Przemysłowy Instytut Automatyki i Pomiarów
Al. Jerozolimskie 202, 02-486 Warszawa

074 A

Zespół Układów i Systemów Sterowania
Nazwa ONB/ZNB

Główny wykonawca ..... prof. dr inż. Tadeusz Missala .....

Wykonawcy: dr inż. Wiesław Stańczak (red.),
dr inż. Andrzej Syryczyński,
mgr inż. Jacek Dunaj

Badania modelu wirtualnego urządzenia wytwórczego VMD zorientowanego na rodzinę robotów URP

DOKUMENT WZORCOWY

(Tytuł pracy, numer i tytuł etapu)

Zleceniodawca ..... Komitet Badań Naukowych .....

Nr zamówienia: 879/T11/95/08

Kierownik Zespołu Realizującego
prof. dr inż. Tadeusz Missala

Dyrektor
doc. dr inż. Stanisław Kaczanowski

Pracę zakończono dnia 31.12.1997

Nr arch. ..... 7553 Nr zlecenia .....

Przemysłowy Instytut Automatyki i Pomiarów
Al. Jerozolimskie 202, 02-486 Warszawa

Zespół Układów i Systemów Sterowania
Nazwa ONB/ZNB

Główny wykonawca prof. dr inż. Tadeusz Missala

Wykonawcy: dr inż. Wiesław Stańczak (red.),
dr inż. Andrzej Syryczyński,
mgr inż. Jacek Dunaj

Badania modelu wirtualnego urządzenia wytwórczego VMD zorientowanego na rodzinę robotów URP

(Tytuł pracy, numer i tytuł etapu)

Zleceniodawca Komitet Badań Naukowych

Nr zamówienia: 879/T11/95/08

Kierownik Zespołu Realizującego
prof. dr inż. Tadeusz Missala

Dyrektor
doc. dr inż. Stanisław Kaczanowski

Pracę zakończono dnia 31.12.1997

Nr arch. 7553

Nr zlecenia

Analiza deskryptorowa

Abstrakt

Tytuły poprzednich sprawozdań

Rozdzielnik

Egz. 1. .....

Egz. 2. .....

Egz. 3. .....

Spis treści

5.3. Funkcje realizujące usługi MMS w modelu VMD robota

5.3.1. Usługi zarządzania ogólnego
Usługa Cancel (Kasowanie)
Usługa Conclude (Zakończenie)
Usługa Initiate (Inicjacja)
Usługa Reject (Wybrakowanie)

5.3.2. Usługi dodatkowe stosowane w VMD
Usługa GetCapabilityList (Pobranie_Listy_Uprawnień)
Usługa GetNameList (Pobranie_Listy_Nazw)
Usługa Identify (Identyfikacja)
Usługa Status (Status)
Usługa UnsolicitedStatus (Status_Spontaniczny)

5.3.3. Usługi zarządzania domeną
Usługa DeleteDomain (Usuwanie_Domeny)
Usługa GetDomainAttributes (Pobranie_Atrybutów_Domeny)
Usługa LoadDomainContent (Ładowanie_Zawartości_Domeny)
Usługa StoreDomainContent (Przechowanie_Zawartości_Domeny)

5.3.4. Usługi zarządzania wywołaniem programu
Usługa CreateProgramInvocation (Utworzenie_Wywołania_Programu)
Usługa DeleteProgramInvocation (Usuwanie_Wywołania_Programu)
Usługa GetProgramInvocationAttributes Pobranie_Atrybutów_Wywołania_ Programu)
Usługa Reset (Zerowanie)
Usługa Resume (Wznowienie)
Usługa Start (Start)
Usługa Stop (Stop)

5.3.5. Usługi dostępu do zmiennej
Usługa GetVariableAccessAttributes (Pobranie_Atrybutów_Dostępu_Do_ Zmiennej)
Usługa InformationReport (Raportowanie_Informacji)
Usługa Read (Czytanie)
Usługa Write (Zapisanie)

5.3.6. Usługi zarządzania semaforem
Usługa RelinquishControl (Zaniechanie_Sterowania)
Usługa ReportPoolSemaphoreStatus (Raportowanie_Statusu_Semafora_ Puli)
Usługa ReportSemaphoreEntryStatus (Raportowanie_Statusu_Pozycji_ Semafora)
Usługa ReportSemaphoreStatus (Raportowanie_Statusu_Semafora)
Usługa TakeControl (Objęcie_Sterowania)

5.3.7. Usługi komunikacji operatorskiej
Usługa Input (Wejście)
Usługa Output (Wyjście)

6.4. Sprawdzenie spełnienia wymagań bezpieczeństwa

6.4.1. PN-M-42087

6.4.2. PN-EN 60204-1

6.4.3. EN 954-1

6.4.4. IEC DIS 61508-3

LITERATURA

1. WSTĘP

Praca była realizowana na podstawie projektu badawczego (grantu) nr 8 T11A 054 08, pt. *"Badania modelu wirtualnego urządzenia wytwórczego VMD zorientowanego na rodzinę robotów URP"* zgodnie z Umową Nr PB 879/T11/95/08 zawartą w dniu 1995.04.20. Zakres badań i ich cel zostały ustalone treścią załączników do wspomnianej umowy.

1.1. Cel pracy

Celem pracy według opisu projektu badawczego było sformułowanie abstrakcyjnego modelu urządzenia wytwórczego (sterowników robotów przemysłowych) w postaci sformalizowanej, tzn. przy użyciu aparatu teorii grafów, a także opracowanie narzędzi do weryfikacji modelu oraz przeprowadzenie badań poprawności i (sub)optymalizacja. Następnie zwiększenie szczegółowości modelu przez zapisanie go w języku C, przeprowadzenie badań symulacyjnych zgodności z normami. Wreszcie posadowienie modelu na platformie sprzętowej sterownika robota URP i dokonanie badań funkcjonalnych w środowisku sieci lokalnych.

Tematyka projektu jest związana z realizacją protokołów komunikacji sieciowej w środowisku wytwarzania. W licznych przemysłowych sieciach lokalnych, zarówno zgodnych z normami międzynarodowymi, jak i w sieciach firmowych, przyjęto protokoł Specyfikacji Komunikacji w Procesie Wytwarzania MMS (Manufacturing Message Specification). Powstał on jako protokoł najwyższej warstwy (aplikacyjnej) systemu MAP (Manufacturing Automation Protocol). Obecnie MMS jest stosowany we wszystkich systemach otwartych obsługujących proces produkcyjny, skonstruowanych zgodnie z siedmiowarstwowym modelem OSI (Open Systems Interconnections) wg ISO.

Z drugiej strony tematyka projektu wpisuje się w rozwój robotyki. Połączone sieciami roboty przemysłowe wykorzystuje się najczęściej w liniach produkcyjnych. Współpracują one, powiązane za pomocą sieci lokalnych, z innymi urządzeniami produkcyjnymi, sterującymi, kontrolnymi i transportowymi.

Podstawową strukturą w protokole MMS jest wirtualne urządzenie wytwórcze VMD (Virtual Manufacturing Device). VMD stanowi abstrakcyjną reprezentację rzeczywistego urządzenia wykonawczego (np. obrabiarki sterowanej numerycznie, centrum obróbczego, sterownika PLC, robota) wykorzystywanego w procesie produkcyjnym zorganizowanym zgodnie z ideą komputerowo zintegrowanego wytwarzania CIM (Computer Integrated Manufacturing). Istnienie i prawidłowe funkcjonowanie konkretnej implementacji VMD stanowi warunek konieczny uzyskania dostępu do rzeczywistego urządzenia wykonawczego w ramach realizacji MMS.

Znormalizowane słowne określenie VMD zamieszczono w dokumentach EN oraz ISO / IEC [7, 8, 16, 17, 18, 19, 20]. W początkowej fazie pracy wykorzystywano oryginalną, angielską wersję dokumentów [7, 8]. W trakcie dalszej realizacji uwzględniono - przede wszystkim przy przedsięwzięciach natury implementacyjnej - polskojęzyczną wersję [7, 8]. Uściślenie, a zarazem rozszerzenie znormalizowanego słownego opisu VMD w przypadku robota znajduje się w dokumencie ISO / IEC [20].

Dokumenty [7, 8, 16, 17, 18, 19, 20] stanowią podstawę implementacji szczegółowych, do lokalizacji na platformie sprzętowo - programowej konkretnych rzeczywistych urządzeń wykonawczych. W praktyce realizacją wirtualnego urządzenia wytwórczego jest określona struktura danych i związany z nią program wykonawczy (Executive Function).

Opis słowny z samej swojej istoty jest obszerny, np. dokument [7] zawiera 454 strony, i mało czytelny, co stwarza poważne trudności przy implementacji. W wyraźny sposób daje się odczuć brak modelu - sformalizowanej wersji opisu słownego. Taki model byłby z jednej strony podstawą do szybkiego generowania implementacji szczegółowych. Z drugiej zaś strony stanowiłby bazę do przeprowadzania badań i testów zgodności (Conformance Tests). Opracowanie tego rodzaju modelu jest więc zadaniem badawczym o istotnej wadze poznawczej, zaś jego implementacja na platformie sprzętowo - programowej robotów produkcji krajowej stanowi ważny krok przy wprowadzaniu tychże robotów do nowoczesnych zakładów przemysłowych, wykorzystujących ideę komputerowo zintegrowanego wytwarzania CIM.

W modelu wyróżniono dwie warstwy wnikliwości opisu - warstwę opisu strukturalnego oraz warstwę opisu szczegółowego. Opis strukturalny ujmuje aspekty transakcyjne VMD, a więc poszczególne stany wewnętrzne usług realizowanych przez wirtualne urządzenie wytwórcze, traktowanych jako automaty skończone i przejścia pomiędzy nimi. Opis ten został sformułowany w języku grafów skończonych. Formalizm teorii grafów jest podatny na przekształcenia, udostępnia różnorodną reprezentację struktur danych w komputerze i zmianę formy reprezentacji. Zastosowanie teorii grafów umożliwia dokonanie próby segmentacji czyli dekompozycji opisu strukturalnego oraz wyodrębnienie z niego fragmentów o dużym stopniu podobieństwa. Postępowanie tego typu jest obecnie powszechnie stosowane przy tworzeniu skomplikowanych implementacji programowych, jaką niewątpliwie stanowi implementacja usług realizowanych przez wirtualne urządzenie wytwórcze robota. Zostało ono wykonane rutynowo, jako obowiązujący krok, podczas stosowania omówionego niżej aparatu zespołów minimalnych.

Dalszym etapem postępowania były przekształcenia optymalizujące model. Polegały one na dekompozycji opisu strukturalnego za pomocą metodyki uogólnionych zespołów minimalnych, stanowiącej rozwinięcie pomysłu przedstawionego w pracy [61]. Metoda zespołów minimalnych rozbija graf na fragmenty (segmenty), biorąc za podstawę relacje zachodzące między podzbiorami zbioru wszystkich wierzchołków grafu. Otrzymana w wyniku jej zastosowania segmentacja posiada atrybut częściowego porządku (wzajemna rozłączność albo inkluzywność poszczególnych segmentów), ma więc charakter hierarchiczny [61]. Stanowi to o przydatności metody zespołów minimalnych do manipulacji nad opisem strukturalnym wirtualnego urządzenia wytwórczego z punktu widzenia przedstawionej niżej idei warstwy opisu szczegółowego. Powtórne zastosowanie metodyki uogólnionych zespołów minimalnych, tym razem do wyników segmentacji, umożliwiło dodatkowo wyróżnienie segmentów podobnych, prowadząc w rezultacie do parametryzacji segmentów, a więc zmniejszenia liczby różnych typów segmentów.

Jak wykazano wyżej teoria grafów dostarcza narzędzi pozwalających na formalne, strukturalne zbadanie poprawności modelu VMD i jego (sub)optymalizację. Natomiast badania funkcjonalne realizacji usług MMS można przeprowadzić na symulowanym lub rzeczywistym obiekcie. Do

symulacji i badań posłużyła warstwa druga modelu VMD - warstwa opisu szczegółowego. W warstwie tej model został opisany w języku C, przy zachowaniu uprzednio uzyskanej (sub)optymalnej struktury. Model w języku C odwzorowuje związki strukturalne uzyskane poprzez segmentację i parametryzację segmentów modelu w języku teorii grafów, zarazem cechuje się większą szczegółowością opisu. Obrazuje nie tylko wewnętrzne zmiany stanów VMD, ale również ujmuje te zmiany ilościowo i wiąże (przy uwzględnieniu m.in. liczonej w bajtach długości tzw. PDUs - Protocol Data Units) wirtualne urządzenie wytwórcze ze środowiskiem sieci lokalnych. Model w języku C odnosi się do szerokiej klasy robotów, umożliwia także przykładową lokalizację na platformie sprzętowo - programowej sterownika robotów URP produkcji krajowej.

Do badania realizacji usług opracowano zestaw testów zgodności (Conformance Tests). Badania te przeprowadzono w środowisku sieci lokalnych, w dwu fazach. W fazie pierwszej, symulacyjnej, poszczególnymi węzłami sieci były komputery wyposażone w karty sterowników sieci MAP 3.0, z posadowioną warstwą opisu szczegółowego modelu wirtualnego urządzenia wytwórczego. Po uzyskaniu pozytywnych rezultatów w fazie pierwszej, model VMD został zlokalizowany na platformie sprzętowo - programowej sterownika robotów URP. W fazie drugiej, końcowej, badania przeprowadzono z zastosowaniem rzeczywistych obiektów - robotów z rodziny URP. Pozytywny rezultat tych ostatnich badań stanowił weryfikację końcową modelu VMD dla robota.

1.2. Zakresy tematyczne poszczególnych etapów pracy

Praca była realizowana w ramach 7 następujących etapów:

Nr etapu.	Nazwa etapu
1	Opracowanie opisu strukturalnego (w języku teorii grafów) modelu VMD do robota
2	Opracowanie programowych narzędzi do weryfikacji i (sub)optymalizacji modelu VMD
3	Zakup aparatury
4	Badania weryfikujące opis strukturalny VMD. (Sub)optymalizacja opisu strukturalnego
5	Opracowanie opisu szczegółowego (w języku C) modelu VMD do robota
6	Symulacyjne badania opisu szczegółowego modelu VMD w sieci komputerów
7	Lokalizacja implementacji modelu VMD na platformie sprzętowej sterownika robota URP i badania weryfikujące

Etapy 1 i 2 miały charakter czysto poznawczy i przygotowawczy, jednocześnie silnie rzutowały na dalszy bieg pracy. W ramach etapu 1 została rozpoznana struktura komunikacji w sieciach zgodnych z siedmiowarstwowym modelem ISO/OSI, a następnie opracowana metodologia konstruowania modelu strukturalnego w języku teorii grafów i zastosowanie jej do usług MMS. Wyniki uzyskane w trakcie realizacji etapu 1 opisano w rozdziale 3. Scharakteryzowano w nim, między innymi, różne rodzaje sieci transmisji danych stosowanych w przedsiębiorstwie przemysłowym, siedmiowarstwowy model ISO/OSI, komunikację typu klient-serwer oraz dwa rodzje usług sieciowych: bez potwierdzenia (bezpołączeniowe) i z potwierdzeniem

(połączeniowe). Wprowadzone tam pojęcia wykorzystano potem w treści rozdziałów 5 i 6. Następnie przedstawiono koncepcję protokołu MMS i wirtualnego urządzenia wytwórczego VMD (omawiając relacje między rzeczywistymi i wirtualnymi urządzeniami i obiektami), modelowanie obiektów i urządzeń na poziomie VMD oraz omówiono usługi protokołu MMS. Sposób tworzenia modelu strukturalnego w języku teorii grafów i jego zastosowanie do opisu usług MMS przedstawiono w rozdziale 3 na przykładzie usług Initiate i Conclude, bez których nie jest możliwe włączenie się robota do pracy w sieci MAP i zwykłe (tzn. bez wystąpienia stanów awaryjnych) zakończenie seansu łączności.

Etap 2 dotyczył aspektów programowej realizacji algorytmu wspomagającego metodologię zespołów minimalnych. W jego ramach powstały procedury programowe umożliwiające automatyczne (przy pomocy komputera) generowanie zespołów minimalnych. Wyniki uzyskane w trakcie realizacji etapu 2 opisano w rozdziale 2. Punktem wyjścia było istnienie wielu metod klasyfikacji obiektów. Wykorzystuje się w nich różnorakie sposoby oceny stopnia wzajemnego "podobieństwa" lub "zróżnicowania" obiektów poddawanych klasyfikacji. Werbalnie problem klasyfikacji można ująć w następujący sposób:

- w przypadku "podobieństwa": do tej samej klasy powinny należeć obiekty możliwie wzajemnie jak najbardziej podobne, zaś obiekty o znikomym podobieństwie należy umieścić w różnych, rozłącznych klasach;
- w przypadku "zróżnicowania": do tej samej klasy należy zaliczyć obiekty możliwie wzajemnie jak najmniej różniące się, zaś obiekty znacznie różniące się powinny znaleźć się w różnych, rozdzielnych klasach.

Jako punkt wyjściowy w pracy zastosowano przytoczone wyżej określenia słowne. Postarano się następnie sformalizować pojęcia "podobieństwa" i "zróżnicowania" , biorąc pod uwagę również aspekty interpretacyjne. Zauważono, że pojęcia bardziej lub mniej "podobny" oraz mniej lub bardziej "zróżnicowany" można uogólnić i ujednolicić mówiąc o liniowym uporządkowaniu pewnego zbioru cech (atrybutów wzajemnych powiązań) przypisywanych zarówno pojedynczym parom klasyfikowanym obiektom jak i większym podzbiorom mnogości wszystkich obiektów poddawanych klasyfikacji. Przeformułowano potem w następujący sposób kryterium klasyfikacyjne:

- klasę stanowi każdy niepusty podzbiór mnogości wszystkich, podlegających klasyfikacji obiektów, wtedy i tylko wtedy, gdy atrybut powiązania każdej części właściwej (niepustej) tego podzbioru z resztą podzbioru jest poprzedzany (w sensie ww. liniowego porządku) przez atrybut powiązania wspomnianej części właściwej ze zbiorem wszystkich obiektów nie należących do omawianej klasy.

Innymi słowy (bardziej obrazowo, chociaż mniej dokładnie) klasę stanowi taki podzbiów mnogości wszystkich podlegających klasyfikacji obiektów, który jest związany wewnętrznie silniej (w sensie ww. liniowego porządku) niż z podzbiorem wszystkich obiektów leżących poza omawianą klasą.

W związku z analogią przyjętego określenia do definicji tzw. (klasycznych) zespołów minimalnych omawianą klasę nazwano (uogólnionym) zespołem minimalnym. Udowodniono, że zbiór wszystkich uogólnionych zespołów minimalnych (określonych dla danych: mnogości wszystkich, podlegających klasyfikacji obiektów oraz liniowo uporządkowanej relacji atrybutów wzajemnych powiązań) jest niepusty, a także jest częściowo uporządkowany przez relację inkluzji.

Doprowadziło to do przypuszczenia, że do wyszukiwania uogólnionych zespołów minimalnych przydatny może być hierarchiczny algorytm agregacyjny. Szereg dalszych właściwości omawianej struktury matematycznej, sformułowanych w postaci lematów, twierdzeń oraz wniosków, a następnie udowodnionych potwierdziło ww. przypuszczenie. Opierając się na wspomnianych właściwościach, sformułowano podstawy algorytmu wyszukującego wszystkie uogólnione zespoły minimalne (w przypadku konkretnych danych określających: mnogość wszystkich, podlegających klasyfikacji obiektów oraz liniowo uporządkowaną relację atrybutów wzajemnych powiązań).

Etap 4 miał charakter interakcyjny. Z jednej strony był on kontynuacją etapu 2, polegającą na dobraniu i dopasowaniu odpowiedniej funkcji kryterialnej do uogólnionej metody zespołów minimalnych, a więc na uszczegółowieniu metody zespołów minimalnych. W jego ramach powstały kolejne wersje modułów oprogramowania realizującego algorytm wyszukiwania zespołów minimalnych, stanowiące alternatywę dla modułów opracowanych w ramach etapu 2. Moduły te testowano na danych stanowiących rezultaty cząstkowe etapu 2 (tzn. na zapisanych w języku teorii grafów modelach poszczególnych usług MMS). Zaś wyniki testów posłużyły, po dokonaniu oceny metodami eksperckimi, do konkretyzacji metody zespołów minimalnych. Początkowo przewidywano, że ostateczną postacią stosowanej w metodzie zespołów minimalnych funkcji kryterialnej będzie funkcja addytywna (stanowiąca podstawę tzw. klasycznej metody zespołów minimalnych), funkcja minimaksowa (tzw. zespoły Max-minimalne), bądź kombinacja obu wspomnianych funkcji. Wynikiem realizacji etapu 4 było przyjęcie funkcji minimaksowej. Sprawozdanie z przeprowadzonych w etapie 4 rozważań zawarto w rozdziale 4 niniejszej pracy. Tamże też przedstawiono rezultat (sub)optymalizacji modelu VMD, który doprowadził do wyboru podzbioru usług MMS przeznaczonych do realizacji w postaci modelu szczegółowego. Usługi te wyspecyfikowano dalej w p. 5.3 niniejszego sprawozdania.

Prace w etapie 5 polegały na utworzeniu i zapisaniu w języku C (postać źródłowa) podprogramów (procedur) realizujących usługi MMS w przypadku robota. Uzyskane rezultaty przedstawiono w rozdziale 5. Wyspecyfikowano w nim podstawowe struktury danych wykorzystywane w modelu VMD, a potem funkcje realizujące usługi MMS w modelu VMD robota, podzielone na klasy funkcji dotyczących usług zarządzania ogólnego, usług dodatkowych stosowanych w VMD, usług zarządzania domeną, usług zarządzania wywołaniem programu, usług dostępu do zmiennej, usług zarządzania semaforem, usług komunikacji operatorskiej i usług zarządzania plikami. Wspomniane struktury danych i funkcje wykorzystano w etapie 6 do utworzenia oprogramowania komunikacyjnego z protokołem MMS.

W etapie 6 wygenerowano oprogramowanie komunikacyjne na podstawie modułów programowych usług opracowanych w ramach etapu 5 (por. rozdział 5), posadowiono je na komputerach kompatybilnych z IBM-PC i przeprowadzono szereg seansów łączności pod nadzorem monitora sieci. Uzyskane rezultaty przedstawiono w rozdziale 6 (p. 6.1 i 6.2). Zawarto tam opis strukturalny implementacji oprogramowania realizującego usługi protokołu MMS, omówienie konfiguracji sprzętowo - programowej segmentu sieci IEEE 802.4, który posłużył do badań symulacyjnych przeprowadzanych w celu sprawdzenia poprawności działania wygenerowanego oprogramowania oraz wyniki przeprowadzonych testów. Etap zakończył się stwierdzeniem prawidłowego działania końcowej wersji wytworzonego oprogramowania.

Etap 7 polegał na powiązaniu zweryfikowanego w etapie 6 oprogramowania z platformą sprzętową sterownika robota URP i przeprowadzeniu testów funkcjonalnych, a następnie na dokonywaniu niezbędnych zmian w oprogramowaniu prowadzących w końcu do uzyskania bezawaryjnej pracy w środowisku sieciowym. Uzyskane rezultaty omówiono w rozdziale 6 (p. 6.3 i 6.4). W p. 6.3 przedstawiono sposób lokalizacji modelu VMD na platformie sprzętowej sterownika robota URP. Polega on na sprzężeniu sterownika robota z komputerem IBM PC, gdzie ten ostatni jednocześnie pełni rolę stacji sieciowej z protokołem MMS i zarządza pracą sterownika. Przedstawiono zasady komunikacji między sterownikiem a komputerem i wyspecyfikowano formaty przesyłek między nimi (p. 6.3.1 i 6.3.2). Następnie omówiono testy funkcjonalne (i ich wyniki), które zastosowano w celu weryfikacji przyjętej koncepcji lokalizacji modelu VMD na platformie sprzętowej sterownika robota URP. Punkt. 6.4 poświęcono zagadnieniom spełnienia wymagań bezpieczeństwa lokalizacji modelu VMD na platformie sprzętowej sterownika robota URP.

2. OPISANIE ZWIĄZKÓW MIĘDZY OBIEKTAMI

2.1. Związki między obiektami: podobieństwo i odmienność

Do codziennej praktyki należy próba klasyfikowania zdarzeń, zachowań, obiektów itp. Dotyczy to najczęściej zbioru (grupy) o niewielkiej liczności. O jego elementach formułujemy jeden z przeciwstawnych poglądów typu:

*coś **jest bardziej podobne** do czegoś **niż** do czego innego* (2.1.1a)

lub

*coś **jest mniej odmienne** od czegoś **niż** od czegoś innego* (2.1.1b)

przy czym pisząc (bądź mówiąc): *coś, czegoś, czego innego* mamy na myśli trzy różne elementy rozpatrywanego zbioru (grupy). W przypadku (2.1.1a) mówimy o podobieństwie (lub bliskości) elementów, zaś w przypadku (2.1.1b) mówimy o odmienności (inne terminy to np. niepodobieństwo, zróżnicowanie). W ten to sposób, niejawnie, wprowadzane są aspekty klasyfikacyjne: do tej samej klasy zaliczymy elementy

bardziej podobne a nie ***mniej podobne*** (2.1.2a)

lub

mniej odmienne a nie ***bardziej odmienne*** (2.1.2b)

Oczywiście w dalszym ciągu pozostaje zdecydować na ile elementy mają być do siebie podobne lub jak niewiele powinny być odmienne, ażeby zaliczyć je do jednej i tej samej klasy. Zagadnienie to stanowi jeden z centralnych problemów teorii klasyfikacji, którego rozwiązanie w szczególnym przypadku zostanie przytoczone w p. 2.3 (definicja 1).

W przypadku dążenia do algorytmizacji procesu klasyfikacji podane wyżej reguły (2.1.2a) i (2.1.2b) są z powodu swojej ogólności niewystarczające (dokładniej mówiąc mogą one zostać wykorzystane przy klasyfikacji bazującej na metodologii logik i zbiorów rozmytych, jednakże i w tym przypadku zastosowanie komputerów wymusza wprowadzenie pewnych dodatkowych uściśleń, umożliwiających efektywną realizację programów komputerowych). Dlatego też przystąpi się teraz do sformalizowania (a więc i uściślenia) wprowadzonych wyżej zgodnie z intuicją pojęć.

Symbolem X oznaczymy mnogość zawierającą co najmniej 2 elementy ($|X| > 1$, gdzie $|.|$ - oznaczenie mocy zbioru). Niech istnieje funkcja

$$f: X \times X \to \mathbb{R} \tag{2.1.3}$$

przekształcająca produkt kartezjański $X \times X$ zbioru X w mnogość liczb rzeczywistych $\mathbb{R}$. Jeśli ponadto dla każdej pary $x, y \in X$ zachodzą następujące warunki:

$$f_0 \geq f(x, y), \tag{2.1.4a}$$
$$f(x, x) = f_0, \tag{2.1.4b}$$
$$f(x, y) = f(y, x), \tag{2.1.4c}$$

przy czym $f_0 \in \mathbb{R}$, to funkcję f nazywamy funkcją podobieństwa, zaś wartość wyrażenia f(x, y) - podobieństwem lub stopniem podobieństwa elementów x i y. Sformalizowanym odpowiednikiem poglądu (2.1.1a) przekształconego do postaci:

x ***jest bardziej podobne*** *do* y ***niż*** *do* z,

będzie zatem zapis

$$f(x, y) \geq f(x, z), \tag{2.1.4d}$$

przy czym x, y, z $\in$ X, gdzie relacja $\geq$ odpowiada stwierdzeniu ***jest bardziej podobne***. Piątkę uporządkowaną

$$\langle X, \mathbb{R}, \geq, f, f_0 \rangle \tag{2.1.4e}$$

nazywamy systemem podobieństwa, zaś parę uporządkowaną

$$\langle \mathbb{R}, f_0, \geq \rangle \tag{2.1.4f}$$

- typem podobieństwa.

Przykładem wykorzystania pojęcia podobieństwa jest system stawiania ocen przyjęty w polskich placówkach oświatowych. Polega on na porównywaniu odpowiedzi z uznanym i przyjętym za obowiązujący wzorcem - jeśli stopień zgodności jest wysoki, to oceniany otrzymuje najwyższą ocenę (szóstkę), zaś zachodzenie istotnych różnic między odpowiedzią i wzorcem powoduje wystawienie oceny najniższej (zera lub jedynki). Wystąpienie różnic mniej istotnych pociąga za sobą ocenę pośrednią, mniejszą od najwyższej i większą od najniższej.
Innym przykładem stosowania podobieństwa jest ocenianie metodą testów. Polega ona na przedstawieniu osobie testowanej pewnej liczby pytań, do każdego z których dołączono zestaw dwu lub więcej wariantów odpowiedzi. Wybranie prawidłowej odpowiedzi na dane pytanie nagradzane jest zaliczeniem jednego punktu. Brak odpowiedzi lub zła odpowiedź powoduje otrzymanie zera. Wynik testu stanowi suma zebranych punktów.
Wyznaczanie procentu wyrobów wadliwych w partii wyprodukowanych wyrobów też wykorzystuje pojęcie podobieństwa. Mianowicie na podstawie dostępnych danych np. o wykorzystaniu wyrobów (półproduktów) w dalszym ciągu procesu produkcyjnego lub o warunkach bezpiecznego użytkowania wyrobu określa się granice tolerancji czyli wartość odchyłki parametrów technicznych od wartości nominalnych. W najprostszym przypadku bierze się pod uwagę tylko jeden zestaw granic tolerancji. Jeśli wartości poszczególnych parametrów technicznych wyrobu nie odbiegają od wartości nominalnej więcej niż na to wskazują granice tolerancji, to badany wyrób zalicza się do dobrych. W przeciwnym razie zalicza sę go do wyrobów wadliwych. Istnieją również przypadki, w których dla pewnych parametrów określa się kilka granic tolerancji. Na przykład żywność przeznaczona do bezpośredniego spożycia nie musi spełniać tak ostrych kryteriów jak żywność przeznaczona do długoterminowego przechowywania w ramach tzw. rezerw państwowych, elementy wyposażenia odbiorników radiowych samolotu muszą być bardziej niezawodne i odporne na wpływy środowiskowe niż takie same elementy będące częściami aparatów przeznaczonych do użytku domowego. Wówczas wyroby dzieli się (w zależności od tego, w zakresie których granic tolerancji znajdują się ich parametry techniczne) na kilka klas dobrych i na wyroby wadliwe.

Jeżeli dla każdej pary x, y $\in$ X zachodzą następujące warunki:

$$f_0 \leq f(x, y), \tag{2.1.5a}$$

$$f(x, x) = f_0, \tag{2.1.5b}$$

$$f(x, y) = f(y, x), \tag{2.1.5c}$$

przy czym $f_0 \in \mathbb{R}$, to funkcję f nazywamy funkcją odmienności (zróżnicowania), zaś wartość wyrażenia f(x, y) - odmiennością lub stopniem odmienności elementów x i y. Sformalizowanym odpowiednikiem poglądu (2.1.1b) przekształconego do postaci:

x ***jest mniej odmienne*** *od* y ***niż*** *od* z,

będzie zatem zapis

$$f(x, y) \leq f(x, z), \tag{2.1.5d}$$

przy czym x, y, z $\in$ X, gdzie relacja $\leq$ odpowiada stwierdzeniu ***jest mniej odmienne***. Piątkę uporządkowaną

$$\langle X, \mathbb{R}, \leq, f, f_0 \rangle \tag{2.1.5e}$$

nazywamy systemem odmienności, zaś parę uporządkowaną

$$\langle \mathbb{R}, f_0, \leq \rangle \tag{2.1.5f}$$

- typem odmienności.

Przykładem wykorzystania pojęcia odmienności jest tzw. norma L_p, $p \geq 1$, w przypadku której stopień odmienności między obiektami x i y, x, y $\in$ X, określa wzór:

$$f(x, y) = \|\boldsymbol{x} - \boldsymbol{y}\|_p = \sqrt[p]{\sum_{k=1}^{l} |x_k - y_k|^p},$$

przy czym cechy charakteryzujące obiekty x i y przedstawiane są w postaci ciągów (lub wektorów) l-elementowych, odpowiednio $\boldsymbol{x} = \langle x_1, x_2, ..., x_l \rangle$ oraz $\boldsymbol{y} = \langle y_1, y_2, ..., y_l \rangle$, gdzie x_k, y_k, $\in \mathbb{R}$, $k = 1, 2, ..., l$. Jeśli $p = 2$, to otrzymujemy znaną i powszechnie stosowaną miarę Euklidesową w przestrzeni l-wymiarowej, zaś wariant $p = 1$ odpowiada tzw. metryce miejskiej lub metropolitalnej (znanej także pod nazwą metryki typu Manhattan), często stosowanej do określania długości drogi łączącej dwa punkty w nowoczesnym mieście o prostokątnym planie ulic lub długości ścieżek na płytce obwodów drukowanych.

Pewnym uogólnieniem normy L_p dla przypadku $p = 2$ jest metryka Mahalanobisa, w której odmienność obiektów x oraz y określa się wzorem:

$$f(x, y) = \sqrt{\langle x_1 - y_1, x_2 - y_2, ..., x_l - y_l \rangle * \boldsymbol{B} * \langle x_1 - y_1, x_2 - y_2, ..., x_l - y_l \rangle^T},$$

gdzie górny indeks T oznacza transpozycję, * symbolizuje mnożenie macierzy, zaś $\boldsymbol{B}$ jest macierzą dodatnio określoną (tzn. symetryczną macierzą kwadratową o wymiarze l, spełniającą warunek $\boldsymbol{x} * \boldsymbol{B} * \boldsymbol{x}^T \geq 0$, dla każdego l-wymiarowego wektora $\boldsymbol{x}$, przy czym równość $\boldsymbol{x} * \boldsymbol{B} * \boldsymbol{x}^T = 0$ zachodzi wtedy i tylko wtedy, gdy $\boldsymbol{x} = \langle 0, 0, ..., 0 \rangle$). Metryki Mahalanobisa używa się w przypadku, w którym cechy mają różny wpływ na osąd stanowiący o ocenie stopnia odmienności. Jako przykład można tu przytoczyć sytuację wyboru z grupy kandydatów tłumacza języka angielskiego na podstawie następujących, znanych nam cech osobowych: (1) wieku (co ma pewien związek z dyspozycyjnością), (2) czasu posiadania uznawanego świadectwa znajomości języka angielskiego, (3) czasu spędzonego za granicą w krajach anglojęzycznych oraz (4) odległości miejsca zamieszkania od miejsca pełnienia obowiązków służbowych (co znowu ma związek z dyspozycyjnością). Jak łatwo zauważyć cechy o numerach (2) i (3) mają tu większą "wartość gatunkową". Dlatego też jedną z dopuszczalnych postaci macierzy $\boldsymbol{B}$ będzie:

$$\begin{bmatrix} \frac{1}{3},0,0,\ 0 \\ 0,2,0,\ 0 \\ 0,0,3,\ 0 \\ 0,0,0,\frac{1}{10} \end{bmatrix},$$

zaś na pewno nierozsądne będzie przyjęcie macierzy $\boldsymbol{B}$ w formie:

$$\begin{bmatrix} 20,1,0,\ 0 \\ 0,1,0,\ 0 \\ 0,0,1,\ 0 \\ 0,0,0,200 \end{bmatrix}$$

Łatwo zauważyć, że przyjęcie $\boldsymbol{B} = \boldsymbol{I}$ (gdzie $\boldsymbol{I}$ jest macierzą jednostkową, tzn. macierzą diagonalną z jedynkami na przekątni głównej) sprowadza metrykę Mahalanobisa do miary Euklidesowej w przestrzeni dwuwymiarowej.

2.2. Podobieństwo i odmienność jako szczególny przypadek powiązania

W p. 2.1 przedstawiono powszechnie stosowane pojęcia podobieństwa i odmienności, sformalizowano je i przytoczono przykłady ich zastosowań. Prezentację prowadzono w sposób uwypuklający symetrię i komplementarność omawianych pojęć (por. wzory (2.1.1a), (2.1.2a), (2.1.4a)-(2.1.4f) i ich odpowiedniki (2.1.1b), (2.1.2b), (2.1.5a)-(2.1.5f)). Przede wszystkim zauważymy, że zarówno typ podobieństwa jak i typ odmienności jest szczególnym przypadkiem porządku liniowego (tzn. odpowiednio zbiorem $\{r: r \in \mathbb{R}, r \leq f_0\}$ lub $\{r: r \in \mathbb{R}, r \geq f_0\}$, liniowo uporządkowanym przez określoną na nim relację). W związku z tym pierwszym, narzucającym się uogólnieniem jest wprowadzenie pojęcia związek dwu elementów lub powiązanie dwu elementów (między dwoma elementami) rozumianego w następujący sposób:

są silniej powiązane (mocniej związane) $\Leftrightarrow$ { i/lub ***są bardziej podobne*** / ***są mniej odmienne*** } (2.2.1)

są słabiej powiązane (słabiej związane) $\Leftrightarrow$ { i/lub ***są mniej podobne*** / ***są bardziej odmienne*** } (2.2.2)

We wzorach (2.1.4a), (2.1.4d)-(2.1.4f) odpowiada to konsekwentnemu zastąpieniu znaku $\geq$ znakiem relacji liniowo porządkującej $\preccurlyeq$; zaś we wzorach (2.1.5a), (2.1.5d)-(2.1.5f) - analogicznie - znaku $\leq$ znakiem relacji liniowo porządkującej $\preccurlyeq$. W rezultacie wspomnianych zabiegów dla każdej pary $x, y \in X$ otrzymujemy:

$$f_0 \preccurlyeq f(x, y), \quad (2.2.3a)$$

$$f(x, x) = f_0, \quad (2.2.3b)$$

$$f(x, y) = f(y, x), \quad (2.2.3c)$$

przy czym funkcję f nazywamy funkcją powiązań, zaś wartość wyrażenia f(x, y) - powiązaniem lub siłą powiązań elementów x i y. Sformalizowanym odpowiednikiem poglądu (2.2.1) przekształconego do postaci:

x ***jest silniej powiązane*** *z elementem* y ***niż*** *z elementem* z,

jest zatem zapis

$$f(x, z) \prec f(x, y), \tag{2.2.3d}$$

przy czym x, y, z $\in$ X, gdzie relacja $\prec$ odpowiada stwierdzeniu ***jest silniej powiązane***. Piątkę uporządkowaną

$$<X,\mathbb{R}, \prec, f, f_0> \tag{2.2.3e}$$

nazywamy systemem powiązań, zaś parę uporządkowaną

$$<\mathbb{R}, f_0, \prec> \tag{2.2.3f}$$

- typem powiązań.

2.3. Idea zespołu minimalnego

W drugiej części p. 2.1 (począwszy od wzoru (2.1.3)) zajmowaliśmy się formalizacją intuicyjnie zrozumiałych pojęć podobieństwa i odmienności. Ze względów historycznych (por. [28, 57]) jak również z powodu łatwości interpretacji wyników zakładaliśmy, że podobieństwo lub odmienność dwu elementów jest wyrażana liczbą rzeczywistą. Ogólnie rzecz biorąc założenie to jest sztuczne. Ponadto, jak się zaraz okaże, założenie to wcale nie jest potrzebne.

W dalszym ciągu wykorzystamy wyniki uzyskane w p. 2.2, a więc rozpatrywać będziemy powiązania - podobieństwo i odmienność traktując jako szczególne przypadki.

Rozważymy mnogość X, |X| > 1, liniowy porządek $<\mathbf{R}, \prec>$, a także funkcję

$$g:\{\{A, B\}: A, B \subset X, A \cap B = \varnothing\} \rightarrow \mathbf{R} \tag{2.3.1}$$

Z (2.3.1) wynika, że funkcja g jest symetryczna względem swoich argumentów, tzn.

$$g(A, B) = g(B, A) \tag{2.3.2}$$

Założymy ponadto, iż dla każdej czwórki zbiorów A, B, C, D $\subset$ X takich, że A $\subset$ B i C $\subset$ D, a także wzajemnie rozłącznych B i D (tzn. B $\cap$ D = $\varnothing$), zachodzi relacja:

$$g(A, C) \prec g(B, D) \tag{2.3.3}$$

Jest oczywiste [56], że $<\mathbf{R}/g, \prec>$ jest także porządkiem liniowym, przy czym przez $\mathbf{R}/g$ oznaczono przeciwdziedzinę funkcji g. Ponadto z (2.3.3) wynika, że

$$g(\varnothing, \varnothing) \prec g(A, B) \tag{2.3.4}$$

(gdzie $\varnothing$ jest symbolem zbioru pustego) dla każdej pary wzajemnie rozłącznych podzbiorów A, B mnogości X. Okazuje się zatem, że po wprowadzeniu oznaczenia $g_0 = g(\varnothing, \varnothing)$, wzór (2.3.4) staje się analogonem relacji (2.2.3a) z p. 2.2. Podobnie wzór (2.3.2) stanowi odpowiednik zależności (2.2.3c). Tak jak uprzednio piątkę uporządkowaną

$$<X,\mathbf{R}, \prec, g, g_0>$$

nazwiemy (uogólnionym) systemem powiązań.

Wprowadzimy definicję:

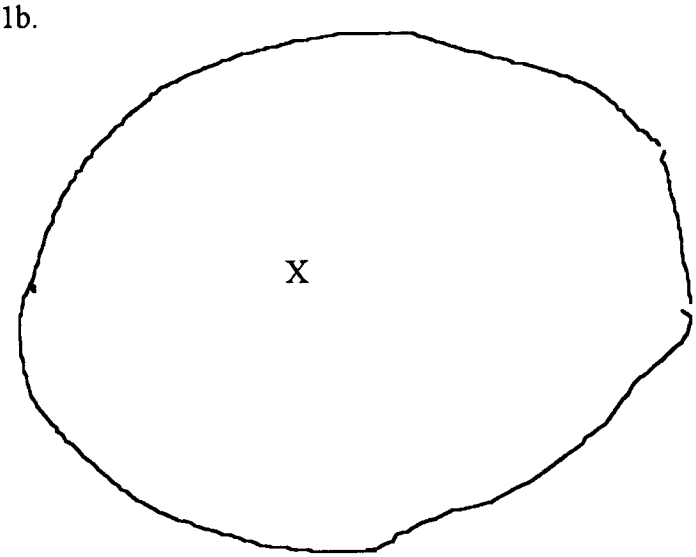
Definicja 1. Jeśli dla pewnego niepustego zbioru $S \subset X$, $S \neq X$, następująca para zależności:

$$g(R, X\text{-}S) \prec g(R, S\text{-}R), \quad (2.3.5a)$$

$$g(R, X\text{-}S) \neq g(R, S\text{-}R), \quad (2.3.5b)$$

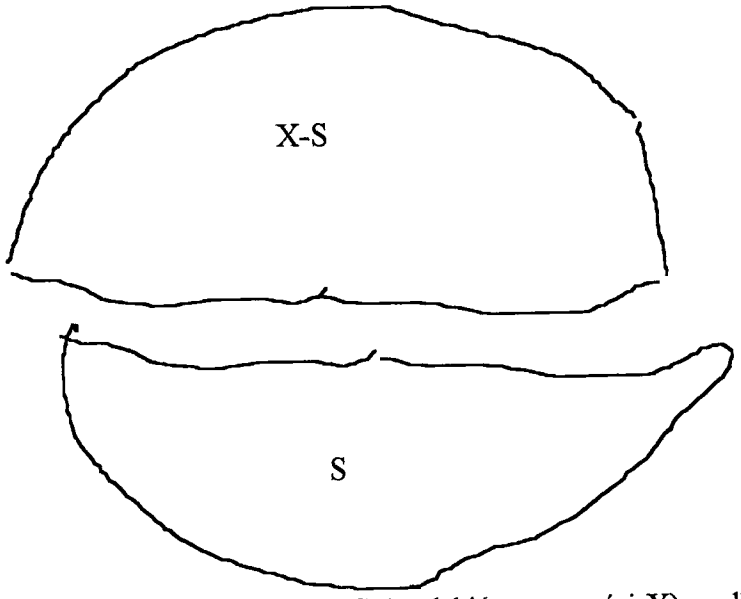
jest spełniona dla każdego niepustego podzbioru $R \subset S$, takiego że $R \neq S$, to zbiór S nazywamy (uogólnionym) zespołem minimalnym.

Spróbujemy teraz zinterpretować wprowadzone pojęcie. W tym celu posłużymy się rysunkami 2.3.1a - 2.3.1e. Na rys. 2.3.1a przedstawiono zbiór wszystkich rozpatrywanych (tzn. podlegających klasyfikacji) obiektów. Powstało podejrzenie, że niepusty podzbiór S zbioru X ($S \neq X$) jest zespołem minimalnym. Zbiór S i jego "zewnętrze" X-S w stosunku do zbioru X pokazano na rys. 2.3.1b.

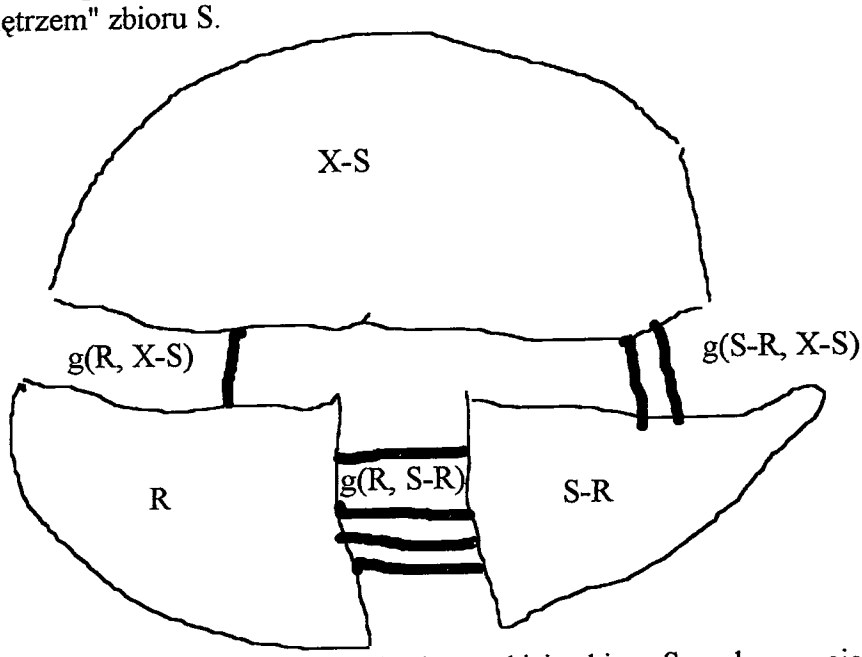


Rys. 2.3.1a. Idea zespołu minimalnego. Zbiór X składający się ze wszystkich rozpatrywanych obiektów.

