typu central telefonicznych określoną liczbą 100000 numerów.

Pomimo że system licencyjny jest systemem bardzo nowoczesnym, to już w chwili obecnej podejmuje się prace naukowo-badawcze nad dalszym rozwijaniem tego systemu po to, aby po kilku czy kilkunastu latach nie okazało się, że produkowany przez nas sprzęt stał się przestarzały.

Jedną z wyraźnie zarysowujących się tendencji rozwojowych w systemach ze sterowaniem programowanym jest wprowadzanie języków programowania wyższego rzędu /high level language - HLL/ zamiast dotychczas najczęściej używanych języków symbolicznych z makrorozkazami /macro-assembly languages/.

Tendencja ta jest uzasadniona, gdyż znaczną częścią pracy, jaką należy wykonać przy projektowaniu i wprowadzaniu do eksploatacji systemu ze sterowaniem programowanym jest opracowanie oprogramowania. Wynika stąd naturalna tendencja do usprawniania narzędzi i metod stosowanych przy jego realizacji. Jednym ze sposobów ułatwiających zarówno realizację jak i utrzymanie oprogramowania jest zastosowanie języka programowania wyższego rzędu.

Faktem jest, że programy pisane w językach wyższego rzędu są mniej efektywne z punktu widzenia wykorzystania czasu procesora, jak również zajmują więcej miejsca w pamięci w porównaniu z programami pisanymi w językach symbolicznych. Należy się jednak spodziewać, że wady te będą miały coraz mniejsze znaczenie, natomiast korzyści wynikające z zastosowania języków wyższego rzędu spowodują ich szerokie rozpowszechnienie.

Przykładem opisanej tendencji może być fakt, że oprogramowanie wersji A centrum eksploatacji systemu CITEDIS wykorzystuje w ograniczonym zakresie język wyższego rzędu - FORTRAN II /tylko dla programów diagnostycznych/. Pozostałe programy napisane są w językach symbolicznych. W nowej opracowywanej obecnie wersji B tego centrum /CTI-B/ oprogramowanie będzie zawierać wiele programów w języku wyższego rzędu o nazwie LPA, który został specjalnie opracowany dla tego celu.

Problemom języków programowania wyższego rzędu poświęconych było wiele referatów przedstawionych na międzynarodowej konferencji pt. "Software Engineering for Telecommunications Switching Systems". Konferencja ta odbyła się w kwietniu 1973 r. na uniwersytecie w Essex w Wielkiej Brytanii. Z materiałów przedstawionych na tej konferencji wynika, że wiele firm i instytucji opracowuje i wprowadza języki wyższego rzędu dla oprogramowania swoich urządzeń. Można sądzić, że najciekawsze prace w tej dziedzinie prowadzi się w Wielkiej Brytanii i w Japonii.

Szczególną uwagę zwracają prace prowadzone na uniwersytecie w Essex, w którym daje się zauważyć duża niezależność poglądów. Studiowane tam języki do celów komutacyjnych nie są bowiem ukierunkowane do zastosowania ich w jednym konkretnym urządzeniu.

We Francji opracowano trzy języki wyższego rzędu. Są to języki PAPE, LPA i ESPL-1 [8] x/.

---

x/ Język PAPE został opracowany w CNET /Centre National d'Etudes des Telecommunications/, język LPA w SLE /Societe Lannionaise d'Electronique/, natomiast język ESPL-1 w LCT /Laboratorie Central de Telecommunications/.

Bardzo interesująco przedstawia się perspektywa opracowania wspólnego języka programowania wyższego rzędu, języka, który mógłby być używany przez wszystkich producentów i użytkowników. Prace prowadzone przez CCITT w tym kierunku prawdopodobnie przyniosą w 1975 r. rezultat w postaci opracowanego i sprawdzonego wspólnego języka programowania wyższego rzędu.

Wydaje się, że problemy języków wyższego rzędu dla celów komutacyjnych powinny być studiowane również w Polsce, a predysponowaną w tym kierunku instytucją jest z pewnością Instytut Łączności.

W niniejszym opracowaniu podano angielskie odpowiedniki i skróty pewnych pojęć ze względu na to, że polska terminologia w tej dziedzinie nie jest jeszcze ostatecznie ustabilizowana.

Należałoby zwrócić uwagę czytelników na fakt, że nie wszystkie tezy zawarte w niniejszym opracowaniu można zastosować do systemu CITEDIS, gdyż nie jest on systemem ze scentralizowanym sterowaniem.

2. ZADANIA JĘZYKA PROGRAMOWANIA

Oprogramowanie systemu telekomutacyjnego składa się z wielu elementów o zdecydowanie różnym charakterze. Oprócz programów obsługi połączeń, które bezpośrednio określają sposób działania urządzenia, występują takie programy, jak programy diagnostyczne, programy przywracania sprawności, cały szereg programów pomocniczych i innych. Oczywiście w skład oprogramowania wchodzi też system operacyjny.

Większość rozważań związanych z językami programowania wyższego rzędu do celów telekomutacyjnych wiąże się z zastosowaniem języka do pisania programów obsługi połączeń, gdyż od tych programów zależą właściwości ruchowe urządzenia telekomutacyjnego. Głównie z punktu widzenia tych programów określa się właściwości języka. Dąży się jednak do tego, aby wiele typów programów można było pisać przy użyciu tego samego języka wyższego rzędu.

Istnieje dość rozpowszechniona opinia, według której języki wyższego rzędu nie powinny być stosowane do pisania systemów operacyj-

nych, gdyż programy te, opracowywane przez producenta sprzętu, powinny być sporządzone bardzo starannie z punktu widzenia oszczędności czasu procesora. W tym celu należałoby stosować języki symboliczne lub nawet kod maszynowy. Niemniej jednak istnieją przykłady wskazujące na użyteczność języków wyższego rzędu i w tym zastosowaniu. Język CORAL został opracowany w Wielkiej Brytanii i stosowany jest do pisania programów dla urządzeń radarowych. Użytkownicy twierdzą, że część pewnego systemu operacyjnego napisana w języku CORAL jest bardziej efektywna od wersji pisanej w języku symbolicznym. Jako przyczynę podają, że przy pisaniu programu w języku wyższego rzędu programista poświęca mniej czasu i uwagi szczegółom i dlatego może bardziej skoncentrować się nad całością problemu.

Rozpatrzmy przedstawiony na rys. 1[x/] uproszczony schemat urządzenia komutacyjnego /np. centrali telefonicznej/ o sterowaniu programowanym. Urządzenie takie składa się z trzech zasadniczych części:

- z sieci dróg rozmównych, do której przyłączone są łącza służące do przesyłania sygnałów,
- urządzenia sterującego, które jest najczęściej zespołem komputerów,
- urządzeń pośredniczących, których zadaniem jest przetwarzanie sygnałów wysyłanych przez każdy z dwóch poprzednio wymienionych bloków w taki sposób, aby je dostosować do bloku, dla którego są przeznaczone.

Całe urządzenie komutacyjne realizuje szereg różnych funkcji zgodnych z wymaganiami funkcjonalnymi /functional requirements/ dla tego urządzenia. Żądany sposób działania urządzenia przedstawia się w postaci algorytmu. Rzeczywiste działanie urządzenia określone jest przez program sterujący umieszczony w pamięciach urządzenia sterującego.

[x/] Rysunki są umieszczone na końcu artykułu.

Można dostrzec tutaj trzy elementy:

- opracowany na podstawie wymagań funkcjonalnych algorytm, zgodnie z którym urządzenie powinno działać,
- program sterujący i
- rzeczywiste działanie urządzenia.

Między tymi elementami musi istnieć ścisła współzależność /rys.2/. Program sterujący jest wytwarzany na podstawie algorytmu działania urządzenia i powoduje on określone działanie urządzenia. Program sterujący zależy nie tylko od wymagań funkcjonalnych urządzenia, ale również od całego szeregu innych czynników, do których należą takie, jak: struktura sieci dróg rozmównych, szczegóły budowy zastosowanego komputera i urządzenia pośredniczącego, systemu operacyjnego itd.

Należy zwrócić uwagę, że pomimo iż program sterujący zawiera wszystkie informacje podane w wymaganiach funkcjonalnych, to jego binarna postać, w jakiej znajduje się on w pamięci urządzenia sterującego, jest niesłychanie odległa od postaci, w jakiej podawane są wymagania funkcjonalne czy algorytm działania urządzenia.

Aby łatwiej wyobrazić sobie rolę języka programowania wyższego rzędu, przedstawiono na rys. 3 /w sposób uproszczony/ proces powstawania programu sterującego. Proces ten składa się z części realizowanych przez człowieka i z części realizowanych przez komputery w sposób automatyczny.

Części wykonywane przez człowieka, to głównie programowanie, to znaczy przetwarzanie przez człowieka informacji do takiej postaci, aby mogły one być zrozumiałe przez maszynę cyfrową. Nie należy mylić jednakże programowania z kodowaniem. Kodowanie polega na pisaniu programu na podstawie gotowego algorytmu. Programowanie jest pojęciem szerszym i obejmuje również etapy pracy prowadzące do uzyskania algorytmu. Informacje są zrozumiałe dla maszyny tylko wówczas, gdy są one zapisane w określonym języku programowania, to znaczy

zgodnie z systemem oznaczeń służącym do formułowania programów, czyli ciągów instrukcji przeznaczonych dla komputera.

Druga część procesu powstawania programu sterującego - część realizowana automatycznie - wykonywana jest głównie nie przez komputer sterujący urządzeniem /on - line/, ale w ośrodku przetwarzania danych przy użyciu komputera off - line. Komputer ten jest często tego samego typu co komputer sterujący, ma natomiast bogatsze wyposażenie i oprogramowanie.

Jest sprawą zrozumiałą, że należy dążyć do przesuwania granicy pomiędzy częścią wykonywaną przez człowieka a częścią wykonywaną przez maszyny w takim kierunku, aby udział człowieka w przetwarzaniu informacji maksymalnie ograniczyć i ułatwić - chociażby dlatego, aby zmniejszyć liczbę błędów, których niestety nie da się uniknąć, gdy pracuje człowiek. Przesunięcie takie następuje w przypadku zastosowania języków wyższego rzędu.

Bardzo korzystną sytuację pokazano na rys. 4 /jest to zmodyfikowany fragment rys. 3/, gdzie program w języku wyższego rzędu A jest pisany wyłącznie na podstawie opisu funkcjonalnego urządzenia komutacyjnego i od tego miejsca proces wytwarzania programu sterującego przebiega automatycznie.

Taka sytuacja jest możliwa do zrealizowania i wówczas zadaniem języka programowania jest umożliwienie przedstawienia - w jak najprostszej dla człowieka, a zrozumiałej dla komputera formie - informacji zawartych w opisie funkcjonalnym urządzenia komutacyjnego.

3. POZIOMY JĘZYKÓW PROGRAMOWANIA

Językiem programowania na najniższym poziomie jest język wewnętrzny maszyny. Program napisany w tym języku, czyli program w kodzie maszynowym /machine code/ jest ciągiem rozkazów określających operacje wykonywane przez maszynę cyfrową. Rozkaz w języku wewnętrznym maszyny ma postać taką samą, w jakiej występuje w pamięci maszyny w czasie wykonywania programu /ciąg zer i jedynek/. Programowanie w ję-

zykach wewnętrznych maszyn jest żmudne, pracochłonne i jest źródłem częstych pomyłek. Każdy program jednakże, aby mógł być wykonany przez maszynę musi być doprowadzony do poziomu języka wewnętrznego. Najczęściej takie tłumaczenie programu wykonywane jest automatycznie przez program nazywany translatorem.

Na następnym z kolei poziomie po języku wewnętrznym znajdują się języki symboliczne /assembler level languages/. W językach tych operacje i adresy są zapisywane w postaci nazw symbolicznych. Jednemu rozkazowi napisanemu w języku symbolicznym odpowiada jeden rozkaz w języku wewnętrznym maszyny. Tłumaczenie programu napisanego przez programistę w języku symbolicznym na język wewnętrzny maszyny wykonuje translator /assembler/. Przykładem języka symbolicznego może być język programowania MITRAS II, w którym napisana jest część programów nowej wersji centrum eksploatacji w systemie CITEDIS.

Aby usunąć istotną wadę języka symbolicznego polegającą na tym, że programista musi napisać taką liczbę instrukcji, jaka jest niezbędna przy wykonywaniu programu, w kodzie maszynowym wprowadza się możliwość definiowania i stosowania makrorozkazów. Każdy makrorozkaz powoduje realizację sekwencji rozkazów maszyny. Makrorozkazy używane są w dwóch postaciach: makrorozkazów o ustalonym znaczeniu, określonym przez producenta oprogramowania maszyny oraz makrorozkazów definiowanych przez programistę.

Języki programowania wyższego rzędu /high level languages/ zbliżone są do sposobu opisu procesów informacyjnych używanego w dziedzinie, w której programy mają być wykorzystywane. Program w języku wyższego rzędu składa się ze zdań. Na ogół jednemu wykonywanemu zdaniu przyporządkowuje się więcej niż jeden rozkaz w języku wewnętrznym maszyny. Tłumaczenie programu napisanego w języku wyższego rzędu na program w języku wewnętrznym maszyn wykonywane jest przez kompilator. Przykładami znanych języków wyższego rzędu są ALGOL i FORTRAN.

W chwili obecnej większość programów w pracujących urządzeniach komutacyjnych napisana jest w językach symbolicznych z makrorozkaza-

mi. Wprowadzenie języków wyższego rzędu jest jednak kwestią najbliższego okresu czasu ze względu na istotne zalety tych języków w stosunku do języków symbolicznych.

Teoretycznie kompilacja programu w języku wyższego rzędu może dawać jako program wynikowy program w języku wewnętrznym maszyny. W praktyce, w wielu istniejących rozwiązaniach kompilacja jest wykonywana do poziomu języka symbolicznego z makrorozkazami lub bez nich. I tak na przykład administracja francuska opracowała język symboliczny z makrorozkazami, nazywany METASYMBOL CT. Praktyczna przydatność tego języka, który ma być uznany jako standardowy język do zastosowań komutacyjnych, została stwierdzona w szczególności w centrali elektronicznej w Roissy-en-France. Równocześnie został opracowany język algorytmiczny /wyższego rzędu/ PAPE, oparty na PL/1. Opracowano kompilatory dla maszyn LCT 3200 i CS40, które będą przetwarzały programy w języku PAPE na programy wynikowe w języku METAL-SYMBOL CT.

Język TL /task-language/, to stosowany w japońskim systemie D-10 język symboliczny z makrorozkazami. Opracowano język wyższego rzędu TGL /task generation language/. Programy pisane w tym języku są tłumaczone automatycznie przez program zwany Task Generator na język TL.

Kompilator języka LPA dla nowej wersji centrum eksploatacji systemu CITEDIS daje program wynikowy w języku symbolicznym MITRAS II.

Interesująca koncepcja dotycząca języków wyższego rzędu powstała w Essex University. M. Bennett [6] i P.C.P. Chung [7] uważają, że korzystne będzie stosowanie równocześnie kilku języków wyższego rzędu ułożonych w porządku hierarchicznym. Taki zespół języków, które autorzy nazywają językami hierarchicznymi /hierarchical languages/, składałby się prawdopodobnie z trzech języków.

Najniższy z tych języków byłby najniższym możliwym językiem niezależnym od maszyny i byłby tłumaczony na język symboliczny lub kod wewnętrzny. Autorzy twierdzą, że takim językiem może być język TPL 2 [2].

Na następnym, wyższym poziomie znalazłby się język, w którym znajdowalibyśmy takie zdania, jak: GET NEW RECORD /QUEUENAME/ czy TIMEOUT /300/. Pierwsza z tych instrukcji powodowałaby umieszczenie nowego bloku danych na liście określonej przez parametr. Druga zaś powodowałaby zatrzymanie dalszej realizacji programu na okres 300 ms.

Z języka poziomu drugiego z pewnymi częściami poziomu pierwszego powstałby język najwyższego poziomu, za pomocą którego łatwo byłyby opisywane algorytmy urządzenia komutacyjnego. Na tym poziomie występowałyby zdania na przykład typu CONNECT DIGIT RECEIVER. Język ten zawierałby pojęcia komutacyjne.

Koncepcje specjalistów w Essex University, którzy współpracują z Pocztą Brytyjską, wykazują dużą niezależność myślenia i dzięki temu szczególnie są godne uwagi.

4. PODSTAWOWE WYMAGANIA DOTYCZĄCE JEZYKÓW WYŻSZEGO RZĘDU DO CELÓW TELEKOMUTACYJNYCH

Mogą istnieć wątpliwości, czy rzeczywiście do celów komutacyjnych trzeba tworzyć nowe, specjalne języki programowania, czy nie należy używać tych języków ogólnego zastosowania, które dają możliwość programowania procesów uwarunkowanych czasowo. Takimi językami są na przykład ALGOL 68, PL/1, czy REAL-TIME FORTRAN. Oczywiście języki te mogłyby być użyte, ale jak pisze P.C.P. Chung [7] "... język programowania powinien być dostosowany do specyficznych wymagań. Istnieje zbyt wiele języków programowania ogólnego zastosowania, które były zaprojektowane w ten sposób, aby zadowolić wielką liczbę użytkowników i nie osiągnęły tego celu z powodu braku możliwości przystosowania ich do żądań większości użytkowników."

Tak więc język dla celów telekomutacyjnych powinien mieć cały szereg cech związanych ze specyfiką jego zastosowania. Wymienimy tu trzy charakterystyczne czynniki rzutujące na wymagania dotyczące takiego języka. Są to:

- konieczność uzyskania wysokiej efektywności wykorzystania czasu

pracy procesora i wysokiej efektywności wykorzystania pojemności pamięci,

- długowieczność urządzeń telekomutacyjnych i ich powszechne zastosowanie,
- niektóre cechy wyróżniające programy urządzeń telekomutacyjnych spośród programów dla procesorów uwarunkowanych czasowo.

Podstawowym problemem, jaki należy rozwiązać wprowadzając język wyższego rzędu do programowania urządzeń telekomutacyjnych, jest problem efektywności programu wynikowego. Program wynikowy jest otrzymywany po przetłumaczeniu przez kompilator programu źródłowego w języku wyższego rzędu. Program wynikowy może być otrzymywany w języku symbolicznym lub w kodzie maszynowym. Problem polega na tym, że program wynikowy przetłumaczony z języka wyższego rzędu zajmuje więcej miejsca w pamięci i pracuje wolniej niż analogiczny program napisany przez doświadczonego programistę w języku symbolicznym. Wynika to z niedoskonałego działania kompilatorów. Należy zatem rozpatrzyć dwie istotne wielkości:

- efektywność wykorzystania czasu pracy procesora,
- efektywność wykorzystania pojemności pamięci.

Oczywiście problem efektywności pojawia się zawsze przy używaniu języka wyższego rzędu do zastosowań czasu rzeczywistego, niemniej jednak w zastosowaniu komutacyjnym wysoka efektywność jest szczególnie pożądana. Można to uzasadnić w następujący sposób. Pamięci operacyjne procesorów służących do sterowania urządzeń komutacyjnych mają bardzo duże pojemności. Pojemności tych pamięci zawierają się w zakresie od 0,5 do wielu milionów bajtów. Biorąc pod uwagę wysoką cenę pamięci operacyjnych, wzrost zapotrzebowania na pamięć może spowodować istotne podniesienie ceny urządzenia komutacyjnego. Z drugiej strony procesor sterujący urządzeniem komutacyjnym jest tak zaprojektowany, aby w godzinach największego ruchu sprawność usługowa centrali pozostawała na odpowiednio wysokim poziomie. Byłoby

bardzo niekorzystne, gdyby wprowadzenie języka wyższego rzędu obniżało sprawność usługową lub pogarszało inne parametry centrali. Nie ulega wątpliwości, że oba wymienione rodzaje efektywności zależą od sposobu, w jakim napisany jest program źródłowy, a więc od programisty oraz od efektywności kompilatora. Niemniej jednak bardzo ważną rolę odgrywa tutaj odpowiedni język programowania.

Tak więc cechą języka programowania dla celów telekomutacyjnych powinno być umożliwienie i ułatwienie uzyskania wysokiej efektywności programu wynikowego. Najczęściej stosowanym mechanizmem wprowadzonym do języka, a służącym do poprawienia efektywności, jest możliwość włączania do programu segmentów w języku symbolicznym. Możliwość taka pozwala na optymalizację programów, szczególnie z punktu widzenia czasu pracy procesora w ten sposób, że pewne fragmenty programów, które występują szczególnie często lub które są kompilowane szczególnie niekorzystnie, można napisać optymalnie w języku symbolicznym.

Należy dodać, że tym sposobem łatwiej poprawić efektywność wykorzystania czasu procesora niż efektywność wykorzystania miejsca w pamięci. Wynika to stąd, że wszystkie programy zajmują miejsce w pamięci, natomiast tylko pewne części programów wpływają silnie na całkowity czas pracy procesora i te części mogą być programowane szczególnie starannie. Specyfika działania urządzenia komutacyjnego jest tutaj czynnikiem korzystnym, gdyż mała część programów obsługi połączeń wykonywana jest szczególnie często. Tak na przykład programy obsługi połączeń centrali miejskiej typu D-10 zawierają 2200 zadań /tasks/. Natomiast tylko 20 zadań najczęściej wykonywanych zajmuje 80% czasu procesora [10].

Niemniej jednak nawet przez ograniczone zastosowanie języka symbolicznego w programach niweluje się w pewnym stopniu niektóre zalety oprogramowania napisanego w języku wyższego rzędu.

Bardzo ważną sprawą jest możliwość pomiaru efektywności otrzymywanego kodu wynikowego. Twórcy języka TPL 2 przewidzieli taką możliwość

i wprowadzili do języka pewne udogodnienia ułatwiające takie pomiary [6]. Przewiduje się w miarę możliwości niezależne pomiary dla określenia efektywności języka i efektywności kompilatora. Przewidywana procedura prowadząca do otrzymania efektywnego programu ma polegać na kolejnych pomiarach efektywności i poprawianiu fragmentów programu odpowiedzialnych za niezadowalającą efektywność. Z doświadczenia wiadomo bowiem, że w skomplikowanych sytuacjach nie jest praktycznie możliwe przewidzenie, jak napisać dany fragment programu, aby uzyskać maksymalną efektywność.

Do zorientowania się w rzędzie wielkości pogorszenia efektywności przez zastosowanie języka wyższego rzędu posłużyć mogą wyniki eksperymentu przeprowadzonego na Uniwersytecie w Essex [1]. Jeden z programów eksploatacyjnych małej centrali telefonicznej został napisany w języku TPL1 i zajmował po kompilacji około 8k słów w pamięci. Następnie przeprowadzono analizę programu wynikowego i usunięto z niego wszystkie zbędne rozkazy w ten sposób, że program ten przybrał postać taką, jakby był pisany przez doświadczonego programistę w języku symbolicznym. W ten sposób stwierdzono, że program pisany jako źródłowy w języku TPL 1 zajmował około 15% więcej miejsca w pamięci, natomiast strata czasu procesora wynosiła około 30%. Równocześnie przewidywano, że użycie doskonalszego kompilatora pozwoli na obniżenie straty miejsca w pamięci do około 3%, natomiast straty czasu procesora do 10%. Trzeba tu przypomnieć, że dane te były opublikowane w pierwszej połowie 1972 r., a język TPL 1 był pierwszym językiem do celów komutacyjnych z serii opracowanej w Essex.

Interesujące informacje na temat efektywności programów komutacyjnych systemu D-10 znaleźć można w pracy [10].

Nie są na razie znane dane liczbowe dotyczące efektywności programów użytkowych centrum eksploatacji CTI-B pisanych w języku LPA, gdyż badania nie zostały jeszcze zakończone.

Pewne trudności przy optymalizacji powstają w wyniku faktu, że ma być ona przeprowadzona z punktu widzenia dwóch niezależnych od

siebie efektywności /to znaczy czasu i miejsca/. Zachodzi pytanie, która z efektywności jest ważniejsza - czasu czy miejsca? Pewna sekcja programu opracowanego w Essex [5], optymalizowanego z punktu widzenia miejsca w pamięci wydawała się szczególnie nieefektywna w czasie. Okazało się, że powiększenie o 5% ilości miejsca zajmowanego w pamięci obniżyło dynamiczny czas przebiegu programu do 45% jego początkowej wartości.

Uzyskanie wysokiej efektywności może utrudnić zbyt bogaty język. Łatwiej jest bowiem opracować efektywny kompilator, gdy nie ma możliwości wybrania wielu różnych sposobów napisania określonego segmentu programu.

Aby uniknąć zdarzających się nieporozumień, należy przypomnieć, że w rozważaniach dotyczących efektywności wykorzystania czasu pracy procesora nie bierze się pod uwagę czasu kompilacji, ponieważ kompilacja wykonywana jest jednokrotnie i to na innej maszynie niż ta, która steruje urządzeniem komutacyjnym. Raz skompilowany program może pracować w urządzeniu komutacyjnym przez wiele lat. Program po kompilacji może być powielony i może być użyty w wielu urządzeniach.

Dalsze wymagane cechy języka wyższego rzędu wynikają z długowieczności urządzeń komutacyjnych. Długi czas życia sprzętu komutacyjnego, wynoszący kilkadziesiąt lat, powoduje istnienie stałego kontaktu z programami. Często zachodzi potrzeba czytania napisanych kiedyś programów, programy są wielokrotnie uaktualniane, modyfikowane lub rozszerzane. Pewne programy są wymieniane pomiędzy użytkownikami podobnego sprzętu. Uaktualnianie, modyfikowanie i rozszerzanie programów napisanych przez producenta sprzętu jest często wykonywane przez użytkownika; a ze względu na długi okres czasu życia sprzętu wykonywanie tych prac przez tę samą osobę jest często niemożliwe.

Powyższe wskazuje przede wszystkim na celowość używania języków wyższego rzędu, ale również określa pewne cechy języka używanego do celów telekomutacyjnych. Wymienić tu można na przykład takie cechy, jak:

- łatwość uczenia się,
- czytelność /zrozumiałość/,
- rozszerzalność /extendibility/.

Należy zwrócić uwagę, że łatwiej osiągnąć pierwsze dwie cechy dla języka, który nie jest zbyt bogaty, a więc nie jest językiem powszechnego zastosowania.

Rozszerzalność powinna polegać na możliwości wprowadzania do języka nowego typu zdań, jak również nowych operatorów i argumentów.

Informacje między urządzeniem sterującym a siecią dróg rozmównych są często przesyłane w postaci meldunków /message/, to znaczy grup bitów o określonej ich liczbie w grupie. W systemie CITEDIS meldunki są przesyłane pomiędzy centralami a CTI w obu kierunkach, przy czym każdy meldunek zawiera tu 180 bitów [4]. Język wyższego rzędu powinien umożliwiać tworzenie takich meldunków w celu ich wysłania, jak również analizowanie i przetwarzanie nadchodzących meldunków.

Dla zarejestrowania wielu danych występujących w procesie przetwarzania wystarczają często ciągi bitów o długości bądź mniejszej od długości słowa maszyny, bądź różnej od wielokrotności takiego słowa. Byłoby rozrzutnością rezerwowanie dla tych wielkości całych słów pamięciowych. W tej sytuacji język powinien umożliwiać deklarowanie danych o różnych długościach wyrażonych w bitach oraz na operacje nad nimi.

I tak na przykład język LPA pozwala na deklarowanie danych trzech rodzajów: FIXED, CHARACTER i BIT/n/. Wielkość zadeklarowana jako FIXED ma określoną długość /równą długości słowa/; dla wielkości typu CHARACTER jest rezerwowanych 8 bitów, wielkości typu BIT/n/ są umieszczane w pamięci stosownie do ich długości określonej liczbą n. Podobne możliwości daje język PAPE. Język TPL 2 umożliwia jeszcze większą elastyczność, szczególnie w sposobach rozmieszczania w pamięci danych o różnych długościach.

Przy sterowaniu urządzeniami komutacyjnymi zachodzi potrzeba two-

rzenia list. Pewne zespoły danych o takim samym charakterze ustawiane są w ciągi, tworząc tzw. listy. Jeśli na przykład każdemu połączeniu przyporządkowuje się pewien zespół danych /blok/, to dane te zajmują pewien obszar w pamięci, przy czym położenie tego obszaru nie jest z góry określone. Informacje umieszczone w tych obszarach są uaktualniane w momentach, gdy zostanie wykryta jakaś zmiana na stanu, upłynął określony czas itp. Język TPL 1 umożliwia tworzenie list o strukturze łańcuchowej, tj. o takiej organizacji, że do każdego bloku jest dodane słowo zawierające adres następnego bloku. Rozróżnia się bloki wolne oraz bloki zajęte. Bloki wolne nie przechowują w danym momencie żadnej użytecznej informacji, natomiast bloki zajęte przechowują takie informacje. Wszystkie bloki wolne tego samego typu mogą tworzyć jedną listę, natomiast wszystkie bloki zajęte - drugą. Dostęp do listy realizowany jest poprzez blok nagłówkowy, który zawiera adres pierwszego bloku listy /lub zero, jeśli lista jest pusta/. Język programowania powinien umożliwiać tworzenie takich list i przeprowadzenie operacji nad nimi.