W celu sprawdzenia warunków (2.3.5) należy teraz dokonać rozbicia mnogości S na dwa niepuste, wzajemnie rozłączne podzbiory R i S-R. Sytuację taką ilustruje rys. 2.3.1c. Zaznaczono tam też symbolicznie wzajemne powiązania występujące między zbiorami R, S-R oraz X-S. Jeśli liczbę pogrubionych kresek interpretować jako siłę wzajemnych powiązań, a za relację porządkującą $\prec$ uznać zwykłą nieostrą nierówność $\leq$, to mamy: $g(R, X\text{-}S) = 1$ oraz $g(R, S\text{-}R) = 4$ ($g(S\text{-}R, X\text{-}S) = 2$ jest tu nieistotne). Jak więc łatwo zauważyć, dla tak wybranego rozbicia mnogości S warunki (2.3.5) są spełnione.



Rys. 2.3.1b. Idea zespołu minimalnego. Zbiór S (podzbiór mnogości X) podlegający testowaniu. X-S jest "zewnętrzem" zbioru S.



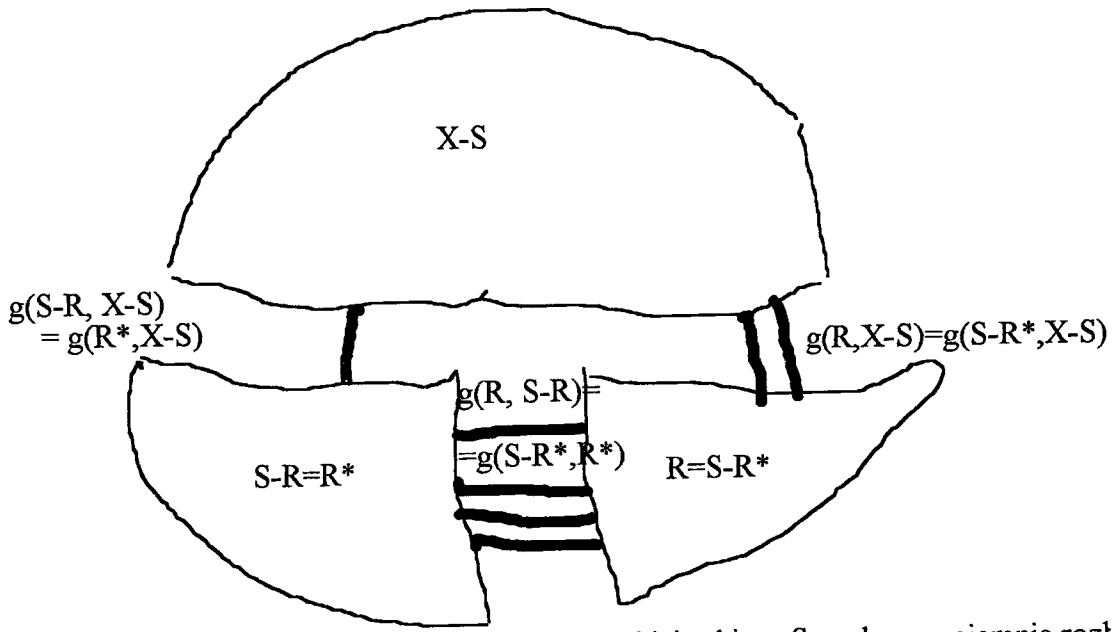
Rys. 2.3.1c. Idea zespołu minimalnego. Arbitralne rozbicie zbioru S na dwa wzajemnie rozłączne i niepuste podzbiory R oraz S-R. Zaznaczono powiązania: g(R, X-S) - między R a X-S, g(S-R, X-S) - między S-R a X-S, g(R, S-R) - między R a S-R.

W przypadku $|S| > 2$, zgodnie z treścią definicji 1 spełnienie warunków (2.3.5) dla pojedynczego, arbitralnie wybranego $R \subset S$, takiego że $\emptyset \neq R \neq S$, pozwala jedynie nie odrzucać hipotezy o minimalności S. Weryfikację należy kontynuować:

- aż do znalezienia takiego R, dla którego warunki (2.3.5) nie zachodzą - i wówczas stwierdzić, że S nie jest zespołem minimalnym;

- albo do wyczerpania wszystkich możliwości, tzn. do pozytywnego wyniku sprawdzenia warunków (2.3.5) w przypadku wszystkich możliwych podzbiorów $R \subset S$, takich że $\emptyset \neq R \neq S$ - i wtedy uzyskujemy pewność, że S jest zespołem minimalnym.

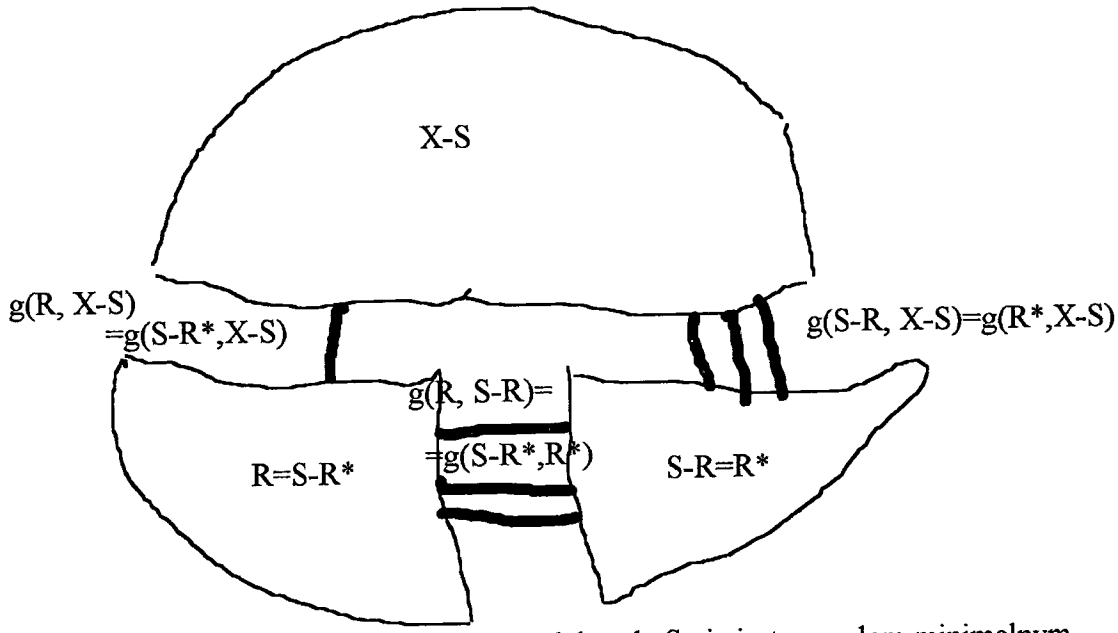
Rysunek 2.3.1d pokazuje dwoistość ukrytą w definicji 1 i wyjaśnia, dlaczego rozpatruje się w niej powiązania występujące między R i X-S oraz R i S-R, nie rozważa się natomiast siły powiązań między S-R i X-S. Mianowicie określenia R implikuje, że $\emptyset \neq S\text{-}R \neq S$, a więc warunki (2.3.5) dla $R^* = S\text{-}R$ były lub będą testowane, co wynika z kwantyfikatora ogólnego zawartego w treści definicji 1.



Rys. 2.3.1d. Idea zespołu minimalnego. Arbitralne rozbicie zbioru S na dwa wzajemnie rozłączne i niepuste podzbiory R oraz S-R. Zaznaczono powiązania: g(R, X-S) - między R a X-S, g(S-R, X-S) - między S-R a X-S, g(R, S-R) - między R a S-R.

Wreszcie na rys. 2.3.1e przedstawiono sytuację, w której S na pewno nie jest zespołem minimalnym. Mamy tutaj g(R, X-S) = 1, g(R, S-R) = 3 oraz g(S-R, X-S) = 3, a więc znowu warunki (2.3.5) są spełnione. Jednakże, zgodnie z uwagami odnoszącymi się do rys. 2.3.1d, w pewnej chwili rozpatrywać się będzie $R^* = S\text{-}R$ i wtedy $g(R^*, X\text{-}S) = g(S\text{-}R, X\text{-}S) = 3$, $g(R^*, S\text{-}R^*) = g(S\text{-}R, S\text{-}(S\text{-}R)) = 3$, a więc $g(R^*, X - S) = g(R^*, S\text{-}R^*)$ co przeczy (2.3.5b).

Podsumowując powyższe uwagi związane z interpretacją definicji 1 można krótko stwierdzić, że zespół minimalny, to taki niepusty podzbiór właściwy S (tzn. $S \neq X$) mnogości X, który jest silniej powiązany wewnętrznie (g(R, S-R)) niż z zewnętrzem (g(R, X-S) oraz g(S-R, X-S)).



Rys. 2.3.1e. Idea zespołu minimalnego. Przypadek, gdy S nie jest zespołem minimalnym.

Po zastosowaniu praw de Morgana do definicji 1 otrzymujemy:

Wniosek 1. Każdy podzbiór jednoelementowy {x} mnogości X (tzn. $x \in X$) jest zespołem minimalnym.

Jak więc widać do konstruowanego modelu nie ma potrzeby wprowadzania analogonu relacji (2.2.3) (w zasadzie relacja (2.2.3b) stanowi, wraz z nierównością (2.2.3a) formalny zapis stwierdzenia faktu, że każdy obiekt jest najsilniej powiązany sam ze sobą).

Można udowodnić następującą właściwość zespołów minimalnych:

Lemat 1. Dwa zespoły minimalne albo są wzajemnie rozłączne, albo też pozostają w relacji inkluzji.

Dowód. Niech S i U będą dwoma różnymi zespołami minimalnymi. Przypadki $S \subset U$ i $U \subset S$ są oczywiste. Zajmiemy się zatem jedynie wariantem, w którym jednocześnie $S \not\subset U$ i $U \not\subset S$. Oznaczmy $T = S \cap U$. Przypadek $T = \emptyset$ jest także oczywisty, więc możemy założyć, iż $T \neq \emptyset$. Dla wygody wprowadzimy oznaczenia: $R = S - T$, $W = U - T$ i $Y = X - (S \cup U)$. Oczywiście są spełnione zależności: $R \neq \emptyset$ i $W \neq \emptyset$. Z definicji 1 wynikają następujące relacje:

$$g(T, Y \cup R) = g(T, X - U) \lessdot g(T, W), \qquad (2.3.6a)$$

$$g(T, Y \cup R) = g(T, X - U) \neq g(T, W), \qquad (2.3.6b)$$

ponieważ $X - U = Y \cup R$. Ponadto $g(T, R) \lessdot g(T, Y \cup R)$, na podstawie (2.3.3), zatem (2.3.6a) i (2.3.6b) implikują, odpowiednio

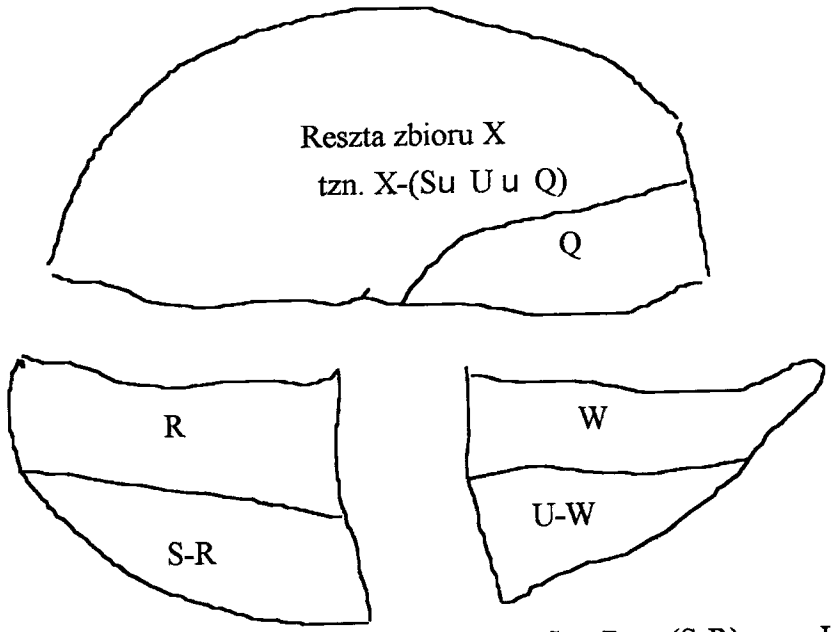
$$g(T, R) \lessdot g(T, W), \qquad (2.3.7a)$$

$$g(T, R) \neq g(T, W). \qquad (2.3.7b)$$

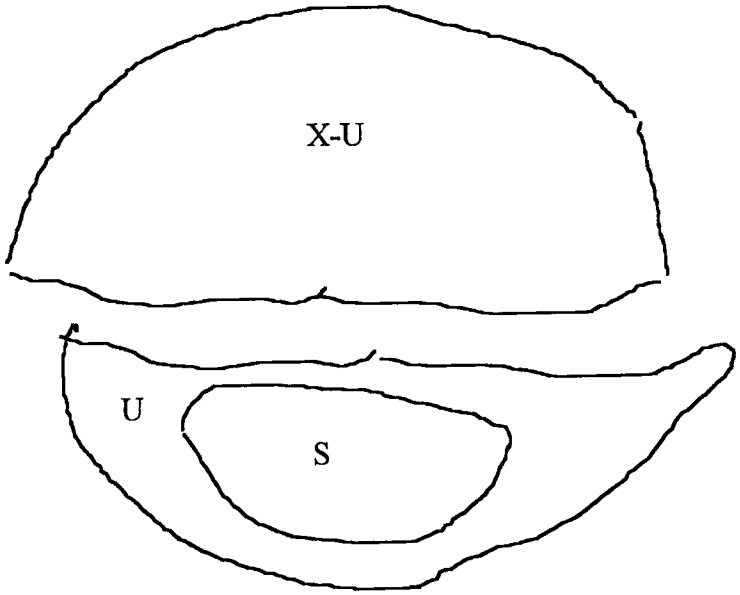
Przeprowadzając analogiczny wywód dla $g(T, Y \cup W) = g(T, X - S)$ oraz $g(T, R)$ otrzymujemy $g(T, W) \prec g(T, R)$, co pozostaje w sprzeczności z (2.3.7a) i (2.3.7b). W związku z tym przyjęte założenie, iż $T \neq \varnothing$ jest fałszywe. Zatem S i U muszą być rozłączne, co kończy dowód.

Treść lematu 1 zilustrowano na rys. 3.2a i 3.2b. Rysunek 3.2a dotyczy pierwszej części lematu. Przedstawiono na nim dwa zespoły minimalne S i U, $S \cap U = \varnothing$, gdzie: $S = R \cup (S-R)$, $\varnothing \neq R \neq S$ i $U = W \cup (U-W)$, $\varnothing \neq W \neq U$, a także rozłączny z nimi, niepusty zbiór Q. Z omawianej części lematu wynika, że żaden z następujących zbiorów: $R \cup W$, $R \cup (U-W)$, $(S-R) \cup W$, $(S-R) \cup (U-W)$, $Q \cup R$, $Q \cup (S-R)$, $Q \cup T$, $Q \cup (U-T)$ nie może być zespołem minimalnym. Jest to ważne sformułowanie, gdyż w teorii klasyfikacji zazwyczaj żąda się, aby ostateczny podział składał się z rozłącznych klas. Ponadto ma ono wydźwięk obliczeniowy: pozwala mianowicie zaniedbać rozpatrywania niektórych zbiorów, gdyż z góry wiadomo, że nie są one zespołami minimalnymi.

Druga część lematu 1 (por. rys. 3.2b) stwierdza, że teorię zespołów minimalnych można zaliczyć do klasy tzw. hierarchicznych technik aglomeracyjnych [25, 57].



Rys. 3.2a. Dwa wzajemnie rozłączne zespoły minimalne: $S = R \cup (S-R)$ oraz $U = W \cup (U-W)$ oraz niepusty zbiór Q, $Q \cap S = Q \cap U = \varnothing$.



Rys. 3.2b. Zespoły minimalne S i U zawarte jeden w drugim ($S \subset U$).

Z lematu 1 i wniosku 1 wynika natychmiast następujące twierdzenie:

Twierdzenie 1. Niech S, $|S| > 1$, będzie zespołem minimalnym w pewnym ustalonym systemie powiązań $\langle X, \mathbf{R}, \prec, g, g_0 \rangle$. Wówczas w tym systemie powiązań istnieje taka niepusta rodzina wzajemnie rozłącznych zespołów minimalnych $S_M = \{S_m: m \in M\}$, $S \notin S_M$, że

$$S = \bigcup_{m \in M} S_m$$

Innymi słowy, jeśli zespół minimalny zawiera więcej niż jeden element, to można go przedstawić jako sumę mnogościową pewnej liczby wzajemnie rozłącznych zespołów minimalnych, z których każdy ma o mniejszą liczbę elementów.

2.4. Zespoły minimalne: aspekty algorytmiczne obliczania siły powiązań

Przedstawiony w p. 2.3. model (uogólnionego) systemu powiązań $\langle X, \mathbf{R}, \prec, g, g_0 \rangle$ i definicja zespołu minimalnego (a także pewne podstawowe właściwości zespołów minimalnych) nie wystarczają do skonstruowania efektywnego algorytmu wyszukiwania zespołów minimalnych. Przyczyna takiego stanu rzeczy tkwi w braku metody wyznaczania wartości funkcji g na podstawie wartości powiązań elementarnych, tzn. obliczania np. $g(A, B)$, $A \cap B = \varnothing$, w oparciu o znajomość poszczególnych $g(\{x\}, \{y\})$, gdzie $x \in A$ i $y \in B$. Bieżący rozdział ma na celu wypełnić wspomnianą lukę.

Niech istnieje operacja binarna

$$\#: \mathbf{R} \times \mathbf{R} \to \mathbf{R}$$

określona w następujący sposób:

$$g(A, B \cup C) = g(A, B) \# g(A, C), \tag{2.4.1}$$

gdzie A, B i C są parami rozłącznymi, niepustymi podzbiorami mnogości X. Ze wzorów (2.3.2), (2.3.3) i (2.4.1) można wyprowadzić następujące właściwości omawianej operacji:

$a \# b = b \# a$, (przemienność) (2.4.2i)

$a \# (b \# c) = (a \# b) \# c$, (łączność) (2.4.2ii)

$a \prec a \# b$, (słaba monotoniczność) (2.4.2iii)

jeśli $b \prec c$, to $a \# b \prec a \# c$, (słaba monotoniczność) (2.4.2iv)

jeśli $a \prec b$ i $a \neq b$, to $a \prec a \# b$ i $a \neq a \# b$, (silna monotoniczność) (2.4.2v)

jeśli $a \prec c$ i $a \neq c$ oraz $b \prec c$ i $b \neq c$, to $a \# b \prec a \# c$ i $a \# b \neq a \# c$, (silna monotoniczność) (2.4.2vi)

jeśli $a \# b \prec a \# c$ i $a \# b \neq a \# c$, to $b \prec c$ i $b \neq c$, dla każdych $a, b, c \in \mathbf{R}$, (2.4.2vii)

natomiast $a \# b \prec a \# c$ i $a \# b \neq a \# c$ w ogólnym przypadku nie pociąga za sobą relacji $a \prec c$.

Wprowadzenie powyższej operacji binarnej # rozwiązuje problem wysłowiony na początku tego rozdziału, a mianowicie wystarcza do skonstruowania efektywnego algorytmu wyszukiwania zespołów minimalnych.

Inną konsekwencją wprowadzenia operacji # jest z jednej strony zautomatyzowanie metody wyliczania siły powiązań między zbiorami obiektów. Na przykład w przypadku skończonych zbiorów A i B takich, że $A, B \subset X$, $A \cap B = \varnothing$, ze wzoru (2.4.1) wprost wynika następująca zależność:

$$g(A, B) = g(\{x_1\}, \{y_1\}) \# g(\{x_2\}, \{y_1\}) \# ... \# g(\{x_{|A|}\}, \{y_1\})$$
$$\# g(\{x_1\}, \{y_2\} \# g(\{x_2\}, \{y_2\}) \# ... \# g(\{x_{|A|}\}, \{y_2\})$$
$$.....$$
$$\# g(\{x_1\}, \{y_{|B|}\}) \# g(\{x_2\}, \{y_{|B|}\}) \# ... \# g(\{x_{|A|}\}, \{y_{|B|}\})$$

gdzie przyjęto, iż $A = \{x_i : i = 1, 2, ... |A|\}$, $B = \{y_j : j = 1, 2, ... |B|\}$.
Z drugiej zaś strony konsekwencją wprowadzenia operacji # jest wyraźne rozdzielenie etapu określania wartości elementarnych powiązań międzyobiektowych (tzn. wartości wyrażeń typu $g(\{x\}, \{y\})$, $x, y \in X$, $x \neq y$) od wartości powiązań między rozłącznymi zbiorami obiektów (tzn. wartości wyrażeń typu $g(A, B)$, $A, B \subset X$, $A \cap B = \varnothing$). Umożliwia to elastyczne stosowanie aparatu zespołów minimalnych, np. wartości $g(\{x\}, \{y\})$ mogą być brane z uprzednio przygotowanych tabel lub też wyznaczane przy użyciu stosownego wzoru (np. jako miara Euklidesowa cech charakteryzujących elementy (obiekty) x i y). Sama zaś operacja # musi być umiejętnie powiązana z porządkiem liniowym $\langle \mathbf{R}, \prec \rangle$ (a więc ze zbiorem $\mathbf{R}$ i określoną na nim liniową relacją porządkującą $\prec$).

2.5. Właściwości zespołów minimalnych prowadzące do zwiększenia efektywności wyszukiwania zespołów minimalnych

Bieżący rozdział poświęcono wyprowadzeniu poszczególnych właściwości zespołów minimalnych. Kolejne lematy i twierdzenia mają tu przede wszystkim znaczenie teoretyczne i są trudno interpretowalne. Ich przytoczenie ma na celu usystematyzowane przedstawienie drogi prowadzącej do głównego rezultatu pracy, którym jest twierdzenie 5.

Lemat 2. S, $\varnothing \neq S \neq X$, jest zespołem minimalnym wtedy i tylko wtedy, gdy

$$g(S, X - S) \prec g(R, X - R), \tag{2.5.1a}$$

$$g(S, X - S) \neq g(R, X - R) \tag{2.5.1b}$$

zachodzi dla każdego niepustego podzbioru R zbioru S, $R \neq S$.

Dowód. Rozważmy niepuste zbiory R i S, $R \subset S \subset X$, $R \neq S \neq X$. Oznaczmy $T = S - R$ i $Y = X - S$. Zbiory T, R i Y są oczywiście niepuste i parami rozłączne, a ponadto $X - R = T \cup Y$ oraz $S = T \cup R$. Ze wzorów (2.3.2) i (2.4.1) wynika więc

$$g(R, X - R) = g(R, Y) \# g(R, T) \quad \text{i} \quad g(S, X - S) = g(R, Y) \# g(T, Y). \tag{2.5.2}$$

Załóżmy teraz, że S jest zespołem minimalnym. Wówczas z definicji 1 mamy:

$$g(R, Y) \prec g(R, T) \tag{2.5.3a}$$

$$g(R, Y) \neq g(R, T) \tag{2.5.3b}$$

oraz

$$g(T, Y) \prec g(R, T) \tag{2.5.4a}$$

$$g(T, Y) \neq g(R, T) \tag{2.5.4b}$$

skąd, po zastosowaniu zależności (2.4.2vi) otrzymujemy:

$$g(R, Y) \# g(T, Y) \prec g(R, Y) \# g(R, T) \tag{2.5.5a}$$

$$g(R, Y) \# g(T, Y) \neq g(R, Y) \# g(R, T) \tag{2.5.5b}$$

Uwzględnienie równości (2.5.2) w (2.5.5a) i (2.5.5b) prowadzi, odpowiednio, do (2.5.1a) i (2.5.1b), co kończy pierwszą część dowodu.

Załóżmy teraz, że spełnione są relacje (2.5.1a) i (2.5.1b), zaś R, S, T i Y są określone jak na początku tego dowodu. Po uwzględnieniu równości (2.5.2) i (2.3.2) w relacjach (2.5.1a) i (2.5.1b) otrzymujemy, odpowiednio, (2.5.5a) i (2.5.5b). Zastosowanie (2.4.2vii) prowadzi następnie do (2.5.4a) i (2.5.4b). Ponieważ R jest dowolnym niepustym podzbiorem mnogości S, więc to samo można stwierdzić o $T = S - R$. Innymi słowy warunki (2.3.5a) i (2.3.5b) są spełnione, co kończy cały dowód.

Lemat 3. Niech S, $S \subset X$, będzie zespołem minimalnym, R jego niepustym podzbiorem właściwym, zaś V, $V \subset X$, niech będzie niepustym zbiorem rozłącznym z S. Wówczas spełnione są następujące relacje:

$$g(V, X - V) \prec g(R \cup V, X - (R \cup V)) \tag{2.5.6a}$$

$$g(V, X - V) \neq g(R \cup V, X - (R \cup V)) \tag{2.5.6b}$$

Dowód. Oznaczmy $T = X - (R \cup V)$. Wówczas $X - V = T \cup R$, a ponadto:

$$g(V, X - V) = g(V, T) \# g(R, V) \tag{2.5.7}$$

$$g(R \cup V, X - (R \cup V)) = g(V, T) \# g(R, T), \tag{2.5.8}$$

z uwagi na (2.4.1). Ponieważ $S \cap V = \varnothing$, więc $V \subset X - S$ oraz $S \subset X - V$. Z tej ostatniej inkluzji wynika zaś $S - R \subset T$. Uwzględniając więc (2.3.3) można napisać:

$$g(R, S - R) \prec g(R, T) \tag{2.5.9}$$

$$g(R, V) \prec g(R, X - S) \tag{2.5.10}$$

Załóżmy teraz, że spełniona jest nierówność (2.5.6b) i (zamiast (2.5.6a)) relacja

$$g(R \cup V, X - (R \cup V)) \prec g(V, X - V). \tag{2.5.11}$$

Po rozwinięciu obu stron tej ostatniej zależności zgodnie z wzorami (2.5.7) i (2.5.8), a następnie po zastosowaniu reguły (2.4.2vii) otrzymujemy $g(R, T) \prec g(R, V)$ oraz $g(R, T) \neq g(R, V)$. Pozwala to na rozwinięcie (2.5.9) i (2.5.10) w następujący sposób: $g(R, S - R) \prec g(R, T) \prec g(R, V) \prec g(R, X - S)$, skąd wynika iż $g(R, S - R) \prec g(R, X - S)$. Zatem przeczy definicji zespołu minimalnego i dowodzi fałszywości wzoru (2.5.11), c.n.d.

Twierdzenie 2. Niech I i J, $J \subset I$, będą niepustymi, skończonymi zbiorami indeksów, $S_I = \{S_i: S_i \subset X, i \in I\}$ - rodziną parami rozłącznych zespołów minimalnych, $R_J = \{R_j: R_j \subset S_j, \varnothing \neq R_j \neq S_j, j \in J\}$ - rodziną niepustych podzbiorów właściwych rodziny S_I, zaś $Q \subset X$ niech będzie dowolną mnogością rozłączną z każdym elementem rodziny S_I. Jeśli zachodzi co najmniej jeden z następujących warunków:

$$|Q|\,|J| > 0 \tag{2.5.12}$$

$$|J| > 0 \tag{2.5.13}$$

to

$$P = Q \cup \bigcup_{j \in J} R_j \tag{2.5.14}$$

nie jest zespołem minimalnym. Ponadto warunek $|Q| > 0$ pociąga za sobą parę poniższych zależności:

$$g(Q, X - Q) \prec g(P, X - P) \tag{2.5.15a}$$

$$g(Q, X - Q) \neq g(P, X - P) \tag{2.5.15b}$$

Dowód. Ze wzorów (2.5.12) i (2.5.13) wynika, że $|J| \geq 1$. Zatem, ponieważ I i J są skończone, więc bez straty ogólności można przyjąć, że $I = \{1, 2, ..., |I|\}$ oraz $J = \{1, 2, ..., |J| < |I|\}$ i dowód przeprowadzić przez indukcję. Oznaczmy

$$P_k = Q \cup \bigcup_{j \in J,\, j \leq k} R_j \tag{2.5.16}$$

dla każdego $k \leq |J|$.

Przypadek 1. Niech $Q \neq \varnothing$. Jeśli $|J| = 1$, to prawdziwość twierdzenia wynika wprost z lematu 3. Załóżmy więc, że istnieje takie $m < |J|$, iż warunki

$$g(Q, X - Q) \prec g(P_m, X - P_m) \tag{2.5.17a}$$

$$g(Q, X - Q) \neq g(P_m, X - P_m) \tag{2.5.17b}$$

są spełnione. Po podstawieniu $V = P_m$ i $R = R_{m+1}$, wzory (2.5.6a) i (2.5.6b) (lemat 3) sprowadzają się do postaci:

$$g(P_m, X - P_m) \prec g(P_{m+1}, X - P_{m+1}), \tag{2.5.18a}$$

$$g(P_m, X - P_m) \neq g(P_{m+1}, X - P_{m+1}), \tag{2.5.18b}$$

a więc, po uwzględnieniu (2.5.17a) i (2.5.17b), otrzymujemy $g(Q, X - Q) \prec g(P_{m+1}, X - P_{m+1})$ oraz $g(Q, X - Q) \neq g(P_{m+1}, X - P_{m+1})$, co kończy drugi krok indukcyjny, dowodząc prawdziwości relacji (2.5.15a) i (2.5.15b). Z tych zaś ostatnich, na podstawie lematu 2, wniskujemy, że P nie jest zespołem minimalnym.

Przypadek 2. Niech $Q = \varnothing$. Wówczas z założeń wynika, że $|J| \geq 2$. Jeśli $|J| = 2$, to P nie jest zespołem minimalnym na podstawie lematu 1. Niech więc $|J| > 2$ i $m = |J| - 1$. Po podstawieniu $V = P_m$ i $R = R_{m+1}$ do wzorów (2.5.6a) i (2.5.6b) (lemat 3) znowu otrzymujemy relacje (2.5.18a) i

(2.5.18b). Zatem na podstawie lematu 2 stwierdzamy, że $P = P_m$ nie jest zespołem minimalnym, co kończy cały dowód.

Uwaga. W dalszej części tego rozdziału rozpatrywane będą elementy maksymalne i minimalne skończonych zbiorów liniowo uporządkowanych. Będą one, odpowiednio, oznaczane MAX i MIN, żeby odróżnić je od zwykłych operacji max i min odnoszących się do liczb rzeczywistych.

Twierdzenie 3. Niech I, S_I i R_I mają to samo znaczenie, co w twierdzeniu 2. Oznaczmy

$$R(J) = \bigcup_{j \in J} R_j \tag{2.5.19}$$

dla pewnego podzbioru J zbioru indeksów I. Zachodzą następujące relacje:

$$\mathrm{MAX}\{g(S_i, X - S_i): i \in I\} \lessdot g(R(I), X - R(I)), \tag{2.5.20a}$$

$$\mathrm{MAX}\{g(S_i, X - S_i): i \in I\} \neq g(R(I), X - R(I)), \tag{2.5.20b}$$

Dowód. Dowód przebiega przez indukcję. W przypadku $|I| = 1$ twierdzenie sprowadza się do treści lematu 2.

Niech $|I| > 1$. Bez straty ogólności można założyć, że $I = \{1, 2, ..., |I|\}$ oraz $J = \{1, 2, ..., |J| < |I|\}$. Przypuśćmy, że

$$\mathrm{MAX}\{g(S_i, X - S_i): i \in J\} \lessdot g(R(J), X - R(J)), \tag{2.5.21a}$$

$$\mathrm{MAX}\{g(S_i, X - S_i): i \in J\} \neq g(R(J), X - R(J)), \tag{2.5.21b}$$

Biorąc $Q = R(J)$ i $P = R(J^*)$, gdzie $J^* = J \cup \{|J| + 1\}$, z twierdzenia 2 otrzymujemy:

$$g(R(J), X - R(J)) \lessdot g(R(J^*), X - R(J^*)) \tag{2.5.22a}$$

$$g(R(J), X - R(J)) \neq g(R(J^*), X - R(J^*)), \tag{2.5.22b}$$

zatem po uwzględnieniu (2.5.21a) i (2.5.21b):

$$\mathrm{MAX}\{g(S_i, X - S_i): i \in J\} \lessdot g(R(J^*), X - R(J^*)), \tag{2.5.23a}$$

$$\mathrm{MAX}\{g(S_i, X - S_i): i \in J\} \neq g(R(J^*), X - R(J^*)), \tag{2.5.23b}$$

Wykorzystując ponownie twierdzenie 2 dla $Q = R_{|J|+1}$ i $P = R(J^*)$ mamy

$$g(R_{|J|+1}, X - R_{|J|+1}) \lessdot g(R(J^*), X - R(J^*)) \tag{2.5.24a}$$

$$g(R_{|J|+1}, X - R_{|J|+1}) \neq g(R(J^*), X - R(J^*)) \tag{2.5.24b}$$

Ponieważ $S_{|J|+1}$ jest zespołem minimalnym, więc lematu 2 (dla $S = S_{|J|+1}$ i $R = R_{|J|+1}$) otrzymujemy

$g(S_{|J|+1}, X - S_{|J|+1}) \lessdot g(R_{|J|+1}, X - R_{|J|+1})$ i $g(S_{|J|+1}, X - S_{|J|+1}) \neq g(R_{|J|+1}, X - R_{|J|+1})$,

co po wykorzystaniu (2.5.24a) i (2.5.24b) prowadzi do:

$$g(S_{|J|+1}, X - S_{|J|+1}) \lessdot g(R(J^*), X - R(J^*)) \tag{2.5.25a}$$

$$g(S_{|J|+1}, X - S_{|J|+1}) \neq g(R(J^*), X - R(J^*)) \tag{2.5.25b}$$

Relacje (2.5.23a) i (2.5.23b) oraz (2.5.25a) i (2.5.25b) implikują

$$\mathrm{MAX}\{g(S_i, X - S_i): i \in J^*\} \lessdot g(R(J^*), X - R(J^*)), \tag{2.5.26a}$$

$$\mathrm{MAX}\{g(S_i, X - S_i): i \in J^*\} \neq g(R(J^*), X - R(J^*)), \tag{2.5.26b}$$

co kończy drugi krok indukcyjny, a zarazem cały dowód.

Twierdzenie 4. Niech I i S_I mają to samo znaczenie co w twierdzeniu 2. Oznaczmy

$$S(J) = \bigcup_{j \in J} S_j \tag{2.5.27}$$

dla pewnego podzbioru J zbioru indeksów I. Niech $S(I) \neq X$. Jeśli S(J) nie jest zespołem minimalnym dla jakiegokolwiek $J \subset I$, $|J| > 1$, to zachodzi następująca relacja:

$$MIN\{g(S_i, X - S_i): i \in I\} \prec g(S(I), X - S(I)), \tag{2.5.28}$$

Dowód. Przyjmujemy, że $\alpha = 0$ i oznaczamy

$$J^\alpha = I \tag{2.5.29}$$

Krok α.

Ponieważ $S(J^\alpha)$ nie jest zespołem minimalnym w przypadku $|J^\alpha| > 1$, więc z lematu 2 wynika istnienie niepustego zbioru $P^\alpha \subset S(J^\alpha)$, $P^\alpha \neq S(J^\alpha)$, dla którego zachodzi relacja

$$g(P^\alpha, X - P^\alpha) \prec g(S(J^\alpha), X - S(J^\alpha)) \tag{2.5.30}$$

W ogólnym przypadku $P^\alpha = R(K^\alpha) \cup S(M^\alpha)$, przy czym K^α i M^α są rozłączne, $\varnothing \neq K^\alpha \cup M \subset J^\alpha$. Stąd i z warunku $P^\alpha \neq S(J^\alpha)$ wynika ostra nierówność

$$|M^\alpha| < |J^\alpha| \tag{2.5.31}$$

Przypadek 1. $K^\alpha \neq \varnothing$ i $M^\alpha = \varnothing$. Na podstawie twierdzenia 3 możemy napisać $MAX\{g(S_i, X - S_i): i \in K^\alpha\} \prec g(R(K^\alpha), X - R(K^\alpha)) = g(P^\alpha, X - P^\alpha)$, skąd, po uwzględnieniu wzoru (2.5.30) otrzymujemy: $MAX\{g(S_i, X - S_i): i \in K^\alpha\} \prec g(S(J^\alpha), X - S(J^\alpha))$. Ponadto $MIN\{g(S_i, X - S_i): i \in J^\alpha\} \prec MIN\{g(S_i, X - S_i): i \in K^\alpha\} \prec MAX\{g(S_i, X - S_i): i \in K^\alpha\}$, a więc w rezultacie $MIN\{g(S_i, X - S_i): i \in J^\alpha\} \prec g(S(J^\alpha), X - S(J^\alpha))$. Jeśli teraz $\alpha = 0$, to po uwzględnieniu równości $J^\alpha = I$, kończymy dowód tego wariantu. Jeśli zaś $\alpha \neq 0$, to przechodzimy do przypadku oznaczonego jako "Wariant iteracyjny".

Przypadek 2. $K^\alpha \neq \varnothing$ i $M^\alpha \neq \varnothing$. Po podstawieniu $Q = S(M^\alpha) \neq \varnothing$ otrzymujemy $g(S(M^\alpha), X - S(M^\alpha)) = g(Q, X - Q) \prec g(P^\alpha, X - P^\alpha)$, zgodnie z treścią twierdzenia 2. Uwzględnienie wzoru (2.5.30) prowadzi do relacji $g(S(M^\alpha), X - S(M^\alpha)) \prec g(S(J^\alpha), X - S(J^\alpha))$. Jeśli teraz $|M^\alpha| = 1$, to dalsza część dowodu sprowadza się do postępowania opisanego niżej jako "Przypadek 3". W przeciwnym razie otrzymujemy przypadek 4.

Przypadek 3. $K^\alpha = \varnothing$ i $|M^\alpha| = 1$. Wówczas warunek (2.5.30) przechodzi w $g(S_j, X - S_j) \prec g(S(J^\alpha), X - S(J^\alpha))$ dla pewnego $j \in J^\alpha$, a ponadto mamy $MIN\{g(S_j, X - S_j): j \in J^\alpha\} \prec g(S_j, X - S_j)$, co po uwzględnieniu poprzedniej relacji prowadzi do $MIN\{g(S_i, X - S_i): i \in J^\alpha\} \prec g(S(J^\alpha), X - S(J^\alpha))$ i kończy dowód tego wariantu w przypadku, w którym $\alpha = 0$. Jeśli $\alpha \neq 0$, to przechodzimy do przypadku oznaczonego jako "Wariant iteracyjny".

Przypadek 4. $K^\alpha = \varnothing$ i $|M^\alpha| > 1$. Zgodnie z założeniem $S(M^\alpha)$ także nie jest zespołem minimalnym. Zwiększamy wartość α o 1, oznaczamy

$$J^{\alpha} = M^{\alpha-1} \qquad (2.5.32)$$

i powracamy do punktu postępowania oznaczonego jako krok α.

Pozostało teraz do wykazania, że opisana wyżej procedura postępowania prowadzi do relacji (2.5.28). Zauważmy przede wszystkim, że koniec postępowania osiąga się po stwierdzeniu przypadku 1 lub 3 i , ewentualnie po opisanym niżej tzw. wariancie iteracyjnym. Przypadek 2 jest typowym wariantem przejściowym, zaś przypadek 4 powoduje wznowienie postępowania dowodowego przy zmodyfikowanych warunkach początkowych.

Rozpoczniemy od wykazania, że postępowanie dowodowe zamyka się w skończonej liczbie kroków. Jeśli stwierdziliśmy przypadek 1 lub 3, a ponadto $\alpha = 0$, to dowód kończy się natychmiast. W przeciwnym razie musieliśmy choć raz wykryć przypadek 4. Ze wzorów (2.5.29), (2.5.30) i (2.5.32) wynika następujący ciąg relacji:

$$|I| = |J^0| > |M^0| = |J^1| > |M^1| = |J^2| > |M^2| \ldots.$$

Ponieważ I jest zbiorem skończonym, a po każdym kroku zakończonym przypadkiem 4 moc zbioru J^{α} ulega zmniejszeniu, więc w myśl nierówności (2.5.31) to samo dotyczy M^{α}. Dlatego też w skończonej liczbie kroków osiągany jest warunek $|M^{\alpha}| = 1$ (definicja przypadku 3), bądź też warunek $M^{\alpha}| = \varnothing$ (definicja przypadku 1).

Wariant iteracyjny. Zauważmy, że przypadek 4 dotyczy dwu sytuacji. W pierwszej z nich jest on stwierdzany bezpośrednio, i wówczas $P^{\alpha} = S(M^{\alpha}) = J^{\alpha+1}$, a następnie, zgodnie ze zmodyfikowanym wzorem (2.5.30):

$$g(P^{\alpha+1}, X - P^{\alpha+1}) \lessdot g(S(J^{\alpha+1}), X - S(J^{\alpha+1})) = g(P^{\alpha}, X - P^{\alpha}) \lessdot g(S(J^{\alpha}), X - S(J^{\alpha})).$$

W sytuacji drugiego rodzaju, która dotyczy stwierdzenia przypadku 2 i przejścia od niego do przypadku 4 mamy co prawda $P^{\alpha} \supset S(M^{\alpha}) = J^{\alpha+1}$ i $P^{\alpha} \neq S(M^{\alpha}) = J^{\alpha+1}$ ale ponadto: $g(P^{\alpha+1}, X - P^{\alpha+1}) \lessdot g(S(J^{\alpha+1}), X - S(J^{\alpha+1})) = g(S(M^{\alpha}), X - S(M^{\alpha})) \lessdot g(P^{\alpha}, X - P^{\alpha}) \lessdot g(S(J^{\alpha}), X - S(J^{\alpha}))$, znowu w myśl zmodyfikowanego wzoru (2.5.30). W obu zatem sytuacjach powstaje następujący ciąg relacji:

$$g(S(J^{\alpha}), X - S(J^{\alpha})) \lessdot g(S(J^{\alpha-1}), X - S(J^{\alpha-1})) \lessdot \ldots$$
$$\lessdot g(S(J^1), X - S(J^1)) \lessdot g(S(J^0), X - S(J^0)) = g(S(I), X - S(I))$$

Wobec tego stwierdzenie, że (por. przypadki 1 i 3)

$$MIN\{g(S_i, X - S_i): i \in J^{\alpha}\} \lessdot g(S(J^{\alpha}), X - S(J^{\alpha}))$$

pociąga za sobą zależność

$$MIN\{g(S_i, X - S_i): i \in J^{\alpha}\} \lessdot g(S(I), X - S(I)), \qquad (2.5.33)$$

na podstawie przechodności relacji liniowo porządkującej. Ponadto, ponieważ dla każdego α mamy $J^{\alpha} \subset I$, więc z właściwości operacji MIN wynika:

$$MIN\{g(S_i, X - S_i): i \in I\} \lessdot MIN\{g(S_i, X - S_i): i \in J^{\alpha}\} \qquad (2.5.34)$$

co, wraz z (2.5.33) prowadzi wprost do (2.5.28) również w przypadku $\alpha \neq 0$. c.n.d.

Twierdzenie 5. Niech I, $|I| > 1$, będzie skończonym zbiorem indeksów, zaś S_I - rodziną parami rozłącznych zespołów minimalnych taką, że $S(I) \neq X$. Niech S(J) nie będzie zespołem

minimalnym dla żadnego $J \subset I$, $1 < |J| < |I|$. Wówczas S(I) jest zespołem minimalnym wtedy i tylko wtedy, gdy:

$$g(S(I), X - S(I)) \prec MIN\{g(S_i, X - S_i): i \in I\} \quad (2.5.35a)$$

$$g(S(I), X - S(I)) \neq MIN\{g(S_i, X - S_i): i \in I\} \quad (2.5.35b)$$

Dowód. Konieczność wynika bezpośrednio z lematu 2.

Dostateczność. Rozważmy niepusty zbiór $P \subset S(I)$, $P \neq S(I)$. W ogólnym przypadku można napisać $P = R(K) \cup S(M)$, przy czym K i M są rozłącznymi zbiorami indeksów, $\emptyset \neq K \cup M \subset I$. Jeśli $K \neq \emptyset$ i $M = \emptyset$, to

$$g(S(I), X - S(I)) \prec g(P, X - P) \quad (2.5.36a)$$

$$g(S(I), X - S(I)) \neq g(P, X - P) \quad (2.5.36b)$$

w myśl twierdzenia 3 oraz wzorów (2.5.35a) i (2.5.35b). Jeśli $K = \emptyset$ i $M \neq \emptyset$, to relacje (2.5.36a) i (2.5.36b) wynikają z zastosowania twierdzenia 4, a następnie (2.5.35a) i (2.5.35b).

Do rozpatrzenia pozostał zatem jedynie wariant, w którym $K \neq \emptyset$ i $M \neq \emptyset$. Pisząc $Q = S(M)$ otrzymujemy (2.5.15a) i (2.5.15b), zgodnie z twierdzeniem 2. Z założeń wynika, że jeśli $1 < |M|$ (a zatem $1 < |M| < |I|$), to S(M) nie jest zespołem minimalnym. Wobec tego zachodzi

$$MIN\{g(S_i, X - S_i): i \in M\} \prec g(S(M), X - S(M)), \quad (2.5.37)$$

w myśl twierdzenia 4, a następnie po wykorzystaniu przed chwilą otrzymanej specjalnej wersji (2.5.15a) i (2.5.15b):

$$MIN\{g(S_i, X - S_i): i \in M\} \prec g(S(M), X - S(M)) \prec g(P, X - P)$$

$$MIN\{g(S_i, X - S_i): i \in M\} \prec g(S(M), X - S(M)) \neq g(P, X - P),$$

skąd, po zauważeniu, że $MIN\{g(S_i, X - S_i): i \in I\} \prec MIN\{g(S_i, X - S_i): i \in M\}$ i po zastosowaniu (2.5.35a) i (2.5.35b) otrzymuje się, odpowiednio, (2.5.36a) i (2.5.36b).
Jeśli zaś $|M| = 1$, to wzór (2.5.37) także zachodzi i całe dotychczasowe rozumowanie pozostaje bez zmian.

Reasumując, dla każdego niepustego $P \subset S(I)$, $P \neq S(I)$ spełnione są warunki (2.5.36a) i (2.5.36b), zatem na podstawie lematu 2 S(I) jest zespołem minimalnym, co kończy dowód warunku dostatecznego, a zarazem całego twierdzenia.

Twierdzenie 5 poddaje sugestie jak weryfikować czy dany zbiór S stanowi zespół minimalny, czy też nie. Metoda ta jest w praktyce znacznie efektywniejsza niż sposób polegający na bezpośrednim wykorzystaniu definicji 1 lub lematu 2. Efektywność metody najłatwiej pokazać na przykładzie. Przypuśćmy mianowicie, że sprawdzamy czy $S = S(\{1, 2, 3, 4\})$ jest zespołem minimalnym, przy czym uprzednio już stwierdzono, że poszczególne jego komponenty S_i, $i = 1, 2, 3, 4$ są zespołami minimalnymi. Niech $|S_i| = 50$, dla kazdego $i = 1, 2, 3, 4$, co w praktyce jest uważane za problem o niewielkiej wymiarowości. Wówczas korzystając z definicji 1 lub z lematu 2 należałoby wykonać $2^{200} - 2 \cong 1{,}606937 * 10^{60}$ testów, zaś metoda sugerowana w twierdzeniu 5 wymaga zastosowania tylko 4 porównań.

2.6. Oszacowanie liczby zespołów minimalnych

Na podstawie właściwości przytoczonych w p. 2.4 i 2.5 można sformułować opisany poniżej "naiwny", iteracyjny algorytm wyszukiwania zespołów minimalnych w przypadku, w którym X jest zbiorem skończonym. Ponieważ X jest zbiorem skończonym, więc istnieje skończona liczba jego niepustych podzbiorów (jest ich dokładnie $2^{|X|}-1$). Zatem zespołów minimalnych jest nie więcej niż $2^{|X|}-1$. Ponadto w myśl Lematu 1 dwa zespoły minimalne są albo wzajemnie rozłączne, albo też jeden z nich w całości zawiera się w tym drugim. Zatem narzuca się następujący sposób wyszukiwania zespołów minimalnych:

Faza wyszukiwania

1. Każdy $\{x\}$, $x \in X$, jest zespołem minimalnym (por. Wniosek 1).
2. Określa się roboczy zbiór U, wstępnie $U \leftarrow X$ (symbol $\leftarrow$ oznacza operację przypisania).
3. Niech $r \leftarrow 2$.
4. Wybieramy kolejną r-kę elementów zbioru U (tzn. podzbiór zbioru U o postaci $\{x_1, x_2,...,x_{r-1}, x_r\}$, gdzie oczywiście $|\{x_1, x_2,...,x_{r-1}, x_r\}| = r$).
5. Jeśli $\{x_1, x_2,...,x_{r-1}, x_r\}$ stanowi zespół minimalny (celu sprawdzenia wykorzystujemy Lemat 2), to modyfikujemy: $U \leftarrow U - \{x_1, x_2,...,x_{r-1}, x_r\}$.
6. Jeśli zbadano wszystkie r-ki, to przechodzimy do kroku 8.
7. Powracamy do kroku 4.
8. Jeśli $U = \varnothing$, to koniec algorytmu.
9. Modyfikujemy $r \leftarrow r + 1$.
10. Jeśli $|U| < r$, to koniec algorytmu.
11. Powracamy do kroku 4.

Łatwo zauważyć, że w opisany wyżej sposób otrzymujemy co najmniej |X| zespołów minimalnych jednoelementowych i zero lub więcej wzajemnie rozłącznych zespołów minimalnych o mocy nie mniejszej niż 2. Jeśli nie znaleziono żadnego zespołu minimalnego o mocy co najmniej dwa, to na tym należy postępowanie zakończyć. W przeciwnym przypadku należy przejść do fazy agregacji.

Faza agregacji

Faza agregacji polega na przeformułowaniu problemu w wyniku zastąpienia każdego znalezionego w kroku 5 fazy wyszukiwania zespołu minimalnego o postaci $A = \{x_1, x_2,...,x_{|A|-1}, x_{|A|}\}$ pojedynczym elementem, równocześnie także na zmianie wartości wzajemnych powiązań między odpowiednikiem wspomnianego A a odpowiednikiem zespołu minimalnego $B = \{y_1, y_2,...,y_{|B|-1}, y_{|B|}\}$ zgodnie ze wzorem:

$$g(A, B) = g(\{x_1\}, \{y_1\}) \# g(\{x_2\}, \{y_1\}) \# ... \# g(\{x_{|A|}\}, \{y_1\})$$
$$\# g(\{x_1\}, \{y_2\} \# g(\{x_2\}, \{y_2\}) \# ... \# g(\{x_{|A|}\}, \{y_2\})$$
$$.....$$
$$\# g(\{x_1\}, \{y_{|B|}\}) \# g(\{x_2\}, \{y_{|B|}\}) \# ... \# g(\{x_{|A|}\}, \{y_{|B|}\})$$

Po dokonaniu fazy agregacji należy przejść do fazy wyszukiwania, itd.

Decyzję o zakończeniu algorytmu podejmuje się, jak widać, po wykonaniu fazy wyszukiwania. Zakończenie następuje po skończonej liczbie naprzemiennie wykonanych faz wyszukiwania i agregacji, gdyż w pewnym momencie nie zostanie znaleziony nowy zespół minimalny o mocy co najmniej 2. Istotnie, w każdej z faz agregacji wymiar problemu (tzn. liczba badanych elementów) zmniejsza się co najmniej o 1.

Ponadto możność zastąpienia testowania wartości powiązań między wszystkimi podzbiorami zespołu minimalnego przez testowanie jedynie wartości powiązań miedzy wzajemnie rozłącznymi zespołami minimalnymi zapewnia twierdzenie 5, co kończy dowód poprawności przytoczonego wyżej algorytmu wyszukiwania zespołów minimalnych.

Przez $a_j^{(i)}$ oznaczymy teraz liczbę zespołów minimalnych skonstruowanych z j elementów zbioru X w wyniku zagregowania uprzednio znalezionych zespołów minimalnych o mocy co najmniej i. Ponieważ dwa zespoły minimalne są albo wzajemnie rozłączne, albo też jeden z nich w całości zawiera się w tym drugim (por. Lemat 1), więc:

$$|X| \geq (i+1)a_{i+1}^{(i)} + (i+2)\, a_{i+2}^{(i)} + \ldots + (|X| - 1)\, a_{|X|-1}^{(i)} \qquad (2.6.1)$$

dla i = 1, 2, ...,|X| - 2. Maksymalizując sumę

$$W(i) = a_{i+1}^{(i)} + a_{i+2}^{(i)} + \ldots + a_{|X|-1}^{(i)} \qquad (2.62)$$

przy uwzględnieniu ograniczenia danego wzorem (2.61) otrzymujemy:

$a_k^{(i)} = 0$ dla każdego $k > i + 1$ oraz $a_{i+1}^{(i)} = \text{entier}\left(\frac{|X|}{i+1}\right)$, gdzie i = 1, 2, ...,|X| - 2. Zatem majorantą dla ciągu X, W(1), W(2),..., W(|X| - 2) jest ciąg o postaci $|X|, \frac{|X|}{2}, \frac{|X|}{3}, \ldots, \frac{|X|}{|X|-1}$. Wobec tego liczba zespołów minimalnych jest nie większa od $|X|[\gamma + \ln(|X|-1) + 1]$, gdzie $\gamma = 0{,}577\ldots$ jest stałą Eulera.

Zauważmy teraz, że zastosowanie wyżej opisanego "naiwnego" algorytmu wyszukiwania zespołów minimalnych w najgorszym przypadku, w którym nie istnieją zespoły minimalne dwu i więcej elementowe, powoduje konieczność sprawdzenia dokładnie $2^{|X|}$-2 niepustych podzbiorów właściwych mnogości X. Zatem, tego typu algorytm jest wysoce nieefektywny. W związku z tym narodziła się konieczność poszukiwania lepszych, efektywniejszych algorytmów, o czym mowa w pod koniec p. 2.5 i w p. 2.7.

2.7. Szczególne rodzaje zespołów minimalnych - zespoły Add - minimalne i zespoły Max - minimalne

Jak wynika z treści p. 2.3 przedstawiony w p. 2.2 model (uogólnionego) systemu powiązań $\langle X, \mathbf{R}, \prec, g, g_0 \rangle$ i definicja zespołu minimalnego (a także przytoczone tam podstawowe właściwości zespołów minimalnych) nie wystarczają do skonstruowania efektywnego algorytmu wyszukiwania zespołów minimalnych. Przyczyna takiego stanu rzeczy tkwi w zbyt ogólnym sformułowaniu metody wyznaczania wartości funkcji g na podstawie wartości powiązań elementarnych, tzn. obliczania np. $g(A, B)$, $A \cap B = \emptyset$, w oparciu o znajomość poszczególnych $g(\{x\}, \{y\})$, gdzie $x \in A$ i $y \in B$ - patrz zamieszczone w p. 2.4 określenie operacji binarnej #. Inną przyczyną jest nieuporządkowanie fazy wyszukiwania w "naiwnym" algorytmie wyznaczania zespołów minimalnych (por. dalej - dyskusja złożoności obliczeniowej algorytmów wyszukiwania zespołów Add - minimalnych i zespołów Max - minimalnych).

Ponadto, mimo że w rozważaniach teoretycznych elegancko i wygodnie jest rozpatrywać przypadki jak najogólniejsze, to w praktyce i tak komputer prowadzi obliczenia wykorzystując konkretną postać operacji #.

Dosyć często używanymi w praktyce przykładami systemów składających się z porządku liniowego $\langle \mathbf{R}, \prec \rangle$ i operacji # są:

Lp.	Zbiór $\mathbf{R}$	Relacja $\prec$	Operacja #
1.	$\mathbb{R}^+ \cup \{0\} = \{r: r \in \mathbb{R}, r \geq 0\}$ (zbiór liczb rzeczywistych nieujemnych)	$\leq$ (większe lub równe)	+ (zwykłe dodawanie)
2.	$\mathbb{R}^+ \cup \{0\} = \{r: r \in \mathbb{R}, r \geq 0\}$ (zbiór liczb rzeczywistych nieujemnych)	$\leq$ (większe lub równe)	max (operacja maksimum)
3.	$\mathbb{R}^+ \cup \{0\} = \{r: r \in \mathbb{R}, r \geq 0\}$ (zbiór liczb rzeczywistych nieujemnych)	$\geq$ (mniejsze lub równe)	min (operacja minimum)

W pierwszym przypadku mówimy o zespołach Add - minimalnych (tzn. o zespołach minimalnych z addytywną wartością powiązań, zwanych niekiedy klasycznymi zespołami minimalnymi) [22, 23, 24, 25, 26, 27, 31, 52, 53, 58]. Wariant drugi odnosi się do tzw. zespołów Max-minimalnych [60, 62, 63, 64].

2.7.1. Zespoły Add - minimalne

Mała efektywność wyznaczania zespołów Add - minimalnych wymusiła prowadzenie badań nad tego rodzaju obiektami. Między innymi w pracy [26] zastanawiano się nad często spotykanym w praktyce przypadkiem, w którym elementarne powiązania międzyobiektowe mają taką samą wartość dla pewnych zbiorów elementów mnogości X. Jednakże otrzymane rezultaty okazały się niezbyt użyteczne w praktyce. Innym kierunkiem badań była próba wykorzystania aparatu teorii grafów. Szczególnie owocne okazały się prace nad związkami między zespołami Add - minimalnymi a przekrojami w grafie. Pierwsze wyniki opublikowano w [24] i [51]. Doprowadziły

one do sformułowania algorytmu o wielomianowej złożoności obliczeniowej (dokładnie o złożoności $O(|X|^5)$), którego pierwszą wersję zaproponowano w [58], zaś zweryfikowaną postać w [59].

Efektywny algorytm wyszukiwania zespołów Add - minimalnych wykorzystuje pojęcie macierzy przepustowości granicznych grafu [33]. Macierzy przepustowości granicznych grafu skończonego o zbiorze wierzchołków X jest macierzą kwadratową $|X| \times |X|$, której (x, y) - ty element, $x, y \in X$, $x \neq y$, jest równy wartości przekroju minimalnego rozdzielającego wspomniane wierzchołki x i y, zaś na głównej diagonali (elementy (x, x) - te) umieszczone są tzw. elementy neutralne.

Przypuśćmy teraz, że $M = [m_{xy}]_{|X|\times|X|}$ jest macierz kwadratową z elementem neutralnym na głównej przekątni. Jeżeli sukcesywna i równoczesna zamiana wierszy i kolumn (tzn. taka, że zamianie miejscami wiersza x z wierszem y towarzyszy równocześnie zamiana miejscami kolumny x z kolumną y) prowadzi do powstania nowej macierzy o następującej postaci blokowej:

$$M^{(P)} = \begin{bmatrix} M_{(a)} & M_{(c)} \\ M_{(d)} & M_{(b)} \end{bmatrix},$$

gdzie $M_{(a)}$ i $M_{(b)}$ są znowu macierzami kwadratowymi zawierającymi na głównych diagonalach element neutralny, a ponadto każdy element macierzy $M_{(c)}$ przybiera tę samą wartość równą $\min\{m_{xy} : x, y \in X, x \neq y\}$, to $M^{(P)}$ nosi nazwę podziału głównego macierzy M, zaś $M_{(a)}$ i $M_{(b)}$ są nazywane głównymi podmacierzami macierzy M. Zachodzi następujące twierdzenie [33]:

Twierdzenie 6. Symetryczna macierz kwadratowa $|X|\times|X|$ jest macierzą przepustowości granicznych grafu ważonego o zbiorze wierzchołków X wtedy i tylko wtedy, gdy istnieje podział główny tej macierzy, a także istnieją podziały główne głównych podmacierzy macierzy pierwotnej, itd. (podziały główne przeprowadzane są do momentu, w którym dzielona podmacierz nie stanie się macierzą jednoelementową składającą się jedynie z elementu neutralnego). Ponadto każda macierz przepustowości granicznych grafu ważonego posiada realizację drzewiastą.

Dowód wspomnianego twierdzenia można znaleźć w [33] na stronach 431-434.

Opierając się na powyżej cytowanym wyniku w pracy [59] (por Proposition 2) udowodniono:

Twierdzenie 7. S jest zespołem Add - minimalnym wtedy i tylko wtedy, gdy jest zespołem Add - minimalnym w dowolnej ścieżce realizującej macierz przepustowości granicznych grafu ważonego o zbiorze wierzchołków X i wagach krawędzi przybierających wartości $g(\{x\}, \{y\})$, $x, y \in X$, $x \neq y$.

Ważny wniosek wynikający z twierdzenia 7 mówi, że zespół Add - minimalny mogą tworzyć jedynie elementy mnogości X reprezentowane jako kolejne, następujące bezpośrednio po sobie wierzchołki ścieżki będącej jedną z realizacji macierzy przepustowości granicznych grafu ważonego o zbiorze wierzchołków X. Zatem (przy szacowaniu złożoności algorytmu, która wynosi $O(|X|^5)$, brano również pod uwagę liczbę elementarnych operacji logicznych i

arytmetycznych wykonywanych w trakcie sprawdzania czy badany podzbiór mnogości X rzeczywiście jest zespołem Add - minimalnym), w najgorszym wariancie, który stanowi przypadek nie istnienia zespołów Add - minimalny dwu i więcej elementowych należy rozpatrzeć |X| - 1 par elementów, |X| - 2 trójek elementów, |X| - 3 czwórek elementów, ..., wreszcie 2 podzbiory (|X| - 1) - elementowe, a więc $O(|X|^2)$ możliwości. Stanowi to istotny postęp w porównaniu z przedstawionym w p. 3 "naiwnym" algorytmem, gdzie w analogicznej sytuacji należało zbadać $O(2^{|X|}-2)$ podzbiorów. Jeśli |X| = 200, to udoskonalony algorytm wymaga przeprowadzenia $4*10^4$ testów, zaś poprzedni około $1{,}606937*10^{60}$, a więc $4*10^{55}$ raza więcej.

Zmodyfikowany algorytm wyszukiwania zespołów Add - minimalnych składa się, podobnie jak "naiwny" algorytm zarysowany w p. 2.6, z fazy wyszukiwania i fazy agregacji, poprzedzonych jednakże fazą wstępną, w której tworzona jest ścieżka realizująca macierz przepustowości granicznych grafu ważonego o zbiorze wierzchołków X i wagach krawędzi przybierających wartości $g(\{x\}, \{y\})$, $x, y \in X$, $x \neq y$. Ponadto zmodyfikowany jest krok 4 fazy wyszukiwania, w którym zamiast wyboru dowolnej, kolejnej r-ki elementów zbioru U (tzn. podzbiór zbioru U o postaci $\{x_1, x_2,...,x_{r-1}, x_r\}$, dokonuje się wyznaczenia również podzbioru r-elementowego mnogości U, ale reprezentowanego przez sąsiadujące za sobą wierzchołki ścieżki realizującej macierz przepustowości granicznych.

2.7.2. Zespoły Max - minimalne

Szukając efektywnego algorytmu wyznaczania zespołów Max - minimalnych wykorzystano doświadczenia nabyte przy zespołach Add - minimalnych. Należało tu jednak zmodyfikować pojęcie wartości przekroju w grafie ważonym. Uczyniono to, wprowadzając następującą definicję:

Definicja 2. Niech C będzie przekrojem w grafie ważonym krawędziowo i niech $\{A_{1C}, A_{2C}\}$ będzie podziałem zbioru wierzchołków grafu generowanym przez przekrój C. Wartością Max - przekroju nazywamy liczbę $V(C) = \max\{v(x, y): x \in A_{1C}, y \in A_{2C}\}$.

Dokładne przestudiowanie dowodu twierdzenia 6 (por. [62]) zaowocowało wnioskiem, że nie bierze się w nim pod uwagę sposobu wyznaczania wartości przekroju. Zatem z twierdzenia 6 można również skorzystać i w przypadku zespołów Max - minimalnych. W rezultacie otrzymujemy [62]:

Twierdzenie 8. S jest zespołem Max - minimalnym wtedy i tylko wtedy, gdy jest zespołem Max - minimalnym w dowolnej ścieżce realizującej macierz przepustowości granicznych grafu ważonego o zbiorze wierzchołków X i wagach krawędzi przybierających wartości $g(\{x\}, \{y\})$, $x, y \in X$, $x \neq y$.

Wnioski wynikające z twierdzenia 8 są analogiczne do wniosków przytoczonych po twierdzeniu 7. Uwaga ta odnosi się również do idei algorytmu wyszukiwania zespołów Max - minimalnych, która jest taka sama jak w przypadku zespołów Add - minimalnych [62, 63].

W pracy [64] przedstawiono modyfikację metody wyznaczania zespołów Max - minimalnych, wykorzystując w tym celu specyficzne właściwości grafów ważonych krawędziowo, w których

operację dodawania zastąpiono operacją maximum. W rezultacie otrzymano algorytm o złożoności $O(n^2)$ (algorytm prezentowany w [63] ma złożoność obliczeniową rzędu $O(n^4)$). Stanowi to postęp w algorytmizowaniu procesu wyznaczania zespołów Max - minimalnych, jednakże sama idea algorytmu w zasadzie pozostaje bez zmian.

3. MMS - SPECYFIKACJA KOMUNIKATÓW W PROCESIE WYTWARZANIA

3.1. Sieci transmisji danych w przedsiębiorstwie przemysłowym

Zakres zastosowań czterech różnych rodzajów sieci w strukturze systemów automatyki na terenie zakładu przemysłowego przedstawia się następująco:

- sieć kręgosłupowa - łączy główny komputer zakładu z komputerami w poszczególnych działach zakładu. Temu rodzajowi sieci poświęcono standard MAP [32, 35, 36, 38, 39, 40, 42]. Obligatoryjną częścią standardu MAP jest MMS, opisany dalej w tym rozdziale.
- sieć procesu - tworzy połączenie między głównym komputerem zakładu i komputerami czasu rzeczywistego rozmieszczonymi na terenie zakładu. Dla tego rodzaju sieci opracowany został standard MiniMAP. Zgodne z nim są magistrale: niemiecka PDV-Bus i japońska FAIS. Jako sieci procesu coraz częściej są obecnie stosowane standardowe lokalne sieci komputerowe LAN (Local Area Network) wykorzystujące CSMA/CD jako metodę dostępu do medium transmisyjnego. Wadą ostatnio przytoczonego rozwiązania jest zatykanie się takich sieci przy dużych obciążeniach. Na tym poziomie można również zastosować sieci standardu PROFIBUS. W przyszłości należy jednak przewidywać wykorzystanie nowych standardowych sieci typu LAN, dostosowanych do aplikacji multimedialnych z możliwością pracy w systemach uwarunkowanych czasowo.
- podstawowa grupa magistral miejscowych - stanowią ją systemy komunikacyjne łączące z jednej strony komputer pracujący w czasie rzeczywistym, z drugiej zaś strony inteligentne urządzenia automatyki takie jak programowalne sterowniki przemysłowe PLC, specjalizowane sterowniki maszyn i napędów, koncentratory danych oraz inteligentne czujniki i zadajniki. Ta grupa magistral miejscowych ma trzy główne klasy zastosowań:
  - magistrale miejscowe ogólnego przeznaczenia (np. PROFIBUS, BITBUS, FIP, Interbus-S),
  - magistrale miejscowe iskrobezpieczne do zastosowań w strefach zagrożenia wybuchem (np. PROFIBUS PA).
  - magistrale miejscowe spełniające ostre wymagania pracy w systemach krytycznych czasowo, np. do sterowania napędami synchronicznymi lub w zastosowaniach z ustalonym czasem zwłoki (np. SERCOS, TTP) ,
- czwartą grupę sieci (i zarazem drugą grupę magistral miejscowych) - tworzą magistrale zadajników i czujników (np. ASI, OPUS). Magistrale te są zwykle sterowane przez sterownik PLC lub przez specjalizowany moduł sterujący, przyłączony do magistrali miejscowej ogólnego przeznaczenia. Możliwe jest połączenie nawet pojedynczych sygnałów binarnych. Magistrale te cechuje niski koszt instalacji.

Ze względu na specyficzne wymagania stawiane niektórym systemom automatyki w klasie magistral ogólnego przeznaczenia można wyróżnić jeszcze szczególne podklasy:

- magistrale miejscowe systemów automatyki w pojazdach,
- magistrale miejscowe systemów automatyki w budynkach.

W zakładach przemysłowych usytuowanych na rozległym terenie lub rozrzuconych na znacznym obszarze wykorzystuje się także sieci metropolitalne MAN (Metropolitan Area Network) i/lub sieci rozlgłe WAN (Wide Area Network).

W zastosowaniach przemysłowych sieci lokalne spełniają inną rolę niż magistrale miejscowe. Stoją one wyżej w hierarchii zakładu w tym sensie, że zazwyczaj nie współpracują bezpośrednio z zadajnikami/czujnikami, ale obsługują większe jednostki wykonawcze, np.

zautomatyzowane centra produkcyjne, bądź też wiążą stacje nadrzędne poszczególnych magistral miejscowych z ośrodkami zarządzania. Tego rodzaju typowym wykorzystaniem sieci lokalnych jest sieć kręgosłupowa i sieć procesu. Sieci lokalne służą również jako element wiążący wyższe szczeble systemu CIM, a mianowicie poziom 6 (przedsiębiorstwa), poziom 5 (zakłady/służby), poziom 4 (wydziały/sekcje), dochodząc do poziomu 3 (gniazda i linie produkcyjne - według modelu odniesienia do celów normalizacyjnych i metodologii identyfikacji wymagań [40, 41, 42].

3.2. Siedmiowarstwowy model ISO/OSI

Potrzebę standaryzacji w zakresie komunikacji systemów komputerowych dostrzeżono stosunkowo szybko. Międzynarodowa Organizacja Standardów (ISO) już w roku 1978 ogłosiła dokument opisujący ogólny model komunikacji systemów niezależnych producentów, znany jako Model Odniesienia dla Współdziałania Systemów Otwartych [21]. Jest to próba ustalenia standardu współdziałania sieci różnych producentów, oparta na nowej, siedmiowarstwowej architekturze. Uogólnia ona sieci znane z firmowych rozwiązania i wprowadza inne, całkowicie nowe. Wieloletni, skomplikowany proces standaryzacji prowadzony przez ISO trwa do dzisiaj, ale już w połowie lat osiemdziesiątych osiągnięto stabilny stan standardu, pozwalający na implementację opartych na nim rozwiązań.

Zaletą modelu OSI jest jego uniwersalność i abstrakcyjność: działanie systemów opisuje się w kategoriach funkcji realizowanych przez poszczególne warstwy, abstrakcyjnych interakcji między warstwami (prymitywów usługowych) i formatów jednostek danych przekazywanych w protokołach różnych warstw.

Podkreśla się często, że takie podejście pozwala stosować określone w modelu zasady komunikacji bez wprowadzania ograniczeń co do sposobu implementacji. Ta zasada, stosowana bezwzględnie w początkowej fazie standaryzacji modelu OSI, ma pewne wyjątki - pojawiają się dokumenty, odnoszące się do interfejsów programowych na poziomie aplikacji API (Application Programming Interface), które w znacznym stopniu narzucają programistom sposób implementacji (przez zdefiniowanie nazw i funkcji podprogramów, używanych struktur danych itp.). Wydaje się, że jest to raczej zaleta niż wada: standaryzacja interfejsów programowych aplikacji ułatwia pracę programistom i umożliwia przenośność oprogramowania aplikacyjnego.

Model OSI często jest używany jako punkt odniesienia przy porównywaniu odmiennych architektur sieciowych. Zdarza się nawet, że niektórzy autorzy próbują uzasadnić tezę o zgodności opisywanych produktów z modelem OSI, posługując się pobieżnym porównaniem warstwowej struktury i funkcji, bez wnikania w stosowane protokoły, co jest ewidentnym nadużyciem i dowodem niekompetencji.

Konsekwencją ogólności modelu jest pewna nadmiarowość, odczuwana często jako jego wada. Mianowicie ścisła implementacja zasad modelu może prowadzić niekiedy do zbędnych narzutów, dublowania niektórych funkcji i innych efektów, wpływających na obniżenie efektywności komunikacji, niepotrzebną komplikację oprogramowania itp. Równocześnie model OSI dopuszcza wiele opcji, co może prowadzić do niemożności współpracy rozwiązań zgodnych z modelem. Odpowiedzią ISO na wspomniane problemy są profile, określające dozwolone kombinacje protokołów poszczególnych warstw i podzbiór realizowanych funkcji. W niektórych przypadkach krytyka modelu OSI doprowadziła do opracowania specjalnych

protokołów, łączących funkcje wielu warstw, niezgodnych z modelem OSI, ale gwarantujących znacznie większą efektywność komunikacji.

Podstawową funkcją dowolnego urządzenia pracującego w sieci jest zapewnienie ścieżki dostępu, dzięki której końcowy użytkownik znajdujący się w dowolnym miejscu może skomunikować się z innym końcowym użytkownikiem, znajdującym się w oddalonym geograficznie punkcie. Para takich końcowych użytkowników to np. użytkownik terminala komputera i wywoływany przez niego zdalny program aplikacyjny. Kolejny przykład to dwa programy aplikacyjne współpracujące ze sobą itd. Przez ścieżkę dostępu należy rozumieć zestaw funkcji, których działanie umożliwia nie tylko fizyczne połączenie obu użytkowników, lecz również ich skomunikowanie się. I to niezależnie od występujących różnego rodzaju błędów podczas przesyłania danych, różnic występujących w formatach danych, różnic w stosowanych platformach sprzętowych i programowych itp.

Systematyczne rozważenie repertuaru funkcji, które muszą być wykonane w celu realizacji ścieżki dostępu jest genezą warstwowego modelu architektury sieci komputerowych (rys. 3.1). Interpretacja rysunku jest następująca:

1. Należy zapewnić zespół sprawnych urządzeń (linii) prowadzących od jednego użytkownika do drugiego (być może przez inne pośrednie urządzenia). Jeżeli użytkownicy przesyłają dane za pośrednictwem kanałów telefonicznych, to na końcu ww urządzeń muszą znajdować się modemy zapewniające konwersję strumienia bitowej informacji cyfrowej na analogowe sygnały elektryczne i odwrotnie.
2. Należy zagwarantować, że otrzymany strumień bitów jest wolną od błędów kopią danych nadanych. Ta funkcja jest jedną z funkcji sterowania łączem danych, do których należą również funkcje grupowania sekwencji bitów w ramki oraz dokonywania retransmisji błędnych ramek.
3. W celu ekonomicznego wykorzystania urządzeń, przez które transmituje się bity, stosuje się różne techniki ich multipleksowania, co oznacza, że wielu użytkowników może korzystać z nich równocześnie. Pociąga to za sobą konieczność stosowania adresów użytkowników. Jeśli stosuje się adresowanie, to naturalnym działaniem jest wybór drogi przesyłania, czyli marszrutyzacja transmitowanych informacji.
4. Należy ustalić sposób prowadzenia dialogu między końcowymi użytkownikami, a odpowiednie funkcje muszą gwarantować realizację przyjętego modelu i jego zmianę. Często spotykanym współdziałaniem jest sesja między użytkownikami, w której przepływ danych w obu kierunkach jest częścią związanych ze sobą serii transakcji.
5. Należy zapewnić uniezależnienie realizowanej komunikacji pomiędzy końcowymi użytkownikami od stosowanych przez nich formatów danych, kodów tych danych itp., muszą więc istnieć funkcje "uniwersalizujące" reprezentację danych.

Istnienie tych wszystkich elementów tworzy kompletną ścieżkę dostępu i umożliwia komunikowanie się użytkowników końcowych.

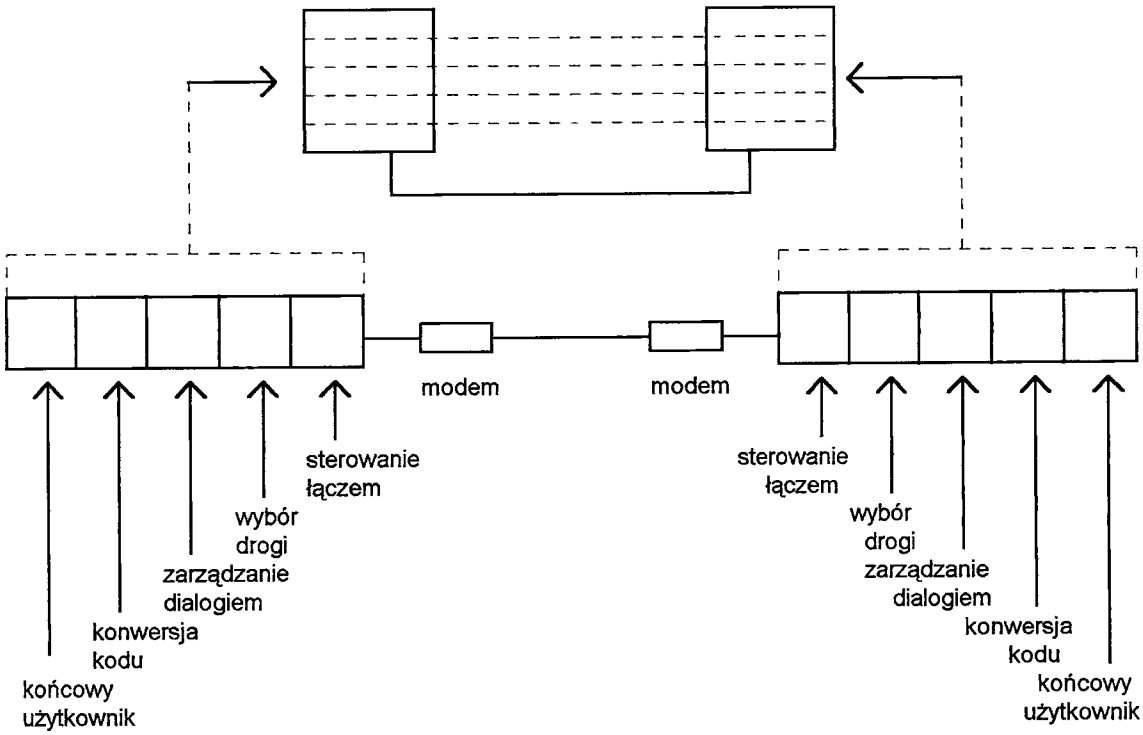
Grupowanie funkcji realizujących wspólny cel to właśnie tworzenie warstw. W konsekwencji prowadzi to do powstania architektury warstwowej sieci komputerowej przedstawionej w postaci pionowych diagramów (górna część rys. 3.1).

Podstawową przesłanką tworzenia warstw było zapewnienie niezależności każdej warstwy w sensie zdefiniowania usług, które ta warstwa oferuje warstwie wyższej, przy czym warstwa wyższa nic nie wie o sposobie ich realizacji. Pozwala to zmienić implementację dowolnej z

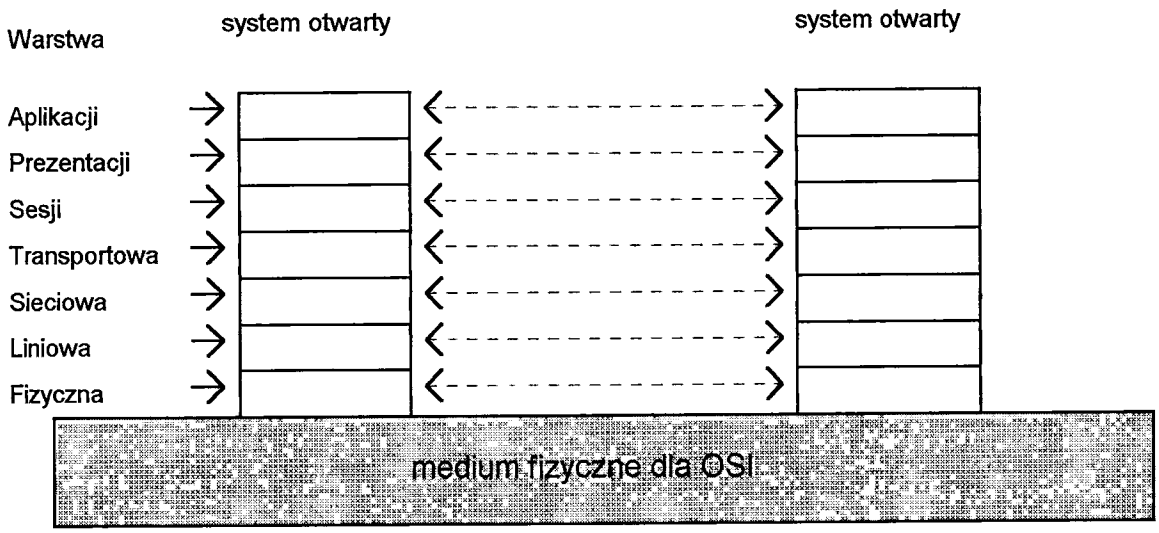
41

warstw, nie zmieniając innych warstw (usługi, czyli sposób porozumiewania się między warstwami, pozostają takie same). Uzyskana w ten sposób dekompozycja dużego zagadnienia inżynierskiego na szereg mniejszych jest dodatkową zaletą tego postępowania.

Wszystkie protokoły, które wypełniają model ISO/OSI bazują na na koncepcji protokołów warstwowych. W tym modelu każdy system składa się z warstw. Użytkownicy komunikują się z usługodawcą poprzez punkt dostępu do usług - SAP (Service Access Point). SAP może przybierać różne formy i podstawowym wymaganiem jest, by zapewniał jednoznaczną identyfikację obiektów z niego korzystających.



Rys. 3.1. Geneza warstwowego modelu architektury sieci komputerowych

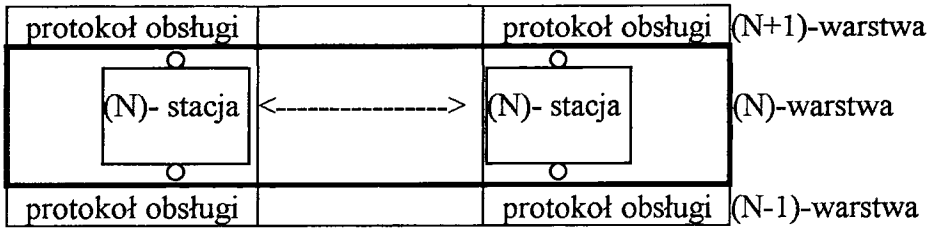


Rys. 3.2. Siedmiowarstwowa architektura modelu odniesienia

Każda warstwa może udostępniać wiele usług. W modelu ISO/OSI przyjęto, że :

- każda warstwa jest podsystemem składającym się z wielu specjalizowanych modułów nazywanych stacjami (entity),
- niektóre warstwy - ze względu na złożoność - mogą być dalej podzielone na wewnętrzne podwarstwy.

Model ISO/OSI systematyzuje i ujednolica pojęcia używane do:
- dostarczania usług między stacjami rezydującymi w przyległych warstwach,
- współdziałania tych stacji poprzez interfejs modelowany przez (N)-SAP,
- współdziałania stacji partnerskich z zastosowaniem protokołu.



Rys. 3.3. Współdziałanie warstw przyległych. Kółeczkami oznaczono SAP - punkty dostępu do usług.

Stacje w warstwach przyległych muszą współdziałać przez uzgodniony interfejs. Stacja (N+1)-warstwy w trakcie elementarnej interakcji ze stacją w (N)-warstwie przez (N)-SAP, tj. łączący je interfejs przesyła jednostkę danych (N)-interfejsu IDU (Interface Data Unit). IDU składa się z jednostki danych (N)-usługi SDU (Service Data Unit) i dodatkowej informacji sterującej protokołu interfejsowego PCI (Protocol Control Information). Ważną cechą modelu jest to, że (N)-stacje przenoszą SDU między sobą (i w rezultacie między nadawczym i odbiorczym (N)-SAP-em) z pomocą jednostek danych (N)-protokołu PDU (Protocol Data Unit). Ze względu na rozmiar SDU (N)-warstwa w stacji nadawczej może dokonywać np. fragmentaryzacji SDU i przesyłać ją do stacji partnerskiej w postaci sekwencji PDU. Scalenie SDU i przesłanie do (N+1)-warstwy następuje w stacji odbiorczej.

Stacje rezydujące w tej samej warstwie i współkomunikujące się są nazywane stacjami partnerskimi (peer entities). Nie jest oczywiście wymagane, by wszystkie stacje należące do wybranej warstwy komunikowały się między sobą. Zwykle jest tak, że komunikują się stacje należące do tej samej podwarstwy, bo wchodzą one we wzajemne związki w celu realizacji zleconych usług.

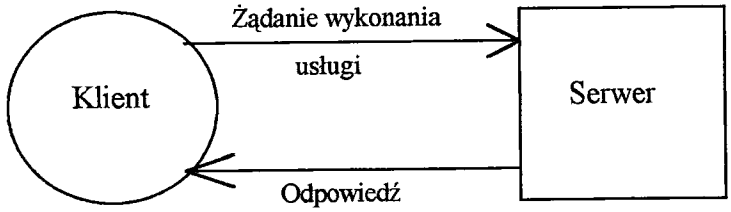
Podstawową ideą leżącą u podstaw modelu warstwowego jest to, że każda warstwa - na bazie usług warstw niższych - wprowadza "nową wartość dodaną" do usług oferowanych przez siebie warstwie wyższej i udostępnia nowe usługi warstwie wyższej przez właściwy SAP. W wyniku tego "dodawania wartości" warstwa najwyższa, która udostępnia usługi aplikacji, dysponuje interfejsem do pełnego zakresu usług oferowanego łącznie przez wszystkie warstwy. Warstwie wyższej jest zwykle przesłaniany sposób realizacji wywoływanej usługi. Jedynym wymaganiem jest to, by usługa została dostarczona i wykonana zgodnie z przekazanymi wartościami parametrów ilościowych i jakościowych.

W większości systemów warstwy, podwarstwy, stacje mają realizację programową. Oznacza to, że oprogramowanie sieciowe jest strukturalizowane wg modelu ISO/OSI i wybrane moduły

programowe wywołują, a inne moduły programowe dostarczają żądane usługi. Konwencja komunikacji między modułami programów jest oczywiście uzależniona od konkretnej implementacji i zastosowanych narzędzi do wytworzenia oprogramowania. Zauważalny z upływem czasu trend to sprzętowa realizacja wybranych stacji, podwarstw czy wreszcie warstw.

3.3. Komunikacja typu klient - serwer i prymitywy ją realizujące

W modelu ISO/OSI powszechne zastosowanie znalazła architektura klient (inicjujący transakcję sieciową i zlecający usługę do wykonania) - serwer (wykonujący zleconą mu usługę) (rys. 3.4). Termin ten jest związany z relacją w obrębie jednej aplikacji między dwoma procesami. W relacji tej klient jest procesem, który zleca wykonanie zadania procesowi serwera, który jest odpowiedzialny za jego wykonanie. Określona relacja obowiązuje we współpracy obu tak zdefiniowanych procesów, ale nie usztywnia roli tych procesów. Znaczy to, że proces pełniący funkcję serwera w stosunku do jednego procesu może w stosunku do innego procesu być klientem, a rola procesu zależy tylko od kontekstu jego pracy.

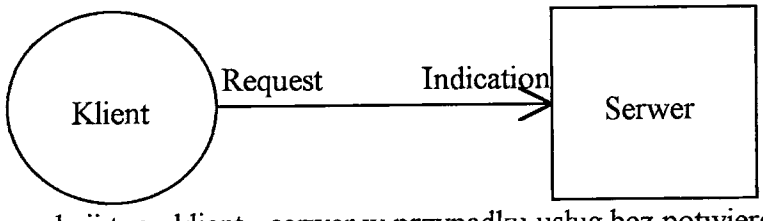


Rys. 3.4 Komunikacja w systemie klient-serwer

Schemat działania klient - serwer realizowany jest przy użyciu prymitywów request, (żądanie), indication (wskazanie), response (odpowiedź) i confirmation (potwierdzenie). Wyróżnienie tych czterech prymitywów jest wygodne z punktu widzenia zarówno analizy jak i syntezy (np. konstrukcji oprogramowania realizującego zadany zbiór usług konkretnego protokołu). Mianowicie w przypadku prymitywów typu request i indication mamy do czynienia z takimi samymi danymi, jednakże w pierwszym przypadku są to dane wyjściowe, w drugim zaś - wejściowe. Podobnie rzecz się ma dla pary prymitywów response (dane wyjściowe) i confirmation (dane wejściowe). Klient na podstawie otrzymanych danych typu confirmation (lub ich braku) podejmuje decyzję o dalszym przebiegu transakcji sieciowej. Serwer zaś, po otrzymaniu danych typu indication, na podstawie swego własnego stanu generuje odpowiedź kierowaną do klienta w postaci danych typu response.

3.4. Usługi bez potwierdzenia (bezpołączeniowe) i z potwierdzeniem (połączeniowe)

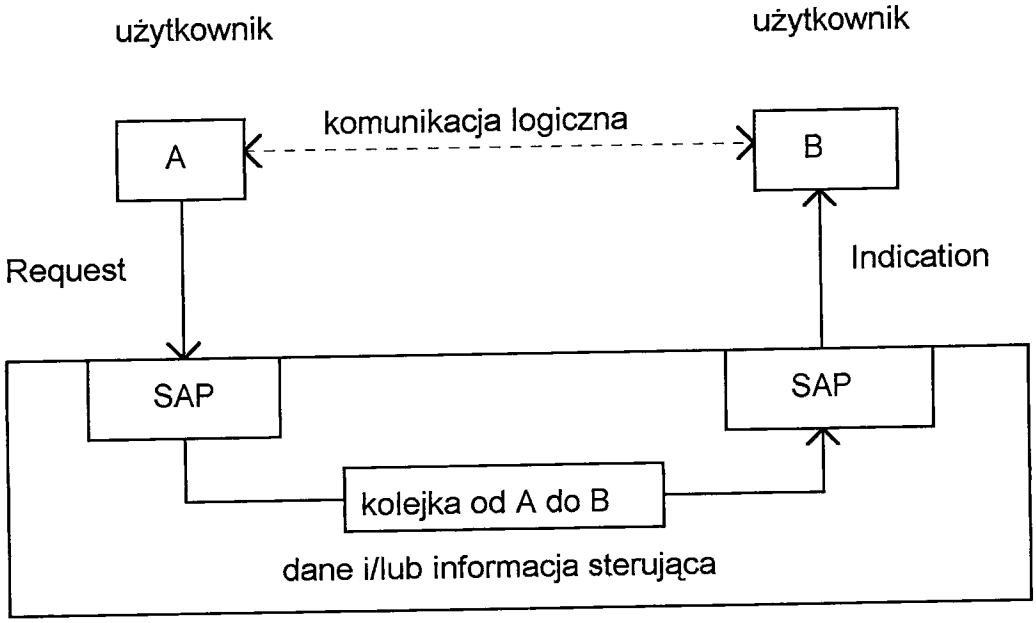
Wyróżniamy 2 typy usług: z potwierdzeniem i bez potwierdzenia. W przypadku usług bez potwierdzenia schemat przepływu informacji (transakcji) przebiega w sposób przedstawiony na rys. 3.5.