W języku TPL 1 na przykład wprowadzono szereg operatorów do przeprowadzania operacji z listami. Operatory te są w zasadzie realizowane przez podprogramy. Operator NEWRECORD dołącza nowy blok z listy wolnych bloków do końca bieżącej listy, operator INSERT ustawia nowy blok z listy wolnych bloków za bieżącym blokiem, operator DELETE usuwa bieżący blok i wstawia go na koniec listy wolnych bloków, operator MOVE /list name/ dołącza bieżący blok na końcu wymienionej listy itd. Ponieważ doświadczenie wykazało, że struktura list możliwych do zrealizowania w języku TPL 1 jest zbyt sztywna, w języku TPL 2 zagadnienia te zostały znacznie rozbudowane.

Ze względu na fakt, że procesor sterujący pracuje w systemie uwarunkowanym czasowo, język programowania musi umożliwiać współzależność programów i przerwań. W tym celu programy pisane w językach TPL mogą być dynamicznie związane z przerwaniami przez zdania:

on /interrupt spec/ do

Ponadto przewiduje się instrukcje blokowania i odblokowywania przerwań:

arm /interrupt spec/
disarm /interrupt spec/

Bardzo istotnym wymaganiem dotyczącym języka używanego do pisania programów obsługi połączeń jest odpowiednie dostosowanie języka programowania do sposobu przedstawiania wymagań funkcjonalnych. Jak już wspomniano poprzednio, język programowania powinien w jak najprostszy sposób pozwolić na wyrażenie informacji zawartych w wymaganiach funkcjonalnych. Im jest prostsze przejście od wymagań funkcjonalnych do programu, tym mniejsza możliwość powstawania błędów. Tak więc problem doboru właściwego języka wyższego rzędu powinien być rozpatrywany łącznie z opracowywaniem sposobu przedstawiania wymagań funkcjonalnych.

Przykład idealnego prawie zbliżenia języka programowania do sposobu przedstawiania wymagań funkcjonalnych jest opisany w opracowaniu japońskim dotyczącym systemu D-10 [9]. Wymagania funkcjonalne w systemie D-10 są przedstawione w postaci tzw. "state transition diagrams", czyli sieci przejść pomiędzy stanami. Działanie urządzenia komutacyjnego w postaci graficznej zawiera stany /states/ i przejścia między tymi stanami /transitions/. Dla opisu stanów wprowadzono język TGLS, a dla opisu przejść między stanami język TGLT. Oba języki są bardzo proste i nauczenie się pisania programów za ich pomocą wydaje się być kwestią kilku godzin. Wydaje się również, że możliwości popełniania błędów są tu bardzo małe. Niestety, prostota i dostosowanie języków do "state transition diagrams" spowodowały ich ograniczone zastosowanie. Języki te mogą służyć tylko do opisu programów obsługi połączeń i tylko około 90% całości tych programów daje się zapisać w ten sposób.

Dalsze wymaganie dotyczące języka wynika z konieczności wysokiej

niezawodności oprogramowania. Większość dużych, pracujących wydawałoby się poprawnie programów zawiera błędy. Źródła amerykańskie szacują liczbę pozostawianych w programach błędów na jeden na tysiąc instrukcji. Ze względu na wymaganą bardzo wysoką niezawodność urządzeń komutacyjnych należy dążyć do usunięcia jak największej liczby błędów z programów. Dlatego język powinien być tak zaprojektowany, aby ułatwił usuwanie błędów /debugging/. Istotne tu jest otrzymywanie odpowiedniej struktury programów, wygodnej dla niezależnego uruchamiania poszczególnych segmentów programów.

Wymienione powyżej wymagania dla języka są często ze sobą sprzeczne. Na przykład wprowadzanie do języka udogodnień pozwalających na łatwiejsze uzyskiwanie wysokiej efektywności może prowadzić do obniżenia jego czytelności lub efektywności. Niestety nie można podać gotowej recepty określającej dokładnie, jakie cechy powinien mieć język dla określonego zastosowania, tym bardziej że cech tych nie można ująć w postaci liczbowej. Dla uzyskania niezbędnej wiedzy w tej dziedzinie potrzebne są studia stanu wiedzy w krajach zaawansowanych, ale konieczna jest również konkretna praca polegająca na tworzeniu programów komutacyjnych w językach wyższego rzędu, i na tej podstawie wnioskowanie o przydatności tych języków i ich niezbędnych właściwościach.

5. OCENA JAKOŚCI JĘZYKA PROGRAMOWANIA

Ogólnie biorąc, jakość programu wynikowego realizującego określone zadanie zależy od trzech istotnych czynników: języka, w którym napisany był program źródłowy, kompilatora oraz od sposobu, w jaki program źródłowy był napisany. Ten ostatni czynnik związany jest z kwalifikacjami programistów oraz ilością czasu, jaką mają oni do dyspozycji. Czynnik ten jest dość elastyczny - programy w języku źródłowym można dość łatwo poprawiać, przerabiać czy uzupełniać. Dwa pierwsze czynniki są dużo bardziej stabilne i dlatego ważna jest umiejętność ich właściwej oceny w momencie, gdy jeszcze nie jest za

późno na wprowadzenie odpowiednich zmian w języku czy w kompilatorze. A zatem potrzeba oceny jakości języka programowania występuje zwykle w początkowym etapie prac nad oprogramowaniem systemu. W tym okresie podejmuje się decyzję, w jakim języku oprogramowanie będzie realizowane. Można wybrać jeden z istniejących języków służących do programowania systemów komutacyjnych, można do dowolnego z tych języków wprowadzać zmiany, można wreszcie zdecydować się na opracowanie całkowicie nowego języka, bądź opracowanie podzbioru któregoś z języków ogólnego zastosowania. W każdym z tych przypadków istnieje potrzeba odpowiadania na pytania: który z wielu języków jest najlepszy do danego zastosowania lub który z dwóch języków, lub która z dwóch wersji języka jest lepsza do danego zastosowania? Wygodnym rozwiązaniem byłoby wyrażenie poszczególnych właściwości języka w postaci liczbowej, a następnie przeprowadzenie operacji arytmetycznych nad tymi liczbami dla otrzymania odpowiedzi na sformułowane pytania.

W przypadku oceny jakości kompilatora możemy posługiwać się takimi wielkościami jak objętość kompilatora /wyrażona liczbą instrukcji lub słów pamięci zajętych przez kompilator/, pojemność pamięci potrzebną do kompilacji, czy prędkość kompilacji. Możemy wreszcie, co jest bardziej istotne w przypadku zastosowań komutacyjnych, posługiwać się takimi wielkościami, jak czas wykonywania programu wynikowego czy liczba instrukcji programu wynikowego. Natomiast istotne właściwości języka programowania nie mają wyrażenia ilościowego w postaci naturalnej i jedyną metodą, jaką można zastosować, jest przypisywanie tym właściwościom wartości liczbowych w sposób arbitralny. Dodatkową trudność sprawia rozdzielenie wpływu języka i wpływu kompilatora na parametry programu wynikowego.

Rozpatrując właściwości języka można rozróżnić właściwości składniowe i właściwości pozaskładniowe. Z punktu widzenia użytkownika wymienimy następujące właściwości pozaskładniowe:

- łatwość czytania programów,

- łatwość pisania programów,
- łatwość uruchamiania programów,
- łatwość uczenia się języka,
- samodokumentacyjność /selfdocumentation/.

Każda z tych właściwości musi być rozpatrywana niezależnie, gdyż to co jest łatwe do czytania, niekoniecznie musi być łatwe do pisania itd. Dalszymi właściwościami są:

- niezależność od maszyny,
- niezależność od systemu operacyjnego,
- naturalność.

Naturalność jest cechą, dzięki której program w języku maszyny jest podobny do instrukcji, jakie przekazywane są pomiędzy ludźmi w celu wyjaśnienia jak wykonać pewne zadania.

Dalej wymienić należy:

- odpowiedniość do obszaru zastosowań,
- odpowiedniość do szczególnego zastosowania.

Jeśli rozpatrujemy jakość języka od strony składni, to należy zwrócić uwagę na takie właściwości, jak:

- zarezerwowane symbole słowne /istnienie, liczba, jakie słowa/,
- rodzaje wyrażeń,
- zestaw stosowanych znaków,
- struktura programu /tj. bloki, procedury, zasady tworzenia sekwencji/,
- rodzaje danych,
- instrukcje,
- deklaracje.

Koncepcja metody pomiarów języków programowania przedstawiona została przez J.E. Sammet [3]. W myśl tej metody pomiary powinny być przeprowadzane z różnych punktów widzenia: z punktu widzenia użytkownika, z punktu widzenia obszaru zastosowań, z punktu widzenia szcze-

gólnego zastosowania itp. Dla każdego punktu widzenia poszczególnym właściwościom przypisać można współczynniki wagowe. Na przykład z punktu widzenia osoby piszącej program list płacy autorka przyporządkowuje właściwości "łatwość pisania" współczynnik wagowy 0.8, a właściwości "łatwość uruchamiania" współczynnik wagowy 0.5. Następnie według umownej, operującej liczbami rzeczywistymi skali ocen poszczególne cechy ocenia dla porównywanych języków. Oceny mnożone są przez współczynniki wagowe, a otrzymane iloczyny sumowane są dla wszystkich branych pod uwagę właściwości danego języka. Wyniki są normalizowane i w ten sposób otrzymuje się liczbowe wyrażenia jakości języka z określonego punktu widzenia. Sposób przeprowadzania obliczeń wyjaśniono w tablicy poniżej, przy czym dla prostoty pokazano tylko dwie właściwości języka.

T a b l i c a

Przykład sposobu oceny jakości języka wg J.E. Sammet

Właściwość	Współczynnik wagowy	Ocena		Ocena ważona	
		COBOL	PL/1	COBOL	PL/1
łatwość pisania	0,8	1	0,5	0,8	0,4
łatwość uruchamiania	0,5	1	0,8	0,5	0,4
			Miara jakości	1,3	0,8

Wydaje się, że metoda ta, jakkolwiek dość subiektywna, może być stosowana jako element pomocniczy przy porównywaniu języków i przy wyborze właściwego języka programowania.

Problem oceny jakości języka wystąpił również w związku z planowanym opracowaniem wspólnego języka CCITT do celów komutacyjnych. W związku z tym Poczta Brytyjska opracowała dokument pt. "Opis pewnych właściwości języka wyższego rzędu z punktu widzenia ich użyteczności do sterowania programowanego". Na razie jest to tylko o-

pracowanie szkicowe, a właściwe, pełne opracowanie znajdzie się w przygotowywanym dopiero dokumencie.

6. ASPEKTY EKONOMICZNE

Rozpatrując zagadnienie wprowadzenia języka wyższego rzędu oraz wyboru optymalnego języka z punktu widzenia ekonomicznego, należy mieć na uwadze różnice pomiędzy warunkami istniejącymi w Polsce a warunkami w rozwiniętych krajach kapitalistycznych, skąd pochodzi większość informacji na ten temat.

W wielu publikacjach zachodnich na przykład [5,10] silnie uwypukla się fakt, że zastosowanie języka wyższego rzędu powiększa wydajność programistów.

Niektórzy producenci zachodni stwierdzają, że wprowadzić język wyższego rzędu można tylko pod warunkiem, że nie spowoduje to ani podwyższenia ceny urządzenia, ani obniżenia jego parametrów ruchowych /patrz np. [5]/. Dzieje się to w sytuacji, gdy centrale ze sterowaniem programowym, zawierające drogie komputery i drogie oprogramowanie, torują drogę na rynku metodami konkurencyjnymi. Inna sytuacja panuje u nas na rynku, który jest rynkiem producenta, a przede wszystkim gdzie można i należy ponieść dość znaczne czasami koszty nowoczesności.

Rozpatrzmy te składniki kosztów związanych z oprogramowaniem urządzenia telekomutacyjnego, które zależą od użytego języka programowania.

Są to składniki następujące:

1. projektowanie systemu,
2. szkolenie pracowników producenta i użytkowników,
3. programowanie,
4. implementacja języka,
5. próby oprogramowania,
6. opracowanie dokumentacji,

7. czas pracy komputera,
8. miejsce w pamięci komputera,
9. utrzymanie oprogramowania.

Wyraźnie daje się zauważyć, że wprowadzenie języka wyższego poziomu obniża koszt wszystkich składników z wyjątkiem składników 4, 7 i 8. Ponieważ implementacja języka jest czynnością jednorazową, natomiast miejsce w pamięci zajęte jest we wszystkich pracujących urządzeniach, a ponadto koszt czasu pracy komputera rośnie z długowiecznością urządzeń, wydaje się więc, że w porównaniu ze składnikiem 4 bardziej istotne znaczenie mają składniki 7 i 8.

Niestety nie opracowano dotychczas żadnej teorii określającej proporcje wymienionych dziewięciu składników kosztu oprogramowania. Niedostępne są także dane liczbowe dotyczące zrealizowanych przedsięwzięć.

Zagadnieniem wyboru optymalnego poziomu języka programowania zajmuje się H.R. Sorgenfrei [11]. Rozpatruje on niezależnie koszty opracowania oprogramowania, to znaczy składniki 1 ÷ 6 i koszty pozostałe, tj. składniki 7 ÷ 9, przy czym utrzymuje on, że składniki 7 ÷ 9 mają większe znaczenie niż pozostałe, gdyż działają one w ciągu całego okresu życia urządzenia komutacyjnego, a poza tym są zwielokratniane z liczbą pracujących urządzeń. Koszty składników 1 ÷ 6 są ponoszone tylko raz w okresie opracowywania systemu.

Rysunek 5 pokazuje zależności kosztu opracowania oprogramowania EK /składniki 1,2,2,5,6/ w funkcji poziomu języka L. β .P/L/ jest rodziną krzywych przedstawiających koszt opracowania programów,gdzie β jest parametrem określającym liczbę instrukcji. I/L/ jest krzywą określającą koszt implementacji wybranego języka programowania. Sumaryczny koszt opracowania oprogramowania EK/L/ = β .P/L/ + I/L/ ma minimum dla pewnego poziomu języka. Można udowodnić, że im liczba instrukcji β jest większa, tym minimum wypada dla wyższego poziomu języka.

Na rysunku 6 przedstawiono zależności kosztu PK składników 7,8,9

również w funkcji poziomu języka L. α .W/L/ jest rodziną krzywych przedstawiających koszt utrzymania, gdzie α jest parametrem określającym liczbę wymaganych czynności utrzymania. γ .S/L/ jest rodziną krzywych przedstawiających koszt czasu pracy komputerów i ich pamięci, przy czym γ jest parametrem określającym liczbę egzemplarzy urządzeń. Tak więc PK/L/ = α. W/L/ + γ . S/L/ określa sumaryczny koszt tych składników, które według opinii autora mają decydujące znaczenie. Funkcja PK/L/ ma również minimum.