Rys. 3.5. Schemat transakcji typu klient - serwer w przypadku usług bez potwierdzenia.

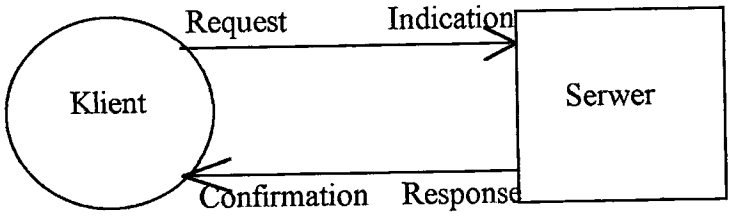
44

Bardziej szczegółowo koncepcję usług bez potwierdzenia zobrazowano na rys. 3.6.

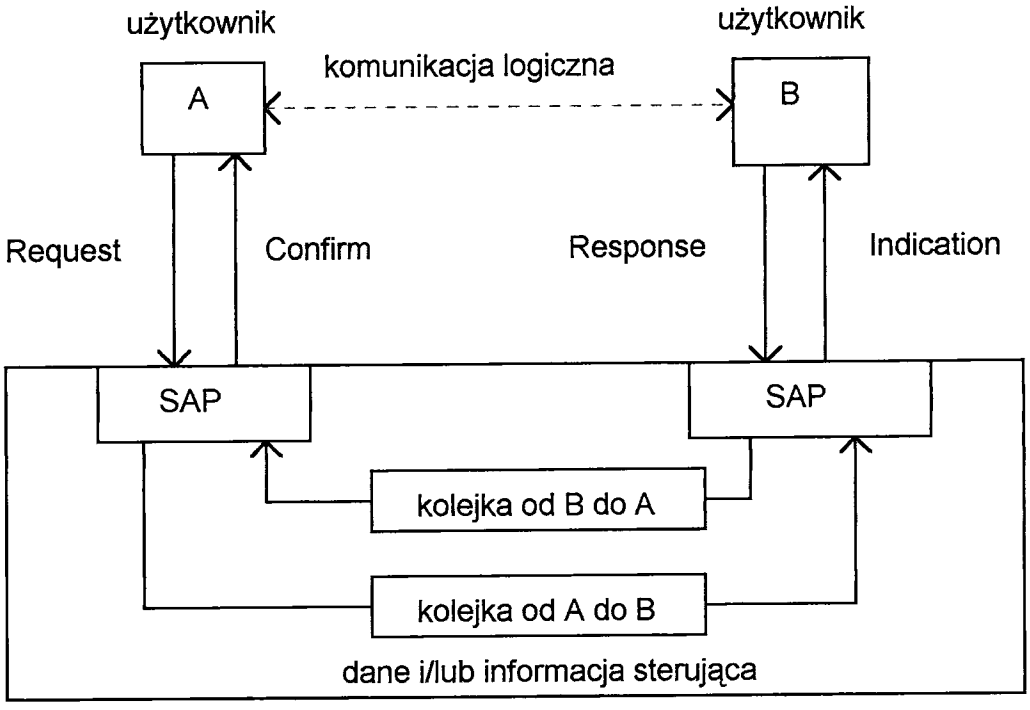


Rys. 3.6. Model warstwy wg ISO/OSI - dostarczanie usług bezpołączeniowych

Sposób realizacji usług z potwierdzeniem pokazano, odpowiednio na rys. 3.7 i 3.8.



Rys. 3.7. Schemat transakcji typu klient - serwer w przypadku usług z potwierdzeniem.



Rys. 3.8. Model warstwy wg ISO/OSI - dostarczanie usług połączeniowych

3.5. Opis warstw modelu ISO/OSI

Poszczególne warstwy modelu ISO/OSI, od najniższej czyli fizycznej do najwyższej czyli aplikacyjnej są opisane w kolejnych, dalszych podrozdziałach.

3.5.1. Warstwa fizyczna

Przeznaczeniem warstwy fizycznej jest dostarczenie środków mechanicznych, elektrycznych, funkcjonalnych i proceduralnych, potrzebnych do uaktywniania, utrzymania oraz deaktywacji połączeń fizycznych kanału transmisji danych zdolnego do przesyłania niestrukturalizowanego strumienia bitów między stacjami liniowymi. Stacje rezydujące w tej warstwie dostarczają warstwie wyższej interfejsu do medium fizycznego. Istnieje wiele standaryzowanych interfejsów opisujących fizyczne, elektryczne/optyczne, mechaniczne charakterystyki dostępu do fizycznego medium. Stacje warstwy fizycznej współdziałają za pomocą medium fizycznego.

W opisie warstwy fizycznej wprowadza się dodatkowo określenie **kanału komunikacyjnego** (data-circuit) - czyli ścieżki komunikacyjnej między dwoma stacjami warstwy fizycznej w medium fizycznym OSI, łącznie z niezbędnymi dla warstwy fizycznej udogodnieniami do transmisji bitów.

Charakterystyki mechaniczne, elektromagnetyczne i inne połączeń przez medium fizyczne są określane na granicy między warstwą fizyczną a medium fizycznym. Definicje takich charakterystyk są specyfikowane w innych normach.

3.5.2. Warstwa liniowa

Stacje warstwy liniowej (łącza danych) tworzą logiczny kanał transmisji danych i dostarczają taki kanał stacjom warstwy sieciowej. Stacje warstwy liniowej realizują funkcje zestawienia,

utrzymania i deaktywacji kanału logicznego. Utrzymanie kanału logicznego to zdolność stacji do uporządkowanego przesyłania danych - na zlecenie stacji sieciowych - wraz ze sterowaniem przepływem, zapobieganiem i wychodzeniem ze stanów błędu (przekłamania, duplikaty, straty danych). Połączenia liniowe są tworzone jako jedno lub kilka połączeń fizycznych.

Warstwa liniowa wykrywa i, w miarę swoich możliwości, koryguje błędy, które mogą powstać w warstwie fizycznej. Dodatkowo warstwa liniowa umożliwia warstwie sieciowej sterowanie współdziałaniem kanałów komunikacyjnych w warstwie fizycznej.

3.5.3. Warstwa sieciowa

Podstawowym zadaniem warstwy sieciowej jest zestawianie, utrzymanie i deaktywacja połączeń w podsieci komunikacyjnej. Warstwa ta oddziela warstwy wyższe od rozwiązywania problemu znalezienia i utrzymania połączenia sieciowego budowanego przez systemy pośredniczące, nazywane podsieciami komunikacyjnymi.

Przez **podsieć komunikacyjną** lub **podsieć** (subnetwork) rozumie się tu zbiór składający się z jednego lub więcej pośrednich systemów otwartych, które spełniają rolę przekaźników i przez które końcowe systemy otwarte mogą ustanawiać połączenia sieciowe. Podsieć jest w modelu odniesienia reprezentacją sieci rzeczywistej, takiej jak np. publiczna sieć transmisji, sieć prywatna albo lokalna sieć komputerowa.

Innymi słowy warstwa sieciowa dostarcza środków do ustanawiania, utrzymania i rozłączania połączeń sieciowych między systemami otwartymi, zawierającymi komunikujące się stacje aplikacyjne. Dostarcza również środków funkcjonalnych i proceduralnych do wymiany, przez połączenia sieciowe, jednostek danych usługi sieciowej między stacjami transportowymi.

Warstwa sieciowa zapewnia stacjom transportowym niezależność od rozważań dotyczących marszrutyzacji i przekazywania, a związanych z ustanowieniem i działaniem danego połączenia sieciowego, włączając w to sytuację, w której wykorzystywanych jest jednocześnie kilka podsieci szeregowo lub równolegle. Warstwa ta przesłania stacjom transportowym problem wykorzystania niżej położonych zasobów (takich jak np. połączenia liniowe) do realizacji połączeń sieciowych.

Wszystkie funkcje przekaźnikowe i protokoły, rozszerzające skokowo (hop-by-hop) usługi, stosowane do utrzymania połączenia sieciowego między końcowymi systemami otwartymi OSI, działają poniżej warstwy transportowej, tzn. w warstwie sieciowej lub niższej.

Nazwą **połączenie w podsieci** (subnetwork-connection) określa się drogę komunikacyjną przez podsieć, która jest wykorzystywana przez stacje warstwy sieciowej do dostarczenia połączenia sieciowego.

Istnieją co najmniej dwa różne podejścia do budowy połączeń w podsieci komunikacyjnej. Warstwa sieciowa może dostarczać **połączeniowych** lub **bezpołączeniowych** usług sieciowych (connection-oriented mode, connectionless mode). W pierwszym przypadku najpowszechniej stosowanymi protokołami do budowy takich podsieci komunikacyjnych jest para protokołów X.25 i X.75. Połączenia są budowane w oparciu o technologię nazywaną komutacją pakietów. Podejściem alternatywnym jest dostarczanie przez warstwę sieciową tzw.

bezpołączeniowej usługi sieciowej. W tym przypadku połączenia w warstwie sieciowej są budowane z wykorzystaniem klasy protokołów datagramowych.

Funkcje warstwy sieciowej pozwalają na utrzymywanie dużej różnorodności połączeń sieciowych, od połączeń dwupunktowych do połączeń typu złożonych kombinacji podsieci o różnych charakterystykach. W celu zapewnienia tak szerokiego zakresu możliwości, funkcje sieciowe powinny być strukturalizowane w podwarstwach. Podział warstwy sieciowej na podwarstwy może być wykonywany tylko wtedy, gdy jest to użyteczne.

Podstawową usługą warstwy sieciowej jest zapewnienie przezroczystego przesyłania danych między stacjami transportowymi. Usługa ta pozwala, by struktura i szczegółowa treść dostarczanych danych, były określane wyłącznie przez warstwy powyżej warstwy sieciowej.

Warstwa sieciowa zawiera funkcje niezbędne do wyposażenia warstwy transportowej w trwałą granicę warstwa sieciowa/warstwa transportowa. Granica ta jest niezależna od niżej położonego medium transmisyjnego we wszystkich jego aspektach, z wyjątkiem jakości usługi. Warstwa sieciowa zawiera więc funkcje niezbędne do zamaskowania różnic występujących w charakterystykach różnych rodzajów transmisji i technologii podsieci, umożliwiając w ten sposób utworzenie spójnej usługi sieciowej.

Usługa dostarczana na każdym końcu połączenia sieciowego jest taka sama, nawet jeśli połączenie sieciowe rozciąga się na kilka podsieci, z których każda oferuje różne usługi.

W czasie ustanawiania połączenia sieciowego, między stacjami transportowymi a stacjami sieciowymi jest negocjowana jakość usługi. Ponieważ ta jakość usługi może zmieniać się dla różnych połączeń sieciowych, więc powinna zostać uzgodniona dla danego połączenia sieciowego i być taka sama w obu jego punktach końcowych.

3.5.4. Warstwa transportowa

Podstawowym zadaniem warstwy transportowej jest aktywacja, utrzymanie i deaktywacja połączenia transportowego między dwoma komunikującymi się systemami otwartymi. Połączenie jest przezroczyste (transparentne), "od końca do końca" (przy czym przez "końce" rozumie się korespondujące ze sobą stacje transportowe). Dlatego o warstwie transportowej mówi się, że jest zorientowana na końcowe systemy otwarte OSI, a protokoły transportowe działają tylko między końcowymi systemami otwartymi OSI. Warstwa transportowa spełnia wymagania jakościowe i efektywnościowe warstwy sesji i, pośrednio, warstwy prezentacji oraz aplikacji. Warstwa transportowa odciąża stacje sesyjne od zajmowania się problemami niezawodnego i efektywnego przesyłania danych.

Usługi dostarczane przez warstwę transportową wypełniają lukę, jaka może istnieć między jakością i efektywnością usług warstwy sieciowej a wymaganą jakością i efektywnością całości usług transportowych żądanych przez warstwy wyższe. Warstwa transportowa optymalizuje wykorzystanie dostępnej usługi sieciowej w celu osiągnięcia, przy minimum kosztów, charakterystyk wydajnościowych wymaganych przez każdą ze stacji sesyjnych. Optymalizację tę osiąga się w warunkach ograniczeń określonych z jednej strony wymaganiami wszystkich współbieżnie działających stacji sesyjnych, a z drugiej jakością i wydajnością usługi sieciowej udostępnianej warstwie transportowej.

Zastosowane w warstwie protokoły mogą być proste lub dość złożone. Pierwsze są implementowane wówczas, gdy podległa warstwa sieciowa dostarcza niezawodnych i wysokiej jakości usług. Jeżeli jakość i niezawodność usług sieciowych jest niska, to aby zagwarantować wymagany poziom jakości i niezawodności usług transportowych, są stosowane bardziej złożone protokoły transportowe.

Warstwa transportowa jest odciążona od wszystkiego, co jest związane z marszrutyzacją i przekazywaniem informacji, ponieważ warstwa sieciowa dostarcza połączeń sieciowych od dowolnej stacji transportowej do dowolnej innej stacji transportowej, nawet w przypadku istnienia między nimi szeregu podsieci.

Funkcje transportowe wywoływane w warstwie transportowej w celu dostarczenia usługi o wymaganej jakości, są zależne od jakości usługi sieciowej. Jakość usługi sieciowej zależy z kolei od sposobu, w jaki realizuje się tę usługę.

Dodatkową przyczyną stosowania różnych protokołów w warstwie transportowej jest typ usługi transportowej. Podobnie jak w warstwie sieciowej, może to być usługa połączeniowa lub bezpołączeniowa.

W trybie połączeniowym (connection-oriented transmission) (patrz też rys. 3.8) połączenie jest związkiem między dwoma lub kilkoma obiektami, ustanowionym okresowo w celu zapewnienia przesłania informacji. Podstawową cechą połączenia jest wyraźnie określony czas trwania. Wyróżnia się fazę jawnego nawiązywania połączenia, fazę przesyłania informacji i fazę rozwiązania połączenia. Nawiązanie połączenia wymaga zgody wszystkich uczestników (usługodawcy i wszystkich usługobiorców) oraz ustalenia przez nich parametrów transmisji.

Nawiązanie połączenia wymaga realizacji dialogu, w ramach którego strona inicjująca wysyła żądanie ze sprecyzowaniem propozycji parametrów transmisji (np. maksymalnej długości wysyłanych ramek). Jeśli usługodawca nie jest zdolny do zapewnienia przepływu informacji na rzecz połączenia, to może wprost wygenerować negatywne potwierdzenie. W przeciwnym razie zawiadomienie zawierające zgłoszone w żądaniu propozycje parametrów dociera do partnera, który po jego przeanalizowaniu generuje odpowiedź, być może zawierającą modyfikację proponowanych parametrów transmisji albo odrzucenie propozycji nawiązania połączenia. Potwierdzenie zawierające zgłoszone w odpowiedzi propozycje dociera do partnera - inicjatora - i połączenie zostaje nawiązane (w przypadku zgody partnera) lub próba nawiązania jest uznana za nieudaną (w przypadku odrzucenia).

Aby realizowany podczas nawiązywania połączenia proces negocjacji parametrów połączenia był jednoznaczny, partner generujący odpowiedź może wyłącznie łagodzić wymagania na transmisję proponowane przez inicjatora, tzn. jeśli inicjator był zdolny do dotrzymania parametrów zaproponowanych w żądaniu (które partner uznał za zbyt wygórowane), to tym bardziej może dotrzymać parametrów proponowanych w odpowiedzi. Z nawiązaniem połączenia wiąże się rezerwacja - zarówno przez partnerów, jak i przez usługodawcę - zasobów niezbędnych do późniejszej realizacji transmisji.

W punktach udostępniania usług (SAP), przez które jest nawiązywane połączenie, tworzy się identyfikatory połączeń. Ich użycie w przesyłanych między partnerami ramkach pozwala skrócić część adresową identyfikującą adresata. Po nawiązaniu połączenia jest możliwe dwukierunkowe przesyłanie wiadomości.

Alternatywą w stosunku do przedstawionej poprzednio procedury przesyłania w trybie połączeniowym jest tryb bezpołączeniowy (connectioless transmission; datagram service), którego istotą jest przekazywanie między źródłowym a docelowym (lub docelowymi) punktami udostępniania usług (N-SAP) pojedynczych, niezależnych jednostek danych, zwanych datagramami (datagrams).

Tryb bezpołączeniowy transmisji danych między obiektami nie wymaga ustanawiania między nimi a usługodawcą żadnych związków i odbywa się przy użyciu dwóch tylko operacji elementarnych usług, zgodnie ze schematem pokazanym wcześniej na rys. 3.6. Tryb bezpołączeniowy może być realizowany do jednego lub wielu partnerów-odbiorców. Tryb ten wymaga, bez względu na liczbę adresatów, wyłącznie dwustronnego uzgodnienia między nadawcą a usługodawcą.

Nadanie datagramu jest związane z pojedynczą interakcją (żądaniem). W ramach tej jednej interakcji muszą być więc każdorazowo, przy każdym żądaniu transmisji datagramu, przekazywane wszystkie parametry niezbędne do określenia adresata (adresatów), opcji związanych z realizacją usługi (np. priorytet wiadomości, sposób obrony przed błędami transmisji). Przesyłane jednostki danych protokołu są całkowicie niezależne i muszą zawierać wszelkie informacje niezbędne do prawidłowej transmisji i doręczenia, w tym pełną identyfikację adresatów. Skutkiem ubocznym tej pełnej niezależności jest możliwość odebrania przez odbiorcę jednostek danych w innej kolejności niż ta, w jakiej byłyby one zlecone do wysłania.

Funkcje realizowane w trybie bezpołączeniowym ograniczają się do właściwego sformatowania ramki oraz obrony przed błędami transmisji, ale bez możliwości retransmisji, a jedynie z odrzucaniem (gubieniem) ramek uznanych przez odbiorcę za błędne. Tryb bezpołączeniowy nie gwarantuje dostarczenia informacji partnerowi, a w przypadku jej dostarczenia nie zapewnia kolejności odbioru zgodnej z kolejnością nadawania.

W dokumencie standaryzacyjnym ISO 8073 opisano połączeniowy protokoł transportowy. W protokole wyróżniono 5 klas implementacji. Charakteryzowane są one następująco:

klasa 0 - klasa prosta,
klasa 1 - klasa z odtwarzaniem po błędach,
klasa 2 - klasa z multipleksowaniem,
klasa 3 - klasa z odtwarzaniem po błędach i multipleksowaniem,
klasa 4 - klasa z detekcją błędów, odtwarzaniem po błędach, multipleksowaniem itd.

Wybór klasy protokołu i obowiązujących opcji jest negocjowany w chwili zestawiania połączeń transportowych. Podstawą są wymagania użytkownika, jakość dostępnych usług warstwy sieciowej i koszt tych usług. Aby podejmować właściwe decyzje co do wyboru klasy protokołu transportowego, jakość usług warstwy sieciowej musi by "obserwowana" i klasyfikowana. Wymieniony standard zakłada trzystopniową klasyfikację połączeń sieciowych (tj. klasy A, B i C) i zaleca wybór klasy protokołu transportowego na bazie tych klasyfikacji (tab. 3.1).

Tablica 3.1. Połączenia sieciowe i klasy transportowe

Typ połączeń sieciowych	A	B	C
Resztkowa stopa błędów	akceptowalna	akceptowalna	nieakceptowalna
Sygnalizowana częstość załamań połączeń	akceptowalna	nieakceptowalna	nieakceptowalna
Zalecana klasa protokołu transportowego	0 lub 2	1 lub 3	4

3.5.5. Warstwa sesji

Podstawowe zadanie warstwy sesji polega na dostarczeniu środków procesom użytkowym (pośrednio przez stacje warstwy prezentacji) do zestawienia, utrzymania i deaktywacji połączenia nazywanego sesją. Warstwa sesji dostarcza współpracującym stacjom prezentacyjnym środków niezbędnych do organizacji i synchronizacji ich dialogu oraz uzyskania wymiany danych między nimi. W tym celu warstwa sesji dostarcza usług do ustanowienia połączenia sesyjnego między dwoma stacjami prezentacyjnymi oraz usług do utrzymywania uporządkowanej interakcji wymiany danych.

Przez odwzorowanie połączenia sesyjnego w połączenie transportowe użytkownicy sesji odwołują się do siebie przez symboliczne nazwy.

Połączenie sesyjne jest tworzone na żądanie stacji prezentacyjnej, zgłoszone w punkcie dostępu do usługi sesyjnej. Podczas trwania połączenia sesyjnego usługi sesyjne są wykorzystywane przez stacje prezentacyjne do sterowania dialogiem tych stacji oraz do zapewnienia uporządkowanej wymiany komunikatów w połączeniu sesyjnym. Połączenie sesyjne istnieje do czasu zwolnienia go przez stację prezentacyjną albo przez stację sesyjną. Podczas trwania połączenia sesyjnego, usługi sesyjne utrzymują stan dialogu nawet wtedy, gdy warstwa transportowa zgubi dane.

Stacja prezentacyjna może mieć dostęp do innej stacji prezentacyjnej tylko przez zainicjowanie lub akceptację połączenia sesyjnego. Stacja prezentacyjna może być skojarzona z wieloma połączeniami sesyjnymi równocześnie.

Stacja prezentacyjna określa docelową stację prezentacyjną przez podanie adresu sesyjnego. W szczególnym przypadku adres transportowy może być stosowany jako adres sesyjny, tzn. istnieje wzajemnie jednoznaczne odwzorowanie między adresem sesyjnym i adresem transportowym. Zasada ta jest wykorzystywana w wielu rzeczywistych systemach. Ogólnie jednak rzecz biorąc, odwzorowanie adresów sesyjnych w adresy transportowe nie jest wzajemnie jednoznaczne. Nie implikuje to konieczności multipleksacji połączeń sesyjnych na połączenia transportowe, ale powoduje, że w czasie ustanawiania połączenia sesyjnego więcej niż jedna stacja prezentacyjna jest potencjalnym celem żądania ustanowienia połączenia sesyjnego, przybywającego w danym połączeniu transportowym.

Nowa jakość warstwy sesji - w stosunku do warstwy transportowej - polega na tym, że funkcje realizowane w tej warstwie pozwalają na:
- wybór modelu dialogu i pomocy w zarządzaniu tym dialogiem,
- strukturalizację całej sesji na mniejsze jednostki wraz z możliwością synchronizacji i resynchronizacji wymiany danych (oznacza to możliwość "cofania" dialogu sesyjnego do ściśle określonych punktów).

Stacje sesyjne mogą przesyłać między sobą 4 rodzaje danych, tj. dane zwykłe, przyspieszone i 2 typy danych związanych z zarządzaniem sesją.

Procesy użytkowe - za pośrednictwem stacji sesyjnych - mogą prowadzić dialog:
a) dwukierunkowy jednoczesny,
b) dwukierunkowy naprzemienny,
c) jednokierunkowy.

Zarządanie dialogiem jest realizowane przez wprowadzenie reguł przekazywania uprawnień do nadawania danych i zamykania połączeń sesyjnych oraz grupowania danych w jednostki atomowe. Ta ostatnia cecha ma znaczenie w tzw. transakcyjnym modelu aplikacji gwarantującym jej integralność. Jeśli transakcja odnosi się do grupy wiadomości/komunikatów, to warstwa sesji udostępnia mechanizmy do przesłania całości danych, zanim transakcja rozpocznie się.

Analogicznie do warstwy transportowej warstwa sesji może być implementowana w różnym zakresie. W przypadku pełnej implementacji standardu strukturalizacja całej sesji jest trzypoziomowa i udostępniany jest najszerszy zakres usług do (re)synchronizacji i zarządzania jej przebiegiem włącznie. Na najwyższym poziomie jest identyfikowana sesja. W trakcie sesji wyróżniane są aktywności (przykładem pojedynczej aktywności może być transmisja pojedynczego pliku). Do zarządzania aktywnościami jest przewidziana cała grupa prymitywów usługowych (Activity Management), z pomocą których użytkownik może:
- kreować i identyfikować aktywności,
- startować, przerywać, zawieszać, wznawiać, zrywać, kończyć aktywności.

Każda ze stacji sesyjnych, na żądanie użytkowników, wtrąca do przesyłanych danych punkty synchronizacyjne, które stacja partnerska sukcesywnie potwierdza. Wyróżniane są małe i duże punkty synchronizacji. Pozwalają one stacjom sesyjnym wrócić do znanego stanu przesyłania danych (sesji) w przypadku błędów i załamań połączeń. Dialog może być "cofany" do wskazanego punktu synchronizacyjnego, ale nie głębiej niż do ostatnio potwierdzonego dużego punktu synchronizacyjnego i nie poza granicę aktywności.

3.5.6. Warstwa prezentacji

W opisie warstwy prezentacji wykorzystuje się dodatkowo następujące pojęcia:
-składnia konkretna (concrete syntax) - te aspekty reguł stosowanych w formalnej specyfikacji danych, które obejmują konkretną reprezentację tych danych.
-składnia transferu (transfer syntax) - składnia konkretna stosowana podczas przesyłania danych między systemami otwartymi.

Warstwa prezentacji zapewnia możliwość reprezentowania informacji, którą posługują się, albo do której odwołują się stacje aplikacyjne podczas komunikacji. Warstwa prezentacji uwzględnia dwa dopełniające się aspekty reprezentacji informacji:

i) reprezentację danych, które mają być przesłane między stacjami aplikacyjnymi,
ii) reprezentację struktury danych (do których odwołują się stacje aplikacyjne w trakcie komunikacji) razem z reprezentacją zbioru działań, które mogą być wykonywane na tej strukturze. Innymi słowy stacje przenoszą reprezentację typów danych i wartości konkretnych zmiennych.

Dzięki uwzględnieniu zarówno aspektu i) jak i ii) stacje aplikacyjne są uwolnione od zajmowania się problemem reprezentacji informacji.

Warstwa prezentacji jest związana tylko ze składnią, to jest z reprezentacją danych, a nie z ich semantyką, czyli ich znaczeniem dla warstwy aplikacji. Znaczenie danych jest znane tylko stacjom aplikacyjnym.

Warstwa prezentacji dostarcza wspólnej reprezentacji danych, która powinna być stosowana między stacjami aplikacyjnymi. Uwalnia to stacje aplikacyjne od zajmowania się problemem reprezentacji informacji. Ta niezależność składni może być opisana dwoma sposobami:

i) warstwa prezentacji dostarcza wspólnych elementów składniowych, które są stosowane przez stacje aplikacyjne,
ii) stacje aplikacyjne mogą stosować dowolną składnię. Warstwa prezentacji dokonuje przekształceń między tymi składniami a składnią wspólną wymaganą do komunikacji między stacjami aplikacyjnymi. Przekształcenie takie jest realizowane wewnątrz systemów otwartych, więc nie jest widoczne dla innych systemów otwartych i dlatego nie ma to wpływu na normalizację protokołów prezentacyjnych.

Stacje aplikacyjne mogą stosować dowolną składnię informacji. W modelu warstwy prezentacji ISO/OSI przyjęto, że warstwa prezentacji dokonuje transformacji ze składni lokalnej (w danym systemie) na składnię wspólną, wymaganą do komunikacji między stacjami aplikacyjnymi.

Dla potrzeb warstwy prezentacji rozwinięto w ramach ISO dwa standardy, tj. ISO 8824 i ISO 8825. Pierwszy standard tzw. ASN.1 (Abstract Syntax Notation One) opisuje abstrakcyjną składnię typów i wartości przesyłanych danych. Standard drugi, tzw. BER (Basic Encoding Rules) zawiera zestaw reguł kodowania składni typów i wartości konkretnych danych.

Istnieje wzajemnie jednoznaczna odpowiedniość adresu prezentacji i adresu sesji. W warstwie prezentacji nie stosuje się multipleksacji ani rozszczepiania.

3.5.7. Warstwa aplikacji

W opisie warstwy aplikacji wprowadza się następujące dodatkowe określenia:

- **stacja aplikacyjna** (application-entity) - aspekty procesu aplikacyjnego, odnoszące się do systemów otwartych.
- **element usługi aplikacyjnej** (application-service-element) - część stacji aplikacyjnej, która dostarcza udogodnień środowiska OSI, odpowiednio wykorzystując w tym celu niżej usytuowane usługi.

- **element użytkownika** (user-element) - reprezentacja tej części procesu aplikacyjnego, która wykorzystuje elementy usługi aplikacyjnej potrzebne do realizacji celów komunikacyjnych tego procesu aplikacyjnego.

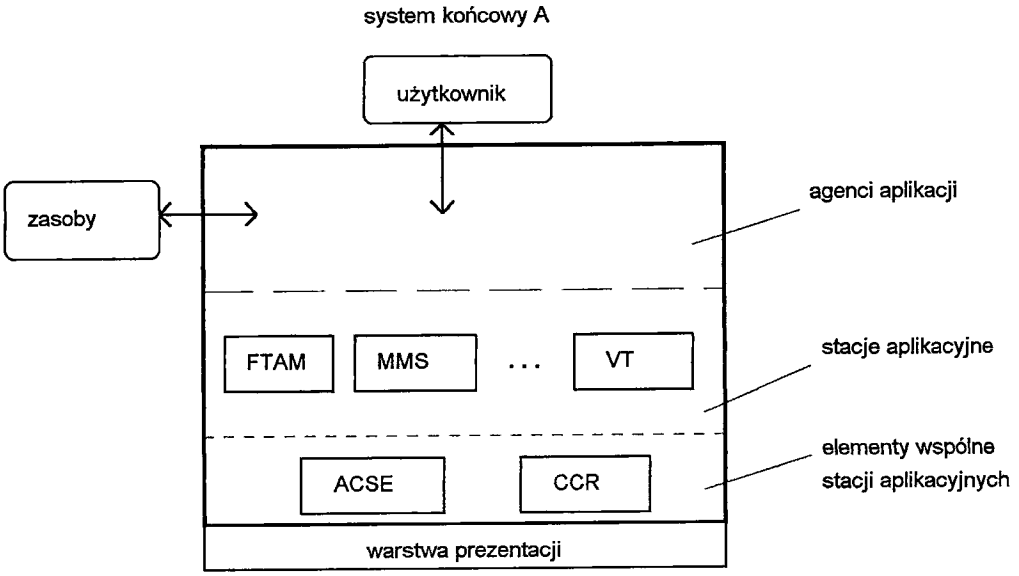
Warstwa aplikacji różni się fundamentalnie od innych warstw modelu ISO/OSI. Jest to warstwa ostatnia i jej użytkownikami nie są stacje w kolejnej wyższej warstwie. Przeznaczeniem warstwy aplikacji jest dostarczanie procesom aplikacyjnym metod dostępu do środowiska OSI. Warstwa aplikacji nie ma interfejsu do warstwy wyższej. Warstwa aplikacji jest jedynym sposobem dostępu procesu aplikacyjnego do środowiska OSI. Warstwa aplikacji pełni rolę okna między korespondującymi procesami aplikacyjnymi, wykorzystującymi OSI do wymiany informacji. Każdy proces aplikacyjny jest reprezentowany w stosunku do partnerskiego procesu aplikacyjnego przez stację aplikacyjną. Wszystkie specyfikowalne parametry procesu aplikacyjnego związane z komunikacją w środowisku OSI są znane temu środowisku dzięki warstwie aplikacji (i w ten sposób są dostępne mechanizmom implementowanym w OSI).

W warstwie aplikacji rezydują procesy aplikacyjne. Procesy aplikacyjne wymieniają informacje za pomocą stacji aplikacyjnych, protokołów aplikacyjnych i usług prezentacyjnych. Warstwa aplikacji jako jedyna warstwa w modelu odniesienia, która bezpośrednio dostarcza usługi procesom aplikacyjnym, z konieczności dostarcza wszystkie usługi OSI bezpośrednio użyteczne procesom aplikacyjnym.

W modelu zakłada się, że proces aplikacyjny składa się z dwóch części, tj. agenta aplikacji (application agent) i stacji aplikacyjnej. Agent aplikacji jest interfejsem do użytkownika i do systemu operacyjnego komputera końcowego. Zapewnia dostęp do zasobów tego systemu i jego specyficznych właściwości. Ta część procesu aplikacyjnego znajduje się poza standardem ISO/OSI.

Stacja aplikacyjna zawiera jeden element użytkownika i zbiór elementów usługi aplikacyjnej. Element użytkownika reprezentuje tę część procesu aplikacyjnego, która stosuje elementy usługi aplikacyjnej do realizacji celów komunikacyjnych procesu aplikacyjnego. Elementy usługi aplikacyjnej spełniają swoje funkcje wywołując się wzajemnie i/lub wywołując usługi prezentacyjne.

Stacje aplikacyjne działają niezależnie od specyficznych cech danego systemu końcowego i, z pośrednictwem agentów aplikacji, udostępniają zestandaryzowane usługi aplikacji, takie jak FTAM (File Transfer, Access and Management), JTM (Job Transfer and Manipulation), VT (Virtual Terminal), MMS czy poczta elektroniczna. Stąd wynika model całej warstwy aplikacji - rys. 3.9. Jest to wersja uproszczona, bo uwzględniono w niej tylko główne elementy stacji aplikacyjnych. Większość stacji aplikacyjnych w trakcie swojego działania zestawia i utrzymuje połączenia - nazywane asocjacjami - ze stacjami partnerskimi ulokowanymi w odległych systemach. Schemat działania jest tu wspólny i zostało to uwidocznione w modelu przez wyodrębnienie tzw. elementu ACSE (Association Control Service Element). Drugą wyodrębnioną częścią wspólną jest element CCR (Commitment, Concurrency and Recovery), z pomocą którego stacje aplikacyjne koordynują swoje współbieżne działanie i zapewniają niezawodne i atomowe działania w środowisku zawodnym i podatnym na awarie. Działania atomowe oznaczają akcje składające się z wielu (współ)zależnych przesłań informacji i wykonywanych działań z gwarancją, że wszystko zostaje wykonane poprawnie lub nastąpi wycofanie do dobrze określonego, poprzedniego stanu.



Rys. 3.9. Uproszczony model warstwy aplikacji

Elementy użytkownika w różnych systemach mogą się komunikować ze sobą tylko przez wymianę jednostek danych protokołu aplikacyjnego. Jednostki te są generowane przez elementy usługi aplikacyjnej.

Warstwa aplikacji zawiera wszystkie funkcje, które są wymagane do komunikacji między systemami otwartymi, a nie są realizowane przez warstwy niższe. Zawiera ona funkcje realizowane przez programy jak również realizowane przez człowieka.

Jeśli konkretny egzemplarz procesu aplikacyjnego chce skomunikować się z egzemplarzem procesu aplikacyjnego w innym systemie otwartym, to musi on wywołać egzemplarz stacji aplikacyjnej w warstwie aplikacji swojego systemu. Egzemplarz tej stacji aplikacyjnej staje się odpowiedzialny za ustanowienie asocjacji z egzemplarzem odpowiedniej stacji aplikacyjnej w docelowym systemie otwartym. Proces ten jest realizowany przez wywoływanie egzemplarzy stacji w niższych warstwach. Gdy asocjacja między dwoma stacjami aplikacyjnymi zostanie ustanowiona, wtedy proces aplikacyjny może się komunikować.

Stacja aplikacyjna może mieć wewnętrzną strukturę funkcjonalną. Zastosowanie pewnego sposobu grupowania funkcji może zależeć od stosowania innych funkcji, a funkcje aktywne mogą się zmieniać podczas trwania połączenia. Strukturalizacja stacji aplikacyjnej w elementy usługi aplikacyjnej i element użytkownika zapewnia organizację funkcji w stacjach aplikacyjnych. Co więcej, dowolny dany podzbiór elementów usługi aplikacyjnej łącznie z elementem użytkownika stanowi pewien typ stacji aplikacyjnej. Każdy typ stacji aplikacyjnej, a więc i każdy jej egzemplarz są jednoznacznie identyfikowalne.

Proces aplikacyjny może określać grupowanie funkcji zawartych w stacji aplikacyjnej.

Rozróżniane są dwie kategorie elementów usługi aplikacyjnej: wspólne elementy usługi aplikacyjnej i specyficzne elementy usługi aplikacyjnej. Wspólne elementy dostarczają udogodnień, które są ogólnie użyteczne dla różnych aplikacji, specyficzne - dostarczają udogodnień wymaganych przez określone potrzeby poszczególnych aplikacji (np. przesyłanie

plików, dostęp do bazy danych, przesyłanie zadań, bankowość, wejście zamówień - order entry). Stacje aplikacyjne mogą zawierać elementy usługi aplikacyjnej obu kategorii.

Podział elementów usługi aplikacyjnej na te dwie kategorie nie implikuje istnienia dwóch niezależnych protokołów.

Funkcje zarządzania systemami i zarządzania aplikacją są umieszczone w warstwie aplikacji. Oprócz zarządzania systemami i aplikacją dostępne są również inne działania, szczególnie te, które są związane z zarządzaniem warstwą aplikacji (takie jak uaktywnianie i sterowanie w sytuacjach błędnych).

3.6. Koncepcja protokołu MMS

MMS [7, 8, 15, 16, 17, 18, 19] jest jedynym obligatoryjnym elementem warstwy aplikacyjnej sieci MAP (Manufacturing Automation Protocol) [32]. Stanowi on normę komunikacyjną przeznaczoną do obsługi transferu danych między programowalnymi urządzeniami przemysłowymi w środowisku komputerowo zintegrowanego wytwarzania CIM (Copmputer Integrated Manufacturing). Podstawowe normy dotyczące MMS nie określają kompletnego zbioru procedur obsługi dla różnych urządzeń programowalnych, na co w przyszłości zostaną skoncentrowane prace zmierzające do standaryzacji. Istniejące normy definiują MMS w środowiska systemów otwartych (OSI) w zakresie:

- abstrakcyjnego modelu definiującego wspólne czynności dla obsługiwanych użytkowników,
- zewnętrznej funkcjonalności narzędzi dostosowanych do norm w formie proceduralnych wymagań związanych z wykonywaną obsługą,
- podstawowych działań wykonywanej obsługi,
- parametrów wejściowych wymaganych dla prawidłowej obsługi,
- wzajemnych relacji między prawidłowo wykonywaną sekwencją obsługi.

Normy te nie określają sposobu wykonania implementacji ani nie narzucają żadnych ograniczeń co do wewnętrznej realizacji danego systemu komputerowego.

Specyfika komunikacji między procesami przemysłowymi polega na tym, że wszystkie urządzenia (np. sterownik robota) w środowisku przemysłowym muszą mieć możliwość jednakowej interpretacji instrukcji (wspólna semantyka), co przy uwzględnieniu dużej różnorodności stosowanego sprzętu powoduje, że protokół musi realizować ogromną ilość usług dla wszystkich możliwych aplikacji. Zasada takiej komunikacji między dwoma urządzeniami polega na przesyłaniu komunikatów zawierających zbiór parametrów, łącznie ze szczegółowo określonymi ich typami i wartościami. MMS dostarcza standardowego języka komend, który umożliwia wysyłanie i jednoznaczną interpretację komunikatów przez urządzenia. Składnia komunikatów wykorzystywanych przez MMS prezentowana jest w notacji składni abstrakcyjnej ASN.1 (por. p. 3.5.6).

MMS zezwala na różne typy sterowania. W szczególności może to być zarówno sposób polegający ma cyklicznym żądaniu od urządzeń (polling) przesłania wartości określonych parametrów (np. stanu urządzenia) lub też praca w trybie sterowania zdarzeniami, polegająca na samoczynnym przekazaniu informacji przez urządzenie w wyniku wystąpienia określonego zdarzenia. MMS również zezwala na wspólne wykorzystywanie zasobów w środowisku przemysłowym, do czego służą mechanizmy semaforowe stosowane w przypadku, w którym

urządzenie otrzymuje równocześnie wiele zgłoszeń, ale może obsłużyć w danej chwili tylko jedno z nich.

MMS dostarcza zbioru ogólnych semantyk przeznaczonych do komunikacji między urządzeniami przemysłowymi. Niektóre z nich wymagają dodatkowej semantyki określonej dla danej aplikacji, co jest wyspecyfikowane w kolejnych dokumentach normy (tzw. normy stowarzyszone - Companion Standards) - w przypadku robota takim dokumentem jest [20]. Przewiduje się rozszerzanie tych dokumentów, tak by powstały odpowiednie zbiory semantyk i składni dla określonych klas zastosowań.

Specyfikacje MMS definiują operacje i semantykę w sposób na tyle ogólny, że mogą one mieć zastosowanie w niemal wszystkich gałęziach przemysłu. Uwzględnienie wszystkich potrzeb potencjalnych użytkowników prowadzi do specyfikacji różnorodnych funkcji i opcji, co ma wpływ na wysoki poziom skomplikowania protokołu

MMS wprowadza następujące definicje:

- **Obiekty (objects).** MMS definiuje zbiór wspólnych obiektów takich jak zmienne, programy, zdarzenia itd i definiuje ich atrybuty możliwe do zidentyfikowania na poziomie sieci (network visible attributes). Atrybutami tymi mogą być nazwy, wartości, typy itd.
- **Usługi (services).** MMS definiuje zbiór usług komunikacyjnych takich jak zapis, odczyt, kasowanie itd. celem zapewnienia dostępu i zarządzania obiektami w środowisku sieciowym (in a networked environmnet).
- **Zachowanie (behavior).** MMS definiuje możliwe do zaobserwowania z poziomu sieci "zachowywanie się" urządzeń przemysłowych podczas ich obsługi.

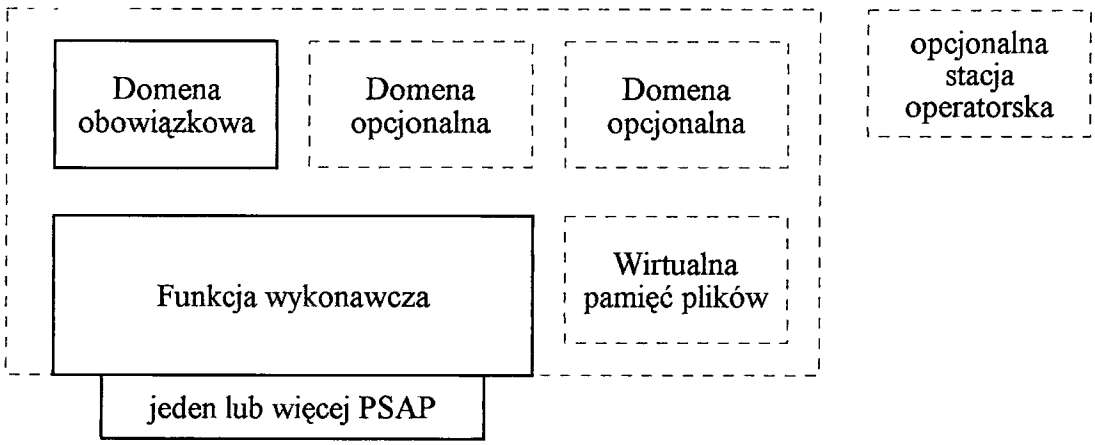
3.7. Wirtualne urządzenie wytwórcze VMD

Wykorzystanie norm przeznaczonych dla określania wzajemnych połączeń urządzeń nadzorujących i sterujących procesem przemysłowym bazuje na modelach abstrakcyjnych. W MMS do określenia zachowania każdego z wzajemnie połączonych urządzeń przemysłowych dla każdego urządzenia sporządza się jego abstrakcyjny model określany mianem wirtualnego urządzenia wytwórczego (lub wirtualnego urządzenia wytwarzania) VMD (Virtual Manufacturing Device). Aby sporządzić taki model niezbędny jest opis urządzenia obejmujący zarówno jego zachowanie jako całości jak i procesów zachodzących wewnątrz. Z punktu widzenia modelu VMD najistotniejsze jest zachowanie urządzenia jako całości. Opis tego co zachodzi wewnątrz w zasadzie powinien być ograniczony do wpływu na jego zewnętrzne zachowanie.

Wspomniane wyżej definicje obiektów, usług i zachowań zawierają także definicję tego, w jaki sposób urządzenia i aplikacje przemysłowe komunikują o fakcie użycia przez MMS wirtualnego urządzenia wytwórczego. VMD określa tylko te aspekty wzajemnej komunikacji, które są możliwe do zaobserwowania z poziomu sieci. Szczegóły takie jak, w jaki sposób wykonać model VMD rzeczywistego urządzenia przemysłowego, tzn. jakiego użyć języka programowania, jakiej jednostki centralnej i w środowisku jakiego systemu operacyjnego, jakich urządzeń wejścia/wyjścia, itd. - nie są określane przez MMS. Poprzez skoncentrowanie uwagi tylko na tych aspektach urządzenia, które można zaobserwować z poziomu sieci, model VMD określa wystarczająco wiele dla uzyskania wysokiej operatywności, nadal pozostawiając spory margines na wprowadzanie dodatkowych innowacji i unowocześnień w samej realizacji urządzenia przemysłowego lub aplikacji przemysłowej.

Wirtualne urządzenie wytwórcze zawiera w sobie informację o zasobach sprzętowych i programowych, właściwych konkretnemu, rzeczywistemu urządzeniu wykonawczemu, ich wzajemnych powiązaniach, funkcjonowaniu oraz bieżąco aktualizowane dane dotyczące zdolności operacyjnej każdego z zasobów. VMD stanowi informatyczny interfejs między rzeczywistym urządzeniem wykonawczym a wyższym szczeblem sterowania procesu wytwórczego.

W obrębie jednego VMD (rys. 3.10) wyróżnia się kilka jednostek funkcjonalnych (entity), takich jak wykonawcza, wirtualna przestrzeń plików, opcjonalne domeny i ewentualnie stacje operatorskie.