Interesujące jest, jak przesuwa się to minimum w zależności od liczby egzemplarzy pracujących urządzeń. W przypadku scentralizowanego utrzymania liczba czynności utrzymania α rośnie wolniej niż liniowo ze wzrostem liczby urządzeń γ . A zatem stosunek α / γ, który decyduje o położeniu minimum funkcji PK/L/ maleje ze wzrostem γ , w wyniku czego minimum przesuwa się w kierunku mniejszych wartości L. Tak więc im większa liczba egzemplarzy urządzeń, tym poziom języka programowania powinien być niższy dla uzyskania optimum tej części kosztów.

Przedstawiona powyżej teoria podana przez Sorgenfrei'a jest dość prymitywna, niemniej jednak przy braku innej teorii pozwala na zorientowanie się w zachodzących zależnościach. Szczególnie daje się odczuć brak danych liczbowych, które określałyby przynajmniej proporcje pomiędzy poszczególnymi składnikami występującymi w przedstawionych wzorach. Przy pewnych proporcjach przewidywane minima mogą wystąpić poza użytecznym zakresem L. Zastrzeżenie może również budzić fakt użycia zmiennej ciągłej, i to jednej zmiennej L dla wyrażenia poziomu języka programowania. W rzeczywistości trudno byłoby w wielu przypadkach odpowiedzieć na pytanie, który z dwóch określonych języków wyższego rzędu reprezentuje poziom wyższy. Szczególnie trudne stałyby się tego typu rozważania przy zastosowaniu języków hierarchicznych.

Wydaje się, że celowe byłoby oszacowanie wartości wymienionych dziewięciu składników w warunkach polskich.

7. WSPÓLNY JĘZYK PROGRAMOWANIA CCITT

Problemem wspólnego języka programowania zajmuje się od kilku lat komisja XI CCITT. Na podstawie dotychczasowych badań ankietowych wśród producentów i administracji już eksploatujących lub mających zamiar wprowadzić centrale ze sterowaniem programowanym stwierdzono, że opracowanie takiego języka jest pożądane. Pewna liczba organizacji mających doświadczenie w opracowywaniu języków zgłosiła chęć uczestniczenia w tej pracy, podczas gdy niewielka część organizacji wyraziła opinię, że w chwili obecnej powinna być jedynie studiowana możliwość zalecenia stosowania wspólnego języka. Organizacje te twierdzą, że standaryzacja jest niebezpieczna albo nieosiągalna. Niebezpieczeństwo mogłoby polegać na tym, że zbyt wcześnie zdefiniowany wspólny język mógłby ograniczyć niezależność koncepcji przyszłych systemów. Jego słabe miejsca mogłyby spowodować ograniczenia w odpowiednim wykorzystaniu procesorów. Należy tu jednak pamiętać o tym, że przeciwnicy wspólnego języka wiedzą, że systemy komutacyjne produkowane po jego ustaleniu mogłyby być konkurencyjne w stosunku do systemów produkowanych wcześniej, a do których język taki nie byłby wprowadzony.

Stan zaawansowania prac w końcu 1973 roku był następujący. Ustalono założenia dla opracowywanego języka, jego zakres zastosowania oraz przyjęto procedurę służącą do określenia jakości języka. Opracowano także harmonogram dalszych prac.

Zgodnie z założeniami wspólny język wyższego rzędu powinien być na poziomie proceduralnym. Powinna istnieć możliwość jego rozszerzania w kierunku uzależniania go od maszyny, jak i w kierunku orientowania go problemowo. Powinna być zapewniona możliwość włączania do programów w języku wyższego rzędu segmentów w języku symbolicznym z makrorozkazami.

Wspólny język powinien umożliwiać kompilację w taki sposób, aby programy wynikowe wykorzystywały zasoby maszyny w sposób efektywny.

Wspólny język powinien być łatwy do uczenia się, do użytkowania, do tłumaczenia i do rozszerzania z punktu widzenia zainteresowanych organizacji. Pojęcie użytkowania zawiera tu również proces uruchamiania programów /debugging/.

Poza tym wspólny język powinien być w znacznym stopniu niezależny od maszyny dla klasy maszyn używanych w chwili obecnej i przewidywanych do zastosowania w urządzeniach telekomutacyjnych w przyszłości.

Język ten powinien nadawać się również do symulacji otoczenia procesorów sterujących. Symulacja taka jest często stosowana przy badaniach mających na celu określanie różnych właściwości urządzeń sterujących bez potrzeby dołączania ich do sterowanych urządzeń.

Przewiduje się, że proponowany język będzie używany do pisania następujących kategorii programów:

1. programy obsługi połączeń /call handling/,
2. programy automatycznych prób wewnętrznych i programy utrzymania,
3. programy implementacji języka konwersacyjnego /nie sam język konwersacyjny/,
4. system operacyjny oraz programy diagnostyki uszkodzeń i przywracania sprawności,
5. programy pomocnicze on-line,
6. programy pomocnicze off-line,
7. próby używane dla odbioru urządzeń.

Nie zostało zdecydowane, czy będzie podjęte opracowanie całkowicie nowego języka, czy też rozwinie się i udoskonali jeden z istniejących języków. W każdym z tych przypadków wspólny język ma być opracowany do końca 1975 roku. Jeśli zapadnie decyzja o opracowaniu nowego języka, dotrzymanie tego terminu będzie znacznie trudniejsze.

Aby zbadać wstępnie przydatność istniejących języków, została sporządzona lista języków proponowanych do przestudiowania. Lista ta

jest obszerna i zawiera 25 pozycji. Występują tu zarówno znane języki ogólnego zastosowania, takie jak ALGOL 68 czy PL/1, jak również języki stworzone specjalnie do celów komutacyjnych, takie jak wspomniany już uprzednio język PAPE czy grupa języków TPL. Lista zawiera takie języki, które powstały przy okazji innych zastosowań, ale wykazują cechy, które pozwalają przypuszczać, że ich zastosowanie do celów komutacyjnych może okazać się korzystne.

Według planu prac CCITT ma być zorganizowana mała grupa robocza, w skład której będą wchodzić specjaliści z dziedziny języków programowania i z dziedziny kompilatorów. Grupa ta opracuje pierwszą wersję nowego języka lub zaproponuje zmiany w którymś z istniejących języków. Następnie zostaną opracowane kompilatory dla tego języka i zostanie zrealizowanych możliwie dużo programów, tak aby można było określić jego przydatność. Po tym okresie będzie opracowana ostateczna wersja języka. Odpowiednie zalecenie przedstawione będzie do rozpoznania na sesji plenarnej CCITT w 1976 roku.

Problem opracowania wspólnego języka jest problemem dość trudnym, niemniej jednak znacznie trudniejsze może się okazać nakłonienie zainteresowanych organizacji do stosowania tego języka. Trudno na przykład sobie wyobrazić, aby firma NTT wprowadziła inny język niż Task Generation Language, który z jednej strony jest doskonale przystosowany do używanego przez tę firmę sposobu przedstawiania wymagań funkcjonalnych, z drugiej zaś strony jest związany z językami symbolicznymi TSL i TLT. Przeszkodą mogą być takie względy jak konieczność przekwalifikowania dużej kadry programistów i innych specjalistów, zmiany sposobu wykonywania dokumentacji i samej dokumentacji itp.

Znacznie łatwiejsze będzie wprowadzenie wspólnego języka w krajach RWPG, gdzie w chwili obecnej nie ma jeszcze central ze sterowaniem programowanym. W przypadku Polski istnienie współpracy z francuskim instytutem CNET w tej dziedzinie sugerowałoby, aby przyjąć u nas języki PAPE i LPA.

Interesujące jest stanowisko RFN w sprawie standaryzacji języka wyższego poziomu przedstawione w jednym z dokumentów CCITT. Czytamy w tym dokumencie: "Język programowania opracowywany przez CCITT powinien zawierać bardzo prosty rdzeń języka. Wystarczająca będzie międzynarodowa standaryzacja tylko tego rdzenia".

8. TENDENCJE ROZWOJOWE

Wyraźną tendencją występującą w dziedzinie języków programowania dla urządzeń telekomutacyjnych jest coraz szersze stosowanie języków wyższego rzędu i zawężanie zakresu zastosowania języków symbolicznych. Zalety języków wyższego rzędu są niewątpliwe. Wydaje się, że przyszłe prace będą koncentrowały się nad usunięciem kłopotów związanych z używaniem tych języków, a więc będą opracowywane metody pozwalające w sposób automatyczny uzyskiwać w procesie kompilacji kody optymalne pod względem efektywności. Prawdopodobnie programiści będą mogli również automatycznie uzyskiwać informacje wskazujące jak poprawić program źródłowy, aby uzyskać lepszą efektywność.

Będą również prowadzone równoległe prace nad metodami opisu wymagań funkcjonalnych i nad językami programowania wyższego rzędu, które doprowadzą do takiej sytuacji, że łatwy do opracowania i przejrzysty opis wymagań funkcjonalnych będzie w prosty sposób mógł być opisany w języku wyższego rzędu. W konsekwencji dojdzie do całkowitej automatyzacji tej części procesu kodowania, co oznacza, że opis wymagań funkcjonalnych stanie się zrozumiały dla maszyny.

Równocześnie będą trwały prace nad zwiększaniem uniwersalności oprogramowania nie tylko w skali międzynarodowej, co jest celem CCITT, ale również w zakresie poszczególnych firm czy zespołów firm. Uniwersalność taka będzie uzyskiwana przez opracowywanie języków hierarchicznych o standardowych interfejsach i w dużym stopniu niezależnych od maszyny, jak również niezależnych od systemów operacyjnych.

Duży postęp będzie osiągnięty przez opracowanie przenośnych kom-

pilatorów, to znaczy kimpilatorów mogących wytwarzać program wynikowy dla dowolnej maszyny pewnej klasy.

9. ZAKOŃCZENIE

W opracowaniu niniejszym przedstawiono pewne problemy związane z zastosowaniem odpowiedniego języka programowania dla urządzeń telekomutacyjnych. Można śmiało twierdzić, że wybór odpowiedniego języka stanowi istotne ogniwo w procesie opracowywania urządzeń i systemów telekomutacyjnych. Właściwości języka mają wpływ nie tylko na koszty i czas opracowania systemu, ale również na jego parametry ruchowe, a także na koszty utrzymania systemu w czasie jego długoletniej eksploatacji. Właściwy poziom języka i jego uniwersalność może ułatwić w przyszłości modernizację istniejących systemów, jak również opracowywanie nowych systemów telekomutacyjnych.

Wydaje się, że w tej sytuacji należy starannie śledzić prace nad językami programowania prowadzonymi przez Komisję XI CCITT i w miarę możliwości brać w nich aktywny udział. Prace takie powinny być również prowadzone w ramach współpracy z Francją, która ma już niemałe osiągnięcia w tym zakresie.

Zainstalowany w Zakładzie Telekomutacji IŁ minikomputer R-10 może ułatwić realizację niektórych prac doświadczalnych nad językami programowania.

WYKAZ LITERATURY

1. Hills M.T., Constantine H., Chung P.: Telecommunication - oriented programming language for switching systems. Proc. IEE 1972 nr 4, s. 409-415.

2. Bennett M.: TPL2 Reference Manual. PO Research Dept. Ipswich.

3. Sammet J.E.: Problems in, and a pragmatic approach to, programming language measurement. Fall Joint Computer Conference 1971, s. 243-251.

4. Bajurski W., Bogobowicz M., Hildebrandt A.: Centrum eksploatacji w systemie CITEDIS. Prz. Telekom. 1974 nr 1, s. 5-11.

5. Allen R.C., Coackley F.P.: Experience gained in the use of a high - level language in experimental switching system. Software Engineering for Telecommunication Switching Systems, Essex 1973, s. 144-151.

6. Bennett M.: Design and implementation of TPL2. Software Engineering for Telecommunication Switching Systems, Essex 1973, s. 21-30.

7. Chung P.C.P.: Hierarchical languages for the implementation of SPC software systems. Software Engineering for Telecommunication Switching Systems, Essex 1973, s. 100-107.

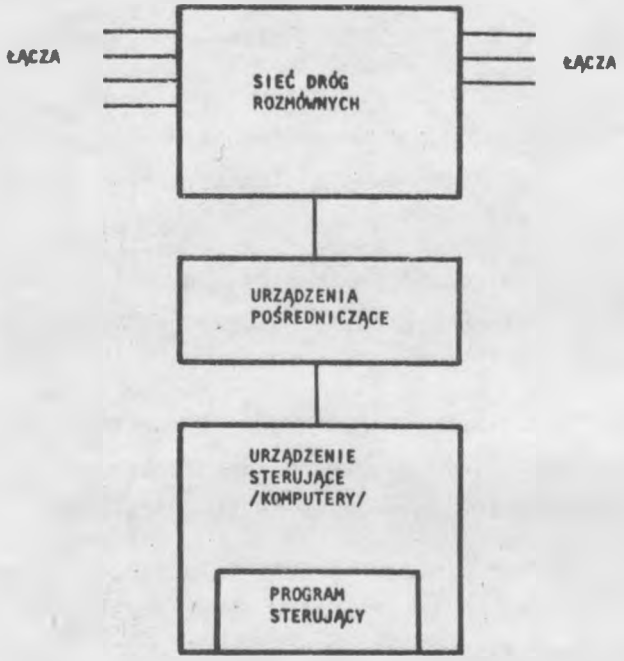
8. Cohen D.: ESPL - 1 - A programming language for switching systems. Software Engineering for Telecommunication Switching Systems, Essex 1973, s. 13-20.

9. Hori Y., Kano S., Kamiyama K., Koizumi K.: High level language for the generation of telephone switching task program. Software Engineering for Telecommunication Switching Systems, Essex 1973, s. 58-67.

10. Kawashima H., Ishino F.: A method of efficiently introducing a high level language into SPC switching system. Software Engineering for Telecommunication Switching Systems, Essex 1973, s. 68-79.

11. Sorgenfrei H.R.: Peripheral requirements relating to the development of a programming language for SPC systems and initial steps toward a solution. Software Engineering for Telecommunication Switching Systems, Essex 1973, s. 223-231.