Rys. 3.10. Uproszczony model wirtualnego urządzenia wytwórczego (VMD) (PSAP - punkt dostępu do usług prezentacji)

Na rys. 3.10 przedstawiono następujące bloki modelu:

- Funkcja wykonawcza - jest używana do sterowania VMD za pomocą MMS i dostarczenia wszystkich wspólnych elementów usługi (to znaczy wszystkich tych elementów, które nie są związane z określoną domeną lub wirtualnym systemem plików).
- Domeny - służą do reprezentacji funkcji urządzenia przemysłowego. Funkcje te mogą być opisane w jednej domenie lub mogą być zdefiniowane jako zbiór domen, co pozwala na większą elastyczność w definiowaniu. Każda domena zawiera informacje przechowywaną w postaci tablic, instrukcji programowych lub struktur danych.
- Wirtualna pamięć plików - zawiera programy i dane. Przyjęta tu koncepcja jest identyczna z zastosowaną w FTAM. Wirtualna pamięć plików MMS jest podzbiorem wirtualnej pamięci plików w aplikacji transferu plików FTAM.
- Stacja operatorska - w modelu VMD dostarcza podstawowych mechanizmów wejścia i wyjścia dla interakcji z człowiekiem. Nie jest w swoim założeniu pełną konsolą operatorską, ale czymś w rodzaju portu informacyjnego, jaki może być stosowany w kontakcie z danym urządzeniem. Do korzystania z pełnej konsoli należy użyć usług i protokołu wirtualnego terminala.

Kluczowym aspektem modelu VMD jest relacja pomiędzy "klientem a "urządzeniem obsługującym" czyli relacja pomiędzy aplikacjami (usługami) sieciowymi a/lub urządzeniami przemysłowymi (por. p. 3.3). Urządzenie obsługujące czyli tzw. serwer jest urządzeniem przemysłowym lub aplikacją przemysłową, zawierającą obiekty i własny model VMD. Klient natomiast jest aplikacją sieciową lub urządzeniem żądającym od serwera danych lub obsługi. W

rozumieniu ogólnym mianem klienta można określić jednostkę sieci która żąda obsługi MMS ze strony serwera, a mianem serwera - jednostkę sieci, która świadczy tę usługę. MMS definiuje usługi dla obu tych jednostek tzn. klienta i serwera, model VMD definiuje tylko możliwe do zaobserwowania z poziomu sieci zachowania serwerów.

Wiele aplikacji MMS i kompatybilnych z MMS urządzeń przemysłowych (np. robot [20]) spełnia funkcje zarówno klienta jak i serwera. Model VMD definiuje tylko te funkcje aplikacji, które dotyczą serwera. Każda aplikacja MMS, zawierająca funkcje serwera, musi więc zawierać także wirtualny model urządzenia wytwórczego dotyczący wszystkich możliwych do zaobserwowania z poziomu sieci aspektów aplikacji i urządzeń przemysłowych. Od klientów MMS wymaga się tylko dostosowania do zasad wzajemnej wymiany komunikatów czyli do przestrzegania określonych sekwencji i protokołów.

3.7.1. Rzeczywiste i wirtualne urządzenia i obiekty

Należy odróżnić obiekty rzeczywiste (np. sterowniki PLC, roboty przemysłowe, itd.), obiekty rzeczywiste będące częściami ww (zmienne, procedury, itd.), urządzenia wirtualne i obiekty zdefiniowane w modelu VMD. Rzeczywiste urządzenia przemysłowe i obiekty mają swoje własne cechy szczególne (np. właściwości danego produktu) związane z nimi w sposób jednoznaczny. Urządzenia i obiekty wirtualne są dostosowane do modelu VMD i są całkowicie niezależne od sposobu wyprodukowania, użytego języka programowania, systemu operacyjnego, środowiska sprzętowego itd. Każdy opis serwera MMS lub aplikacji serwera MMS odpowiada ukrytym szczegółom zawartych w nich prawdziwych urządzeń i obiektów ze względu na wykonywaną przez nich funkcję. Wykonywana funkcja pozwala przekształcić urządzenie lub obiekt rzeczywisty w jego wirtualny odpowiednik zdefiniowany jako model VMD komunikujący się z klientem MMS. Ponieważ klient MMS oraz wirtualny obiekt lub urządzenie zawsze oddziaływują wzajemnie na siebie poprzez model VMD, to aplikacja klienta jest izolowana od specyficznych cech obiektu bądź urządzenia rzeczywistego.

Właściwie zaprojektowana aplikacja klienta MMS może komunikować się w identyczny sposób z różnymi typami urządzeń, ponieważ szczegóły dotyczące rzeczywistych obiektów i urządzeń są ukryte dla tej aplikacji za funkcjami wykonawczymi zdefiniowanymi w modelu VMD. To wirtualny dostęp do opisu zachowania się serwera nie przeszkadza we wprowadzaniu innowacji związanych z unowocześnianiem i ulepszaniem danego urządzenia. MMS model VMD wprowadza tylko pewne ograniczenia w aspektach dotyczących wirtualnych obiektów i urządzeń, możliwych do zaobserwowania na poziomie sieci, a nie do ich rzeczywistych odpowiedników.

3.7.2. Modelowanie obiektów i urządzeń na poziomie VMD

Przy implementowaniu funkcji wykonawczych urządzenia trzeba zdecydować, w jaki sposób modelować obiekt rzeczywisty jako obiekt wirtualny. Sposób w jaki zostanie to zrealizowane ma bowiem największy wpływ na uzyskanie wysokiej operatywności działania pomiędzy klientami a serwerami modelowanymi przez różnych projektantów. Nieodpowiedni lub niewłaściwy model może prowadzić do implementacji, która jest trudna w zastosowaniu.

Na przykład załóżmy, że dla sterownika PLC napisano program w postaci drabinkowej (obiekt rzeczywisty). Osoba, która będzie dokonywała jego implementacji MMS (czyli projektant funkcji wykonawczej) życzy sobie przyporządkować aplikacji zewnętrznej (klientowi) kopię

tego programu jako program dla innego komputera. Dla uproszczenia tego przykładu załóżmy, że model VMD pozwala tej osobie na wybór sposobu przedstawienia programu w postaci drabinkowej jako zmiennej lub domeny obiektu (domain object). Sposób w który wirtualny obiekt można odnieść do rzeczywistego, drabinkowego programu jest krytyczny, ponieważ MMS dostarcza szeregu usług do manipulowania zmiennymi, które są całkiem różne od usług używanych do manipulowania domenami. Zmienne mogą być odczytywane pojedynczo lub jako cała lista danych. Domeny natomiast można kopiować tylko w całości. Jeśli drabinkowy program dla sterownika PLC jest przedstawiony jako zmienna MMS, tworzy to zadanie wykonania prostego kopiowania programu, ponieważ rodzaj danych drabinkowego programu sterownika PLC (jest to zwykle duży, ciągły blok w pamięci) ma wpływ na krańcowo dużą zmienną, do której dostęp może być znacznie utrudniony. Jeśli drabinkowy program jest przedstawiony jako domena, to istnieją określone usługi MMS, które pozwalają na ładowanie i zwalnianie dużych, nietypowych bloków pamięci charakterystycznych dla programów drabinkowych. Nieprawidłowy wybór sposobu utworzenia modelu urządzenia rzeczywistego może utrudnić dostęp do rzeczywistego obiektu.

W pewnych przypadkach ma sens przedstawienie tego samego obiektu rzeczywistego w postaci dwóch różnych obiektów MMS. Na przykład duży blok zmiennych można modelować jako domenę. Umożliwia to klientowi MMS wybór usługi dla uzyskania dostępu do danych. Obsługa zmiennych zapewnia dostęp do poszczególnych elementów w bloku. Obsługa domeny pozwala na odczytanie lub zapisanie całego bloku jako element wywołania programu.

3.8. Obiekty MMS i obiekty specyficzne dla VMD

MMS definiuje różne obiekty, które można znaleźć w wielu typowych urządzeniach i aplikacjach wymagających wymiany informacji w czasie rzeczywistym. Lista tych obiektów została udostępniona poniżej w tablicy 3.2. Dla każdego obiektu określono odpowiadającą mu usługę MMS, która pozwala aplikacji klienta na dostęp do niej i wykorzystywanie danego obiektu.

Tablica 2.3. Obiekty MMS

VMD	urządzenie jako takie jest obiektem,
DOMENA (DOMAIN)	reprezentuje środek np. program komputerowy zawarty modelu VMD
WYWOŁANIE PROGRAMU (PROGRAM INVOCATION)	możliwy do uruchomienia program, zawierający co najmniej jedną domenę
ZMIENNA (VARIABLE)	element należący do pewnego typu danych (tj. liczba całkowita, liczba zmiennoprzecinkowa, tablica itd.).
TYP (TYPE)	opis formatu zmiennych
NAZWA LISTY ZMIENNYCH (NAMED VARIABLE LIST)	lista zmiennych określona pojedynczą nazwą
SEMAFOR (SEMAPHORE)	obiekt używany do kontrolowania dostępu do wspólnych środków
STACJA OPERATORSKA (Operator Station)	np. monitor i klawiatura wykorzystywane przez operatora

WARUNEK ZAJŚCIA ZDARZENIA (Event Condition)	obiekt, który określa stan zdarzenia
ZDARZENIE DZIAŁANIA (EVENT ACTION)	określa działanie podejmowane w momencie zmiany stanu warunku zajścia zdarzenia
ZAREJESTROWANIE ZDARZENIA (EVENT ENROLLMENT)	która aplikacja sieciowa zarejestrowała kiedy warunek zajścia zdarzenia zmienił stan
DZIENNIK (JOURNAL)	podstawowy czas rejestracji zdarzeń lub zamiennych
PLIK (FILE)	zbiór w pamięci do zapisywania plików lub w pamięci serwera
TRANSAKCJA (TRANSACTION)	określa indywidualne żądanie usługi MMS. Nie jest to obiekt, który został nazwany indywidualnie (not a named object).

Z każdym obiektem są związane zbiory atrybutów opisujących dany obiekt. Obiekty MMS mają swoją nazwę i inne atrybuty zmieniające się w zależności od obiektu. Podobnie atrybuty mają także zmienne (nazwa, wartość, typ, itd.), wywołania programów (nazwa, aktualny stan) itd. Obiekty podrzędne (subordinate objects) mogą istnieć tylko wewnątrz zakresu innych obiektów. Na przykład wszystkie obiekty są obiektami podrzędnymi dla (lub zawartymi wewnątrz) modelu VMD. Pewne obiekty, które są zawarte wewnątrz innych obiektów takich jak zmienne są także zawarte w domenie. Ten atrybut obiektu jest nazywany zakresem obiektu (scope). Zakres obiektu odzwierciedla także czas istnienia (lifetime) obiektu. Jako zakres obiektu można zdefiniować:

- **specjalny model VMS** (VMD-Specific) - obiekt ma znaczenie i istnieje przez cały czas istnienia modelu VMD (jest obiektem podrzędnym w stosunku do modelu VMD). Taki obiekt istnieje tak długo jak długo istnieje model VMD.
- **specjalna domena** (domain-specific) - obiekt jest zdefiniowany jako obiekt podrzędny pewnej wyszczególnionej domeny. Taki obiekt istnieje tak długo jak istnieje domena.
- **specjalnie skojarzona aplikacja** (application-association-specific) - określana także skrótem AA-Specific. Jest to obiekt całkowicie zdefiniowany przez klienta jako specjalnie skojarzona aplikacja i może być wykorzystywany tylko przez specjalnego klienta. Taki obiekt istnieje tak długo jak długo istnieje skojarzenie pomiędzy klientem a serwerem.

Nazwa obiektu MMS musi także odpowiadać zakresowi obiektu. Na przykład nazwa obiektu dla domeny specjalnej nie powinna określać tylko nazwy zmiennej wewnątrz tej domeny, ale także nazwę domeny. Nazwy dla danego zakresu powinny być nazwami unikalnymi. Na przykład, nazwa specjalnej zmiennej dla danej domeny musi być unikalną nazwą dla wszystkich specjalnych zmiennych w tej domenie. Pewne obiekty, takie jak zmienne mogą być definiowane z każdymi zakresami opisanymi powyżej. Inne, obiekty np. semafory nie mogą być przyporządkowane specjalnie skojarzonej aplikacji (AA-specific). W momencie usunięcia takiego obiektu jak domena, wszystkie obiekty podrzędne dla tej domeny także muszą być usunięte.

Model VMD jest sam w sobie obiektem i posiada swoje atrybuty. Niektórymi z tych atrybutów, możliwych do zaobserwowania z poziomu sieci są:

- **możliwości** (capabilities) - Możliwość modelu VMD jest zasobem (resource) lub pojemnością zdefiniowaną przez urządzenie rzeczywiste. Modelowi VMD można przyporządkować więcej niż jedną pojemność. Pojemności są opisywane przez ciąg liter.

Są one także zdefiniowane przez osobę wykonującą implementację modelu VMD i mogą być źródłem użytecznej informacji na temat urządzenia lub aplikacji rzeczywistej.

- **stan logiczny** (logical status) - Stan logiczny odnosi się do statusu systemu komunikacyjnego MMS dla modelu VMD. Stanami mogą być: STATE-CHANGES-ALLOWED, NO-STATE-CHANGES-ALLOWED lub ONLY-SUPPORT-SERVICES-ALLOWED.
- **stan fizyczny** (physical status) - Stan fizyczny odnosi się do stanu wszystkich możliwości i jest pojmowany jako całość. Stanem fizycznym może być: OPERATIONAL, PARTIALLY-OPERATIONAL, INOPERABLE lub NEEDS-COMMISSIONING.

3.9. Usługi MMS

W wersji podstawowej MMS definiuje 90 usług. Zestawiono je w tablicy 3.3.

Użyteczne kombinacje usług spośród wielu oferowanych przez MMS zostały rozdzielone między siedem klas implementacyjnych. Do wszystkich klas stosują się następujące reguły:

- Klasa implementacyjna jest niezależna od urządzenia i użytkownik decyduje o jej wyborze.
- Klasa implementacyjna definiuje minimum funkcjonalności - znaczy to, że aplikacja może wykorzystywać dowolną kombinację usług, wykraczającą poza określoną klasę.

Klasy implementacyjne, nazwane w kolejności od MAP1 do MAP7, zostały ustalone na podstawie wymagań stawianych poszczególnym urządzeniom. I tak:

- MAP1 - dostosowane jest do wymagań maszyn sterowanych numerycznie (NC),
- MAP2 i MAP3 - wywodzą się z wymagań stawianych sterownikom programowalnym,
- MAP4 - powstało z wymagań stawianych sterownikom robotów,
- MAP5, MAP6 i MAP7 - dostosowane są do wymagań systemów sterowania procesami.

W tablicy 3.3 użyto następujących oznaczeń:

- x - usługa musi być stosowana w danej klasie,
- y - usługa musi być stosowana w danej klasie zarówno w sytuacji klient jak i w sytuacji serwer,
- A i B - w danej klasie muszą być uwzględnione albo wszystkie usługi oznaczone symbolem A, albo też wszystkie usługi oznaczone symbolem B,
- brak oznaczenia - usługa nie musi być stosowana w danej klasie.

Tablica 3.3. Alfabetyczny wykaz usług oferowanych przez MMS i klasy usług

Lp	Nazwa usługi	Nazwa usługi wg PN	MAP 1	MAP 2	MAP 3	MAP 4	MAP 5	MAP 6	MAP 7
01	Abort	Przerwanie	x	x	x	x	x	x	x
02	AcknowledgeEventNotification	Potwierdzenie_Zgłoszenia_Zdarzenia							
03	AlterEventConditionMonitoring	Zmiana_Monitorowania_Warunku_Zdarzenia							
04	AlterEventEnrollment	Zmienianie_Wpisu_Zdarzenia							
05	AttachToEventCondition	Przywiązanie_Do_Warunku_Zdarzenia							
06	AttachToSemaphore	Przywiązanie_Do_Semafora							
07	Cancel	Kasowanie		x	x	x	x	x	x
08	Conclude	Zakończenie	y	y	y	y	x	y	y
09	CreateJournal	Utworzenie_Dziennika						x	x
10	CreateProgramInvocation	Utworzenie_Wywołania_Programu	x		x	x			
11	DefineEventAction	Definiowanie_Akcji_Zdarzenia							
12	DefineEventCondition	Definiowanie_Warunku_Zdarzenia							
13	DefineEventEnrollment	Definiowanie_Wpisu_Zdarzenia							
14	DefineNamedType	Definiowanie_Typu_Nazwanego							
15	DefineNamedVariable	Definiowanie_Zmiennej_Nazwanej							
16	DefineNamedVariableList	Definiowanie_Listy_Zmiennych_Nazwanych							
17	DefineScatteredAccess	Definiowanie_Dostępu_Rozproszonego							
18	DefineSemaphore	Definiowanie_Semafora							
19	DeleteDomain	Usuwanie_Domeny			x	x			
20	DeleteEventAction	Usuwanie_Akcji_Zdarzenia							
21	DeleteEventCondition	Usuwanie_Warunku_Zdarzenia							
22	DeleteEventEnrollment	Usuwanie_Wpisu_Zdarzenia							
23	DeleteJournal	Usuwanie_Dziennika						x	x

Tablica 3.3.(cd) Alfabetyczny wykaz usług oferowanych przez MMS i klasy usług

Lp	Nazwa usługi	Nazwa usługi wg PN	MAP1	MAP2	MAP3	MAP4	MAP5	MAP6	MAP7
24	DeleteNamedType	Usuwanie_Typu_Nazwanego							
25	DeleteNamedVariableList	Usuwanie_Listy_Zmiennych_Nazwanych							
26	DeleteProgramInvocation	Usuwanie_Wywołania_Programu	x		x	x			
27	DeleteSemaphore	Usunięcie_Semafora							
28	DeleteVariableAccess	Usuwanie_Dostępu_Do_Zmiennej							
29	DownloadSegment	Wprowadzanie_Segmentu	A		x	A	A	A	A
30	EventNotification	Zgłoszenie_Zdarzenia							
31	ExchangeData								
32	FileClose	Zamknięcie_Pliku							
33	FileDelete	Usuwanie_Pliku							
34	FileDirectory	Katalogowanie_Pliku							
35	FileOpen	Otwarcie_Pliku							
36	FileRead	Czytanie_Pliku							
37	FileRename	Przemianowanie_Pliku							
38	GetAlarmEnrollmentSummary	Pobranie_Zestawienia_Wpisów_Alarmu							
39	GetAlarmSummary	Pobranie_Zestawienia_Alarmu					x	x	x
40	GetCapabilityList	Pobranie_Listy_Uprawnień	x	x	x	x	x	x	x
41	GetDataExchangeAttributes								
42	GetDomainAttributes	Pobranie_Atrybutów_Domeny			x	x	x	x	x
43	GetEventActionAttributes	Pobranie_Atrybutów_Akcji_Zdarzenia							
44	GetEventConditionAttributes	Pobranie_Atrybutów_Warunku_Zdarzenia					x	x	x
45	GetEventEnrollmentAttributes	Pobranie_Atrybutów_Wpisu_Zdarzenia							

Tablica 3.3.(cd) Alfabetyczny wykaz usług oferowanych przez MMS i klasy usług

Lp	Nazwa usługi	Nazwa usługi wg PN	MAP1	MAP2	MAP3	MAP4	MAP5	MAP6	MAP7
46	GetNamedTypeAttributes	Pobranie_Atrybutów_Typu_Nazwanego							
47	GetNamedVariableListAttributes	Pobranie_Atrybutów_Listy_Zmiennych_Nazwanych							
48	GetNameList	Pobranie_Listy_Nazw	x	x	x	x	x	x	x
49	GetProgramInvocationAttributes	Pobranie_Atrybutów_Wywołania_Programu			x	x			x
50	GetScatteredAccessAttributes	Pobranie_Atrybutów_Dostępu_Rozproszonego							
51	GetVariableAccessAttributes	Pobranie_Atrybutów_Dostępu_Do_Zmiennej		x	x	x			
52	Identify	Identyfikacja	x	x	x	x	x	x	x
54	InformationReport	Raportowanie_Informacji	x			x			
53	InitializeJournal	Inicjacja_Dziennika						x	x
55	Initiate	Inicjacja	y	y	y	y	x	y	y
56	InitiateDownloadSequence	Inicjacja_Sekwencji_Wprowadzania	A		x	A	A	A	A
57	InitiateUploadSequence	Inicjacja_Sekwencji_Wyprowadzania	A		x	A	A	A	A
58	Input	Wejście	x			x			
59	Kill	Zniszczenie							x
60	LoadDomainContent	Ładowanie_Zawartości_Domeny	B			B	B	B	B
61	ObtainFile	Uzyskanie_Pliku	B		x	B			
62	Output	Wyjście	x			x			
63	Read	Czytanie	x	x	x	x	x	x	x
64	ReadJournal	Czytanie_Dziennika						x	x
65	Reject	Wybrakowanie	x	x	x	x	x	x	x
66	RelinquishControl	Zaniechanie_Sterowania				x	x	x	x
67	Rename	Przemianowanie							

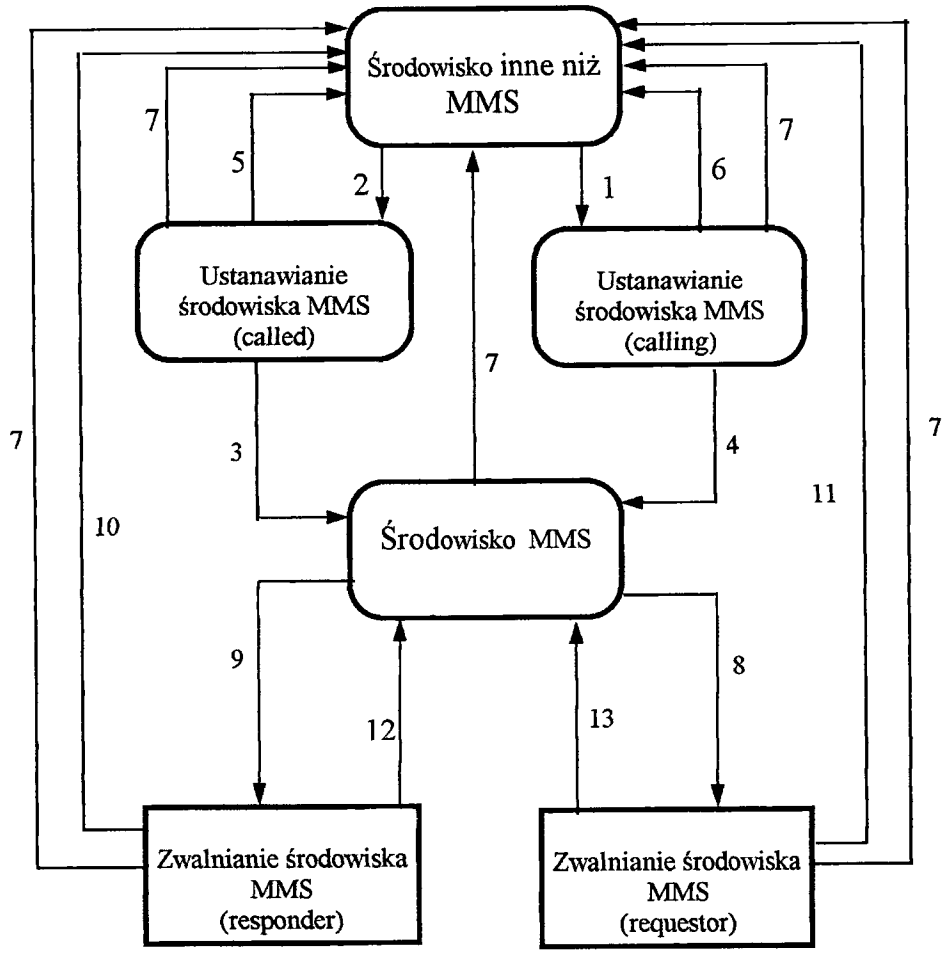
Tablica 3.3.(cd) Alfabetyczny wykaz usług oferowanych przez MMS i klasy usług

Lp	Nazwa usługi	Nazwa usługi wg PN	MAP1	MAP2	MAP3	MAP4	MAP5	MAP6	MAP7
68	ReportEventActionStatus	Raportowanie_Statusu_Akcji_Zdarzenia							
69	ReportEventConditionStatus	Raportowanie_Statusu_Warunku_Zdarzenia					x	x	x
70	ReportEventEnrollmentStatus	Raportowanie_Statusu_Wpisu_Zdarzenia							
71	ReportJournalStatus	Raportowanie_Statusu_Dziennika						x	x
72	ReportPoolSemaphoreStatus	Raportowanie_Statusu_Semafora_Puli				x	x	x	x
73	ReportSemaphoreEntryStatus	Raportowanie_Statusu_Pozycji_Semafora				x	x	x	x
74	ReportSemaphoreStatus	Raportowanie_Statusu_Semafora				x	x	x	x
75	RequestDomainDownload	Żądanie_Wprowadzania_Domeny	A			A			
76	RequestDomainUpload	Żądanie_Wyprowadzania_Domeny	A			A			
77	Reset	Zerowanie			x	x			x
78	Resume	Wznowienie	x		x	x			x
79	Start	Start	x		x	x			x
80	Status	Status	x	x	x	x	x	x	x
81	Stop	Stop	x		x	x			x
82	StoreDomainContent	Przechowanie_Zawartości_Domeny	B			B	B	B	B
83	TakeControl	Objęcie_Sterowania				x	x	x	x
84	TerminateDownloadSequence	Zaprzestanie_Sekwencji_Wprowadzania	A		x	A	A	A	A
85	TerminateUploadSequence	Zaprzestanie_Sekwencji_Wyprowadzania	A		x	A	A	A	A
86	TriggerEvent	Wyzwalanie_Zdarzenia							
87	UnsolicitedStatus	Status_Spontaniczny	x		x	x			
88	UploadSegment	Wyprowadzanie_Segmentu	A		x	A	A	A	A
89	Write	Zapisanie	x	x	x	x	x	x	x
90	WriteJournal	Zapisanie_Do_Dziennika						x	x

Usługi MMS można sklasyfikować w dziesięciu następujących grupach. Omówiono je pokrótce w kolejnych punktach.

3.9.1. Usługi zarządzania ogólnego

Do grupy tej zalicza się usługi dostarczające mechanizmów ustanawiania i zwalniania środowiska MMS, jak Initiate, Conclude, Abort, Cancel oraz Reject.



Przejścia:

1 - initiate request
2 - initiate.indication
3 - initiate.response (+)
4 - initiate.confirm (+)
5 - initiate.response (-)
6 - initiate.confirm (-)
7 - abort.request lub abort.indication
8 - conclude request
9 - conclude.indication
10 - conclude.response (+)
11 - conclude.confirm (+)
12 - conclude.response (-)
13 - conclude.confirm (-)

Rys. 3.11. Diagram stanów zarządzania środowiskiem

Usługi zarządzania ogólnego są konieczne do każdorazowego nawiązania komunikacji między użytkownikami MMS. Komunikacja taka może odbywać się tylko w środowisku MMS. Jako

środowisko MMS rozumie się tu wszystkie wyspecyfikowane elementy usług MMS oraz semantyki komunikacji, jakie zostaną użyte w czasie trwania asocjacji aplikacyjnej.

Wejście, przebywanie i wyjście z tego środowiska umożliwia zestaw usług środowiska i zarządzania. Dokładniej, usługi te służą do zagwarantowania użytkownikowi MMS zainicjowania, utrzymania i zwolnienia asocjacji aplikacyjnych między dwoma węzłami wyposażonymi w MMS. Węzeł, który rozpoczyna asocjację z innym węzłem pełni rolę węzła wywołującego, węzeł odpowiadający (respondent) przyjmuje rolę węzła wywoływanego. Na tym etapie współpracy obu węzłów nie jest istotne który z komunikujących się węzłów przyjmie rolę klienta, a który serwera.

Diagram stanów zarządzania środowiskiem przy użyciu usług zarządzania ogólnego przedstawiono na rys. 3.11.

W procedurze nawiązania asocjacji kluczowym zagadnieniem jest wymiana i negocjacja pewnych parametrów pomiędzy dwoma jednostkami, chcącymi nawiązać ze sobą asocjację. Parametry te dotyczą głównie możliwości i ograniczeń w realizacji usług w poszczególnych węzłach. Część przesyłanych przez węzeł wywołujący parametrów nie podlega negocjacji, ponieważ charakteryzuje ona możliwości węzła i nie może ulec zmianie, inne parametry przekazywane mogą być jako propozycje. Węzeł wywoływany analizuje wartości otrzymanych parametrów, dostosowuje je do swoich potrzeb i tak zmodyfikowane parametry odsyła nadawcy. Węzeł inicjujący sprawdza czy może zaakceptować wersje parametrów otrzymanych od węzła wywoływanego i odsyła potwierdzenie propozycji. Od tego momentu oba węzły mogą wymieniać między sobą informacje z tym, że zarówno rola klienta jak i serwera może być podjęta przez dowolny z dwóch partnerów.

3.9.2. Usługi dodatkowe stosowane w VMD

Usługi dodatkowe stosowane w VMD dostarczają mechanizmów zarządzania VMD poprzez takie możliwości, jak zapis, pobranie listy obiektów w VMD, modyfikację nazw obiektów oraz obsługi specyficznych atrybutów producenta.

Usługa GetCapabilityList (Pobranie_Listy_Uprawnień) - jest wykorzystywana w celu uzyskania pełnej lub częściowej listy możliwości definiowanych w VMD, co w praktyce oznacza przesłanie wartości atrybutu listy możliwośći (List of Capabilities) obiektu VMD. Jako możliwości rozumie tu się lokalnie zdefiniowane zasoby (fizyczne i logiczne), najczęściej w postaci zbioru, który może być przesłany jako ciąg znaków.

Usługa GetNameList (Pobranie_Listy_Nazw)
- pozwala użytkownikowi MMS na żądanie od respondenta (użytkownika MMS) pełnej lub częściowej listy obiektów nazw definiowanych w VMD.

Usługa Identify (Identyfikacja) - może być wykorzystywana przez użytkownika MMS w celu uzyskania od respondenta informacji identyfikacyjnych obiekt VMD, takich jak: nazwa producenta, nazwa modelu, numer wersji oraz lista składni abstrakcyjnych.

Usługa Status (Status) - pozwala użytkownikowi MMS określić ogólny stan VMD. W wyniku zapytania o status urządzania, użytkownik otrzymuje informację o jego stanie logicznym, stanie fizycznym oraz ewentualnie innych szczegółach podanych przez producenta VMD.

Stan fizyczny odzwierciedla stan urządzenia rzeczywistego i może nim być:

- działający (operational) - urządzenie zdolne do wykonania zadania, do jakiego jest przeznaczone;
- częściowo działający (partialy-operational) - niektóre funkcje nie mogą być wykonane ze względu na ograniczenia lub uszkodzenia sprzętowe;
- niedziałający (inoperable) - urządzenie rzeczywiste nie może wykonać żadnej z funkcji.

Jako **stan logiczny** rozumie się jeden z czterech zdefiniowanych poziomów funkcjonalności VMD. Są to:

- dozwolone zmiany stanów (State-changes-allowed) - w tym przypadku realizowane mogą być wszystkie usługi specyfikowane przez MMS;
- nie zezwala się na zmiany stanów (No-state-changes-allowed) - mogą być wykonywane tylko te usługi MMS (pod warunkiem, że są one implementowane w VMD), które dotyczą odczytywania wartości zmiennych, przesyłania raportów, identyfikacji, informacji o stanach oraz Abort, Cancel i Conclude.
- ograniczenie dozwolonych usług (Limitted-serwices-permitted) - zezwala się tylko na wykonanie następujących usług: Abort, Conclude, Status i Identify;
- dozwolone usługi pomocnicze (Support-services-allowed) - mogą być wykonywane wszystkie usługi VMD za wyjątkiem: Start, Stop, Reset, Resume i Kill.

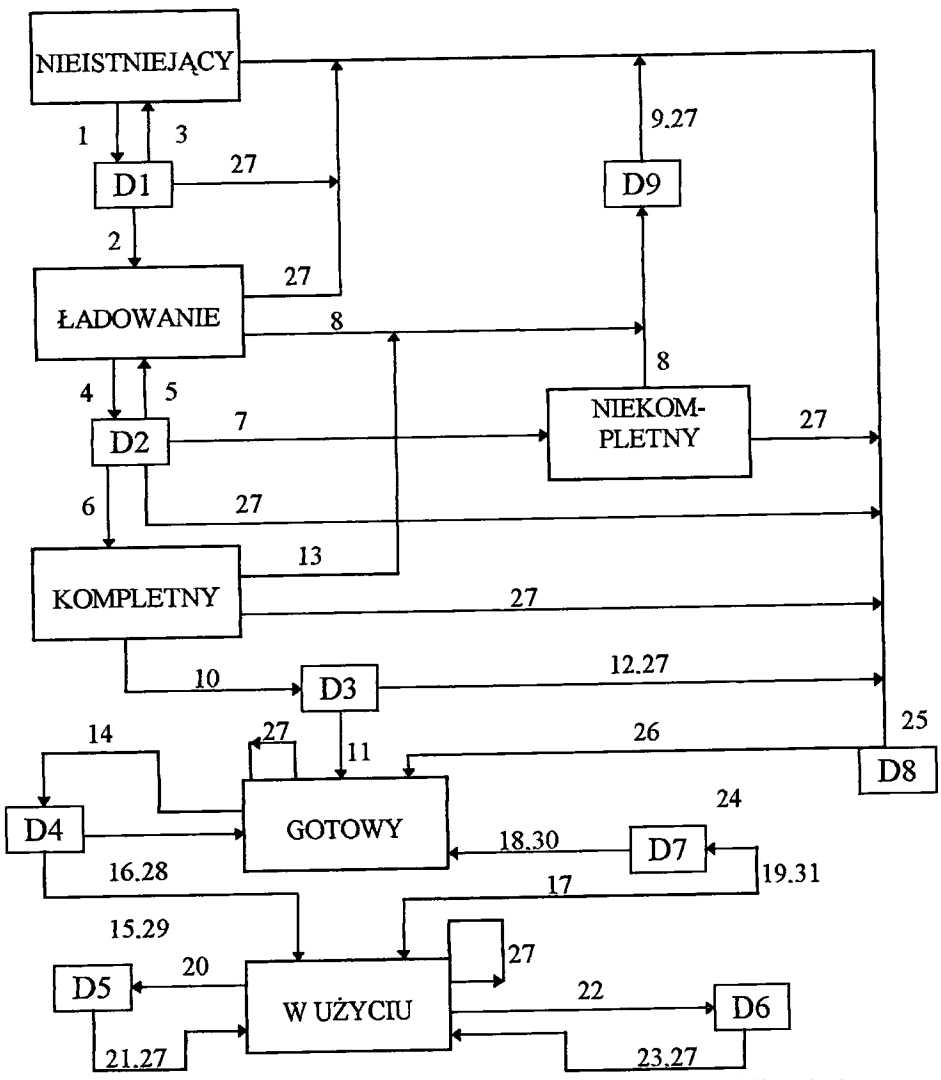
Usługa UnsolicitedStatus (Status_Spontaniczny) - użytkownik MMS może wykorzystywać tę usługę do spontanicznego raporotowania swojego statusu. Ta "spontaniczność" wynikać musi z posiadanej przez użytkownika możliwości detekcji wszelkich zmian swojego stanu. Usługa ta w swoim chrakterze jest podobna do powiadomień o zaistniałych zdarzeniach (Event Notification) związanych z usługami zarządzania, z tą jednak różnicą, że użytkownik powiadamiany o zaistniałym zdarzeniu usługą Event Notification oczekuje na tą informację (to znaczy jest zainteresowany wystąpieniem określonego zdarzenia i ma wpływ na to, czy ma być o tym powiadomiony), natomiast usługa Unsolicited Status informuje o zdarzeniach zdefiniowanycjh lokalnie i użytkownik otrzymujący takie informacje nie ma wpływu na ich definiowanie lub zmianę. Przekazywane informacje są takie same jak w usłudze Status.

Usługa Rename (Przemianowanie) - jest wykorzystywana, jeśli użytkownik chce zmienić identyfikator określonego obiektu. Użytkownik w tym celu musi wyspecyfikować w żądaniu usługi nazwę obiektu, jaki ma zostać zmieniony oraz nową nazwę tego obiektu. Do zadań systemu odpowiadającego należy weryfikacja czy taki obiekt istnieje i czy proponowana nowa nazwa może być użyta (to znaczy, czy nie ma innego obiektu danej klasy o takiej nazwie).

3.9.3. Usługi zarządzania domeną

Domena jest to abstrakcyjny obiekt reprezentujący możliwości VMD ukierunkowane na realizację określonego celu. Domena zawiera więc wszystkie zasoby wymagane do wykonania specyficznego aspektu aplikacji (np. sterowania lub monitorowania). Przykładem domeny jest model ramienia robota, na który składa się komplet zasobów koniecznych do realizacji operacji na rzeczywistym ramieniu wraz z danymi i kodem programu sterującego. Domena może być statyczna, jeśli istnieje zawsze w VMD lub dynamiczna, jeśli może być tworzona i usuwana w miarę potrzeby aplikacji. Przykładowe atrybuty domeny to:

- nazwa identyfikująca domenę
- lista możliwości domeny
- stan
- statyczność/dynamiczność
- zdolność do współdzielenia domeny przez różne inwokacje programów
- lista podległych domenie obiektów
- zdefiniowanych lista inwokacji programów (PI)
- listy zdarzeń związanych z domeną



Rys. 3.12. Diagram stanów domeny - symbole przejść zebrano w tablicy 3.4.

Tablica 3.4. Przejścia między stanami domeny.

1 - InitiateDownload Sequence.indication	17 - DeleteProgramInvocation.indication
2 - InitiateDownload Sequence.response (+)	18 - DeleteProgramInvocation.response(+)
3 - InitiateDownload Sequence.response (-)	19 - DeleteProgramInvocation.response(-)
4 - DownloadSegment.request	20 - CreateProgramInvocation.indication
5 - DownloadSegment.confirm (+) More Follows = true	21 - CreateProgramInvocation.response(+) lub (-)
6 - DownloadSegment.confirm (+) More Follows = false	22 - DeleteProgramInvocation.indication
7 - DownloadSegment.confirm (-)	23 - DeleteProgramInvocation.response(+) lub(-)
8 - TerminateDownloadSequence.request. Discard present	24 - DeleteDomain.indication
9 - TerminateDownloadSequence.confirm (+) (-)	25 - DeleteDomain.response(+)
10 - TerminateDownloadSequence.request. Discard not present	26 - DeleteDomain.indication(-)
11 - TerminateDownloadSequence.confirm (+)	27 - Abort.indication
12 - TerminateDownloadSequence.confirm (-)	28 - Abort.indication (utworzenie inwokacji programu zakończyło się niepowodzenim)
13 - TerminateDownloadSequence.request. Discard present	29 - Abort.indication (utworzenie inwokacji programu zakończyło się powodzeniem)
14 - CreateProgramInvocation.indication	30 - Abort.indication (usunięcie inwokacji programu zankończone powodzeniem)
15 - CreateProgramInvocation.response(+)	31 - Abort.indication (usunięcie inwokacji programu zankończone niepowodzeniem)
16 - CreateProgramInvocation.response(-)	

Domena może znajdować się w jednym z pięciu stanów:

- - ŁADOWANIE (LOADING) - stan wskazujący na ładowanie domeny;
- - KOMPLETNY (COMPLETE) - stan wskazujący na załadowanie ostatniego segmentu, ale przed wygenerowaniem komendy zamykającej operację ładowania;
- - NIEKOMPLETNY (INCOMPLETE) - stan, w którym operacja ładowania jest zakończona przed zakończeniem całego procesu ładowania;
- - GOTOWY (READY) - stan będący konsekwencją poprawnie zakończonej sekwencji ładowania;
- - W UŻYCIU (IN-USE) - stan wskazujący na istnienie co najmniej jednej zdefiniowanej inwokacji programu wykorzystującej tę domenę.

Domena nie istnieje, jeżeli nie zostanie utworzona. Dla pełności definicji diagramu stanów domeny przyjmuje się stan nieistnienia NIEISTNIEJĄCY (NON-EXISTENT), odzwierciedlający brak domeny.

Przejścia między poszczególnymi stanami domeny zobrazowano na rys. 3.12. Poza stanami wymienionymi wyżej uwzględniono tam także stany pośrednie, oznaczone D1..D9.

Usługi zarządzania domeną obejmują modelowanie i utrzymanie domen w VMD. Usługi tej grupy definiują obiekt domeny z jej atrybutami oraz pozwalają na utworzenie i załadowanie obiektów domeny.

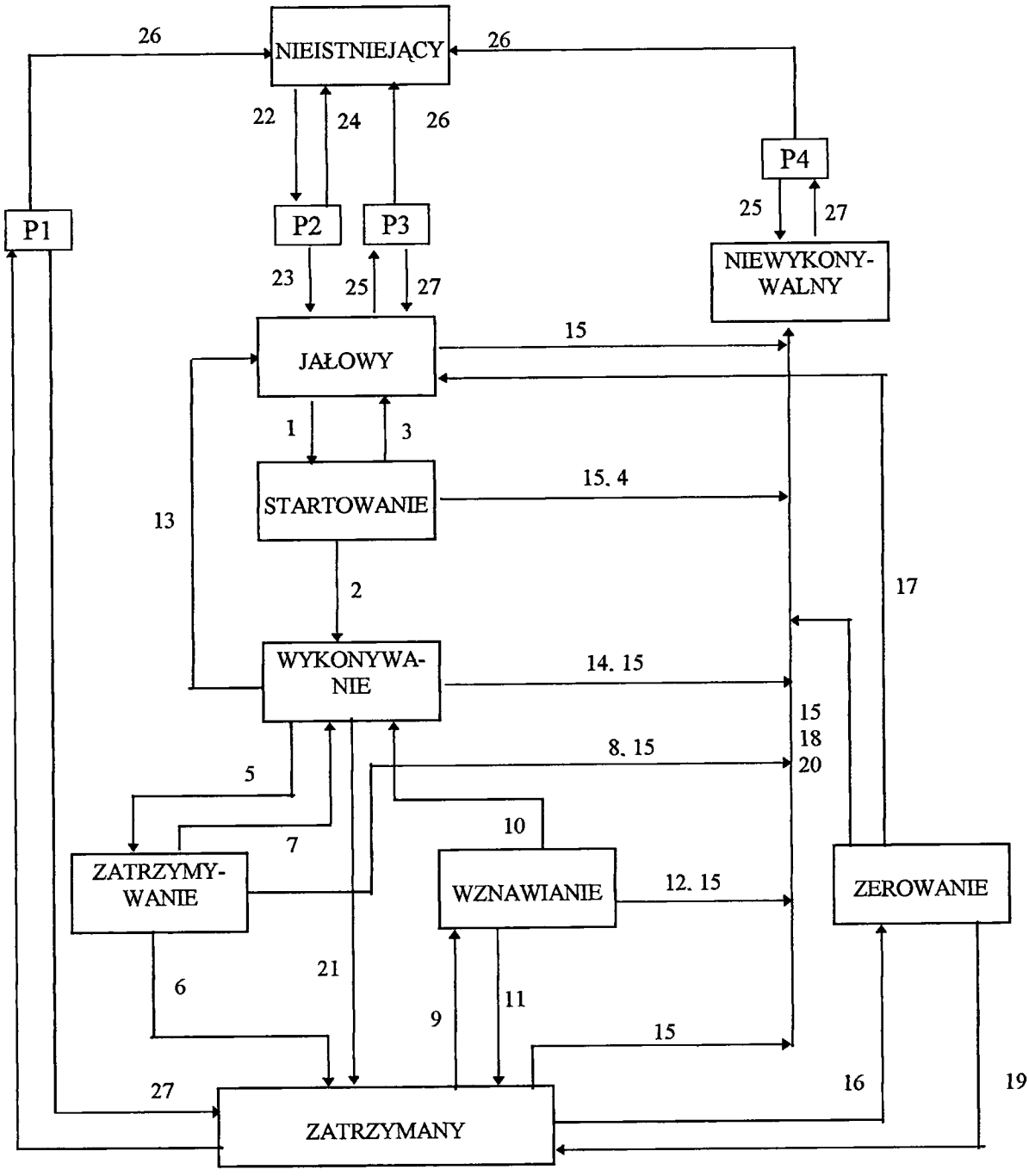
3.9.4. Usługi zarządzania wywołaniem (inwokacją) programu

Pojęcie wywołania lub inwokacji programu najprościej może być zinterpretowane jako odpowiednik wątku wykonania programu w środowisku wieloprogramowym. Wywołanie programu może być zdefiniowane, a następnie powoływane do wykonania lub też może być tworzone dynamicznie lokalnie lub zdalnie przez usługi MMS. Na element ten składa się zbiór domen łącznie z przynależnymi do nich informacjami sterującymi koniecznymi do ich wykonania. Zadania związane z daną inwokacją programu dotyczą implementacji określonej funkcji sterującej, np. zadania sterowania ruchem ramienia robota mogą być pogrupowane jako jedno wywołanie programu.

Możliwość wykonania wywołania programu uwarunkowana jest istnieniem co najmniej jednej związanej z nim domeny. Domeny te powinny zawierać wszelkie informacje potrzebne do realizacji inwokacji (np. kod programu i segmenty danych) za wyjątkiem aktualnych parametrów, które dostarczane są osobno.

Wywołanie programu jest jednostką dynamiczną i w czasie swojego istnienia może znajdować się w różnych stanach. Brak wywołania programu uwzględnia stan NIEISTNIEJĄCY (NON-EXISTENT). Pozostałe stany to:

- JAŁOWY (IDLE) - jest to stan, w którym wywołanie programu jest utworzone, ale nie zostało jeszcze uaktywnione.
- WYKONYWANIE (RUNNING) - stan ten określa, że inwokacja programu jest wykonywana. Nie ma tu ścisłej definicji wykonania, ponieważ jest to sprawą lokalną i nie należy tego stanu utożsamiać z podobnym w odniesieniu do procesu w systemie wieloprogramowym. Jako "wykonanie" rozumie się tu fakt wystąpienia zmian w jakiejkolwiek domenie związanej z wywołaniem programu.
- ZATRZYMANY (STOPPED) - stan ten jest osiągnięty, gdy wywołanie programu kończy swoje wykonanie w sposób normalny lub zostaje zatrzymane przez klienta. W tym stanie, żadna z domen związanych z inwokacją programu nie ulega zmianie.
- NIEWYKONYWALNY (UNRUNNABLE) - osiągnięcie tego stanu spowoduje, że wywołanie programu nie może być dalej wykonywane, ale nie zostało jeszcze usunięte.
- STARTOWANIE (STARTING) - stan rozpoczynania uzyskuje wywołanie programu jako stan przejściowy między IDLE i RUNNING. Otrzymany został prymityw (operacja elementarna) Start.indication, ale jeszcze nie wysłano jego potwierdzenia. W tym stanie VMD realizuje wszelkie procedury inicjujące, konieczne do wykonania wywołania programu.
- ZATRZYMYWANIE (STOPPING) - podobnie jak poprzednio, jest to stan przejściowy między stanami RUNNING i STOPPED, występujący w czasie pomiędzy przyjęciem Stop.indication, a wygenerowaniem potwierdzenie (Stop.confirm). W tym stanie VMD może wykonywać procedury zatrzymujące operacje, które mogą być zdefiniowane lokalnie lub wymagane przez standard.



Rys 3.13. Diagram stanów wywołania programu. Numery oznaczają zdarzenia powodujące zmianę stanów i opisane są w tabl. 3.5.

- WZNAWIANIE (REASUMING) - jest stanem przejściowym między STOPPED i RUNNING - wywołanie programu znajdujące się w tym stanie wykonuje procedury wznawiające wykonanie począwszy od stanu, w którym wywołanie znalazło się w stanie STOPPED.
- ZEROWANIE (RESETTING) - stan przejściowy między stanami STOPPED i IDLE, który może być uzyskany tylko przez wywołanie programu utworzone z przeznaczeniem do wielokrotnego używania (re-usable). Każde wykonanie wywołania programu jest niezależne

od wykonania innych inwokacji i każde z nich rozpoczyna się od początku, co jest znaczącą różnicą w stosunku do operacji wznawiania (resume).

Poza stanami wymienionymi wyżej na rys. 3.13 uwzględniono również stany pośrednie, oznaczone jako P1..P4.

Tablica 3.5. Zdarzenia powodujące zmianę stanów automatu opisującego wywołanie programu

1. Start.indication	15.Kill.response(+)
2. Start.response (+)	16.Reset.indication
3. Start.response(-) niedestrukcyjny	17.Reset.response (+) Reusable=true
4. Start.response(-) destrukcyjny	18.Reset.response (+) Reusable=false
5. Stop.indication	19.Reset.response(-) niedestrukcyjny
6. Stop.response (+)	20.Reset.response(-) destrukcyjny
7. Stop.response(-) niedestrukcyjny	21.zatrzymanie programu
8. Stop.response(-) destrukcyjny	22.CreateProgramInvocation.indication
9. Resume.indication	23.CreateProgramInvocation.response(+)
10.Resume.response (+)	24.CreateProgramInvocation.response(-)
11.Resume.response(-) niedestrukcyjny	25.DeleteProgramInvocation.indication
12.Resume.response(-) destrukcyjny	26.DeleteProgramInvocation.response(+)
13.koniec programu: Reusable=true	27.DeleteProgramInvocation.response(-)
14.koniec programu: Reusable-false	

3.9.5. Usługi dostępu do zmiennej

Usługi dostępu do zmiennej wykorzystywane są do specyficznej wymiany informacji, pozwalając na podstawowe czynności manipulacyjne.

Przez **zmienną** rozumie się abstrakcyjny element VMD, który może dostarczyć (odczyt) lub zaakceptować (zapis) wartość danej określonego typu. **Typ danych** definiuje abstrakcyjny opis, obejmujący składnię, dopuszczalny zakres wartości oraz reprezentację tej zmiennej w trakcie komunikacji MMS.

Usługa dostępu do zmiennych MMS pozwala na odczyt i zapis wartości zmiennych zlokalizowanych w VMD. MMS definiuje pięć **obiektów dostępu** do zmiennych MMS:

- obiekt zmiennej nienazwanej (Unnamed Variable object)
- obiekt zmiennej nazwanej (Named Variable object)
- obiekt dostępu rozrzuconego (Scattered Access object)
- obiekt nazwanej listy zmiennych (Named Variable List object)
- obiekt nazwy typu. (Named Type object)

Dostępu do obiektu zmiennej (nazwanej lub nienazwanej) dotyczą dwa następujące założenia:

- wykonanie każdej funkcji dostępu do obiektu zmiennej może zakończyć się albo pomyślnie albo takiego dostępu można nie uzyskać; wyklucza się sytuacje pośrednie,
- operacja dostępu do obiektu zmiennej jest nieprzerywalna.

Powiązania między zmiennymi rzeczywistymi i obiektami MMS określają dwie abstrakcyjne funkcje: V-Pobranie (V-Get) i V-Oddanie (V-Put). Funkcja V-Get pozwala na pobranie

wartości zmiennej MMS z VMD, a funkcja V-Put służy do wprowadzenia do VMD nowej wartości zmiennej MMS, przez co wykonanie jej modyfikuje stan VMD.

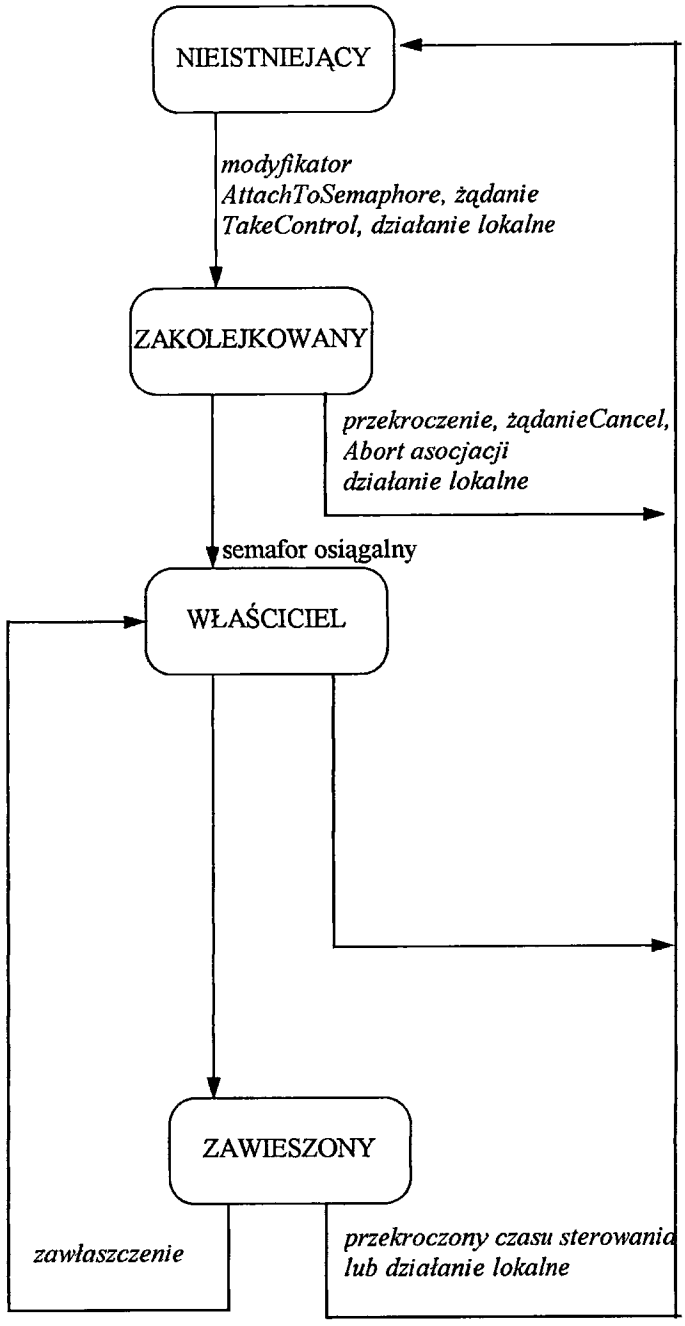
Odwzorowanie między zmienną MMS i zmienną rzeczywistą w VDM definiowane jest na dwóch poziomach abstrakcji. Służą do tego dwa obiekty: obiekt zmiennej nazwanej i obiekt zmiennej nienazwanej. Obiekty **zmiennych nienazwanych** modelują wszelkie aspekty związane z rzeczywistym urządzeniem, a więc dotyczą adresowalnych jednostek VMD, natomiast obiekty **zmiennych nazwanych** modelują sposób widzenia zmiennej w VMD przez aplikację. Taki obiekt związany jest więc bezpośrednio z domeną asocjacji aplikacyjnej lub domeną VMD. Oba typy obiektów modelują dostęp do zmiennej VMD, jednak istnieje dość znaczna różnica między nimi oraz ich oddziaływaniu na klienta MMS. Różnica polega na sposobie odwołania się do zmiennej. Obiekt zmiennej nazwanej posługuje się nazwą, określoną w procesie aplikacyjnym. Natomiast obiekt zmiennej nienazwanej opisuje dostęp do wartości rzeczywistej używając adresu, który jest elementem specyficznym danego urządzenia. W konsekwencji obiekt zmiennej nazwanej pozwala opisać zmienną, której adres nie jest znany, zaś obiekt zmiennej nienazwanej wymaga znajomości ustalonego w urządzeniu adresu zmiennej.

Obiekt zmiennej nazwanej może być wykorzystany zarówno do opisu jednej zmiennej jak i zbioru danych związanych z aplikacją. W tym ostatnim przypadku są to zbiory danych, do których można odwołać się za pomocą jednej nazwy (np. tablica). Obiekt zmiennej nienazwanej modeluje dostęp do wbudowanych, adresowalnych elementów VMD, zależnych od rodzaju urządzenia i implementacji.

3.9.6. Usługi zarządzania semaforem

Aplikacja MMS może żądać ściśle kontrolowanego dostępu do obszarów pamięci w VMD lub też konieczna może być synchronizacja operacji realizowanych w różnych zadaniach aplikacji. Do tego celu stosuje się mechanizmy zwane semaforami. W MMS semafor może być wykorzystany przez użytkowników do kontrolowania dostępu do określonego podzbioru VMD, np. kiedy równocześnie pracuje kilka robotów lub do synchronizacji współdziałania zdalnych aplikacji. Samo VMD nie zabezpiecza obszaru pamięci i dlatego za koordynację procesów odpowiada aplikacja. Reguły stosowania semaforów w zakresie danej aplikacji powinny być wynikiem uzgodnień między użytkownikami tej aplikacji.

Zasobem krytycznym jest każdy zasób, do którego równocześnie usiłuje uzyskać dostęp co najmniej dwu użytkowników. Uzyskanie dostępu do zasobu krytycznego (np. może to być operacja modyfikacji zmiennej VMD) wymaga otrzymania przez użytkownika zezwolenia na wejście do sekcji krytycznej (fragmentu procesu). Zezwoleniem tym steruje mechanizm semaforowy. W ogólnym pojęciu **semaforem** jest całkowita liczba przyjmująca wartości dodatnie. Jeżeli liczba ta ograniczona jest do dwóch wartości: 0 i 1, to mówi się o semaforze binarnym. W MMS semafor modelowany jest jako zbiór uprawnień i udzielenie dostępu do zasobu jest równoznaczne z przydzieleniem użytkownikowi jednego z wolnych uprawnień, jeśli takie istnieje. Dostęp do zasobu krytycznego (np. dokonanie modyfikacji zmiennej) możliwy jest tylko dla użytkownika posiadającego uprawnienie. Semafor dysponuje ograniczoną liczbą uprawnień, odzwierciedlającą maksymalną liczbę użytkowników, którzy mogą równocześnie znajdować się w swojej sekcji krytycznej, czyli korzystać z zasobu krytycznego.



Rys. 3.14. Model pozycji semafora.

Każde żądanie sterowania semaforowego generuje nową, tak zwaną **pozycję semafora** (*semaphore entry*) (patrz rys. 3.14). Jeśli dany semafor dysponuje wolnym uprawnieniem, to wtedy przydziela je tak utworzonej pozycji. W przeciwnym zaś przypadku pozycja ta zostaje ustawiona w kolejce do określonego semafora. Jeśli uprawnienie zostaje zwolnione, to pierwsza pozycja listy oczekujących staje się jego właścicielem.
MMS definiuje dwa typy semaforów:
1. semafory uprawnień (*token semaphores*)
2. semafory puli (*pool semaphores*)

W pierwszym przypadku wszystkie uprawnienia traktowane są jednakowo. Jeśli więc klient żąda usługi semaforowej, to otrzymuje pierwsze wolne uprawnienie. Natomiast każdy uprawnienie z puli ma przypisaną nazwę i żądanie może dotyczyć dynamicznego lub bezpośredniego przydziału określonego nazwą uprawnienia. Semantyka takiego uprawnienia zdefiniowana jest lokalnie i nie podlega usługom zarządzania oferowanym przez MMS.

Konsekwencją sterowanego dostępu do zasobów jest konieczność zabezpieczenia użytkowników aplikacji przed wzajemną blokadą. Mechanizmami stosowanymi w MMS w celu niedopuszczenia do blokady są operacje związane z upływem czasu oraz pewne możliwości dawane aplikacji takie jak np. przejęcie kontroli nad semaforem lub anulowanie bieżącego żądania.

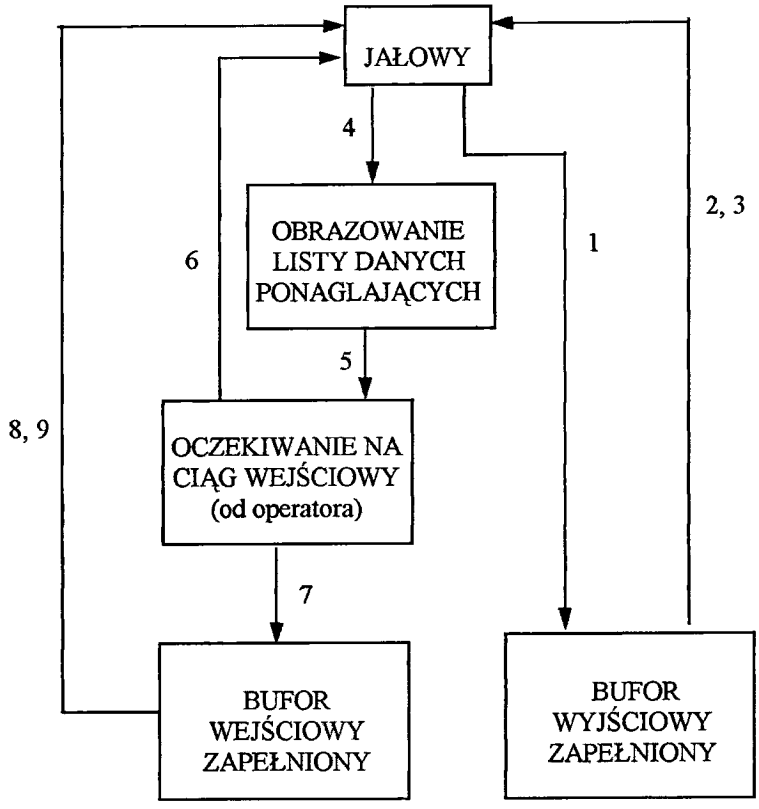
Dwie klasy obiektów semaforowych, jakimi może zarządzać MMS, to znaczy semafory uprawnienia oraz semafory puli, definiowane są za pomocą automatu skończonego oraz charakteryzowane zbiorem określonych atrybutów. Obiekt semafora określony jest jako wystąpienie konkretnych wartości atrybutów ogólnej klasy obiektu, jakim jest zdefiniowana klasa semafora. Odwołanie do semafora następuje przez nazwę, której dotyczą takie same zasady, jak każdej innej nazwy w VMD. Konkretny semafor może być już uprzednio zdefiniowany w VMD, może też powstać w wyniku utworzenia domeny lub w wyniku realizacji usługi DefineSemaphore.

Działanie semafora modelowane jest jako przetwarzanie kolejki bazującej na dwóch listach: listy właścicieli uprawnień oraz listy zgłoszeń do semafora. Każda pozycja w takiej kolejce jest tworzona jest za pomocą żądania TakeControl lub innych środków, jakimi mogą być lokalne lub zdalne żądania modyfikujące. Tak długo, jak długo obsługiwany jest dana pozycja semafora należeć on będzie do listy właścicieli semafora, jeżeli zaś zgłoszenie nie może być w danej chwieli obsłużone, to zajmie ono kolejną pozycję na liście zgłoszeń.

Właścicielem semafora jest proces aplikacyjny (w środowisku OSI), który może być zarówno lokalny jaki i zdalny w stosunku do VMD. W obu przypadkach musi być utrzymana asocjacja aplikacyjna z serwerem, trwająca przynajmniej od chwili żądania semafora do jego zwolnienia. Proces aplikacyjny może wysłać wiele zgłoszeń żądania tego samego semafora na tej samej lub na różnych asocjacjach. Na tej samej asocjacji, sterowanie semaforami leży w gestii tylko partnerskich jednostek takiej asocjacji. W wyniku ponawianych żądań uprawnienia, proces aplikacyjny może stać się właścicielem wielu uprawnień lub wspólnej puli semaforowej. Jednak serwer rozróżnia właścicieli semaforów tylko wtedy, gdy są związani z różnymi asocjacjami.

3.9.7. Usługi komunikacji operatorskiej

Usługi komunikacji operatorskiej wspomagają lokalny interfejs człowiek - urządzenie.



Przejścia:

1 - Output.indication
2 - Output.response(+)
3 - Output.response(-)
4 - Input.indication
5 - D-Put
6 - Input.response(-) przy upływie czasu
7 - E-Get
8 - Input.response(+)
9 - Input.response(-)

Rys. 3.15. Diagram stanów stanowiska operatorskiego

Usługa komunikacji operatorskiej służy do wymiany danych między lokalnym VMD i operatorem. Wyróżnia się dwa typy usług komunikacji operatorskiej: wejście i wyjście.

- Input (wejście) - klient za pomocą tej usługi otrzymuje strumień danych z urządzenia wejściowego.
- Output (wyjście) - usługa pozwalająca na wyświetlenie sekwencji znaków wyjściowych na stacji operatora.

Specyfikacja MMS nie precyzuje mechanizmu sterowania przepływem koniecznym do kontroli funkcji wejściowych i wyjściowych. Utrzymanie integralności danych w stacji operatorskiej dla pojedynczego użytkownika musi zapewnić implementacja. W przypadku wielu użytkowników,

korzystających z jednego obiektu stacji operatorskiej należy wykorzystać usługi semaforowe do zapewnienia wyłączności sterowania stacją operatorską przez poszczególnych użytkowników.

Obiekt stacji operatorskiej opisywany jest następującymi parametrami:
- Nazwa stacji
- Typ stacji - wskazuje na typ stacji obiektu stacji operatorskiej. Wyróżnia się trzy typy stacji: WEJŚCIOWA (ENTRY), akceptująca tylko żądanie usługi wejścia, OBRAZOWANIA (DISPLAY), akceptująca tylko żądania usługi wyjściowej oraz WEJŚCIOWA-OBRAZOWANIA (ENTRY-DISPLAY), która dopuszcza zarówno żądania usługi wejścia jak i wyjścia.
- Bufor wejściowy - atrybut konieczny dla stacji typu ENTRY i ENTRY-DISPLAY i zawiera wartość parametru określonego w usłudze wejścia wskazującego na strumień wejściowy.
- Lista bufora wyjściowego - parametr konieczny dla stacji typu ENTRY-DISPLAY i DISPLAY, zawierający wartość parametru określonego w usłudze wyjścia wskazującego na listę danych wyjściowych.
- Stan - określa bieżący stan stacji operatorskiej (rys. 3.15).

3.9.8. Usługi zarządzania dziennikiem

MMS umożliwia gromadzenie w dzienniku istotnych informacji. Usługi tej grupy pozwalają na inicjację dziennika, odczytanie, zapis oraz sprawdzenie statusu dziennika.

Dzienniki pozwalają na zapisywanie i odtwarzania informacji zapamiętanej w kolejności chronologicznej. Informacja ta, zorganizowana w rekordy i etykietowana czasem, dotyczyć może np. historii wartości zmiennych, stanów zdarzeń lub danych od operatora (adnotacji operatora). Dziennik taki, na ogół, przechowuje się w pamięci dyskowej, ale jest to kwestia implementacji i nie podlega normalizacji. Dziennik definiowany jest za pomocą dwóch typów obiektów:
- obiekt dziennika (*jurnal object*)
- obiekt pozycji dziennika (*jurnal entry object*)

Pierwszy z nich dotyczy ogólnych cech dziennika takich jak nazwa i czas utworzenia dziennika, a drugi obiekt pozwala na zapis poszczególnych rekordów dziennika jako informacji cechowanych znacznikiem czasu.

Obiekt dziennika charakteryzuje się szeregiem atrybutów, z których podstawowym jest nazwa, a poza tym są to:
- usuwalność - informuje, czy dziennik może (TRUE) czy nie (FALSE) być usunięty za pomocą odpowiedniej usługi.
- lista pozycji - atrybut stanowiący listę odwołań do obiektów pozycji dziennika. Liczba takich pozycji może również wynosić zero.

Obiekt pozycji dziennika jest rekordem informacyjnym ze znacznikiem czasu. Informacja przechowywana w jednej pozycji dziennika może być jedną z trzech typów:
1. ADNOTACJA (ANNOTATION) - stanowiąca tekstowy komentarz (najczęściej pochodzący od operatora), który może służyć do dokumentacji pewnych zdarzeń, warunków, czy też akcji.

2. DANE-ZDARZENIA (EVENT-DATA) - odniesienie do zapisywanego zdarzenia oraz jego stanu.
3. DANE (DATA) - odniesienia (etykiety) do wartości jednej lub więcej zmiennych.

Każdy obiekt pozycji dziennika charakteryzuje się takimi atrybutami jak:

- odniesienie do dziennika - jednoznaczna identyfikacja obiektu dziennika, z którym obiekt pozycji dziennika jest związany
- identyfikator pozycji - wartość tego atrybutu przydzielona powinna być przez serwer, co gwarantuje jednoznaczność i niepowtarzalność identyfikatora. Atrybut ten daje możliwość rozróżnienia poszczególnych obiektów pozycji dziennika wtedy, gdy etykietowane są takim samym znacznikiem czasu.
- identyfikacja procesu aplikacyjnego - identyfikator procesu, na którego żądanie został utworzony obiekt dziennika pozycji.
- znacznik czasu - atrybut zawierający czas
- kolejność przyjmowania - atrybut reprezentujący kolejność w jakiej został utworzony obiekt pozycji dziennika wśród wszystkich innych obiektów o tym samym znacznikiem czasu oraz tej samej wartości atrybutu odniesienia do dziennika.
- szczegóły dodatkowe - są to dodatkowe atrybuty, których znaczenie i składnia definiowane są w dokumentach standardów towarzyszących.
- typ informacji - określa typ informacji zawartej w obiekcie pozycji dziennika. Może to być: ANNOTATION, EVENT-DATA, DATA.

3.9.9. Usługi zarządzania zdarzeniem

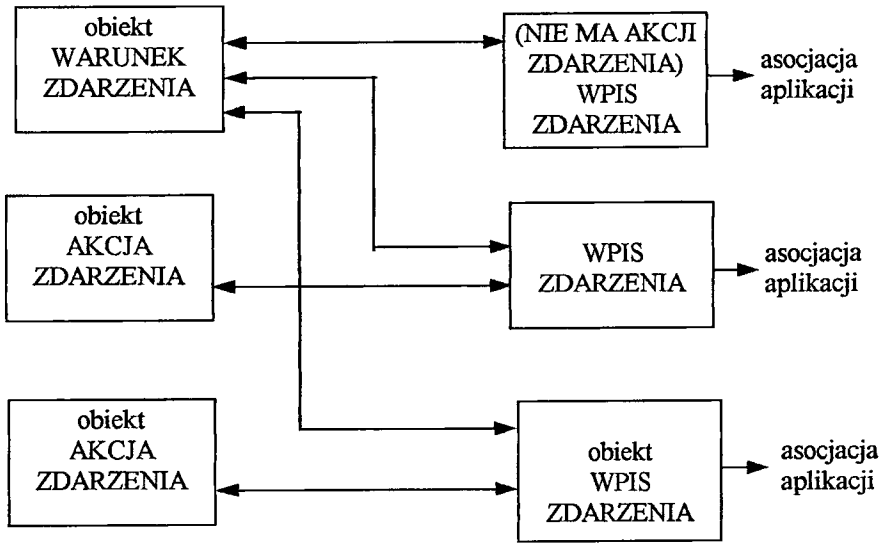
W przemysłowych systemach sterowania, zdarzeniem może być wysłanie sygnału (alarmu) informującego o zaistniałej sytuacji, która najczęściej wiąże się z przekroczeniem określonych wartości krytycznych, np. temperatury, ciśnienia, napięcia czy poziomu cieczy. W sieciowym środowisku przemysłowym informacja o wystąpieniu zdarzenia przekazywana jest przez sieć i zdarzenie to obsługiwane jest zgodnie z regułami komunikacji sieciowej.

Usługi zarządzania zdarzeniami pozwalają użytkownikowi MMS na sterowanie zdarzeniami w VMD, a w szczególności dostarczają mechanizmów służących do definicji i kontroli zdefiniowanych obiektów zdarzeń oraz służą do powiadamiania aplikacji o wystąpieniu określonego zdarzenia. Z punktu widzenia VMD realizacja takich możliwości wymaga rozszerzenia zdefiniowanego modelu VMD o dodatkowe obiekty i usługi.

Związki między zdarzeniami i akcjami określają 3 nowe obiekty, a na manipulację na tych obiektach pozwala zdefiniowanych w tym celu 19 usług (patrz tabl. 3.3). Wspomniane trzy obiekty to:

- warunek zdarzenia - definiuje typ zdarzenia, którym zainteresowany jest klient; odpowiedzialne są równocześnie za detekcję i priorytet zdarzenia,
- akcja zdarzenia - specyfikuje akcję, którą powinien wykonać serwer jeśli wystąpi określone zdarzenie (wykonanie usług MMS),
- wpis zdarzenia - obiekty przypisujące akcje do zdarzeń, odpowiedzialne też za powiadomienie właściwego procesu (klienta) o wystąpieniu zdarzenia.

Związki między wspomnianymi obiektami przedstawiono na rys. 3.16.



Rys. 3.16. Relacje między obiektami zarządzania zdarzeniami

3.9.10. Usługi zarządzania plikami

Usługi zarządzania plikami pozwalają na dokonanie przez procesy aplikacyjne pewnych operacji na plikach. Usługi te dostarczają dość ograniczonego zbioru funkcji do manipulacji na nazwach i atrybutach, pozwalają jednak na pełny transfer plików. Pliki zawierające programy sterujące i dane mogą przyjmować tylko format niestrukturalnego pliku binarnego. Interpretacja zawartości plików należy do systemów je wykorzystujących. Wszystkie te ograniczenia odnośnie reprezentacji plików oraz funkcjonalności usługi zarządzania plikami są zamierzone ze względu na podstawowe wymaganie stawiane usłudze, jakim jest jej prostota.

Klient MMS może żądać następujących operacji na plikach znajdujących się w wirtualnej pamięci pików serwera:

- FileOpen - otwarcie pliku konieczne do identyfikacji pliku w wirtualnej przestrzeni plików oraz ustawienia pliku w stan otwarcia.
- FileRead - czytanie pliku używane do transferu części lub całości otwartego pliku od serwera MMS do klienta MMS. Dane przesyłane są sekwencyjnie.
- FileClose - zamknięcie pliku jest operacją służącą do zwolnienia zasobów związanych z transferem pliku.
- FileRename - zmiana nazwy pliku znajdującego się w wirtualnej pamięci plików serwera
- FileDelete - usunięcie pliku z pamięci serwera.
- FileDirectory - usługa katalogowa może być żądana przez klienta MMS w celu otrzymania nazwy i atrybutów pliku lub grupy plików.
- ObtainFile - przesłanie pliku, jest to usługa związana z dostępem do pliku. Klient MMS może za jej pomocą poinformować odpowiadający serwer MMS, że chce otrzymać określony plik z określonego serwera. Serwer, który otrzymał takie zlecenie (odpowiadający) jest odpowiedzialny za interakcję z wyznaczonym serwerem, zweryfikowanie nazwy pliku oraz przesłanie jako odpowiedź pliku źródłowego do klienta żądającego wykonania właśnie omawianej usługi.

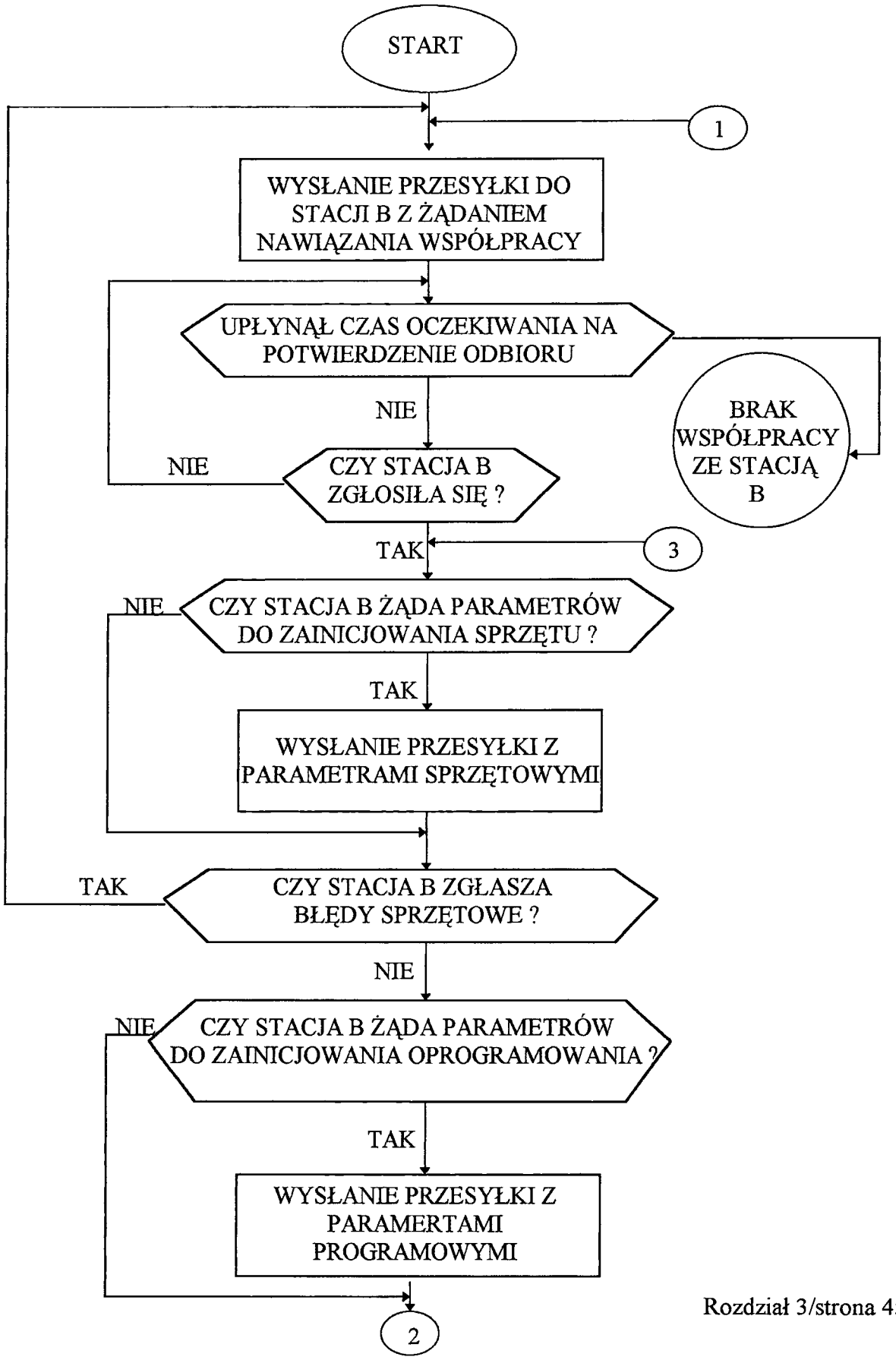
3.10. Przykład VMD dla sterownika wsadowego (batch controller)

Rozpatrzmy teraz dla przykładu model VMD utworzony dla sterownika wsadowego. Sterownik wsadowy pozwala kontrolować dwa identyczne procesy, nazywane dalej procesem A i B. Domena WE/WY jest specyficzna dla każdego procesu. Domena programu sterującego jest wspólna i zawiera algorytm sterowania. Klient MMS może tworzyć oddzielne wywołania programu dla sterowania każdym procesem. Pozwala to klientowi na kontrolowanie receptury i wykonywanie sterowania nadrzędnego (start, stop, itd.) obu procesów niezależnie od siebie. Model VMD opisanego sterownika zapewnia klientowi odpowiednie metody sterowania procesem wsadowym poprzez użycie usług MMS. Dla uruchomienia i sterowania tymi dwoma procesami klient podejmuje następujące czynności:

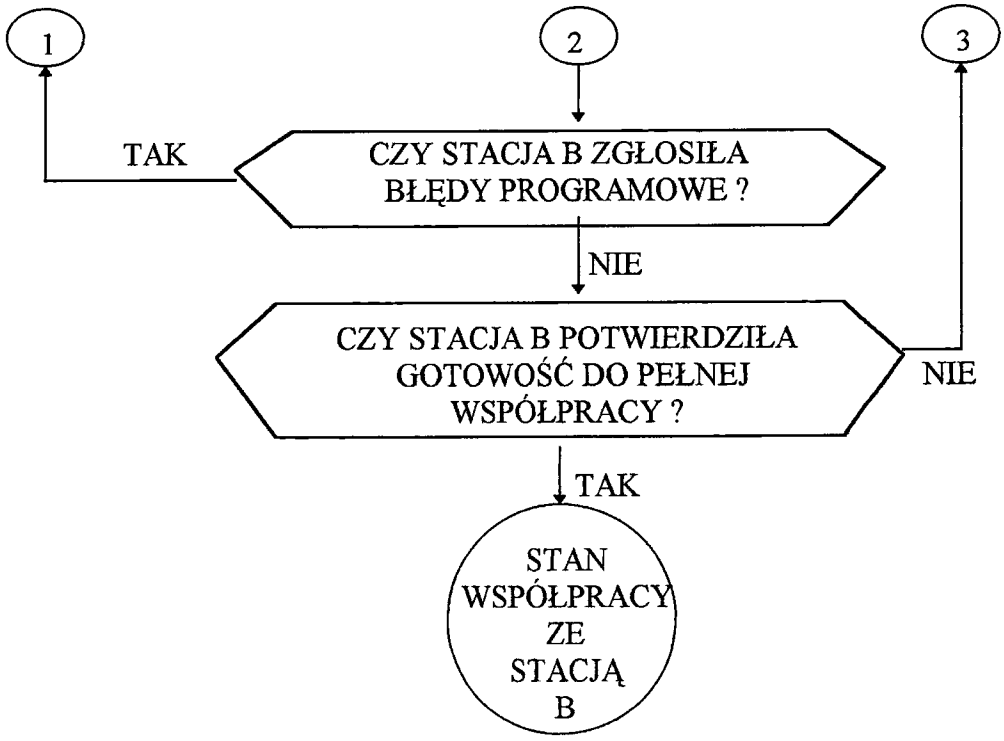
- dla obu procesów A i B inicjuje wprowadzanie do pamięci domen z danymi, domen WE/WY i domeny zawierającej program sterujący;
- tworzy wywołanie programu A składającego się z domeny WE/WY, domeny programu sterującego oraz domeny zawierającej dane dla procesu A;
- uruchamia wywołanie programu A;
- tworzy wywołanie programu B składającego się z domeny WE/WY, domeny programu sterującego oraz domeny zawierającej i dane dla procesu B,
- uruchamia wywołanie programu B.

3.11. Opis strukturalny usług MMS na przykładzie usług Initiate i Conclude

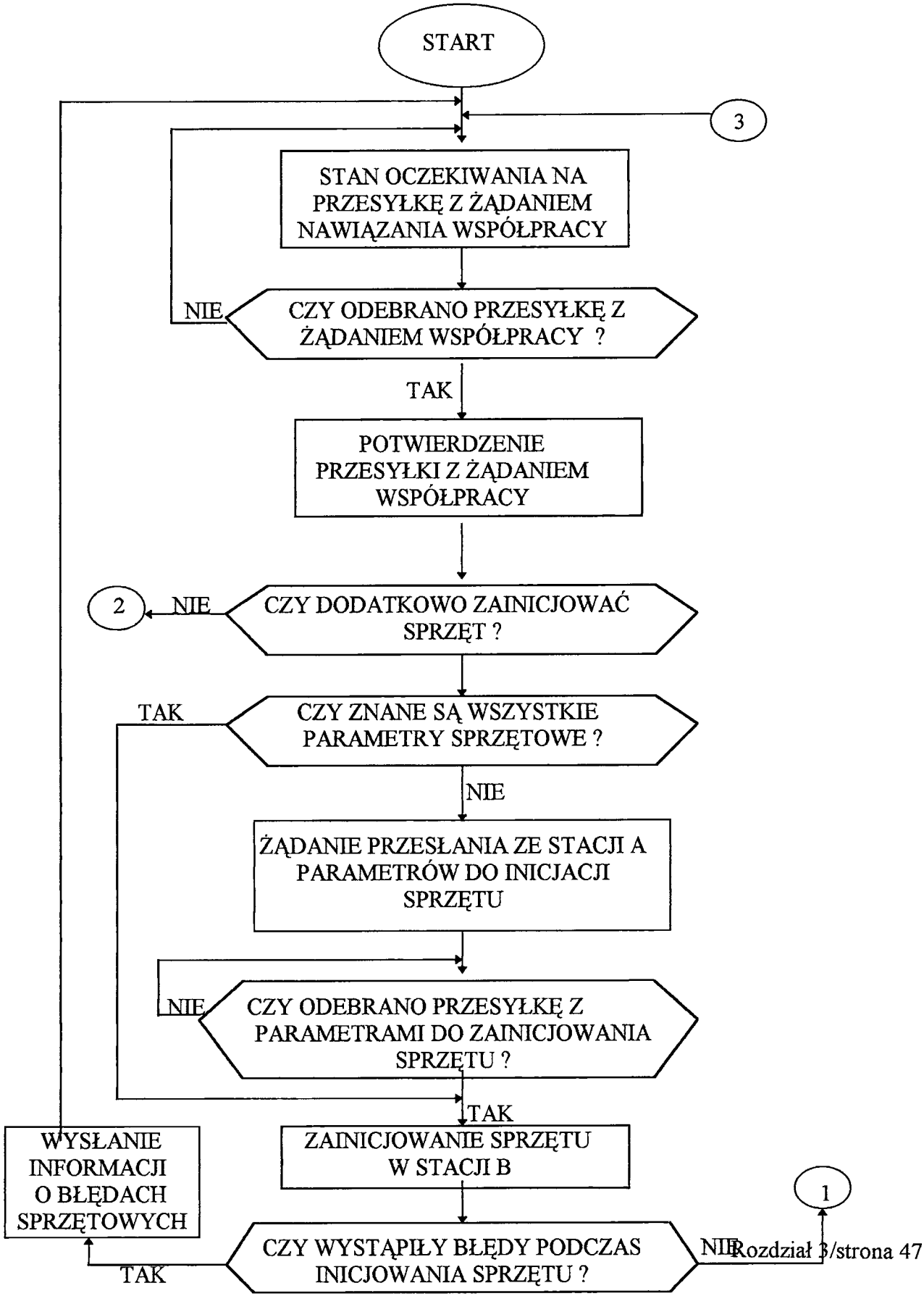
Przedstawiony w poprzednim punkcie model VMD dla sterownika wsadowego można łatwo uogólnić i zastosować do zmodelowania przebiegu realizacji poszczególnych usług MMS (a więc i VMD). Takie podejście umożliwia stworzenie graficznej reprezentacji działania - w postaci tzw. flowdiagamu - grafu przepływu sygnałów. Jest to jedna z postaci zapisu uwzględniana w teorii grafów i daje się ona w sposób automatyczny, przy użyciu znanych algorytmów teorii grafów przekształcać do innej postaci zapisu, np. do tzw. tablicy incydencji. tablicy przyległości lub zapisów listowych. Umożliwia to późniejsze łatwe manipulacje nad odpowiednimi grafami i ich analizę. Celem przykładu, poniżej przedstawiono grafy opisujące usługi Initiate (rys. 3.17 i 3.18) i Conclude (rys. 3.19 i 3.20). W obydwu przypadkach przyjęto, że stacja oznaczona symbolem A jest inicjatorem transakcji sieciowej, tzn. wysyła ona żądanie (prymityw request).