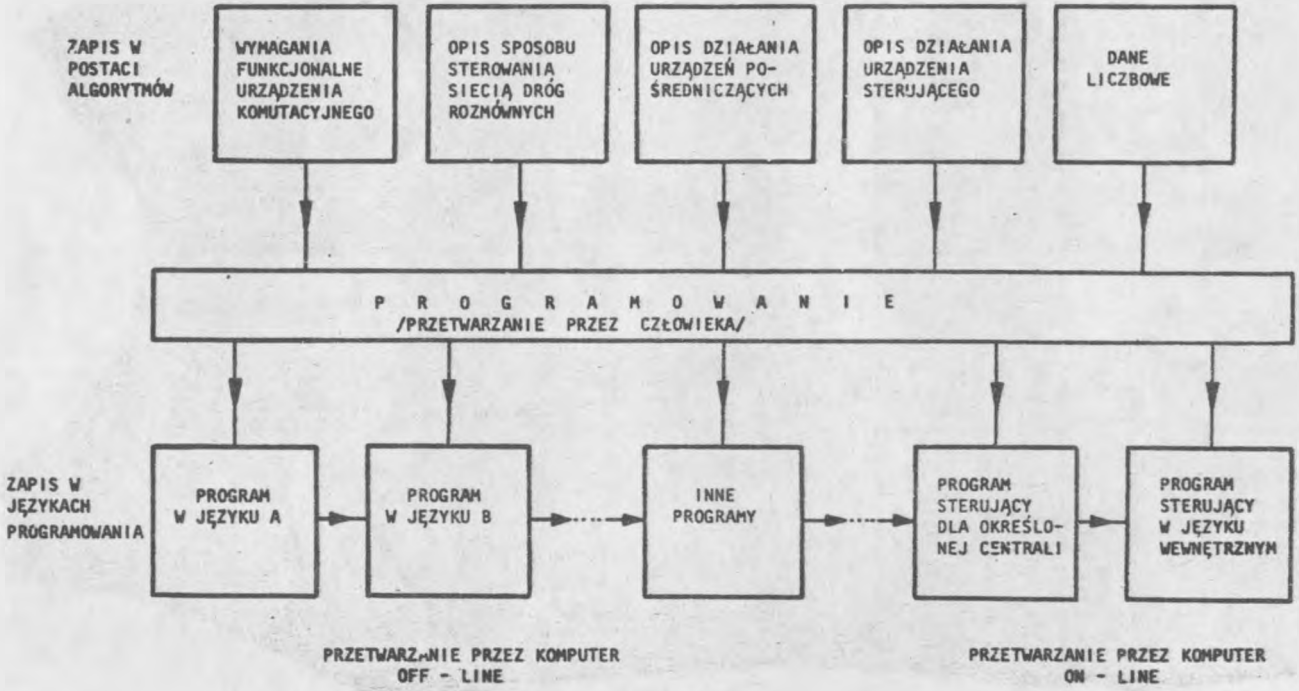
12. Dokumenty Komisji XI CCITT z okresu 1972-1974.


ŁĄCZA
SIEĆ DRÓG ROZMÓWNYCH
ŁĄCZA
URZĄDZENIA POŚREDNICZĄCE
URZĄDZENIE STERUJĄCE /KOMPUTERY/
PROGRAM STERUJĄCY

Rys. 1.

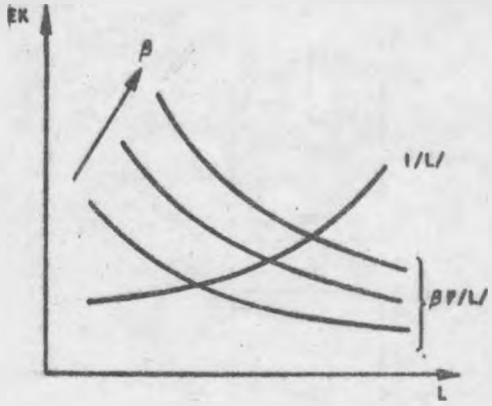

ALGORYTM DZIAŁANIA URZĄDZENIA
PROGRAM STERUJĄCY
WYTWARZANIE PROGRAMU STERUJĄCEGO
PROGRAM STERUJĄCY
RZECZYWISTE DZIAŁANIE URZĄDZENIA
PRACA PROGRAMU STERUJĄCEGO

Rys. 2.

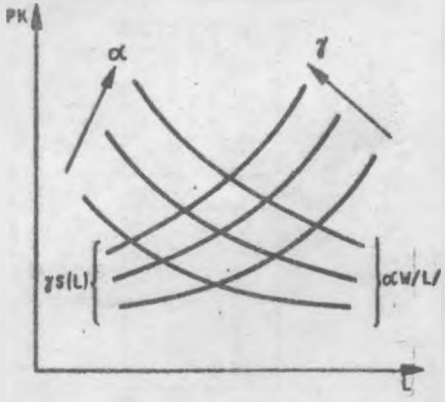

ZAPIS W POSTACI ALGORYTMÓW
WYMAGANIA FUNKCJONALNE URZĄDZENIA KOMUTACYJNEGO
OPIS SPOSOBU STEROWANIA SIECIĄ DRÓG ROZMÓWNYCH
OPIS DZIAŁANIA URZĄDZEŃ POŚREDNICZĄCYCH
OPIS DZIAŁANIA URZĄDZENIA STERUJĄCEGO
DANE LICZBOWE
P R O G R A M O W A N I E
/PRZETWARZANIE PRZEZ CZŁOWIEKA/
ZAPIS W JĘZYKACH PROGRAMOWANIA
PROGRAM W JĘZYKU A
PROGRAM W JĘZYKU B
INNE PROGRAMY
PROGRAM STERUJĄCY DLA OKREŚLONEJ CENTRALI
PROGRAM STERUJĄCY W JĘZYKU WEWNĘTRZNYM
PRZETWARZANIE PRZEZ KOMPUTER OFF - LINE
PRZETWARZANIE PRZEZ KOMPUTER ON - LINE

Rys. 3

Rys. 4.



Rys. 5.



Rys. 6.

Ryszard Kowalik

621.395.3-.503.55

DIAGNOSTYKA STEROWANYCH PROGRAMOWO URZĄDZEŃ TELEKOMUTACYJNYCH

1. WSTĘP

W krajach rozwiniętych oprogramowanie stało się składnikiem urządzeń komutacyjnych w ostatnich latach, w Polsce zaś, dzięki zakupowi licencji na nowoczesny system E-10, w roku bieżącym.
Fakt ten zaistniał w tym samym momencie, gdy nastąpił olbrzymi skok w zakresie technologii sprzętu urządzeń komutacyjnych, a w szczególności w zakresie urządzeń sterujących. Przeskoczone zostały dwie generacje urządzeń: lampowa i tranzystorowa.

Skomplikowany, a równocześnie bardzo niezawodny sprzęt zbudowany z obwodów scalonych spowodował konieczność powstania nowych metod diagnostycznych, istnienie zaś oprogramowania znacznie ułatwiło przeprowadzanie złożonych testów.

W Pracowni Oprogramowania Urządzeń Telekomutacyjnych IŁ prowadzona była praca mająca na celu zebranie informacji o nowoczesnych metodach diagnozowania. Informacje te są niezbędne dla właściwej eksploatacji i dla dalszego rozwijania systemu E-10.

Niniejsze opracowanie stanowi zbiór uporządkowanych pojęć i podstawowych informacji z dziedziny diagnostyki programowania sterowanych urządzeń komutacyjnych, przy czym najwięcej uwagi poświęcono programowo sterowanym centralom telefonicznym.

Dużą niezawodność "a posteriori" w połączeniu z atrakcyjną ceną sprzedaży, prostotą utrzymania przy niskich kosztach eksploatacji można osiągnąć w programowo sterowanych centralach telefonicznych przy zastosowaniu w nich układowo programowych środków diagnostycznych. Przyjęcie takiego rozwiązania umożliwia powierzenie nadzoru

nad złożoną nowoczesną centralą telefoniczną mniej doświadczonemu personelowi, znającemu tylko jej podstawowe zasady funkcjonowania. Dopuszczenie badania podzespołów centrali metodą klasyczną typu człowiek-miernik byłoby zbyt uciążliwe, czasochłonne i ryzykowne - w wyniku ingerencji człowieka wewnątrz skomplikowanej struktury systemu.

Duża niezawodność stosowanych elementów sprawia, że uszkodzenia w nowoczesnym sprzęcie występują bardzo rzadko. Personel nie miałby więc doświadczenia w szybkim lokalizowaniu uszkodzeń. W tej sytuacji wprowadzenie automatycznych układowo-programowych środków diagnostycznych stało się koniecznością.

Do wykrywania i lokalizowania uszkodzeń stosowane są różnorodne techniki [7], jak:

- diagnozowanie w warunkach pracy /on-line/,
- diagnozowanie w warunkach wyłączenia /off-line/,
- technika zapisu błędu,
- automatyczne procedury przywracania sprawności,
- mikrodiagnostyka,
- panele badaniowe,
- układy kontrolne.

2. POJĘCIA PODSTAWOWE

Pod pojęciem diagnostyki rozumiemy: metody, środki i czynności umożliwiające wykrywanie i lokalizowanie uszkodzonych elementów w badanym systemie. Uszkodzenie jest to awaria w strukturze systemu, natomiast błąd jest funkcjonalnym objawem uszkodzenia. Diagnozowanie jest działaniem zmierzającym do wykrycia uszkodzenia i jeżeli obecność jego zostanie wykryta, to diagnozowanie zawiera w sobie również czynności zmierzające do zlokalizowania uszkodzenia. Efektem diagnozowania jest postawienie diagnozy, tj. decyzji o stanie systemu i miejscu jego ewentualnego uszkodzenia. Rozróżniamy:

- diagnozę stanu i
- diagnozę uszkodzenia.

Diagnoza stanu jest to decyzja o stanie systemu, która może być dwojaka: system sprawny, system uszkodzony. Diagnoza uszkodzenia jest decyzją o rodzaju i miejscu uszkodzenia w strukturze systemu. Jest to decyzja typu: element X ma uszkodzenie rodzaju Y [21].

Czynności poprzedzające diagnozę stanu nazywamy procesem kontroli, a czynności poprzedzające diagnozę uszkodzenia - procesem lokalizacji. Efektem procesu kontroli jest detekcja /wykrycie/ - czyli stwierdzenie obecności uszkodzenia w systemie, a efektem procesu lokalizacji jest identyfikacja /zlokalizowanie/, tzn. określenie miejsca uszkodzenia w strukturze systemu. Zlokalizowanie może być jednoznaczne - z dokładnością do pojedynczego uszkodzenia lub wieloznaczne - z dokładnością do podzbioru uszkodzeń.

Proces diagnozowania systemu polega na testowaniu, tzn. na wprowadzeniu różnych sygnałów testowych w odpowiednie wejściowe punkty testowe oraz kontrolowaniu i ocenianiu wynikających z tego odpowiedzi w określonych wyjściowych punktach testowych.

W przypadku dużych systemów /ze strukturalnego punktu widzenia/ czas testowania i długość sekwencji testowej mogą być duże. Z tych przyczyn system powinien być podzielony na mniejsze segmenty - jednostki funkcjonalne /podsystemy/, testowalne pojedynczo lub w kombinacji z innymi jednostkami funkcjonalnymi [1].

3. PROGRAMY DIAGNOSTYCZNE

Diagnozowanie programowo sterowanych central telefonicznych przeprowadzane jest przy użyciu tzw. programów diagnostycznych, które z punktu widzenia ich przeznaczenia dzielą się na:

- detekcyjne,
- lokalizujące i
- uniwersalne.

Programy detekcyjne przeznaczone są do diagnozowania stanu centrali. Obejmują one swym zasięgiem cały kompleks centrali lub jej poszczególne jednostki. Programy detekcyjne uruchamiane są zwykle z ustaloną częstotliwością powtarzania /np. 2 razy dziennie/ lub na życzenie personelu. Działanie tych programów polega na przeprowadzaniu pewnej liczby sekwencji funkcjonalnych i badaniu ich poprawnego wykonania. Sekwencje funkcjonalne są to zbiory elementarnych operacji wykonywanych przez system w normalnych warunkach pracy. Do programów detekcyjnych zaliczane są, między innymi, programy weryfikujące faktyczny stan sieci dróg rozmównych z danymi o tej sieci zapisanymi w pamięci centralnej jednostki sterującej. Innym przykładem są programy weryfikujące działanie systemu operacyjnego również na bazie tych samych danych. Poza okresowo uruchamianymi programami detekcyjnymi, programowo sterowane centrale telefoniczne bywają wyposażone w krótkie procedury detekcyjne wtrącone do programów operacyjnych. Tego typu procedury wykonywane są w sposób dynamiczny /w czasie normalnej pracy systemu/ i wprost proporcjonalny do natężenia ruchu obsługiwanego przez centralę.

Wszystkie programy detekcyjne oparte są na procedurze kombinacyjnej /patrz pkt. 4/.

Diagnozowanie stanu centrali środkami programowymi jest w zasadzie zabiegiem profilaktycznym, który eksploatuje system w sposób nieefektywny, przyspieszając proces starzenia się niektórych jego elementów.

Uszkodzenia wykryte przez programy lub procedury detekcyjne są następnie identyfikowane przez programy lokalizujące. Działanie programów lokalizujących trwa znacznie dłużej od działania programów detekcyjnych. Pomimo że lokalizowanie uszkodzenia zawiera w sobie również jego detekcję, to jednak ze względu na kryterium czasu nie stosuje się programów lokalizujących do diagnozowania stanu systemu. Lokalizowanie uszkodzenia jest przeprowadzane zwykle w warunkach wyłączenia podsystemu z pracy. Programy lokalizujące mogą

być oparte zarówno na procedurze sekwencyjnej, jak i kombinacyjnej.

Istnieją również takie programy, które z jednakowym powodzeniem mogą być wykorzystywane do diagnozowania stanu i do lokalizowania ewentualnych uszkodzeń. Programy tego typu, stosowane zwykle na etapie instalowania centrali, oparte są na procedurze BOOTSTRAP, której opis znajduje się w punkcie 4.

Trudniejszą do opracowania grupę programów diagnostycznych stanowią programy uniwersalne, oparte zawsze na procedurze sekwencyjnej. Przeznaczone są one do diagnozowania stanu i dynamicznego lokalizowania wykrytych uszkodzeń. Opis zastosowanej w nich procedury sekwencyjnej znajduje się również w punkcie 4.

Specyficznym rodzajem diagnostyki programowej w programowo-sterowanych centralach telefonicznych jest diagnozowanie systemu przy obniżonych lub podwyższonych poziomach napięć zasilających, w zakresie dopuszczalnych tolerancji. Celem tych zabiegów jest prognozowanie uszkodzeń, umożliwiające wcześniejszą wymianę podzespołów o dużym prawdopodobieństwie awarii w najbliższej przyszłości.

Każdy program diagnostyczny składa się z co najmniej dwóch podstawowych elementów - procedury diagnostycznej i zbioru danych. W zbiorze danych, stanowiącym element tablicowy programu diagnostycznego, zawarte są /rys. 1/[x]:

- dane sygnałów testowych /wejścia/,
- wzorcowe odpowiedzi testów /wyjścia/,
- komunikaty do wydrukowania,
- numery kolejnych testów /ale tylko w przypadku procedury sekwencyjnej/.

Opcjonalnym elementem programu diagnostycznego jest mniej lub bardziej złożona procedura sterująca testowaniem, która równocześnie stanowi interfejs pomiędzy programem a operatorem.

Testy procedury diagnostycznej przedstawiają sobą prawdopodobne

---

[x] Wszystkie rysunki są zamieszczone na końcu artykułu.

lub nieprawdopodobne /przy normalnej pracy systemu/ kombinacje sygnałów testowych, wprowadzanych w wejściowe punkty testowe diagnozowanego podsystemu. Uzyskane odpowiedzi są następnie porównywane z odpowiedziami wzorcowymi, dając w efekcie wyniki testów. Otrzymywane kolejne wyniki są zapisywane lub nie, w zależności od rodzaju programu i rodzaju zastosowanej procedury.

4. PROCEDURY DIAGNOSTYCZNE

Ze względu na złożoność podzespołów programowo sterowanej centrali telefonicznej wykrycie lub zlokalizowanie w niej ewentualnego uszkodzenia wymaga wykonania dostatecznie dużej liczby testów uporządkowanych w procedurę diagnostyczną. Istnieją dwa podstawowe rodzaje procedur diagnostycznych [3,14,20,22]:

- procedury kombinacyjne i
- procedury sekwencyjne.

W procedurze kombinacyjnej /rys. 2/ wykonywana jest zawsze stała seria N testów na testowanym podsystemie, w kolejności ustalonej na etapie projektowania procedury. Wyniki przeprowadzanych testów, w przypadku programu lokalizującego, są na bieżąco zapisywane w pamięci testera. Przypisując pozytywnemu wynikowi testu "0" oraz "1" wynikowi negatywnemu po wykonaniu wszystkich testów otrzymuje się N-bitową kombinację /ze zbioru 2^N możliwych kombinacji/. Ze względu na taką formę wyniku diagnozowania procedurę tę nazwano procedurą kombinacyjną. Otrzymany wynik diagnozowania jest następnie przekształcany do postaci cyfrowej w zapisie dziesiętnym lub ósemkowym. Zabieg ten jest niezbędny ze względu na to, że liczba testów w niektórych przypadkach może być bardzo duża, rzędu kilkuset. Ostateczne postawienie diagnozy co do miejsca ewentualnego uszkodzenia polega na wyszukaniu otrzymanego wyniku w słowniku diagnostycznym i odczytaniu skojarzonej z nim informacji. Sam słownik diagnostyczny stanowi listę wyników i komentarzy, uszeregowanych w kolejności wy-

godnej do wyszukiwania. Zdarzyć się może, szczególnie w początkowym etapie eksploatowania systemu, że otrzymany wynik diagnozowania wcale nie istnieje w dysponowanym słowniku. Sytuacja taka możliwa jest wówczas, gdy ewentualność wystąpienia pewnego uszkodzenia została przeoczona podczas projektowania procedury. W takim przypadku zachodzi potrzeba interwencji personelu, który za pomocą testów ręcznych lub innych procedur zlokalizuje uszkodzenie i przeprowadzi aktualizację słownika /przez dopisanie nowego, nieznanego dotychczas wyniku/. Bezowocne poszukiwanie wyniku w słowniku diagnostycznym może mieć również miejsce wówczas, gdy w systemie wystąpi uszkodzenie o charakterze przemijającym lub krótkotrwałym.

W procedurze sekwencyjnej /rys. 3/, zwanej inaczej dynamiczną, wykonywane testy nie stanowią, jak w przypadku poprzednim, sztywnej serii, lecz kolejny test uzależniony jest od wyniku testu poprzedniego. Ideą tej procedury jest stopniowe ograniczanie podzbioru elementów, wśród których znajdować się może element uszkodzony. Dla podzbioru uszkodzeń, dających identyczny wynik testu, przeprowadza się kolejny test, umożliwiający bliższe określenie zaistniałego uszkodzenia. Procedurę tę nazwano sekwencyjną ze względu na sekwencyjny sposób analizy wyników testów.

Procedura sekwencyjna usuwa potrzebę stosowania słownika diagnostycznego, ale z drugiej strony wymaga większej pamięci testera. Główną zaletą procedury sekwencyjnej jest to, że znacznie redukuje liczbę testów potrzebnych do zlokalizowania uszkodzenia, co niejednokrotnie pozwala na skrócenie czasu diagnozowania.

Jeżeli założyć, że w systemie wystąpiło uszkodzenie pojedyncze, to procedura kombinacyjna będzie nieefektywna, ponieważ wymaga wykonywania wszystkich testów za każdym razem, gdy wystąpi uszkodzenie. W takiej sytuacji bardziej uzasadnione byłoby stosowanie procedury sekwencyjnej. W przypadku uszkodzeń złożonych rzecz ma się zupełnie odwrotnie.

Obydwa omawiane tu rodzaje procedur opracowywane są dla znanych uszkodzeń, a więc niekoniecznie dla wszystkich. Przewidzenie wszy-

stkich możliwych uszkodzeń jest zadaniem wyjątkowo trudnym, o ile w ogóle możliwym. Dlatego też procedura kombinacyjna, która zwykle charakteryzuje się nadmiarem, stanowi lepsze rozwiązanie problemu. Przez nadmiar należy rozumieć tu wielokrotne /przy jednorazowym wykonaniu programu diagnostycznego/ obejmowanie pewnych uszkodzeń przez kilka testów /np. uszkodzenie X może być objęte testem 2,10..., n-tym/. Istniejący w procedurze kombinacyjnej nadmiar pozwala na efektywniejsze wykrywanie i lokalizowanie uszkodzeń krótkotrwałych.

Procedurą scalającą w sobie cechy procedury kombinacyjnej i sekwencyjnej jest procedura BOOTSTRAP /rys. 4/. Pierwszy test tej procedury obejmuje swym zasięgiem niewielką liczbę elementów z całego podsystemu. Każdy następny test dodaje niewielką liczbę elementów do już sprawdzonych. Jeżeli wynik któregokolwiek testu okaże się negatywny, oznaczać to będzie, że uszkodzenie znajduje się w podzbiorze elementów dodanym w tym teście. Procedura tego typu jest szczególnie efektywna w przypadku uszkodzeń złożonych. Zlokalizowanie uszkodzenia przy użyciu procedury BOOTSTRAP następuje równocześnie z jego wykryciem. Ze względu na zdolność "wychwytywania" pojedynczego uszkodzenia w zbiorze dużej liczby uszkodzeń procedura BOOTSTRAP stosowana jest powszechnie na etapie instalowania centrali w tzw. testach systemu, przed przystąpieniem do którego zakłada się istnienie w systemie względnie dużej liczby uszkodzeń.

Procedura uniwersalna /rys. 5/ ze względu na swoją topologię ma charakter procedury sekwencyjnej. Pierwszy test w tej procedurze obejmuje swym zasięgiem dużą liczbę elementów systemu, a po uzyskaniu pozytywnego wyniku z tego testu następuje kolejny test, obejmujący inną dużą liczbę elementów. Procedura ta w warunkach braku uszkodzenia pozwala zrealizować szybkie badanie systemu, natomiast w przypadku istnienia uszkodzenia doprowadza do szybkiej lokalizacji.

Ze względu na złożoność procedur sekwencyjnych i uniwersalnych ich błędne zaprojektowanie może doprowadzić do sytuacji, w której lokalizowane będą uszkodzenia nie istniejące.

5. TESTER

Do przeprowadzania testów niezbędne jest urządzenie testujące - tester, generujące sygnały testowe i oceniające otrzymywane odpowiedzi jako pozytywne albo negatywne. Ocena testera - wynik diagnozowania będzie wiarygodna, o ile tester będzie nie uszkodzony. Oznacza to, że tester powinien charakteryzować się niezawodnością znacznie przewyższającą niezawodność testowanego systemu. Z tego względu tester jako część systemu nazywany jest jego niezawodnym rdzeniem. Wymaganą niezawodność testera osiąga się w praktyce przez stosowanie w nim elementów o dużej niezawodności, elementów nadmiarowych, układów o logice głosującej [28], a nawet nadmiaru w jego systemie operacyjnym. Dla przykładu porównanie otrzymanej odpowiedzi z odpowiedzią wzorcową może polegać na wykonaniu operacji odejmowania od siebie dwóch wartości binarnych albo na wykonaniu operacji SUMA MODULO 2 i badaniu tak uzyskanego wyniku [4].

Aby tester wykrywał i lokalizował uszkodzone elementy w testowanym systemie, musi mieć do niego odpowiedni dostęp, tzn. dostateczną liczbę punktów testowych. Problem dostępu uzależniony jest od lokalizacji testera względem testowanego systemu. Rozróżnia się testery:

- zewnętrzne i
- wewnętrzne.

Tester zewnętrzny jest fizycznie oddalony od systemu i przyłączany do niego na czas testowania. Przy użyciu testera zewnętrznego możliwe jest w zakresie diagnozowania stanu równoległe testowanie kilku identycznych podsystemów. Testerem zewnętrznym jest zwykle mały komputer, który może być równocześnie wykorzystany do lokalizowania uszkodzeń w wyłączonym podzespole systemu. Systemy z zewnętrznym niezawodnym rdzeniem diagnozowane są najczęściej w oparciu o strategię rozszerzania testera /patrz pkt. 6/.

Punktami testowymi mogą być funkcjonalne punkty wejściowo-wyjścio-

we danego podsystemu lub specjalne punkty testowe zaprojektowane z myślą o zwiększeniu efektywności i diagnozowania. Wprowadzenie dodatkowych punktów testowych może okazać się niebezpieczne ze względu na potencjalne obniżenie niezawodności całego systemu.

Zagadnienie lokalizacji testera i jego dostępu do systemu może być rozwiązane na drodze samodiagnostyki. W takim przypadku tester nie stanowi odrębnego urządzenia, lecz wszystkie jego elementy biorą czynny udział w normalnej pracy systemu. Diagnozowanie systemu testerem wewnętrznym ma trzy podstawowe zalety:

- nie jest potrzebne dodatkowe urządzenie testujące,
- uzyskuje się bardziej bezpośredni dostęp do elementów systemu,
- upraszczają się procedury diagnostyczne.

Ogólnie za system samodiagnostyczny uważa się taki, który samodzielnie w sposób automatyczny jest w stanie:

- wykryć obecność uszkodzenia, a następnie
- zlokalizować je z dokładnością do wymienialnego modułu lub podzbioru takich modułów.

W obrębie procesorów central coraz częściej spotyka się niższy, ale za to o wiele precyzyjniejszy poziom diagnostyki, tzw. mikrodiagnostykę. Mikrodiagnostyka możliwa jest do zastosowania tylko w procesorach mikroprogramowanych, tj. sterowanych mikroprogramami umieszczonymi w pamięci mikroprogramów. Mikrodiagnostyka ma kapitalne znaczenie w przypadku uszkodzeń katastrofalnych powodowanych awarią centralnego procesora. W razie wystąpienia takiej sytuacji diagnozowanie procesora może być przeprowadzane po przyłączeniu dodatkowej pamięci z mikroprogramami diagnostycznymi.

6. STRATEGIE DIAGNOZOWANIA

Strategia diagnozowania systemu uzależniona jest od jego struktury, wymaganej niezawodności i orientacji testera względem systemu.

Od struktury systemu uzależniona jest elastyczność i rozmiar programowej diagnostyki oraz głębokość wykrywania i lokalizowania uszkodzeń. Wymagana niezawodność systemu determinuje rodzaj i charakter programów diagnostycznych, natomiast orientacja testera określa efektywność i sposób przeprowadzania testów.

Możliwych jest co najmniej pięć podstawowych strategii diagnozowania programowo sterowanej centrali telefonicznej:

1/ strategia rozszerzania testera,
2/ strategia kolejnych przybliżeń,
3/ strategia zapisu błędu,
4/ strategia słownika diagnostycznego i
5/ strategia detekcji - obejmowania uszkodzeń.

W celu zastosowania strategii rozszerzania testera system musi być podzielony w sposób logiczny - a zatem i strukturalny - na kilka poziomów testowania. Przed przystąpieniem do diagnozowania w sposób ręczny wykonywana jest weryfikacja stanu niezawodnego rdzenia. W przypadku gdy istnieją w systemie dwa lub więcej niezawodnych rdzeni, nie zachodzi potrzeba wykonywania testów ręcznych [3,5]. Po uzyskaniu pozytywnego wyniku weryfikacji przechodzi się do fazy automatycznego diagnozowania. Na wstępie, niezawodny rdzeń testuje elementy poziomu 1 i, o ile nie występuje w nich uszkodzenie, niezawodny rdzeń wraz z elementami poziomu 1 przystępuje do testowania elementów poziomu 2 itd. /rys. 6/.

Strategia rozszerzania testera jest szczególnie efektywna i uzasadniona dla uszkodzeń złożonych. Opiera się ona na procedurze BOOTSTRAP. Strategia ta stosowana jest w większym lub mniejszym stopniu we wszystkich sterowanych programowo centralach telefonicznych. Dla przykładu, diagnozowanie pewnego obszaru może być realizowane np. wówczas, gdy osiągnięty zostanie do niego odpowiedni dostęp poprzez inne nie uszkodzone podzespoły centrali. Wówczas w pierwszej kolejności testowane są podzespoły dostępu, a dopiero po otrzymaniu pozytywnych wyników z tak przeprowadzonych testów przystępuje się do testowania wybranego obszaru.

W strategii kolejnych przybliżeń /stosowanej do lokalizowania uszkodzeń/ programy diagnostyczne oparte są na procedurze sekwencyjnej.

W strategii zapisu błędu [16] stan systemu diagnozowany jest w sposób dynamiczny bądź przy użyciu układów kontrolnych, bądź w sposób programowy przy użyciu programów lub procedur detekcyjnych; strategia ta stosowana jest najczęściej w odniesieniu do podsystemu urządzeń peryferyjnych. Realizacja jej polega na kontrolowaniu działania całego systemu poprzez nadzorowanie wszystkich jednostek funkcjonalnych w sposób statystyczny, tzn. mierząc ich stopy błędów /stopa błędu jest to stosunek liczby operacji kontrolnych o wynikach negatywnych do liczby wszystkich operacji/. W strategii tej, każdej jednostce funkcjonalnej przydzielony jest licznik operacji kontrolnych o wynikach negatywnych i licznik wszystkich operacji kontrolnych. Stany liczników uaktualniane są wówczas, gdy jednostka funkcjonalna jest objęta operacją kontrolną. Identyfikowanie uszkodzonej jednostki polega na porównywaniu obliczanych aktualnych stóp błędów ze stopami progowymi /ustalonymi wcześniej na drodze teoretycznej lub na podstawie doświadczeń z eksploatacji systemu/. Porównywania i obliczenia aktualnych stóp błędów dokonuje uruchamiany okresowo niewielki program analizujący. Zaletami strategii zapisu błędu są:

- prostota i elastyczność,
- krótki, a przy tym niezmienny czas analizy liczników /zarówno w warunkach uszkodzenia, jak i nieuszkodzenia/,
- wykrywanie i lokalizowanie uszkodzeń wszystkich typów z dokładnością do jednostki funkcjonalnej,
- brak priorytetów diagnostycznych, gdyż jest to metoda pasywna,
- dowolność ustalania czułości reakcji /tzn. poziom progowej stopy błędu może być zmieniany dowolnie dla każdej jednostki/,
- brak wzajemnego oddziaływania procesów połączeniowych i procesu zapisu błędu.