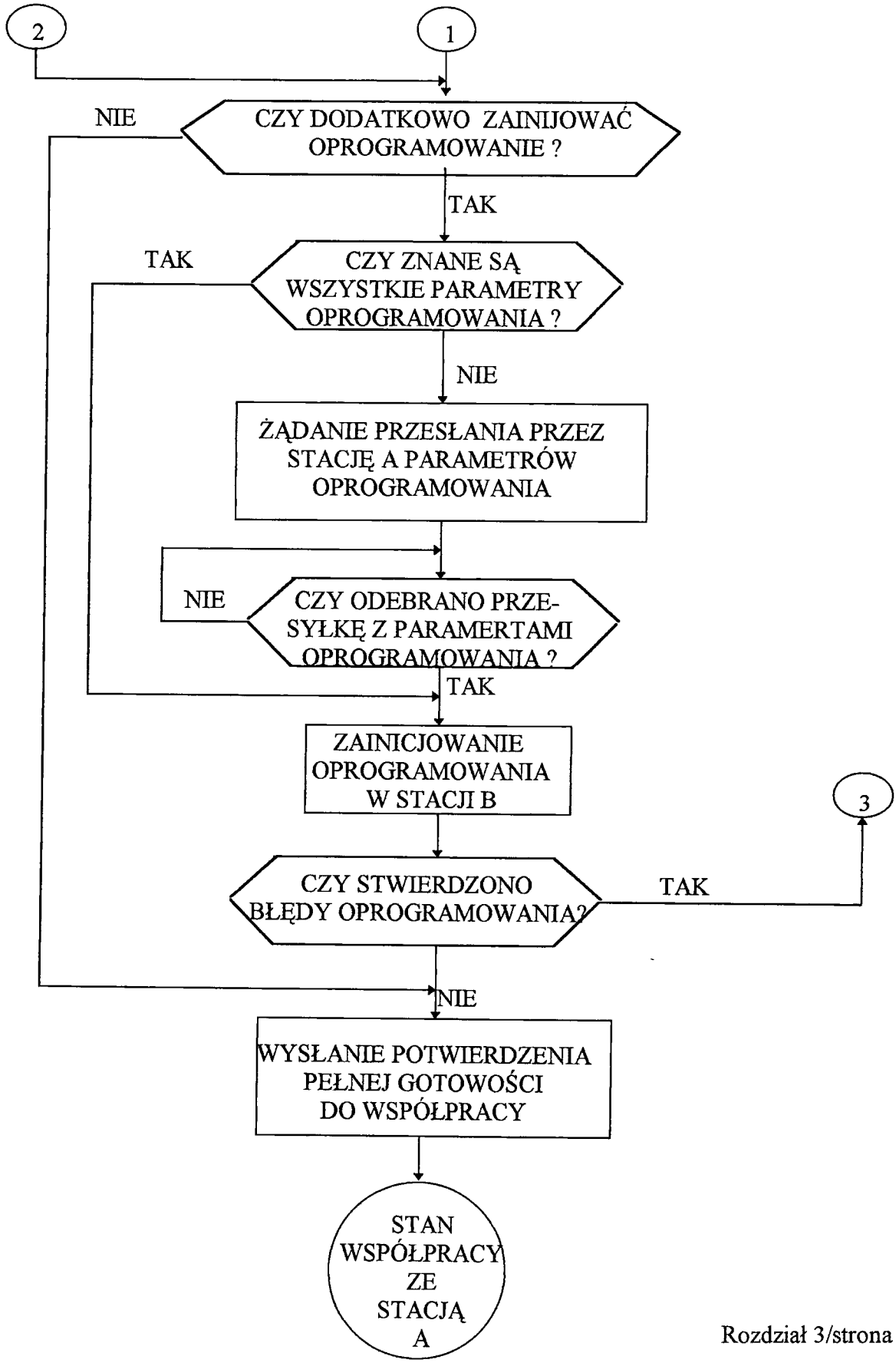
Rys. 3.17. (cz.1) Realizacja usługi Initiate przez stację A



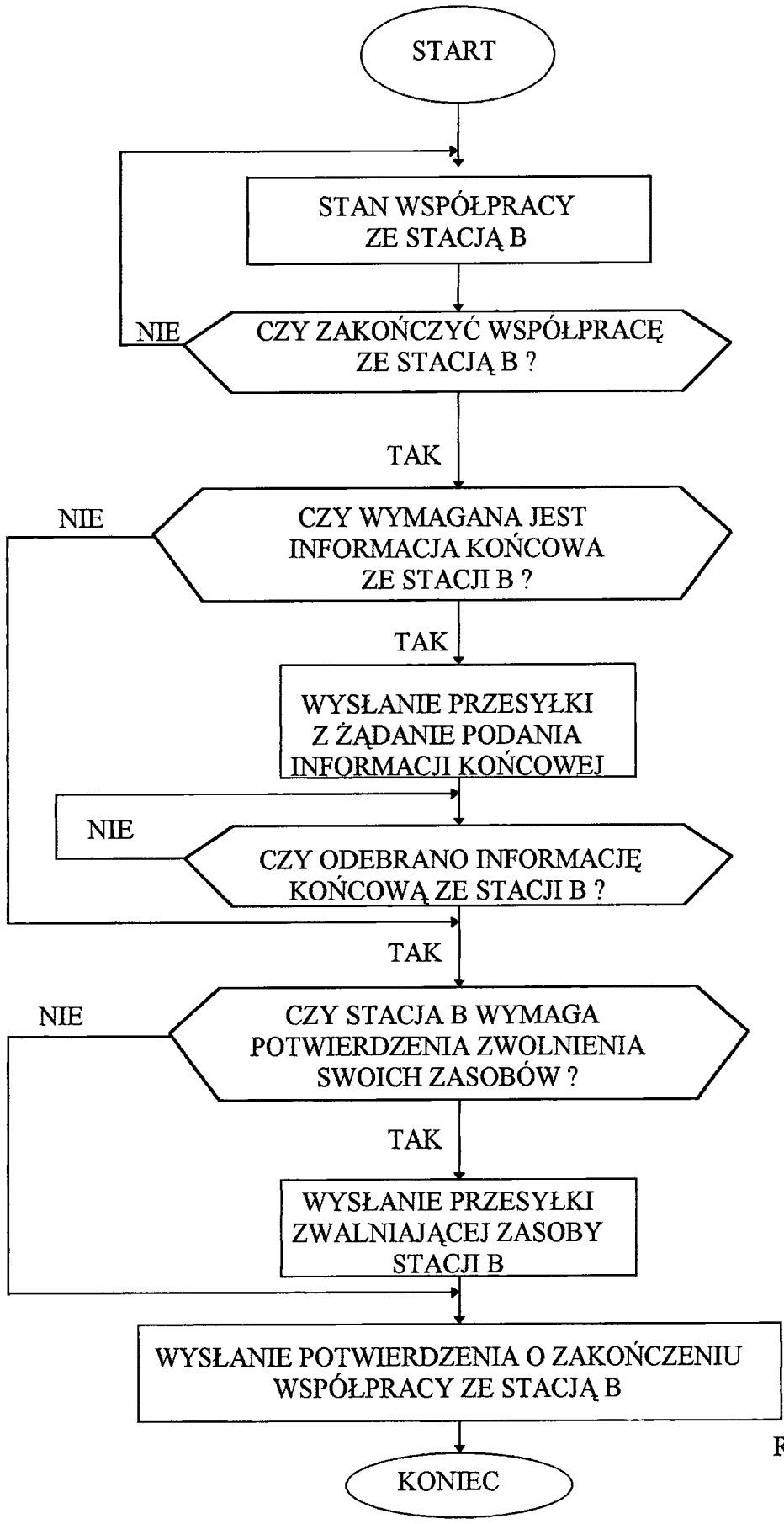
Rys. 3.17. (cz.2) Realizacja usługi Initiate przez stację A



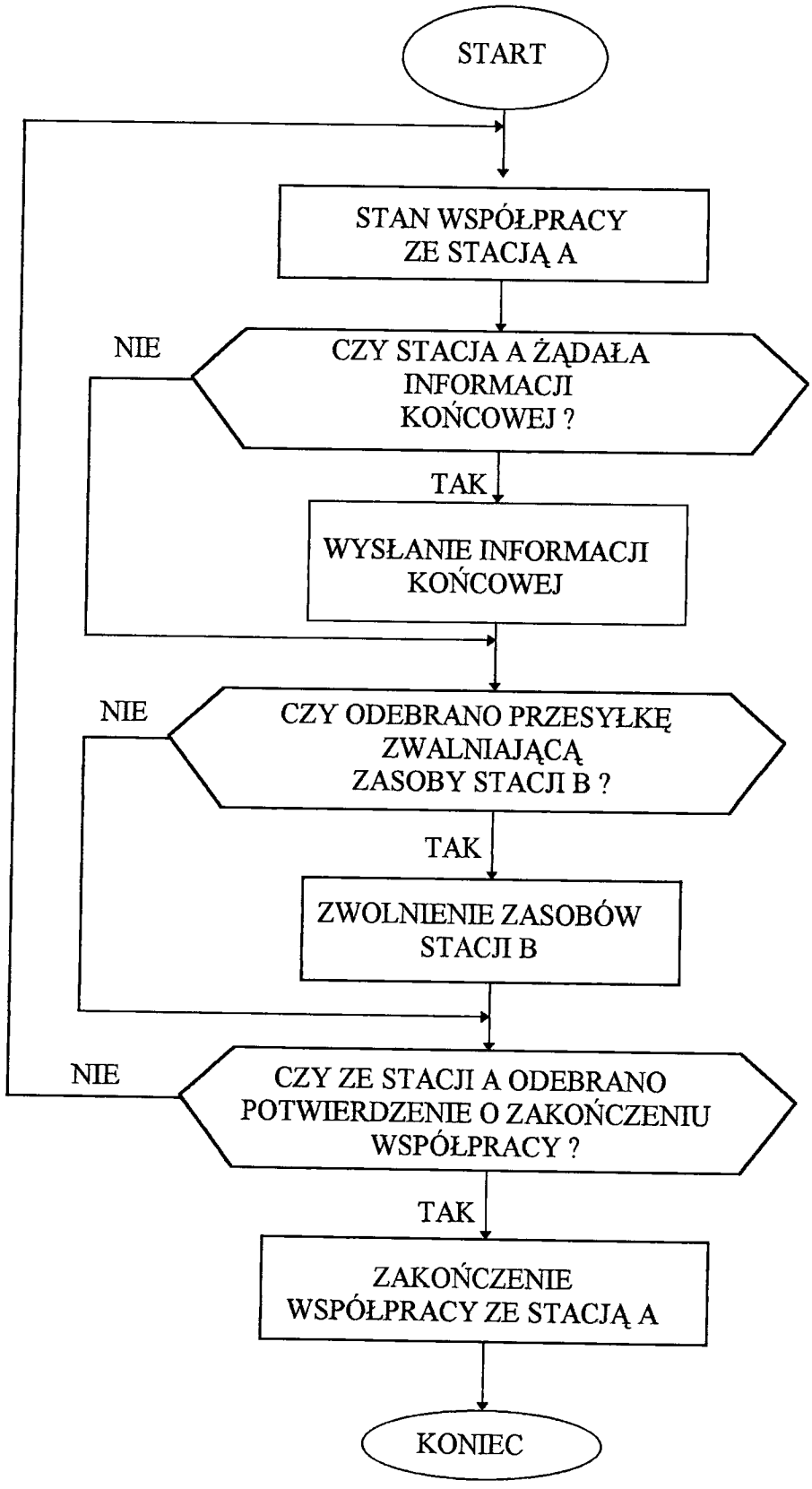
Rys. 3.18. (cz.1) Realizacja usługi Initiate przez stację B



Rys. 3.18. (cz.2) Realizacja usługi Initiate przez stację B



Rys. 3.19. Realizacja usługi Conclude przez stację A



Rys. 3.20. Realizacja usługi Conclude przez stację B

4. ZASTOSOWANIE METODY ZESPOŁÓW MINIMALNYCH DO (SUB)OPTYMALIZACJI MODELU VMD

W bieżącym rozdziale przez przekształcenia (sub)optymalizujące rozumie się (zgodnie z treścią załączników do umowy Nr PB 879/T11/95/08) dekompozycję zbioru usług MMS na podzbiory, na podstawie wzajemnych relacji zachodzących między tymi usługami. Z góry zakładano, że dekompozycji dokona się za pomocą metodyki uogólnionych zespołów minimalnych opisanych w rozdziale 2 tego sprawozdania. Metoda zespołów minimalnych rozbija graf na fragmenty (segmenty), biorąc za podstawę relacje zachodzące między podzbiorami zbioru wszystkich wierzchołków grafu. Otrzymana w wyniku jej zastosowania segmentacja posiada atrybut częściowego porządku (wzajemna rozłączność albo inkluzywność poszczególnych segmentów), ma więc charakter hierarchiczny [61].

Początkowo próbowano zastosować metodę zespołów minimalnych wprost do opisu strukturalnego (patrz p. 3.11). Jednakże z punktu widzenia zagadnień klasyfikacyjnych opis strukturalny okazał się mało przydatny. Przyczyną były istotne trudności powstające przy próbie oszacowania stopnia powiązania (np. podobieństwa - por. rozdział 2) dwu grafów przepływowych, a następnie wyrażenia go w sposób umożliwiający przetwarzanie przy użyciu komputera. Co prawda autorom znane są metody przekształcania grafów do postaci równoważnej (stosowane np. przy zagadnieniach związanych z tzw. problemem pokolorowania), jednakże są one po pierwsze nieefektywne obliczeniowo, po drugie zaś przy ich pomocy uzyskuje się jedynie odpowiedź na pytanie, czy rozpatrywane grafy są izomorficzne. W przypadku odpowiedzi negatywnej nie pozwalają one jednak na wyrażenie stopnia odmienności dwu grafów. Z tej też przyczyny zaproponowano innego rodzaju opis, który przedstawiono w p. 4.1.

4.1. Opis VMD podatny na obróbkę przy wykorzystaniu metody zespołów minimalnych

Jednym z celów tworzenia opisu strukturalnego było umożliwienie wyszukania w VMD części podobnych, typowych, a przez to uniknięcie w trakcie dalszych prac zbędnego powtarzania zabiegów programistycznych. Mając powyższe na uwadze skonstruowano model o czterech cechach. Wartości przyjmowane przez cechy dla poszczególnych usług MMS zgromadzono w kolumnach 4...7 tablicy 4.1. Mają one następujące znaczenie:

- cecha 1 - grupa usług zgodnie z podziałem na grupy określonym w [7], przy czym użyto oznaczeń kodowych zamieszczonych w tablicy 4.2;
- cecha 2 - Down - usługa związana z operacja wprowadzania domeny lub segmentu, Upl - usługa związana z operacja wyprowadzania domeny lub segmentu, ( ) - tzn. brak opisu - pozostałe przypadki;
- cecha 3 - Get - usługa związana z pobraniem obiektu, Del - usługa związana z usuwaniem obiektu, Def - usługa związana z definiowaniem obiektu, Rep - usługa związana z raportowaniem , ( ) - tzn. brak opisu - pozostałe przypadki;
- cecha 4 - usługa z potwierdzeniem (P) i bez potwierdzenia (N).

Tablica 4.1. Specyfikacja cech opisujących usługi MMS

Lp.	Nazwa usługi	MAP 4*/	Grupa usług	Ch-ka Wpro/ Wypro	Typ akcji	Usługa P/N
---	---	---	cecha 1	cecha 2	cecha 3	cecha 4
1	2	3	4	5	6	7
01.	Abort		A			N
02.	AcknowledgeEventNotification		H			P
03.	AlterEventConditionMonitoring		H			P
04.	AlterEventEnrollment		H			P
05.	AttachToEventCondition		H			N
06.	AttachToSemaphore		F			N
07.	Cancel	x	A			P
08.	Conclude	y	A			P
09.	CreateJournal		I			P
10.	CreateProgramInvocation	x	D			P
11.	DefineEventAction		H		Def	P
12.	DefineEventCondition		H		Def	P
13.	DefineEventEnrollment		H		Def	P
14.	DefineNamedType		E		Def	P
15.	DefineNamedVariable		E		Def	P
16.	DefineNamedVariableList		E		Def	P
17.	DefineScatteredAccess		E		Def	P
18.	DefineSemaphore		F		Def	P
19.	DeleteDomain	x	C		Del	P
20.	DeleteEventAction		H		Del	P
21.	DeleteEventCondition		H		Del	P
22.	DeleteEventEnrollment		H		Del	P
23.	DeleteJournal		I		Del	P
24.	DeleteNamedType		E		Del	P
25.	DeleteNamedVariableList		E		Del	P
26.	DeleteProgramInvocation	x	D		Del	P
27.	DeleteSemaphore		F		Del	P
28.	DeleteVariableAccess		E		Del	P
29.	DownloadSegment	A	C	Down		P
30.	EventNotification		H			N
31.	ExchangeData		K			P
32.	FileClose		J			P
33.	FileDelete		J			P
34.	FileDirectory		J			P
35.	FileOpen		J			P

Tablica 4.1.(cd.) Specyfikacja cech opisujących usługi MMS

Lp.	Nazwa usługi	MAP 4*/	Grupa usług	Ch-ka Wpro/ Wypro	Typ akcji	Usługa P/N
---	---	---	cecha 1	cecha 2	cecha 3	cecha 4
1	2	3	4	5	6	7
36.	FileRead		J			P
37.	FileRename		J			P
38.	GetAlarmEnrollmentSummary		H		Get	P
39.	GetAlarmSummary		H		Get	P
40.	GetCapabilityList	x	B		Get	P
41.	GetDataExchangeAttributes		K		Get	P
42.	GetDomainAttributes	x	C		Get	P
43.	GetEventActionAttributes		H		Get	P
44.	GetEventConditionAttributes		H		Get	P
45.	GetEventEnrollmentAttributes		H		Get	P
46.	GetNamedTypeAttributes		E		Get	P
47.	GetNamedVariableListAttributes		E		Get	P
48.	GetNameList	x	B		Get	P
49.	GetProgramInvocationAttributes	x	D		Get	P
50.	GetScatteredAccessAttributes		E		Get	P
51.	GetVariableAccessAttributes	x	E		Get	P
52.	Identify	x	B			P
54.	InformationReport	x	E	Rep		N
53.	InitializeJournal		I			P
55.	Initiate	y	A			P
56.	InitiateDownloadSequence	A	C	Down		P
57.	InitiateUploadSequence	A	C	Upl		P
58.	Input	x	G			P
59.	Kill		D			P
60.	LoadDomainContent	B	C			P
61.	ObtainFile	B	J			P
62.	Output	x	G			P
63.	Read	x	E			P
64.	ReadJournal		I			P
65.	Reject		A			N
66.	RelinquishControl	x	F			P
67.	Rename		B			P
68.	ReportEventActionStatus		H		Rep	P
69.	ReportEventConditionStatus		H		Rep	P
70.	ReportEventEnrollmentStatus		H		Rep	P

Tablica 4.1.(cd.) Specyfikacja cech opisujących usługi MMS

Lp.	Nazwa usługi	MAP 4*/	Grupa usług	Ch-ka Wpro/ Wypro	Typ akcji	Usługa P/N
---	---	---	cecha 1	cecha 2	cecha 3	cecha 4
1	2	3	4	5	6	7
71.	ReportJournalStatus		I		Rep	P
72.	ReportPoolSemaphoreStatus	x	F		Rep	P
73.	ReportSemaphoreEntryStatus	x	F		Rep	P
74.	ReportSemaphoreStatus	x	F		Rep	P
75.	RequestDomainDownload	A	C	Down		P
76.	RequestDomainUpload	A	C	Upl		P
77.	Reset	x	D			P
78.	Resume	x	D			P
79.	Start	x	D			P
80.	Status	x	B			P
81.	Stop	x	D			P
82.	StoreDomainContent	B	C			P
83.	TakeControl	x	F			P
84.	TerminateDownloadSequence	A	C	Down		P
85.	TerminateUploadSequence	A	C	Upl		P
86.	TriggerEvent		H			P
87.	UnsolicitedStatus	x	B			N
88.	UploadSegment	A	C	Upl		P
89.	Write	x	E			P
90.	WriteJournal		I			P

*/ Uwaga: MAP 4 - klasa zgodności 4 - usługi z punktu widzenia ich wykorzystania w robocie (por. z tabl. 3.3):

- x - usługa musi być stosowana w MMS robota,
- y - usługa musi być stosowana w MMS robota zarówno w sytuacji klient jak i w sytuacji serwer,
- A i B - w MMS robota muszą być uwzględnione albo wszystkie usługi oznaczone symbolem A, albo też wszystkie usługi oznaczone symbolem B,
- brak oznaczenia - usługa nie musi być stosowana w MMS robota.

Tablica 4.2. Grupy usług MMS i ich oznaczenia kodowe użyte w tabl. 4.1

Kod	Nazwa grupy usług	Nazwa grupy usług wg PN
A.	General Management Services	Usługi Zarządzania Ogólnego
B.	VMD Support Services	Usługi Stosowane w VMD
C.	Domain Management Services	Usługi Zarządzania Domeną
D.	Program Invocation Management Services	Usługi Zarządzania Wywołaniem Programu
E.	Variable Access Services	Usługi Dostępu do Zmiennej
F.	Semaphore Management Services	Usługi Zarządzania Semaforem
G.	Operator Communication Services	Usługi Komunikacji Operatorskiej
H.	Event Management Services	Usługi Zarządzania Zdarzeniem
I.	Journal Management Services	Usługi Zarządzania Dziennikiem
J.	File Management Services	Usługi Zarządzania Plikami
K.	Exchange Data Services	Usługi Wymiany Danych

Uwzględnienie cechy 1 zostało spowodowane używaniem odmiennego modelu opisu w każdej ze wspomnianych grup. Cecha 4 uwzględnia inny rodzaj przebiegu transakcji między stacjami sieci. Rozpatrując cechy 2 i 3 starano się wziąć pod uwagę spodziewane podobne sposoby realizacji każdej z wyróżnionych danym atrybutem grupy usług.

Wymienione wyżej cechy mają charakter tzw. cech nominalnych. Oznaczmy przez C^{m_k} zbiór ciągów m_k - elementowych o postaci $x_k = <x_{k_1}, x_{k_2}, ..., x_{k_m}>$, przy czym dokładnie jeden element x_{k_i}, $1 \leq i \leq m_k$, jest równy 1 (co odpowiada przybieraniu przez cechę wartości o numerze i), a pozostałe elementy przybierają wartość zero. Zatem w przypadku cechy nr 1 ($k = 1$) parametr m_k przybiera wartość 11, w przypadku cechy nr 2 parametr m_k przybiera wartość 3, w przypadku cechy nr 3 parametr m_k przybiera wartość 5, zaś w przypadku cechy nr 4 parametr m_k przybiera wartość 2. Stopień powiązania (o charakterze podobieństwa) obiektów x oraz y ze względu na k-tą cechę wynosi:

$$g_k(x, y) = \sum_{i=1}^{m_k} (1 - |x_{k_i} - y_{k_i}|) * \min(x_{k_i}, y_{k_i}) \qquad (4.1)$$

Wartość powiązania elementarnego obiektów x oraz y opisywanych jak podano wyżej przez l cech nominalnych (tutaj $l = 4$) wyznaczono stosując wzór:

$$g(\{x\}, \{y\}) = \sum_{i=1}^{l} w_i g_i(x, y), \qquad (4.2)$$

gdzie poszczególne $g_i(x, y)$ są dane wzorem (4.1), zaś w_i, $|w_i| \geq 0$, jest tak zwaną wagą lub stopniem preferencji i-tej cechy.

Wprowadzenie parametru w_i umożliwia zróżnicowanie wpływu poszczególnych cech obiektów na system powiązań. W szczególności $w_i = 0$ oznacza pominięcie i-tej cechy. W wariancie $w_i = 1$ dla

każdego i = 1, 2, ..., l zakłada się równoważność wszystkich cech. Zwykle stopnie preferencji cech dobiera się doświadczalnie, w procesie iteracyjnym typu eksperckiego. Mianowicie dokonuje się obliczeń dla różnych próbnych zestawów wartości w_i, i = 1, 2, ...,l, interpretuje się wyniki, ocenia się je, itd. aż do pozytywnego skutku.

W rozważanym modelu stopnie preferencji każdej z cech dobierano w procesie iteracyjnym. Przy wszystkich wagach równych jedności (przypadek cech równoważnych) nie można było w logiczny sposób zinterpretować wyników. Z uwagi na domniemaną rolę cechy nr 1 - sukcesywnie zwiększano odpowiadający jej stopień preferencji. Zmniejszano jednocześnie stopień preferencji przypisywany cesze 4. Spowodowane to było w zasadzie jednowartościowością tejże cechy. Zmniejszano także (chociaż znacznie wolniej niż w przypadku cechy 4) stopnie preferencji związane z cechami 2 i 3. W trakcie wielokrotnie powtarzanego eksperymentu obliczeniowego dobrano następujące wartości: $w_1 = 3{,}4$; $w_2 = 0{,}8$; $w_3 = 0{,}8$ oraz $w_4 = 0{,}1$.

Do obliczeń przyjęto model zespołów Max-minimalnych. Uzasadnienie takiego wyboru przedstawiono poniżej.

Podobnie jak w rozdziale 2 rozważymy system powiązań $\langle X, \mathbf{R}, \prec, g, g_0 \rangle$, w którym $\mathbf{R} = \mathbb{R}^+ \cup \{0\}$, relację porządkującą $\prec$ stanowi $\leq$ (większe lub równe), operacją # jest zwykłe dodawanie, $g_0 = 0$, a ponadto wartości funkcji g określone są następująco: $g(\{x\}, \{y\}) = w(x, y)$, $x \neq y$, $g(A, B) = \sum_{x \in A} \sum_{y \in B} w(x, y)$, gdzie A i B są niepustymi, wzajemnie rozłącznymi podzbiorami mnogości X.

Dla uproszczenia założymy również, że dla każdej pary $x, y \in X$, $x \neq y$, $w(x, y) = w$, tzn. wartości wzajemnych powiązań są stałe. Niech A, B i C będą niepustymi, wzajemnie rozłącznymi podzbiorami mnogości X. Zachodzi następująca zależność:

$$\max\{g(A, B), g(A, C)\} = |A| * \max\{|B|, |C|\} * w < |A|(|B| + |C|) * w = g(A, B \cup C)$$

co oznacza, że $B \cup C$ jest silniej powiązane z A niż samo B ewentualnie samo C. Wyprowadzona zależność świadczy o tym, że w przypadku zespołów Add - minimalnych ważne są nie tylko wartości wzajemnych powiązań elementarnych. Wręcz przeciwnie, o tym, czy dany podzbiór mnogości wszystkich obiektów stanowi zespół Add - minimalny (czy też nie stanowi) może decydować nie tylko siła wzajemnych powiązań, ale również i liczba obiektów w rozpatrywanym podzbiorze. Dlatego też w przypadkach badania powiązań typu podobieństwo między przedmiotami lub obiektami, co do których trudno jest wnosić o ich zależności funkcjonalnej wygodniej jest stosować wariant zespołów Max-minimalnych.

Dla następujących wartości: $w_1 = 3{,}4$; $w_2 = 0{,}8$; $w_3 = 0{,}8$ oraz $w_4 = 0{,}1$ otrzymano podział ostateczny na zespoły Max-minimalne w postaci (liczby są numerami usług MMS - kolumna 1 w tabl. 4.1):

$S_1 = \{01, 07, 08, 55, 65\}$,
$S_2 = \{40, 48, 52, 67, 80, 87\}$,
$S_3 = \{19, 29, 42, 56, 57, 60, 75, 76, 82, 84, 85, 88\}$,
$S_4 = \{10, 26, 49, 59, 77, 78, 79, 81\}$,
$S_5 = \{14, 15, 16, 17, 24, 25, 28, 46, 47, 50, 51, 54, 63, 89\}$,
$S_6 = \{06, 18, 27, 66, 72, 73, 74, 83\}$,
$S_7 = \{58, 62\}$,

S_8 = {02, 03, 04, 05, 11, 12, 13, 20, 21, 22, 30, 38, 39, 43, 44, 45, 68, 69, 70, 86},
S_9 = {09, 23, 53, 64, 71, 90},
S_{10} = {32, 33, 34, 35, 36, 37, 61},
S_{11} = {31, 41},
przy czym w fazach pośrednich uzyskano:
wewnątrz S_2 : {40, 48},
wewnątrz S_3 :{ 29, 56, 75, 84} i {57, 76, 85, 88},
wewnątrz S_5 : {14, 15, 16, 17}, {24, 25, 28} i {46, 47, 50, 51},
wewnątrz S_6 :{72, 73, 74}
wewnątrz S_8 : {11, 12, 13}, { 20, 21, 22}, {38, 39, 43, 44, 45} oraz {68, 69, 70}.

Uzyskane wyniki wykorzystano przy opracowaniu modelu szczegółowego VMD (patrz rozdział 5). Ze względu na specyfikę robota URP zdecydowano się zaimplementować w VMD usługi oznaczone w klasie zgodności MAP 4 (patrz kolumna 3 w tabl. 4.1) symbolami x, y oraz B.

5. MODEL (OPIS) SZCZEGÓŁOWY VMD

5.1. Realizacja komunikacji klient-serwer

Jak już wspomniano w p. 3.3 schemat działania klient - serwer realizowany jest przy użyciu prymitywów request, indication (ind), response (resp) i confirmation (conf). W przypadku prymitywów typu request i indication mamy do czynienia z takimi samymi danymi, przy czym w pierwszym przypadku są to dane wyjściowe, w drugim zaś - wejściowe. Podobnie jest dla pary prymitywów response (dane wyjściowe) i confirmation (dane wejściowe).

W dalszej części tego rozdziału, przy omawianiu funkcji realizujących poszczególne usługi MMS, posłużono się następującą konwencją:

- nazwy funkcji realizujących prymityw response będą kończyły się podkreśleniem i przyrostkiem resp, np. (w przypadku usługi Initiate) mv_init_resp;
- nazwy funkcji realizujących prymityw indication będą kończyły się podkreśleniem i przyrostkiem ind, np. u_init_ind;
- nazwy funkcji realizujących prymityw confirmation będą kończyły się podkreśleniem i przyrostkiem conf, np. u_mv_init_conf;
- nazwy funkcji realizujących prymityw request pozostaną bez dodatkowego podkreślenia i przyrostka, np. mv_init.

5.2. Podstawowe struktury danych wykorzystywane w modelu VMD

Z punktu widzenia przenoszalności (portability) oprogramowania wygodnie jest wprowadzić następujące definicje typów i stałych:

```
#define BOOLEAN    unsigned char    /* Two valued (TRUE/FALSE)*/
#define DEFAULT    int              /* machine-dependent int value*/
#define BYTE       char             /* Signed byte (8 bits)      */
#define UBYTE      unsigned char    /* Usigned byte              */
#define SHORT      int              /* Signed word (16 bits)     */
#define USHORT     unsigned int     /* Unsigned word             */
#define LONG       long             /* Signed long (32 bits)     */
#define ULONG      unsigned long    /* Usigned lone              */
#define DOUBLE     double           /* 8 bytes                   */
#define FLOAT      float            /* 4 bytes                   */
#define VOID       void             /* Void function return      */
#define TRUE       1                /* Function TRUE value       */
#define FALSE      0                /* Function FALSE value      */
#define SUCCESS    0
#define FAILURE    1
#define NO_ERROR   0
```

5.2.1. Zmienne konfiguracyjne

Poniżej zestawiono zmienne istotne dla konfiguracji środowiska MMS. Standardowo zmiennym tym są nadawane wartości domniemane (stałe języka C).

1) Maksymalna liczba kanałów
   SHORT max_mms_chan = DEFAULT_NUM_MMS_CHAN;
2) Maksymalna liczba na wszystkich kanałach nieobsłużonych żądań Request

SHORT max_req_pend = 16;

3) Maksymalna liczba na wszystkich kanałach nieobsłużonych wskazań Indication
   SHORT max_ind_pend = 16;
4) Maksymalna liczba plików otwartych przez urządzenie wirtualne w lokalnym systemie
   SHORT max_loc_open = MAXFILESOPEN;
5) Maksymalna liczba plików otwartych przez urządzenie wirtualne w zdalnym systemie
   SHORT max_rem_open = MAXFILESOPEN;
6) Maksymalna liczba nazwanych typów MMS jaką urządzenie wirtualne może zdefiniować w całym środowisku MMS
   SHORT max_mmsease_types = 50;
7) Maksymalna liczba nazwanych zmiennych MMS jaką urządzenie wirtualne może zdefiniować
   SHORT max_mmsease_vars = 50;
8) Maksymalna liczba nazwanych domen jaką urządzenie wirtualne może zdefiniować
   SHORT max_mmsease_doms = 20
9) Maksymalny rozmiar komunikatu PDU
   SHORT mms_max_msgsize = DEFAULT_MAX_MSGSIZE;

5.2.2. Struktury danych urządzenia wirtualnego

Poniższa struktura jest wykorzystywana jako element wielu innych struktur do tworzenia zagnieżdżonych struktur w postaci list pierścieniowych.

```
struct dbl_lnk
 {
 struct  dbl_lnk *next;
 struct  dbl_lnk *prev ;
 };
```

Struktura informacyjna kanału

Kanał komunikacyjny reprezentuje połączenie pomiędzy dwiema aplikacjami. Każdej aplikacji przydzielony jest dokładnie jeden kanał. Kanały są identyfikowane za pomocą numerów od 0 do max_mms_chan - 1.

Struktury danych opisujące kanały są zorganizowane w postaci tablicy mms_chan_info[] struktur indeksowanej numerem kanału.

```
struct mchaninfo
        {
                struct
                        {
                        SHORT           llp_type;
                        ULONG           chan_state;
                        SHORT           xmit_pend;
                        USHORT      contexts;
                        BYTE        local_ar[MAX_AR_LEN+1];
                        BYTE        rem_ar[MAX_AR_LEN+1];
                        } ctxt;
                struct
                        {
```

```
                struct          vmd_ctrl        *vmd;
                struct          domain_objs     *aa_objs;
                } objs;
        SHORT       version;
        LONG            segsize;
        SHORT       maxpend_req;
        SHORT       maxpend_ind;
        BYTE        max_nest;
        UBYTE       param_supp[2];
        UBYTE       service_req[11];
        UBYTE       service_resp[11];
        USHORT      file_blk_size;
        USHORT      download_blk_size;
        BYTE        *user_info;
    };
struct mchaninfo    *mms_chan_info;
```

gdzie:
llp_type - rodzaj usług niższego poziomu; aktualnie = ACSE30_LLP,
chan_state - aktualny stan kanału,
xmit_pend - licznik komunikatów wysłanych w sieć i oczekujących na obsłużenie,
contexts - konteksty prezentacji wynegocjowane dla danego kanału,
ctxt.local_ar - nazwa lokalnej aplikacji zarejestrowanej w kanale,
ctxt.rem_ar - nazwa zdalnej aplikacji, z którą zostało nawiązane połączenie,
vmd - wskazanie na strukturę kontrolną VMD typu vmd_ctrl odpowiadającą urządzeniu wirtualnemu będącemu serwerem w danym kanale,
aa_objs - wskazanie na strukturę obiektów domenowych typu domain_objs zawierającą listę obiektów zdefiniowanych dla danego kanału,
version- wersja standardu MMS wynegocjowana dla danego kanału (0 -DIS, 1 - IS),
seg_size - maksymalny rozmiar segmentu (jednostki PDU),
maxpend_req - maksymalna liczba wysłanych żądań Request będących w trakcie obsługi,
maxpend_ind - maksymalna liczba wskazań Indication wysłanych przez zdalny węzeł będących w trakcie obsługi,
max_nest - maksymalna liczba poziomów zagnieżdżania struktur danych,
param_supp - jakie opcje związane z obsługą parametrów są realizowane w kanale,
service_req - jakie usługi mają być wspierane w kanale (żądanie zdalnego węzła)
service_resp - jakie usługi mają być wspierane w kanale (odpowiedź lokalnego węzła),
file_blk_size - maksymalna liczba bajtów jaka może być wysłana/odebrana w jednym segmencie w trakcie operacji File Read PDU. Jest obliczany w oparciu o seg_size,
download_blk_size - maksymalna liczba bajtów jaka może być załadowana w jednym segmencie danych w trakcie operacji Download Segment PDU. Obliczany w oparciu o seg_size,
user_info - wskazanie na dane użytkownika, które może on związać z kanałem.

Struktury zarządzania danymi

Struktura definicji typów

Do definiowania typów zmiennej nazwanej służy struktura named_type. Struktura zawiera nazwę definiowanego typu oraz szereg jego atrybutów. Struktura ta jest pośrednio powiązana

ze strukturą named_var. Jest ona wykorzystywana zarówno do definicji zmiennych lokalnych jak i zmiennych zdalnych.

```
struct named_type
        {
        struct      dbl_lnk *link;
        BYTE        typename[MAX_IDENT_LEN+1];
        BOOLEAN     deletable;
        BYTE        protection;
        BOOLEAN     erased;
        SHORT       nref;
        struct      runtime_type *rt_head;
        SHORT       rt_num;
        SHORT       asn1len;
        BYTE        *asn1ptr;
        SHORT       blocked_len;
        BYTE        *(*read_ind_fun)(BYTE *,SHORT);
        SHORT       (*write_ind_fun)(BYTE *,BYTE *,SHORT);
        };
```

gdzie
typename - nazwa typu,
deletable - czy dany typ może być skasowany zdalnie,
protection - kontrola dostępu do zmiennych danego typu przez zdalne aplikacje,
erased - wartość TRUE oznacza, że dany typ został skasowany. Może on jednak nadal istnieć w sieci o ile zmienna tego typu istnieje (nref $\neq$0),
nref - liczba nazwanych zmiennych zdefiniowanych wg. danego typu,
asn1len - długość (w bajtach) definicji typu w notacji ASN.1,
asn1ptr- wskazanie na definicję typu w notacji ASN.1,
blocked_len - długość (w bajtach) lokalnej reprezentacji zmiennej,
(*read_ind_fun) (*write_ind_fun) - są to wskazania na funkcje służące do czytania/zapisu zmiennych danego typu.

Specyfikacja typów MMS

Urządzenie wirtualne na podstawie definicji typów ASN.1 tworzy rzeczywiste definicje typów MMS na czas wykonywania programu (konwersja na format wewnętrzny).

Struktury danych zmiennych nazwanych

Do definiowania zmiennych nazwanych służy struktura named_var. Jest ona wykorzystywana tylko do opisu zmiennych lokalnych.

```
struct      named_var
        {
        struct      dbl_lnk     link;
        BYTE        varname[MAX_IDENT_LEN+1];
        BOOLEAN     invalid;
        struct      named_type  *type;
        BOOLEAN     deletable;
        BYTE        rd_pro;
        BYTE        wr_pro;
```

```
        struct      var_acc_addr  addr;
        BYTE        *(*read_ind_fun)    (BYTE       *,SHORT);
        SHORT       (*write_ind_fun)    (BYTE       *,BYTE*,SHORT);
        };
```

gdzie
varname - nazwa zmiennej,
invalid - wartość TRUE oznacza, że zmienna jest niepoprawna (adres, zmienna lub definicja typu są niepoprawne lub zostały skasowane),
type - wskazanie na strukturę named_type zawierającą definicję typu,
deletable - wartość TRUE oznacza, że zmienna może zostać skasowana z sieci,
rd_pro - przywilej kontrolowania dostępu odczytu do zmiennej, aktualnie nie wykorzystywana,
wr_pro- przywilej kontrolowania dostępu zapisu do zmiennej, aktualnie nie wykorzystywana,
addr - zawiera specyfikację adresową zmiennej (umieszczenie tu adresu oznacza pokazanie go innym węzłom sieci). Nie musi to być aktualny fizyczny adres zmiennej.
(*read_ind_fun) oraz (*write_ind_fun) podczas nazywania zmiennej wpisywane są tu wartości wzięte z odpowiedniej struktury named_type.

Struktury list danych

Jest to struktura używana przez urządzenie wirtualne do operowania na listach zmiennych zdalnych. Struktury te zawierają aktualną listę zmiennych nazwanych.

```
struct      mv_vardesc
        {
        struct      object_name   name;
        struct      object_name   type;
        BYTE        *data;
        };
```

gdzie:
name - zawiera nazwę zdalnej zmiennej, która ma być odczytana/zapisana,
type - zawiera nazwę lokalnie zdefiniowanego nazwanego typu odpowiadającego typowi zmiennej w węźle zdalnym,
data - wskazanie adresu w pamięci lokalnej dokąd ma być skierowany wynik żądania odczytu lub skąd mają być pobrane dane w przypadku operacji zapisu.

Struktury obiektów typu domena

Stacja przechowuje alfabetyczną listę nazwanych domen dla każdego zdefiniowanego urządzenia wirtualnego. Poniższa struktura jest używana do definiowania różnych obiektów MMS jak zmienne, typy, zdarzenia, semafory, które mogą być elementami domen. Struktura zwraca wskazanie na listy różnych obiektów zawartych w domenie.

```
struct      domain_objs
        {
            struct  named_var     *var_list;
            struct  named_var_list      *var_list_list;
            struct  named_type    *type_list;
            struct  scat_acc            *scat_acc_list;
            struct  sem           *sem_list;
```

```
        struct  event_cond      *ev_cond_list;
        struct  event_action    *ev_act_list;
        struct  journal         *jour_list;
        BYTE rsrvd[8];
    };
```

gdzie:
var_list - wskazanie na listę struktur zawierających definicje wszystkich nazwanych zmiennych związanych z domeną,
var_list_list - wskazanie na listę struktur zawierających definicje wszystkich nazwanych list zmiennych związanych z domeną,
type_list - j.w. dla nazwanych typów,
scat_acc_list - j.w. dla zmiennych o dostępie rozproszonym,
sem_list - j.w. dla semaforów,
ev_cond_list - j.w. dla warunkowych zdarzeń,
ev_act_list - j.w. dla akcji warunkowych,
jour_list - j.w. dla dzienników.

Struktura kontrolna nazwanych domen

Poniższa struktura służy do definiowania nazwanej domeny. Zawiera ona wskazania na obiekty MMS zawarte w danej domenie. Struktura taka jest traktowana jako element listy nazwanych domen.

```
struct      named_dom_ctrl
    {
        struct      dbl_link        link;
        BYTE        dom_name[MAX_IDENT_LEN+1];
        BYTE        protection;
        BOOLEAN     deletable;
        BOOLEAN     sharable;
        BYTE        state;
        BOOLEAN     upl_in_prog;
        LONG        octet_pos;
        SHORT           npi;
        BYTE        reserved[4];
        BYTE        user_rsrvd[8];
        BYTE        *dom_content;
        struct      domain_objs   objs;
        BOOLEAN     dl_detail_pres;
        SHORT           dl_detail_len;
        UBYTE           *dl_detail;
        SHORT           num_of_capab;
        /* BYTE     *capab_list[num_of_capab];      */
    };
```

gdzie
dom_name - nazwa domeny,
protection - kod protekcji, aktualnie nie używany,
deletable - wartość TRUE oznacza, że domena może być skasowana przez zdalną aplikację,
sharable - wartość TRUE oznacza możliwość dostępu przez kilka inwokacji programów,

state - określa aktualny status domeny:
DOM_NONEXISTENT(0)
DOM_LOADING(1)
DOM_READY(2)
DOM_IN_USE(3)
DOM_COMPLETE(4)
DOM_INCOMPLETE(5)
upl_in_progr - domena jest w trakcie ładowania (TRUE),
octet_pos - liczba załadowanych oktetów,
npi - licznik zawierający liczbę inwokacji programów odwołujących się do domeny,
user_rsrvd - informacje użytkownika,
dom_content - wskazanie przeznaczone do ewentualnego specyficznego wykorzystania przez aplikację,
objs - zawiera listę nazwanych obiektów związanych z domeną,
dl_detail_pres = 0 wskazanie dl_detail nie może być włączone do PDU,
≠0 włączenie wskazania dl_detail do PDU,
dl_detail_len - długość (w bajtach) danych wskazywanych przez dl_detail,
dl_detail - wskazanie na dane ładowane, które muszą być w formacie ASN.1,
num_of_capab- liczba łańcuchów znakowych wskazywanych przez elementy capab_list,
capab_list - tablica wskazań bajtowych na specyficzne dla implementacji łańcuchy znakowe opisujące możliwości VMD. Są one częścią ładowanej domeny.

Struktura kontrolna urządzenia wirtualnego

Poniższa struktura danych jest używana przez system zarządzania urządzenia wirtualnego. Dla każdego zdefiniowanego urządzenia wirtualnego istnieje jedna taka struktura.

```
struct      vmd_ctrl
    {
        struct  dbl_lnk             link;
        struct  domain_objs  vmd_wide;
        struct  named_dom_ctrl      *dom_list;
        struct  prog_inv     *prog_inv_list;
        struct  oper_stat    *op_stat_list;
        struct  ae_ctrl      *ae_list;
        BYTE *user_info;
    };
```

gdzie
vmd_wide - zawiera listę obiektów związanych z danym VMD,
dom_list - wskazanie na początek listy domen zdefiniowanych dla danego VMD,
prog_inv_list - wskazanie na początek listy inwokacji programów zdefiniowanych dla danego VMD,
op_stat_list - wskazanie na początek listy stacji operatorskich zdefiniowanych dla danego VMD,
ae_list - wskazanie na początek listy inwokacji AE zdefiniowanych dla danego VMD,
user_info - wskazanie na informacje użytkownika, które może on związać z danym VMD.

Struktura kontrolna inwokacji programu

Poniższa struktura służy do śledzenia inwokacji programów.

```
struct      prog_inv
    {
        struct      dbl_link            link;
        BYTE        pi_name[MAX_IDENT_LEN+1];
        BYTE        protection;
        BYTE        state;
        BOOLEAN     deletable;
        BOOLEAN     reusable;
        BOOLEAN     monitor;
        BYTE        *start_arg;
        SHORT           start_len;
        SHORT           ndom;
        BYTE        reserved[4];
        BYTE        user_rsrvd[8];
        /* struct           named_dom_ctrl      *dom_list[];   */
    };
```

gdzie
pi_name - nazwa inwokacji programu,
protection - kontrola dostępu do inwokacji (w powiązaniu z przywilejem nadanym zdalnej jednostce komunikacyjnej),
state - status inwokacji programu:
PI_NON_EXISTENT(0)
PI_UNRUNNABLE(1)
PI_IDLE(2)
PI_RUNNING(3)
PI_STOPPED(4)
PI_STARTED(5)
PI_STOPPING(6)
PI_RESUMING(7)
PI_RESETTING(8)
deletable - TRUE oznacza, że inwokacja może być skasowana z sieci,
reusable - TRUE oznacza, że inwokacja może być zresetowana i uruchomiona od początku,
monitor - TRUE oznacza, że urządzenie wirtualne powinno informować host o statusie tej inwokacji programu (za pomocą usługi Event Notification),
start_arg - wskazuje na argument startowy ostatnio wydanego żądania Start,
start_len - długość (w bajtach) łańcucha start_arg (włącznie z terminatorem zero),
ndom - liczba domen obecnych w inwokacji programu,
user_rsrvd - dane definiowane przez użytkownika dla dowolnego wykorzystania,
dom_list - tablica wskazań na struktury domenowe typu named_dom_ctrl, do których odwołuje się dana inwokacja.

Struktura raportu informacyjnego

Poniższa struktura opisuje jedną zmienną.

```
struct      mv_info_req_info
    {
        BYTE        *data_ptr;
        struct      named_type   *type;
        BOOLEAN     alt_acc_pres;
        BOOLEAN     alt_acc_data_packed;
        struct      alternate_access      alt_acc;
    };
```

gdzie:
data_ptr - wskazanie na dane w lokalnym formacie utworzone w wyniku pozytywnego wykonania żądania raportu,
type - struktura zawierająca opis typu danej zmiennej,
alt_acc_pres - TRUE - alt_acc jest obecny,
alt_acc_data_packed - TRUE - dane w alt_acc są spakowane,
alt_acc - struktura zawierająca opis alternatywnego dostępu

Struktura nazwy obiektu

Jest to uniwersalna struktura służąca do specyfikacji nazw różnych obiektów jak typ, zmienna, semafor itp.

```
struct      object_name
    {
        SHORT       object_tag;
        union
        {
            BYTE        vmd_spec[MAX_IDENT_LEN+1];
            BYTE        item_spec[MAX_IDENT_LEN+1];
            BYTE        aa_spec[MAX_IDENT_LEN+1];
        }obj_name;
        BYTE        domain_id[MAX_IDENT_LEN+1];
    };

object_tag=  VMD_SPEC(0)
             DOM_SPEC(1)
             AA_SPEC(2)
```

5.2.3. Struktury danych do śledzenia Request i Indication

Po wywołaniu funkcji typu Request system zwraca do programu użytkownika wskazanie na strukturę pozwalającą śledzić wykonanie żądania. Struktura jest aktywna aż do odebrania potwierdzenia na wysłane żądanie. Podobnie jest dla funkcji typu Indication.