W strategii słownika diagnostycznego [5] programy diagnostyczne oparte są na procedurach kombinacyjnych. Strategia ta stosowana jest jedynie w odniesieniu do lokalizowania uszkodzeń wykrytych wcześniej przez programy detekcyjne. Program lokalizujący wykonywany jest zawsze w całości w każdym przypadku wystąpienia uszkodzenia.

Strategia obejmowania uszkodzeń stosowana jest w zakresie diagnostyki stanu. Opiera się zawsze na procedurze kombinacyjnej.

7. PROCES DIAGNOSTYCZNY W PROGRAMOWO STEROWANEJ CENTRALI TELEFONICZNEJ

Typowy proces diagnostyczny w programowo sterowanej centrali telefonicznej podzielić można na cztery, zwykle oddzielnie realizowane etapy /rys. 7/. Na etapie pierwszym diagnozowany jest stan centrali i, o ile wówczas nastąpi detekcja uszkodzenia, to następnie metodą kolejnych przybliżeń /etapy 2, 3 i 4/ ma miejsce proces lokalizowania uszkodzenia.

Ponieważ wstrzymanie pracy centrali jest niedopuszczalne nawet wówczas, gdy chcemy dokonać jej przeglądu, diagnozowanie jej stanu odbywać się musi w warunkach dynamicznych, na jednostkach funkcjonalnych, biorących aktywny udział w procesach komutacyjnych.

Z silną koncentracją funkcjonalną sterowania w programowo sterowanych centralach telefonicznych związane jest potencjalne niebezpieczeństwo znacznego zmniejszenia operatywności centrali w przypadku wystąpienia nawet pojedynczego uszkodzenia. Obecność uszkodzenia, szczególnie w jednostkach sterujących, powinna być wykryta natychmiast z chwilą jego wystąpienia, aby w porę dokonać odpowiedniej rekonfiguracji systemu, nie dopuszczając w ten sposób do gromadzenia się uszkodzeń i ich wpływu na pracę centrali.

Natychmiastowe wykrycie uszkodzenia jest nieodzowne głównie w odniesieniu do trudnych do zlokalizowania uszkodzeń krótkotrwałych, nawiasem mówiąc uszkodzeń stanowiących "lwią część" wszystkich uszkodzeń występujących w systemach cyfrowych.

W odniesieniu więc do diagnostyki stanu wymaganie takie wyklucza możliwość stosowania programów testowych, jako pierwszoplanowego środka do wykrywania uszkodzeń. Muszą być stosowane układy kontrolne /testery układowe [1,9], a zasada ta dotyczy wszystkich aktualnie eksploatowanych programowo sterowanych central telefonicznych.

Diagnostyka układowa ma przewagę nad programową właśnie w układowym wykrywaniu uszkodzeń. Działa ono w sposób ciągły i nie wymaga zużywania na cele kontrolne efektywnego czasu pracy ewentualnie sprawnej jednostki centralnej.

Specyficznym rodzajem diagnozowania stanu jest weryfikacja naprawionych podsystemów.

Testery układowe pracujące w centralach elektronicznych przeznaczone są głównie do kontrolowania informacji przesyłanych w jednostkach funkcjonalnych lub wymienianych pomiędzy jednostkami centrali. Testery te wykrywają błędy /a w ten pośredni sposób występujące uszkodzenia/ w oparciu o kody arytmetyczne lub kody funkcjonalne. Spośród kodów arytmetycznych, najczęściej stosowanym ze względu na swoją prostotę jest kod parzystości.

Jednak nie każda operacja czy informacja może być sprawdzona jedynie w oparciu o układy kontrolne [16], a stosowanie ich w nadmiernej ilości doprowadzić mogłoby do obniżenia niezawodności systemu [10]. Z drugiej strony uszkodzenia w samych układach kontrolnych mogą spowodować to, że układy te nie wygenerują sygnału błędu w warunkach uszkodzenia. Muszą więc być stosowane testy programowe sprawdzające przynajmniej zdolność układów kontrolnych do wykrywania uszkodzeń [1]. W związku z tym daje się zauważyć tendencję do ograniczania liczby układów kontrolnych do niezbędnego minimum, widząc w tym możliwość polepszenia niezawodności systemu. Nie oznacza to bynajmniej, że w zakresie diagnozowania stanu rezygnuje się całkowicie z układów kontrolnych i przekazuje funkcje przez nie spełniane procedurom diagnostycznym. Ograniczenie liczby układów kontrolnych możliwe jest jedynie wówczas, gdy zapewnimy dużą jednorodność struktu-

ry jednostek centrali bądź wówczas, gdy układom kontrolnym przypiszemy funkcje centralne. Dla przykładu pojedynczy układ kontrolny nadzorować może równocześnie kilka jednostek funkcjonalnych centrali. Taka funkcja układów kontrolnych stosowana bywa najczęściej w odniesieniu do pól komutacyjnych i ich wyposażeń.

Natychmiast po wykryciu uszkodzenia musi być podjęte działanie, mające na celu przywrócenie sprawności centrali. Stosuje się w tym celu automatyczne procedury przywracania sprawności. Zadaniem procedur przywracania sprawności jest zlokalizowanie uszkodzenia na poziomie podsystemu, a następnie przeprowadzenie rekonfiguracji systemu przy użyciu jednostek rezerwowych. Przywracanie sprawności może przebiegać w dwojaki sposób:

- układowo-programowy lub
- programowo-układowy.

Układowo-programowe przywracanie sprawności polega na tym, że każda nowa konfiguracja systemu zestawiana jest automatycznie środkami układowymi, po czym utworzona konfiguracja jest diagnozowana w sposób programowy. W programowo-układowym przywracaniu sprawności najpierw lokalizowany jest uszkodzony podsystem, po czym przeprowadzana jest rekonfiguracja systemu. Zakończenie działania tych procedur następuje po przetestowaniu utworzonej konfiguracji i postawieniu pozytywnej diagnozy stanu.

8. ZAKOŃCZENIE

Diagnostyka jest jeszcze młodą, rozwijającą się dziedziną nauki i na swoim aktualnym poziomie nie dysponuje metodami umożliwiającymi opracowywanie efektywnych testów dla wszystkich typów uszkodzeń.

Programy diagnostyczne dla większości systemów cyfrowych projektowane są zwykle przy założeniu, że podczas testowania istnieć może co najwyżej uszkodzenie pojedyncze. Sprawność tak opracowanych programów może się znacznie obniżyć, gdy w systemie wystąpi jedno-

cześnie więcej niż jedno uszkodzenie. Dostatecznie częste uruchamianie profilaktycznych programów testujących umożliwia obniżenie prawdopodobieństwa wystąpienia takiej sytuacji.

Spodziewana gęstość uszkodzeń w systemie determinuje wybór odpowiedniej procedury testującej. Dla uszkodzeń złożonych istnieje wyjątkowo efektywna strategia diagnozowania, oparta wyłącznie na środkach programowych. Jest nią tzw. technika rozszerzania testera /bootstrap/. Programy diagnostyczne mogą być efektywne jedynie wówczas, gdy będą projektowane równolegle z projektowaniem systemu. Taka współbieżność jest o tyle istotna, że niejednokrotnie w celu zwiększenia skuteczności programu diagnostycznego okaże się wręcz niezbędne umieszczenie dodatkowych punktów testowych lub wprowadzenie modyfikacji w strukturze systemu. Dla przykładu, w testowaniu wielopoziomowym /ułatwiającym diagnozowanie uszkodzeń złożonych/ z góry zakłada się, że każda jednostka funkcjonalna musi być wyposażona w swój własny tester układowy zaprojektowany oczywiście wówczas, gdy powstaje projekt tej jednostki.

Analizując dostępne publikacje traktujące o diagnostyce stwierdzono, że czasy "życia" programów diagnostycznych w ich niezmienionej formie mogą być względnie krótkie, a wynikać to może z następujących czterech przyczyn [1,26]:

- z niekompletności programów diagnostycznych określonej liczbą uszkodzeń wykrywanych i lokalizowanych,
- z małej rozróżnialności uszkodzeń /np. pewne uszkodzenia mogą być lokalizowane na poziomie 2,3 lub więcej pakietów, wówczas gdy zainteresowani jesteśmy w lokalizowaniu ich na poziomie jednego pakietu/,
- z faktu, że programy diagnostyczne opracowywano równolegle z projektowaniem systemu, nie znając wówczas ani rozkładu prawdopodobieństwa występowania uszkodzeń, ani czasów wykonywania testów,
- ze zmiany rozkładu prawdopodobieństwa uszkodzeń obiektu, wynika-

jącej z zastosowania pewnej liczby elementów elementami o niezawodności na przykład wyższej lub z dokonanej modyfikacji w strukturze obiektu.

W systemie central bezobsługowych, w którym jedno centrum eksploatacji nadzoruje względnie dużą liczbę central /np. osiem, jak w systemie E-10/, dużym udogodnieniem byłaby możliwość dokonywania /w sposób zdalny/ zmian w programach rezydujących w pamięci /lub w pamięciach/ centrali. W przypadku istnienia takiej możliwości pewna część programów diagnostycznych /np. wszystkie programy lokalizujące uszkodzenie na poziomie pakietu/ mogłaby być przechowywana w jednym centrum, wspólnym dla kilku central, i dopiero wówczas, gdy zachodziłaby potrzeba, programy te mogłyby być wpisywane do pamięci w miejsce rezydujących programów operacyjnych. Oczywiście wymazany fragment programów operacyjnych centrali byłby wpisany ponownie po ukończeniu diagnozowania. Rozwiązanie takie jest proste oraz atrakcyjne z ekonomicznego punktu widzenia. Rozwiązanie takie z pewnymi ograniczeniami istnieje już w systemie EWS 1.

WYKAZ LITERATURY

1. Chang H.Y., Heimbinger G.W., Sense D.J., Smith T.L.: Maintenance techniques of a microprogrammed self-checking control complex of an electronic switching system. IEEE Trans. Comp. 1973 nr 5.

2. Bashkow T.R., Priets J., Karson A.: A programming system for detection and diagnosis of machine mulfunctions. IEEE Trans. Electron. Comp. 1963 nr 2.

3. Ramamoorthy C.V., Chang L.C.: System modelling and testing procedures for microdiagnostics. IEEE Trans. Comp. 1972 nr 11.

4. Corvec R., Fer A.: Test et localisation d'avarie dans l'unite logique d'un multie-uegistreur Pericles. Commut. et Electron. 1972 nr 7.

5. Downing W.R., Nowak J.S., Tuomenoksa L.S.: No 1 ESS Maintenance Plan. BSTJ 1964 nr 5, Part. 1.

6. Hilsley R.W.: Martex engineering and maintenance facilities. Point Point Commun. 1972 nr 1.

7. Dent J.J.: Diagnostic engineering requirements. AFIPS Proc. of SJCC 1968.

8. Starski M.: Niezawodność i eksploatacja urządzeń elektronicznych. Warszawa: WNT 1972.

9. Cook R.W., Sissin W.H., Storey T.F., Toy W.N.: Design of a self-checking microprogramm control. IEEE Trans. Comp. 1973 nr 3.

10. Cillot J.W., Simon T.: Telemaintenance des ensembles Platon. Commut. et Electron. 1970 nr 31.

11. Preparata F.P., Gernot Metze, Cmien R.T.: On the connection assignment problem of diagnosable systems. IEEE Trans. Comp. 1967 nr 6.

12. Beuscher H.J., Fessler G.E., Huffman D.W., Kennedy P.J., Nussbaum E.: Administration and maintenance plan. BSTJ 1969 nr 8.

13. Chang H.Y.: Figures of merit for the diagnostics of digital system. IEEE Trans. Reliab. 1968 nr 3.

14. Chang H.Y., Tholms W.: Methods of interpreting diagnostic data for locating faults in digital machines. BSTJ 1967 nr 2.

15. Tsiang S.H., Urlich W.: Automatic trouble diagnosis of computer logic circuits. BSTJ 1962 nr 4.

16. Treves S.R., Dupieux J.G., Henrion H.A.: Exploratory pulse code modulation integrated switching and transmission system for local networks. Electr. Commun. 1972 nr 2.

17. Rousier M., Grall Ph.: Maintenance du systeme Platon. Commut. et Electron. 1971 nr 32.

18. Martin J.T.: Programowanie maszyn cyfrowych w systemach uwarunkowanych czasowo. Warszawa: WNT 1970.

19. Avizienis A.: Arithmetic error codes - cost and efectiveness studies for application in digital system design. IEEE Trans. Comp. 1971 nr 11.

20. Brule J.D., Jonnson R.A., Kletsky E.J.: Diagnosis of equipment failure. IRE Trans. Reliab. 1960 vol. RQC-9.

21. Klimowicz J.: Samodiagnostyka maszyn cyfrowych. Nowości ETO 1970 nr 2.

22. Bennetts R.G., Lewin D.W.: Fault diagnosis of digital systems. Brit. Comp. J. 1971 nr 2.

23. Hangk G., Tsiang S.H., Zimmerman L.: System testing of the No 1 ESS. BSTJ 1964 nr 5, Part 2.

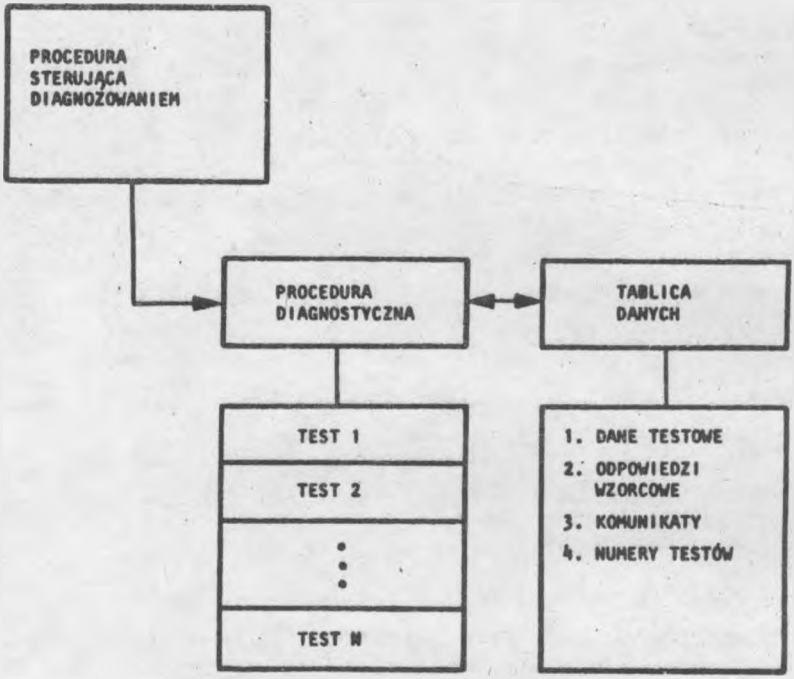
24. Tokuyama G., Honma T.: Diagnosis in the DEX-1 central control. Rev. ECL 1970 nr 9-10.

25. Fukuda S., Suzoki S., Murata T.: Test programs for DEX-1 speech path equipment. Rev. of ECL 1969 nr 10.

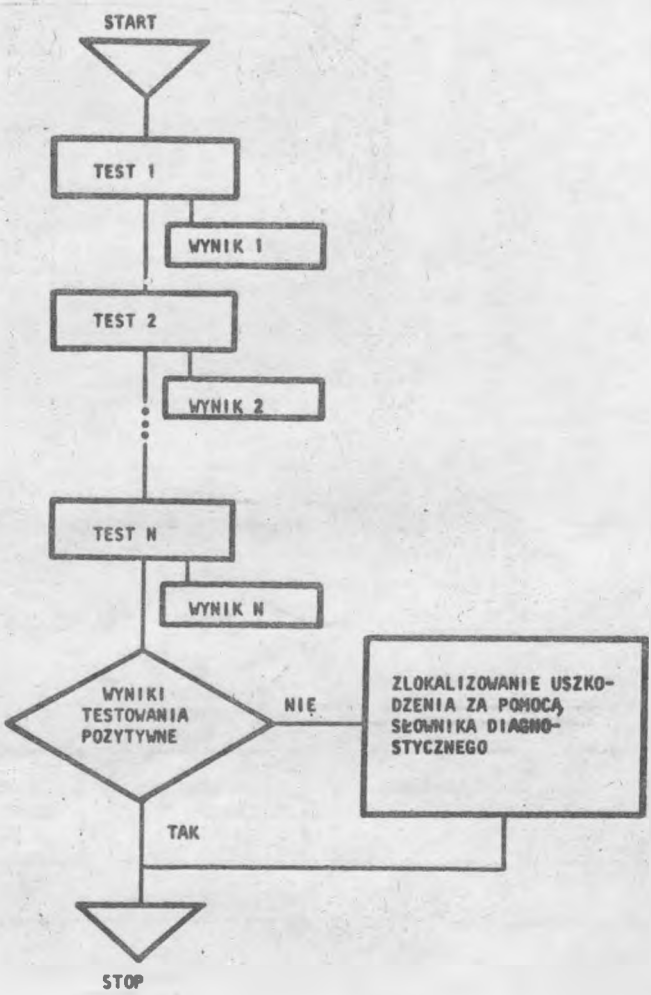
26. Manning E.: On computer self - diagnosis. Part 1 and Part II. IEEE Trans. Comp. 1966 nr 6.

27. Symons F.J.W., Ward M.K.: Programme control of the IST project. Telecomm. J. of Australia 1970 nr 3.

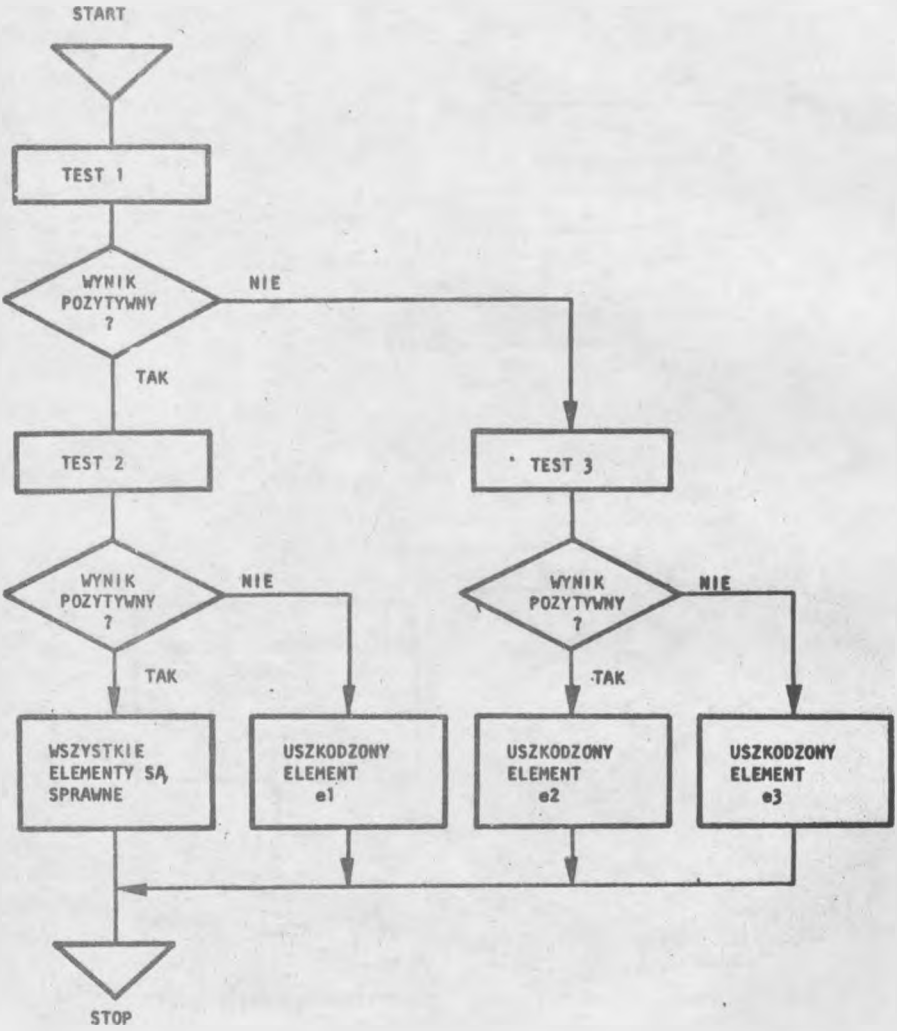
28. Fredman A., Menon P.: Wykrywanie uszkodzeń w układach cyfrowych. Warszawa: WNT 1974.



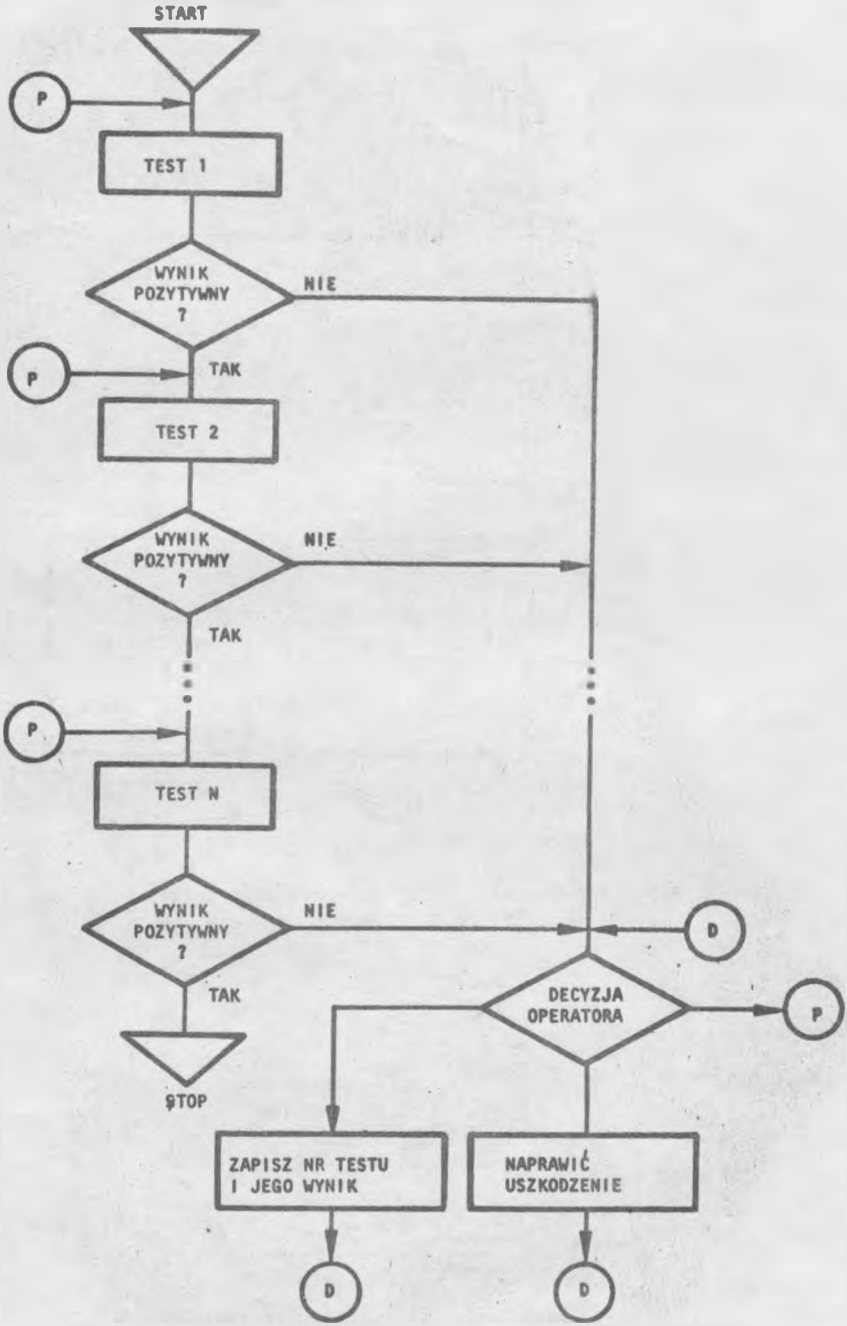
Rys. 1. Struktura programu diagnostycznego



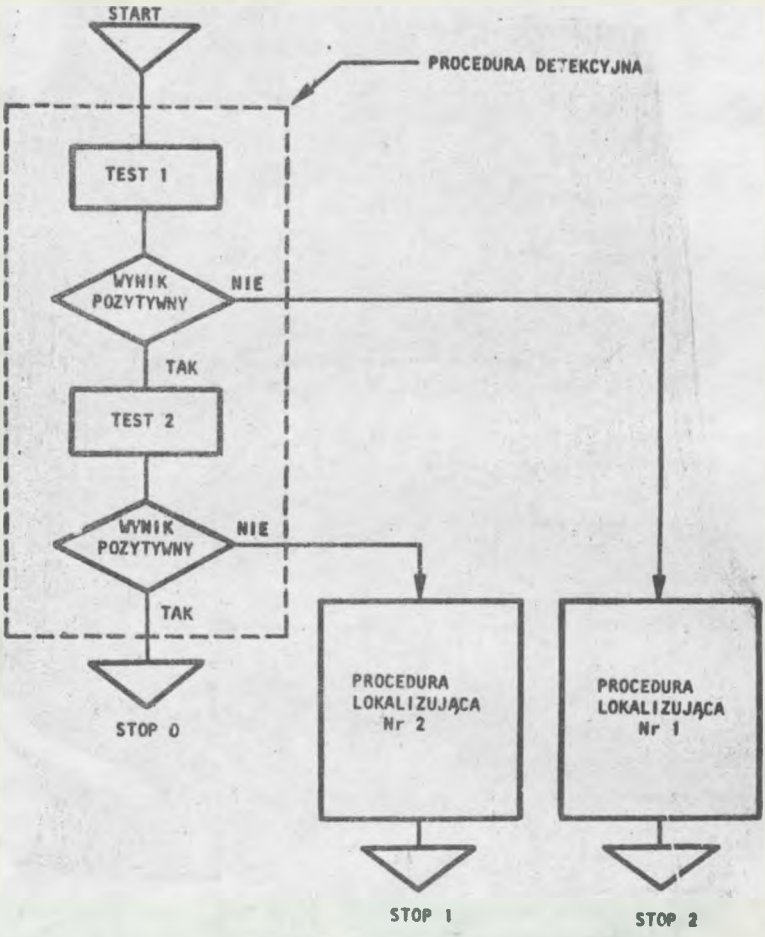
Rys. 2. Sieć działań programu diagnostycznego z procedurą kombinacyjną



Rys. 3. Sieć działań przykładowej procedury sekwencyjnej

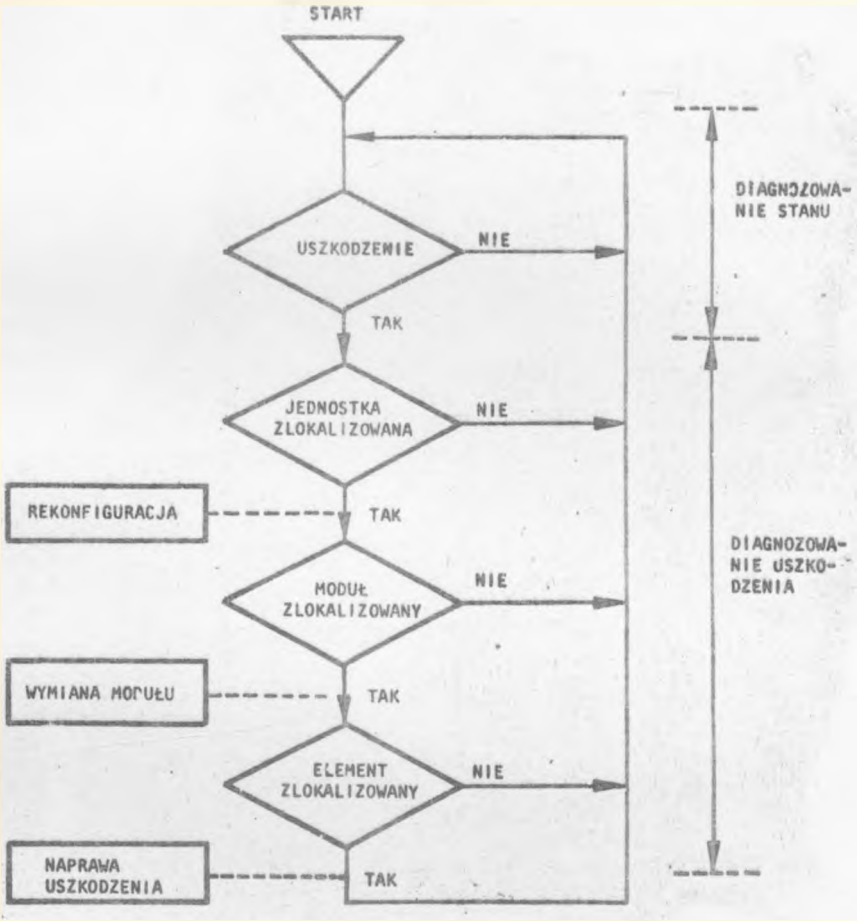


Rys. 4. Sieć działań procedury BOOTSTRAP



Rys. 5. Przykładowa sieć działań uniwersalnego programu diagnostycznego

Rys. 6. Idea strategii rozszerzania testera



Rys. 7. Proces diagnostyczny w programowo sterowanej centrali telefonicznej