Struktura dla Request - MMSREQ_PEND

Jest to struktura tworzona przy wysyłaniu żądania i pozostaje aktywna aż do odbioru i obsługi potwierdzenia

```
typedefstruct        rqdat
        {
        struct       dbl_lnk        link:
        SHORT        pri;
        SHORT        chan;
        SHORT        llp;
        USHORT       context;
        USHORT       op;
        ULONG        id;
        SHORT        resp_err;
        SHORT        cancl_state;
        LONG                req_time;
        BOOLEAN      req_info_pres;
        BYTE         *req_info_ptr;
        struct       list_of_mods   mods;
        LONG                resp_time;
        BOOLEAN      resp_info_pres;
        BYTE         *resp_info_ptr;
        SHORT        *rxbuf;
        VOID         (*resp_rcvd_fun)();
        struct       csi            cs;
        struct       cmd_service    s;
        } MMSREQ_PEND;
```

gdzie:
pri- priorytet żądania; aktualnie niewykorzystywany.
chan - numer kanału, w którym wysłano żądanie.
llp - rodzaj usługi warstw niższych (LLP); aktualnie dostępna jest MAP V3.0 ACSE (ACSE30_LLP).
context - kontekst prezentacyjny, w jakim wysłano żądanie.
op - kod operacyjny żądania (dla celów systemowych).
id - InvokeID żądania
resp_err - kod błędu wskazujący czy odebrano poprawne potwierdzenie. Może być:
RESP_OK - odebrano poprawną odpowiedź,
PARSE_ERR - wykryto błąd w trakcie lokalnej analizy danych,
REJ_ERR - żądanie zostało odrzucone przez partnera,
ERR_OK - odebrano odpowiedź będącą błędem,
DISCONNECTED - połączenie rozłączone lub zerwane,
CHAN_OP_ERR - niepoprawna operacja dla invoke id
CANST_ERR - błąd skasowania stanu,
ASS_REQ_REJECTED - żądanie inicjacji odrzucone,
ASS_RESP_PARAM - odebrano niepoprawne potwierdzenie inicjacji,
ASS_USER_REJ_CONF - potw. inicjacji odrzucone przez użytkownika
cancl_state - aktualny stan operacji kasowania:
CANCEL_REJECTED (-2) - próba kasowania; wystąpiło odrzucenie - stan nieznany,

CANCEL_FAILED (-1) - nieudana próba kasowania; powrót do stanu no-cancel
NO_CANCEL (0) - nie podjęto próby kasowania,
CANCEL_REQUESTED (1) - wysłano żądanie kasowania, brak odpowiedzi,
POS_CANCEL_RCVD (2) - odebrano (+) potwierdzenie kasowania, ale nie odebrano jeszcze (-) potwierdzenia usługi,
POS_CANCEL_DUE (3) - odebrano (-) potwierdzenie usługi kasowania, ale nie odebrano (+) potwierdzenia kasowania,
NEG_CANCEL_DUE (4) - z powodów innych niż kasowanie odebrano potw. usługi (+) lub potw. usługi (-), ale nieodebrano potw. kasowania (-),
CANCELED (5) - odebrano obie usługi potw. usługi (-) i potw. kasowania

req_time - czas wysłania żądania,
req_info_pres = 0 - brak dodatkowej informacji w PDU,
≠0 - jest dodatkowa informacja wskazywana przez req_info_ptr,
req_info_ptr - wskazuje na dodatkowe informacje zawarte w PDU,
mods - struktura typu list_of_mods zawierająca informacje o modyfikatorze, jaki mógł być wysłany w żądaniu
resp_time - czas odebrania potwierdzenia,
resp_info_pres = 0 - brak dodatkowych informacji zawartych w PDU,
≠0 - jest dodatkowa informacja w PDU wskazywana przez resp_info_ptr,
rxbuf_type - dla użytku wewnętrznego,
rxbuf - dla użytku wewnętrznego,
cs - struktura typu csi; zawiera informacje związane ze standardem firmowym,
s - struktura typu cmd_service; zawiera dodatkowe informacje, podane przez użytkownika

Struktura dla Indication - MMSREQ_IND

```
typedefstruct  rspdat
        {
        struct          dbl_lnk         link;
        SHORT           pri;
        SHORT           chan;
        SHORT           llp;
        USHORT          context;
        USHORT          op;
        ULONG           id;
        BOOLEAN         req_info_pres;
        BYTE        *req_info_ptr;
        struct          list_of_mods    mods;
        SHORT           cancl_state;
        LONG            ind_time;
        SHORT           rxbuf_type;
        BYTE        *rxbuf;
        struct          cmd_service     s;
        UBYTE           add_addr_info_pres;
        struct          llp_addr_info   *add_addr_info;
        struct          csi                     cs;
        } MMSREQ_IND;
```

Znaczenie większości parametrów jest analogiczne jak dla wyżej opisanej struktury MMSREQ_PEND za wyjątkiem:

cancl_state - stan aktualny operacji kasowania dotyczącej wskazania:
CANCEL_FAILED (-1) - odmowa kasowania; powrót do stanu no-cancel,
NO_CANCEL (0) - no-cancel,
CANCELING (1) - odebrano wskazanie kasowania, nie wysłano odpowiedzi,
POS_CANCEL_SENT (2) - wysłano Odpowiedź Kasowania (+), ale jeszcze nie wysłano odpowiedzi usługi (-),
add_addr_info_pres = 0 PDU nie zawiera specyficznej dla LLC informacji adresowej
≠0 PDU zawiera informacje adresowe specyficzne dla LLC wskazywane przez add_addr_info

Struktura cmd_service

Jest to struktura będąca elementem zarówno MMSREQ_PEND jak i MMSREQ_IND. Zawiera informacje specyficzne dla użytkownika dotyczące usług żądania/potwierdzenia. Jest przeznaczona dla programu użytkownika.

```
struct  cmd_service
        {
        SHORT       (*serve_fun)();
        BYTE        *cmd_info;
        LONG                cmd_stat;
        SHORT       aux_flags[6];
        };
```

serve_fun() - to wskazanie na funkcję dotyczy właśnie wywoływanej, odpowiedniej funkcji obsługi,
cmd_info, cmd_stat, aux_flags zmienne zawierające (dowolne) informacje dodatkowe.

5.3. Funkcje realizujące usługi MMS w modelu VMD robota

5.3.1. Usługi zarządzania ogólnego

Usługa Abort (Przerwanie)

SHORT	mp_abort(SHORT chan, SHORT reason) - przerywa (kończy) przypisanie do kanału; SHORT chan - nr kanału; SHORT reason - ignorowane w przypadku ACSE.
VOID	u_abort_ind(SHORT chan, SHORT reason, SHORT au_flag) SHORT chan - numer kanału z którego właśnie został otrzymany sygnał zaniechania; SHORT reason - kody zaniechania wg. specyfikacji MAP. SHORT au_flag - źródło zaniechania;

Usługa Conclude (Zakończenie)

MMSREQ_PEND *	mp_conclude(SHORT chan) - przesyła PDU zlecenia Conclude w celu pozytywnego zakończenia przypisania do kanału; SHORT chan - numer kanału, którego przypisanie zakończono.
SHORT	mp_conclude_resp(struct MMSREQ_IND *req_ptr) - przesyła PDU z (pozytywnym) potwierdzeniem wykonania zlecenia Conclude; struct MMSREQ_IND *req_ptr - wskazanie na strukturę będącą argumentem funkcji u_conclude_ind.
VOID	u_conclude_ind(MMSREQ_PEND *req_ptr)
VOID	u_mp_conclude_conf(MMSREQ_PEND *)

Usługa Initiate (Inicjacja)

MMSREQ_PEND *	mv_init(SHORT chan, BYTE *partner) - wykonanie usługi Initiate przez odwołanie się do VMD; SHORT chan - kanał, który posłuży do wykonania usługi; BYTE *partner - wskazanie na nazwę (AR Name) procesu.
SHORT	mv_init_resp(struct MMSREQ_IND *req_ptr) - przesłanie potwierdzenia na odpowiedź o wykonaniu usługi Initiate przez odwołanie się do VMD; struct MMSREQ_IND *req_ptr - wskazanie na strukturę uprzednio przekazaną w argumencie funkcji u_init_ind.
VOID	u_init_ind(MMSREQ_IND *)
VOID	u_mv_init_conf(MMSREQ_PEND *)

Usługa Reject (Wybrakowanie)

SHORT	mp_reject_ind(MMSREQ_IND *req_ptr, struct reject_resp_info *info_ptr)
	- przesyła PDU zlecenia Reject zamiast pozytywnego lub negatywnego potwierdzenia; MMSREQ_IND *req_ptr - wskazanie na strukturę odpowiadającą zleceniu, którego wykonanie ma być zaniechane; struct reject_resp_info *info_ptr - wskazanie na strukturę, która zawiera informacje szczegółowe dla PDU zlecenia Reject.
VOID	u_reject_ind(SHORT chan, struct reject_resp_info *info_ptr) SHORT chan - numer kanału, w którym otrzymano Reject; struct reject_resp_info *info_ptr - wskazanie na strukturę zawierającą informacje wyszczególnione w otrzymanym PDU dla Reject.

```
struct reject_resp_info
 {
 UBYTE detected_here;      /* flag indicating where error occurred */
 UBYTE invoke_known;           /* flag indicating if invoke ID known*/
 ULONG invoke;             /* invoke ID of service rejected      */
 USHORT pdu_type;          /* pdu type of service rejected       */
 SHORT rej_class;          /* reject class                 */
 SHORT rej_code;           /* reject code                        */
 };
```

Usługa Cancel (Kasowanie)

MMSREQ_PEND *	mp_cancel(struct MMSREQ_PEND *req_ptr) - przesyła PDU zlecenia Cancel, dla odwołania uprzednio wysłanego zlecenia;
SHORT	mp_cancel_resp(struct MMSREQ_IND *req_ptr) - przesyła PDU z (pozytywnym) potwierdzeniem wykonania zlecenia Cancel; struct MMSREQ_IND *req_ptr - wskazanie na strukturę będącą argumentem funkcji u_cancel_ind.
VOID	u_cancel_ind(MMSREQ_IND *req_ptr)
VOID	u_mp_cancel_conf(MMSREQ_PEND *req_ptr, UBYTE errdata_pres, struct err_info *err_ptr). - służy do przetwarzania informacji zwracanej przez zlecenie Cancel; MMSREQ_PEND *req_ptr - wskazanie na strukturę zwróconą w wyniku wykonania uprzednio wywołanej funkcji mp_cancel; UBYTE errdata_pres - numer błedu powstały przy realizacji zlecenia Cancel (w przypadku braku błędu = 0); struct err_info *err_ptr -wskazanie na strukturę zawierającą opis błędu związany z realizacją zlecenia Cancel (a nie uprzedniej usługi, do której Cancel ma zastosowanie)

```
struct err_info
 {
 SHORT class;                          /* error class                    */
 SHORT code;                           /* error code                     */
 struct adtnl_err_resp_info adtnl;     /* additional error information */
 };

struct adtnl_err_resp_info
 {
 BOOLEAN mod_pos_pres; /* modifier position pres. flag                   */
 LONG mod_pos;      /* modifier position                          */
 BOOLEAN info_pres;        /* additional information present indicator  */
 BOOLEAN code_pres;        /* additional code present indicator         */
 LONG code;         /* additional code                            */
 BOOLEAN descr_pres;       /* additional description present indicator  */
 BYTE*descr;        /* pointer to the additional description      */
 BOOLEAN ssi_pres; /* service specific info present indicator     */
 SHORT service;     /* number indicating the specific service     */
 SHORT ss_error_val;       /* service specific error value               */
                    /* for companion standard error fields        */
 SHORT ss_error_len;       /* service specific error length              */
 UBYTE *ss_error_data;     /* service specific error data                */
 };
```

5.3.2. Usługi dodatkowe stosowane w VMD

Usługa GetCapabilityList (Pobranie_Listy_Uprawnień)

MMSREQ_PEND *	mp_getcl(SHORT chan, struct getcl_req_info *info_ptr) - przesyła PDU zlecenia GetCapabilityList; SHORT chan - numer kanału, którym zachodzi przesłanie; struct getcl_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.
SHORT	mp_getcl_resp(struct MMSREQ_IND *req_ptr, struct getcl_resp_info *info_ptr) - przesyła PDU z (pozytywnym) potwierdzeniem wykonania zlecenia GetCapabilityList; struct MMSREQ_IND *req_ptr - wskazanie na strukturę zwracaną przez funkcję u_getcl_ind; struct getcl_resp_info *info_ptr - wskazanie na strukturę zawierającą informacje o zleceniu.
VOID	u_getcl_ind(MMSREQ_IND *)
VOID	u_mp_getcl_conf(MMSREQ_PEND *)

```
struct getcl_req_info
 {
 BOOLEAN cont_after_pres;        /* flag if continue param pres. */
 BYTE *continue_after;           /* pointer continue-after name      */
 };

struct getcl_resp_info
 {
 BOOLEAN more_follows;           /* end of list boolean value    */
 SHORT num_of_capab;                     /* number of capabilities       */
 };
```

Usługa GetNameList (Pobranie_Listy_Nazw)

MMSREQ_PEND *	mp_namelist(SHORT chan, struct namelist_req_info *info_ptr) - przesyła PDU zlecenia GetNameList; SHORT chan - numer kanału, którym zachodzi przesłanie; struct namelist_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.
SHORT	mp_namelist_resp(struct MMSREQ_IND *req_ptr, struct namelist_resp_info *info_ptr) - przesyła PDU z (pozytywnym) potwierdzeniem wykonania zlecenia GetNameList; struct MMSREQ_IND *req_ptr - wskazanie na strukturę zwracaną przez funkcję u_namelist_ind; struct namelist_resp_info *info_ptr - wskazanie na strukturę zawierającą dane o zleceniu.
VOID	u_namelist_ind(MMSREQ_IND *)
VOID	u_mp_namelist_conf(MMSREQ_PEND *)

```
struct namelist_req_info
 {
 BOOLEAN cs_objclass_pres;              /* set for use CS object class  */
 union
        {
        SHORT mms_class;                /* object class          */
        struct                  /* CS context object sel        */
                {
                SHORT len;              /* length of ASN.1 CS class  */
                UBYTE *class;                   /* CS object class ASN.1      */
                } cs;
        } obj;
 SHORT objscope;                /* object scope                         */
 BYTE dname[MAX_IDENT_LEN+1];           /* domain name, for scope = dom     */
 BOOLEAN cont_after_pres;       /* flag if continue param pres. */
 BYTE continue_after[MAX_IDENT_LEN+1]; /* continue-after object name     */
 };

struct namelist_resp_info
 {
 BOOLEAN more_follows;          /* end of list boolean value            */
 SHORT num_names;       /* number of names                      */
 };
```

Usługa Identify (Identyfikacja)

MMSREQ_PEND *	mp_ident(SHORT chan) - przesyła PDU zlecenia Identify; SHORT chan - numer kanału, którym zachodzi przesłanie.
SHORT	mv_ident_resp(struct MMSREQ_IND *req_ptr) - przesłanie potwierdzenia na odpowiedź o wykonaniu usługi Identify przez odwołanie się do VMD; struct MMSREQ_IND *req_ptr - wskazanie na strukturę uprzednio przekazaną w argumencie funkcji u_ident_ind.
VOID	u_ident_ind(MMSREQ_IND *)
VOID	u_mp_ident_conf(MMSREQ_PEND *)

Usługa Status (Status)

MMSREQ_PEND *	mp_status(SHORT chan, struct status_req_info *info_ptr) - przesyła PDU zlecenia Status; SHORT chan - numer kanału, którym zachodzi przesłanie; struct status_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.

111

SHORT	mp_status_resp(struct MMSREQ_IND *req_ptr, struct status_resp_info *info_ptr) - przesyła PDU z (pozytywnym) potwierdzeniem wykonania zlecenia Status; struct MMSREQ_IND *req_ptr - wskazanie na strukturę będącą argumentem funkcji u_status_ind; struct status_resp_info *info_ptr - wskazanie na strukturę zawierającą informacje o zleceniu.
VOID	u_status_ind(MMSREQ_IND *)
VOID	u_mp_status_conf(MMSREQ_PEND *)

```
struct status_req_info
 {
 BOOLEAN extended;              /* flag indicating if extended status   */
 };                             /* is desired                           */

struct status_resp_info
 {
 SHORT logical_stat;            /* logical status - not optional*/
 SHORT physical_stat;           /* physical status - not optional       */
 BOOLEAN local_detail_pres;     /* flag if logical detail is present    */
 SHORT local_detail_len;        /* length of logical detail (in bits)   */
 UBYTE local_detail[16];        /* vendor-specific detailed info        */
 BOOLEAN addl_detail_pres;      /* flag if additional detail is present */
 SHORT addl_detail_len;         /* length of additional detail          */
 UBYTE *addl_detail;            /* ptr to raw ASN.1 defined by cmpn std */
 };
```

Usługa UnsolicitedStatus (Status_Spontaniczny)

MMSREQ_PEND *	mp_ustatus(SHORT chan, struct ustatus_req_info *info_ptr) - przesyła PDU zlecenia UnsolicitedStatus; SHORT chan - numer kanału, którym zachodzi przesłanie; struct ustatus_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.
VOID	u_ustatus_ind(MMSREQ_IND *)

```
struct ustatus_req_info
 {
 SHORT logical_stat;            /* logical status - not optional*/
 SHORT physical_stat;           /* physical status - not optional       */
 BOOLEAN local_detail_pres;     /* flag if logical detail is present    */
 SHORT local_detail_len;        /* length of logical detail (in bits)   */
 UBYTE local_detail[16];        /* vendor-specific detailed info        */
 BOOLEAN addl_detail_pres;      /* flag if additional detail is present */
 SHORT addl_detail_len;         /* length of additional detail          */
 UBYTE *addl_detail;            /* ptr to raw ASN.1 defined by cmpn std */
 };
```

5.3.3. Usługi zarządzania domeną

Usługa DeleteDomain (Usuwanie_Domeny)

MMSREQ_PEND *	mp_deldom(SHORT chan, struct deldom_req_info *info_ptr) - przesyła PDU zlecenia DeleteDomain; SHORT chan - numer kanału, którym zachodzi przesłanie; struct deldom_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.
SHORT	mp_deldom_resp(struct MMSREQ_IND *req_ptr) - przesyła PDU z (pozytywnym) potwierdzeniem wykonania zlecenia DeleteDomain; struct MMSREQ_IND *req_ptr - wskazanie na strukturę będącą argumentem funkcji u_deldom_ind.
VOID	u_deldom_ind(MMSREQ_IND *)
VOID	u_mp_deldom_conf(MMSREQ_PEND *)

```
struct deldom_req_info
 {
 BYTEdname [MAX_IDENT_LEN +1];    /* domain name                    */
 };
```

Usługa GetDomainAttributes (Pobranie_Atrybutów_Domeny)

MMSREQ_PEND *	mp_getdom(SHORT chan, struct getdom_req_info *info_ptr) - przesyła PDU zlecenia GetDomainAttributes; SHORT chan - numer kanału, którym zachodzi przesłanie; struct getdom_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.
SHORT	mp_getdom_resp(struct MMSREQ_IND *req_ptr, struct getdom_resp_info *info_ptr) - przesyła PDU z (pozytywnym) potwierdzeniem wykonania zlecenia GetDomainAttributes; struct MMSREQ_IND *req_ptr - wskazanie na strukturę zwracaną przez funkcję u_getdom_ind; struct getdom_resp_info *info_ptr - wskazanie na strukturę zawierającą informacje o zleceniu.
VOID	u_getdom_ind(MMSREQ_IND *)
VOID	u_mp_getdom_conf(MMSREQ_PEND *)

```
struct getdom_req_info
 {
 BYTEdname [MAX_IDENT_LEN +1];    /* domain name                    */
 };

struct getdom_resp_info
 {
 SHORT num_of_capab;              /* number of capabilities         */
 BOOLEAN mms_deletable;           /* MMS deletable, no default      */
 BOOLEAN sharable;                /* boolean, no default            */
```

```
SHORT num_of_pinames;        /* number of pi names                */
SHORT state;                 /* 0 : non-existent          */
                             /* 1 : loaded                */
                             /* 2 : ready                 */
                             /* 3 : in use                */
                             /* 4 : complete              */
                             /* 5 : incomplete            */
BYTE upload_in_progress;     /* upload in progress        */
BOOLEAN dom_detail_pres;             /* domain detail used        */
SHORT dom_detail_len;        /* the length of dom detail  */
UBYTE        *dom_detail;            /* ptr to dom detail         */
};
```

Usługa LoadDomainContent (Ładowanie_Zawartości_Domeny)

MMSREQ_PEND *	mp_loaddom(SHORT chan, struct loaddom_req_info *info_ptr) - przesyła PDU zlecenia LoadDomainContent; SHORT chan - numer kanału, którym zachodzi przesłanie; struct loaddom_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.
SHORT	mp_loaddom_resp(struct MMSREQ_IND *req_ptr) - przesyła PDU z (pozytywnym) potwierdzeniem wykonania zlecenia LoadDomainContent; struct MMSREQ_IND *req_ptr - wskazanie na strukturę będącą argumentem funkcji u_loaddom_ind.
VOID	u_loaddom_ind(MMSREQ_IND *)
VOID	u_mp_loaddom_conf(MMSREQ_PEND *)

```
struct loaddom_req_info
 {
BYTEdname [MAX_IDENT_LEN +1];    /* domain name                       */
BOOLEAN sharable;            /* boolean, no default       */
BOOLEAN ld_detail_pres;      /* download detail present   */
SHORT ld_detail_len;                 /* download detail ptr & len */
UBYTE *ld_detail;
BOOLEAN third_pty_pres;      /* third party used          */
SHORT third_pty_len;                 /* the length of third party */
UBYTE        *third_pty;             /* ptr to third party ASN.1  */
SHORT num_of_capab;                  /* number of capabilities    */
SHORT num_of_fname;                  /* num of fname elements     */
};
```

Usługa StoreDomainContent (Przechowanie_Zawartości_Domeny)

MMSREQ_PEND *	mp_storedom(SHORT chan, struct storedom_req_info *info_ptr) - przesyła PDU zlecenia StoreDomainContent; SHORT chan - numer kanału, którym zachodzi przesłanie; struct storedom_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.
SHORT	mp_storedom_resp(struct MMSREQ_IND *req_ptr) - przesyła PDU z (pozytywnym) potwierdzeniem wykonania zlecenia StoreDomainContent; struct MMSREQ_IND *req_ptr - wskazanie na strukturę będącą argumentem funkcji u_storedom_ind.
VOID	u_storedom_ind(MMSREQ_IND *)
VOID	u_mp_storedom_conf(MMSREQ_PEND *)

```
struct storedom_req_info
 {
 BYTEdname [MAX_IDENT_LEN +1];    /* domain name                    */
 SHORT num_of_fname;              /* num of fname elements    */
 BOOLEAN third_pty_pres;          /* third party used         */
 SHORT third_pty_len;             /* the length of third party  */
 UBYTE      *third_pty;           /* ptr to third party ASN.1   */
 };
```

5.3.4. Usługi zarządzania wywołaniem programu

Usługa CreateProgramInvocation (Utworzenie_Wywołania_Programu)

MMSREQ_PEND *	mp_crepi(SHORT chan, struct crepi_req_info *info_ptr) - przesyła PDU zlecenia CreateProgramInvocation; SHORT chan - numer kanału, którym zachodzi przesłanie; struct crepi_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.
SHORT	mp_crepi_resp(struct MMSREQ_IND *req_ptr) - przesyła PDU z (pozytywnym) potwierdzeniem wykonania zlecenia CreateProgramInvocation; struct MMSREQ_IND *req_ptr - wskazanie na strukturę będącą argumentem funkcji u_crepi_ind.
VOID	u_crepi_ind(MMSREQ_IND *)
VOID	u_mp_crepi_conf(MMSREQ_PEND *)

```
struct crepi_req_info
 {
 BYTEpiname [MAX_IDENT_LEN+1];    /* program invocation name  */
 SHORT num_of_dnames;             /* number of domain names   */
 BOOLEAN create_detail_pres;      /* create detail present    */
 SHORT create_detail_len;         /* create detail length */
 UBYTE      *create_detail;       /* pointer to create detail */
 BOOLEAN monitor_pres;            /* monitoring present       */
 BOOLEAN monitor;                 /* TRUE : permanent monitoring */
```

```
                                        /* FALSE: current monitoring        */
};
```

Usługa DeleteProgramInvocation (Usuwanie_Wywołania_Programu)

MMSREQ_PEND *	mp_delpi(SHORT chan, struct delpi_req_info *info_ptr) - przesyła PDU zlecenia DeleteProgramInvocation; SHORT chan - numer kanału, którym zachodzi przesłanie; struct delpi_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.
SHORT	mp_delpi_resp(struct MMSREQ_IND *req_ptr) - przesyła PDU z (pozytywnym) potwierdzeniem wykonania zlecenia DeleteProgramInvocation; struct MMSREQ_IND *req_ptr - wskazanie na strukturę będącą argumentem funkcji u_delpi_ind.
VOID	u_delpi_ind(MMSREQ_IND *)
VOID	u_mp_delpi_conf(MMSREQ_PEND *)

```
struct delpi_req_info
 {
 BYTEpiname [MAX_IDENT_LEN+1];     /* program invocation name  */
 };
```

Usługa GetProgramInvocationAttributes
(Pobranie_Atrybutów_Wywołania_Programu)

MMSREQ_PEND *	mp_getpi(SHORT chan, struct getpi_req_info *info_ptr) - przesyła PDU zlecenia GetProgramInvocationAttributes; SHORT chan - numer kanału, którym zachodzi przesłanie; struct getpi_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.
SHORT	mp_getpi_resp(struct MMSREQ_IND *req_ptr, struct getepi_resp_info *info_ptr) przesyła PDU z (pozytywnym) potwierdzeniem wykonania zlecenia GetProgramInvocationAttributes; struct MMSREQ_IND *req_ptr - wskazanie na strukturę zwracaną przez funkcję u_getpi_ind; struct getepi_resp_info *info_ptr - wskazanie na strukturę zawierającą informacje o zleceniu.
VOID	u_getpi_ind(MMSREQ_IND *)
VOID	u_mp_getpi_conf(MMSREQ_PEND *)

```
struct getpi_req_info
 {
 BYTEpiname [MAX_IDENT_LEN+1];     /* program invocation name  */
 };

struct getpi_resp_info
 {
 SHORT state;                      /* program invocation state   */
```

```
BOOLEAN mms_deletable;          /* MMS deletable          */
BOOLEAN reusable;               /* Reusable               */
BOOLEAN monitor;                /* TRUE : permanent monitoring */
                                /* FALSE: current monitoring   */
BYTE*start_arg;                 /* Start Argument         */
BOOLEAN pi_detail_pres;         /* PI detail present      */
SHORT pi_detail_len;            /* PI detail length       */
UBYTE      *pi_detail;          /* pointer to PI detail   */
SHORT num_of_dnames;            /* number of domain names */
};
```

Usługa Reset (Zerowanie)

MMSREQ_PEND *	mp_reset(SHORT chan, struct reset_req_info *info_ptr) - przesyła PDU zlecenia Reset; SHORT chan - numer kanału, którym zachodzi przesłanie; struct reset_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.
SHORT	mp_reset_resp(struct MMSREQ_IND *req_ptr) - przesyła PDU z (pozytywnym) potwierdzeniem wykonania zlecenia Reset; struct MMSREQ_IND *req_ptr - wskazanie na strukturę będącą argumentem funkcji u_reset_ind.
VOID	u_reset_ind(MMSREQ_IND *)
VOID	u_mp_reset_conf(MMSREQ_PEND *)

```
struct reset_req_info
 {
BYTEpiname [MAX_IDENT_LEN+1];   /* program invocation name */
BOOLEAN reset_detail_pres;      /* reset detail present    */
SHORT reset_detail_len;         /* reset detail length     */
UBYTE      *reset_detail;       /* pointer to reset detail */
};
```

Usługa Resume (Wznowienie)

MMSREQ_PEND *	mp_resume(SHORT chan, struct resume_req_info *info_ptr) - przesyła PDU zlecenia Resume; SHORT chan - numer kanału, którym zachodzi przesłanie; struct resume_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.
SHORT	mp_resume_resp(struct MMSREQ_IND *req_ptr) - przesyła PDU z (pozytywnym) potwierdzeniem wykonania zlecenia Resume; struct MMSREQ_IND *req_ptr - wskazanie na strukturę będącą argumentem funkcji u_resume_ind.
VOID	u_resume_ind(MMSREQ_IND *)
VOID	u_mp_resume_conf(MMSREQ_PEND *)

```
struct resume_req_info
 {
 BYTEpiname [MAX_IDENT_LEN+1];    /* program invocation name  */
 BOOLEAN resume_detail_pres;      /* resume detail present    */
 SHORT resume_detail_len;         /* resume detail length     */
 UBYTE      *resume_detail;       /* pointer to resume detail */
 };
```

Usługa Start (Start)

MMSREQ_PEND *	mp_start(SHORT chan, struct start_req_info *info_ptr) - przesyła PDU zlecenia Start; SHORT chan - numer kanału, którym zachodzi przesłanie; struct start_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.
SHORT	mp_start_resp(struct MMSREQ_IND *req_ptr) - przesyła PDU z (pozytywnym) potwierdzeniem wykonania zlecenia Start; struct MMSREQ_IND *req_ptr - wskazanie na strukturę będącą argumentem funkcji u_start_ind.
VOID	u_start_ind(MMSREQ_IND *) - Start;
VOID	u_mp_start_conf(MMSREQ_PEND *) - Start;

```
struct start_req_info
 {
 BYTEpiname [MAX_IDENT_LEN+1];    /* program invocation name  */
 BOOLEAN start_arg_pres;          /* start argument present   */
 BYTE*start_arg;                  /* pointer to start argument */
 BOOLEAN start_detail_pres;       /* start detail present     */
 SHORT start_detail_len;          /* start detail length      */
 UBYTE      *start_detail;        /* pointer to start detail  */
 };
```

Usługa Stop (Stop)

MMSREQ_PEND *	mp_stop(SHORT chan, struct stop_req_info *info_ptr) - przesyła PDU zlecenia Stop; SHORT chan - numer kanału, którym zachodzi przesłanie; struct stop_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.
SHORT	mp_stop_resp(struct MMSREQ_IND *req_ptr) - przesyła PDU z (pozytywnym) potwierdzeniem wykonania zlecenia Stop; struct MMSREQ_IND *req_ptr - wskazanie na strukturę będącą argumentem funkcji u_stop_ind.
VOID	u_stop_ind(MMSREQ_IND *)
VOID	u_mp_stop_conf(MMSREQ_PEND *)

```
struct stop_req_info
 {
 BYTEpiname [MAX_IDENT_LEN+1];    /* program invocation name  */
 BOOLEAN stop_detail_pres;        /* stop detail present      */
```

```
SHORT stop_detail_len;          /* stop detail length          */
UBYTE     *stop_detail;                 /* pointer to stop detail      */
};
```

5.3.5. Usługi dostępu do zmiennej

Bezpośrednio poniżej przedstawiono struktury danych wykorzystywane wewnątrz innych struktur danych, specyficznych dla poszczególnych funkcji.

```
struct unconst_addr
 {
 SHORT unc_len;                 /* unconstrained address length */
 UBYTE     *unc_ptr;                   /* unconstrained address ptr  */
 };

struct var_acc_addr
 {
 SHORT addr_tag;                /* address tag                 */
 union
        {
        ULONG      num_addr;           /* numeric address             */
        BYTE*sym_addr;          /* symbolic address            */
        struct unconst_addr unc_addr;       /* unconstrained address       */
        } addr;
};

struct var_acc_spec
 {
 SHORT var_acc_tag;             /* variable access tag         */
 struct object_name vl_name;    /* variable list name          */
 SHORT num_of_variables;        /* number of variables                 */
/*                              list of variables           */
 };

struct var_acc_data
 {
 SHORT len;                     /* len of variable access data  */
 UBYTE     *data;                      /* ptr to variable access data  */
 };
```

Usługa GetVariableAccessAttributes (Pobranie_Atrybutów_Dostępu_Do_Zmiennej)

MMSREQ_PEND *	mp_getvar(SHORT chan, struct getvar_req_info *info_ptr) - przesyła PDU zlecenia GetVariableAccessAttributes; SHORT chan - numer kanału, którym zachodzi przesłanie; struct getvar_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.

SHORT	mv_getvar_resp(struct MMSREQ_IND *req_ptr) - przesłanie potwierdzenia na odpowiedź o wykonaniu usługi GetVariableAcceessAttributes przez odwołanie się do VMD; struct MMSREQ_IND *req_ptr - wskazanie na strukturę uprzednio przekazaną w argumencie funkcji u_getvar_ind.
VOID	u_getvar_ind(MMSREQ_IND *)
VOID	u_mp_getvar_conf(MMSREQ_PEND *)

```
struct getvar_req_info
 {
 SHORT req_tag;                 /* request tag                */
                                /* 0 : object name request    */
                                /* 1 : address request  */
 struct object_name name;       /* object name                */
 struct var_acc_addr address;   /* address                    */
 };
```

Usługa InformationReport (Raportowanie_Informacji)

MMSREQ_PEND *	mv_info(SHORT chan, struct object_name *vname, struct object_name *tname, BYTE *src_data) - wykonanie usługi InformationReport przez odwołanie się do VMD; SHORT chan - kanał, który posłuży do wykonania usługi; struct object_name *vname - wskazanie na strukturę zawierającą nazwę zmiennej włączaną do PDU usługi InformationReport; struct object_name *tname - wskazanie na strukturę zawierającą lokalną nazwę zmiennej. Jeżeli argument ten = NULL, to MMS-EASE używa informacji z bazy danych VMD; BYTE *src_data - wskazanie na miejsce włączenia informacji w PDU dla InformationReport.
VOID	u_info_ind(MMSREQ_IND *)

Usługa Read (Czytanie)

MMSREQ_PEND *	mp_read(SHORT chan, struct read_req_info *info_ptr) - przesyła PDU zlecenia Read; SHORT chan - numer kanału, którym zachodzi przesłanie; struct read_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.
SHORT	mv_read_resp(struct MMSREQ_IND *req_ptr) - przesłanie potwierdzenia na odpowiedź o wykonaniu usługi Read przez odwołanie się do VMD; struct MMSREQ_IND *req_ptr - wskazanie na strukturę uprzednio przekazaną w argumencie funkcji u_read_ind.
VOID	u_read_ind(MMSREQ_IND *)
VOID	u_mp_read_conf(MMSREQ_PEND *)

```
struct read_req_info
 {
 BOOLEAN spec_in_result;          /* specification with result,   */
                                  /* default to false             */
 struct var_acc_spec   va_spec;   /* variable access spec         */
/*                                 which "includes" */
/*                                 a list of variables      */
 };
```

Usługa Write (Zapisanie)

MMSREQ_PEND *	mp_write(SHORT chan, struct write_req_info *info_ptr) - przesyła PDU zlecenia Write; SHORT chan - numer kanału, którym zachodzi przesłanie; struct write_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.
SHORT	mv_write_resp(struct MMSREQ_IND *req_ptr) - przesłanie potwierdzenia na odpowiedź o wykonaniu usługi Write przez odwołanie się do VMD; struct MMSREQ_IND *req_ptr - wskazanie na strukturę uprzednio przekazaną w argumencie funkcji u_write_ind.
VOID	u_write_ind(MMSREQ_IND *)
VOID	u_mv_write_conf(MMSREQ_PEND *)

```
struct write_req_info
 {
 SHORT num_of_data;                 /* number of data               */
 struct var_acc_data *va_data;      /* ptr to list of variable data */
 struct var_acc_spec   va_spec;     /* variable access spec         */
/*                                  which "includes" */
/*                                  a list of variables      */
 };
```

5.3.6. Usługi zarządzania semaforem

Wykorzystywana dalej struktura object_name jest opisana w 5.2.2.

Usługa RelinquishControl (Zaniechanie_Sterowania)

MMSREQ_PEND *	mp_relctrl(SHORT chan, struct relctrl_req_info *info_ptr) - przesyła PDU zlecenia RelinquishControl; SHORT chan - numer kanału, którym zachodzi przesłanie; struct relctrl_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.
SHORT	mp_relctrl_resp(struct MMSREQ_IND *req_ptr) - przesyła PDU z (pozytywnym) potwierdzeniem wykonania zlecenia RelinquishControl; struct MMSREQ_IND *req_ptr - wskazanie na strukturę zwracaną przez funkcję u_relctr_ind.
VOID	u_relctrl_ind(MMSREQ_IND *)

VOID	u_mp_relctrl_conf(MMSREQ_PEND *)

```
struct relctrl_req_info
 {
 struct object_name sem_name;       /* semaphore name              */
 BOOLEAN named_token_pres;          /* named token present ind     */
 BYTEnamed_token [MAX_IDENT_LEN+1]; /* named token                 */
 };
```

Usługa ReportPoolSemaphoreStatus (Raportowanie_Statusu_Semafora_Puli)

MMSREQ_PEND *	mp_rspool(SHORT chan, struct rspool_req_info *info_ptr) - przesyła PDU zlecenia ReportPoolSemaphoreStatus; SHORT chan - numer kanału, którym zachodzi przesłanie; struct rspool_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.
SHORT	mp_rspool_resp(struct MMSREQ_IND *req_ptr, struct rspool_resp_info *info_ptr) - przesyła PDU z (pozytywnym) potwierdzeniem wykonania zlecenia ReportPoolSemaphoreStatus; struct MMSREQ_IND *req_ptr - wskazanie na strukturę zwracaną przez funkcję u_rspool_ind; struct rspool_resp_info *info_ptr - wskazanie na strukturę zawierającą informacje o zleceniu.
VOID	u_rspool_ind(MMSREQ_IND *)
VOID	u_mp_rspool_conf(MMSREQ_PEND *)

```
struct rspool_req_info
 {
 struct object_name sem_name;       /* semaphore name              */
 BOOLEAN start_after_pres;          /* start after name present ind */
 BYTE start_after [MAX_IDENT_LEN+1]; /* name to start after        */
 };

struct rspool_resp_info
 {
 SHORT num_of_tokens;               /* number of named tokens      */
 BOOLEAN more_follows;              /* more follows indicator      */
 };
```

Usługa ReportSemaphoreEntryStatus (Raportowanie_Statusu_Pozycji_Semafora)

MMSREQ_PEND *	mp_rsentry(SHORT chan, struct rsentry_req_info *info_ptr) - przesyła PDU zlecenia ReportSemaphoreEntryStatus; SHORT chan - numer kanału, którym zachodzi przesłanie; struct rsentry_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.

SHORT	mp_rsentry_resp(struct MMSREQ_IND *req_ptr, struct rsentry_resp_info *info_ptr) - przesyła PDU z (pozytywnym) potwierdzeniem wykonania zlecenia ReportSemaphoreEntryStatus; struct MMSREQ_IND *req_ptr - wskazanie na strukturę zwracaną przez funkcję u_rsentry_ind; struct rsentry_resp_info *info_ptr - wskazanie na strukturę zawierającą informacje o zleceniu.
VOID	u_rsentry_ind(MMSREQ_IND *)
VOID	u_mp_rsentry_conf(MMSREQ_PEND *)

```
struct rsentry_req_info
 {
 struct object_name sem_name;     /* semaphore name          */
 SHORT state;                     /* state: 0 : queued       */
                                  /*        1 : owner        */
                                  /*        2 : hung         */
 BOOLEAN start_after_pres;        /* start after id present ind  */
 SHORT sa_len;                    /* length of entry id to start */
                                  /* after                */
 UBYTE       *start_after;              /* pointer to entry id to start */
                                  /* after                */
 };

struct rsentry_resp_info
 {
 SHORT num_of_sent;                    /* number of semaphore entries     */
 BOOLEAN more_follows;                 /* more follows indicator      */
 };
```

Usługa ReportSemaphoreStatus (Raportowanie_Statusu_Semafora)

MMSREQ_PEND *	mp_rsstat(SHORT chan, struct rsstat_req_info *info_ptr) - przesyła PDU zlecenia ReportSemaphoreStatus; SHORT chan - numer kanału, którym zachodzi przesłanie; struct rsstat_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.
SHORT	mp_rsstat_resp(struct MMSREQ_IND *req_ptr, struct rsstat_resp_info *info_ptr) - przesyła PDU z (pozytywnym) potwierdzeniem wykonania zlecenia ReportSemaphoreStatus; struct MMSREQ_IND *req_ptr - wskazanie na strukturę zwracaną przez funkcję u_rsstat_ind; struct rsstat_resp_info *info_ptr - wskazanie na strukturę zawierającą informacje o zleceniu.
VOID	u_rsstat_ind(MMSREQ_IND *)
VOID	u_mp_rsstat_conf(MMSREQ_PEND *)

```
struct rsstat_req_info
 {
 struct object_name sem_name;       /* semaphore name            */
 };

struct rsstat_resp_info
 {
 BOOLEAN   mms_deletable;              /* mms deletable              */
 SHORT     class;                   /* class: token (0) or pool (1) */
 USHORT    num_of_tokens;              /* number of tokens           */
 USHORT    num_of_owned;               /* number of owned tokens     */
 USHORT    num_of_hung;             /* number of hung tokens      */
 };
```

Usługa TakeControl (Objęcie_Sterowania)

MMSREQ_PEND *	mp_takectrl(SHORT chan, struct takectrl_req_info *info_ptr) - przesyła PDU zlecenia TakeControl; SHORT chan - numer kanału, którym zachodzi przesłanie; struct takectrl_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.
SHORT	mp_takectrl_resp(struct MMSREQ_IND *req_ptr, struct takectrl_resp_info *info_ptr) - przesyła PDU z (pozytywnym) potwierdzeniem wykonania zlecenia TakeControl; struct MMSREQ_IND *req_ptr - wskazanie na strukturę zwracaną przez funkcję u_takectrl_ind; struct takectrl_resp_info *info_ptr - wskazanie na strukturę zawierającą informacje o zleceniu.
VOID	u_takectrl_ind(MMSREQ_IND *)
VOID	u_mp_takectrl_conf(MMSREQ_PEND *)

```
struct takectrl_req_info
 {
 struct object_name sem_name;       /* semaphore name            */
 BOOLEAN named_token_pres;             /* named token present ind   */
 BYTE named_token [MAX_IDENT_LEN+1]; /* named token                 */
 UBYTE priority;                    /* priority                  */
                                    /* 0 - highest          */
                                    /* 64 - normal (default)     */
                                    /* 127 - lowest              */
 BOOLEAN acc_delay_pres;               /* acceptable delay present ind */
 ULONG acc_delay;                   /* acceptable delay          */
 BOOLEAN ctrl_timeout_pres;            /* control timeout present   */
 ULONG ctrl_timeout;                   /* control timeout           */
 BOOLEAN abrt_on_timeout_pres;         /* Abort on Timeout present  */
 BOOLEAN abrt_on_timeout;              /* Abort on Timeout          */
 BOOLEAN rel_conn_lost;             /* relenquish if connection löst*/
 BOOLEAN app_preempt_pres;             /* app preempt present ind   */
 SHORT app_len;                     /* length of app. to preempt  */
 UBYTE      *app_preempt;                  /* application to preempt    */
 };
```

```
struct takectrl_resp_info
 {
 SHORT resp_tag;                    /* response tag                */
                                    /* 0 : NULL response           */
                                    /* 1 : named token response    */
 BYTEnamed_token [MAX_IDENT_LEN+1];     /* named token identifier      */
 };
```

5.3.7. Usługi komunikacji operatorskiej

Usługa Input (Wejście)

MMSREQ_PEND *	mp_input(SHORT chan, struct input_req_info *info_ptr) - przesyła PDU zlecenia Input; SHORT chan - numer kanału, którym zachodzi przesłanie; struct input_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.
SHORT	mp_input_resp(struct MMSREQ_IND *req_ptr, struct input_resp_info *info_ptr) - przesyła PDU z (pozytywnym) potwierdzeniem wykonania zlecenia Input; struct MMSREQ_IND *req_ptr - wskazanie na strukturę zwracaną przez funkcję u_input_ind; struct input_resp_info *info_ptr - wskazanie na strukturę zawierającą informacje o zleceniu.
VOID	u_input_ind(MMSREQ_IND *)
VOID	u_mp_input_conf(MMSREQ_PEND *)

```
struct input_req_info
 {
 BYTEstation_name [MAX_IDENT_LEN+1]; /* operator station name   */
 BOOLEAN echo;                       /* echo, default: TRUE                */
 BOOLEAN timeout_pres;               /* input timeout present ind  */
 ULONG timeout;                  /* input timeout              */
 BOOLEAN prompt_pres;                /* prompt data present ind */
 SHORT prompt_count;                 /* number of prompt strings   */
 };

struct input_resp_info
 {
 BYTE *input_resp;               /* input response             */
 };
```

Usługa Output (Wyjście)

MMSREQ_PEND *	mp_output(SHORT chan, struct output_req_info *info_ptr) - przesyła PDU zlecenia Output; SHORT chan - numer kanału, którym zachodzi przesłanie; struct output_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.

SHORT	mp_output_resp(struct MMSREQ_IND *req_ptr) - przesyła PDU z (pozytywnym) potwierdzeniem wykonania zlecenia Output; struct MMSREQ_IND *req_ptr - wskazanie na strukturę zwracaną przez funkcję u_fclose_ind.
VOID	u_output_ind(MMSREQ_IND *)
VOID	u_mp_output_conf(MMSREQ_PEND *)

```
struct output_req_info
 {
 BYTEstation_name [MAX_IDENT_LEN+1]; /* operator station name   */
 SHORT data_count;                   /* number of data strings  */
 };
```

5.3.8. Usługa zarządzania plikami

Usługa ObtainFile (Uzyskanie_Pliku)

MMSREQ_PEND *	mp_obtfile(SHORT chan, struct obtfile_req_info *info_ptr) - przesyła PDU zlecenia ObtainFile; SHORT chan - numer kanału, którym zachodzi przesłanie; struct obtfile_req_info *info_ptr - wskazanie na strukturę zawierającą dane PDU, które mają być wysłane.
SHORT	mp_obtfile_resp(struct MMSREQ_IND *req_ptr) - przesyła PDU z (pozytywnym) potwierdzeniem wykonania zlecenia ObtainFile; struct MMSREQ_IND *req_ptr - wskazanie na strukturę zwracaną przez funkcję u_obtfile_ind.
VOID	u_obtfile_ind(MMSREQ_IND *)
VOID	u_mp_obtfile_conf(MMSREQ_PEND *)

```
struct obtfile_req_info
 {
 BOOLEAN ar_title_pres;          /* application title present     */
 SHORT ar_len;                   /* length of app proc title      */
 UBYTE       *ar_title;                  /* application process title     */
 SHORT num_of_src_fname;         /* num of source fname elements  */
 SHORT num_of_dest_fname;                /* num of dest. fname elements       */
 };
```

6. IMPLEMENTACJA MODELU VMD I JEJ BADANIA

6.1. Implementacja modelu VMD

W skład zestawu sieciowego wchodzą następujące elementy:

- stacje sieciowe,
- media transmisyjne i osprzęt sieciowy.

Media transmisyjne i osprzęt sieciowy to okablowanie, aktywne i pasywne rozgałęźniki, terminatory, itp. Stanowią one element technologii sieciowej. Ich dobór jest silnie uzależniony od zakładanego rodzaju sieci, a w ramach tego ostatniego, także od przyjmowanej metody dostępu do medium transmisyjnego.

Stacja sieciowa, to komputer lub sterownik programowalny (np. sterownik robota, sterownik PLC, sterownik obrabiarki sterowanej numerycznie) wyposażony w odpowiednią kartę sieciową, stosowne oprogramowanie komunikacyjne i oprogramowanie aplikacyjne.

Wyposażenie stacji sieciowej zależy przede wszystkim od rodzaju obsługiwanej przez nią sieci. W przypadku, w którym rolę stacji sieciowej pełni sterownik programowalny, całość sprzętowego i programowego wyposażenia sieciowego dostarczana jest najczęściej przez producenta konkretnego rodzaju sterownika. Spowodowane jest to specyficzną wewnętrzną architekturą sterownika programowalnego, która w zasadzie u każdego producenta jest odmienna.

Inaczej dzieje się w przypadku komputerów kompatybilnych z IBM PC (także w wykonaniu przemysłowym). Wewnętrzna architektura sprzętowa jest zazwyczaj taka sama (występują różnice dotyczące stosowanej szyny systemowej). Oprogramowanie systemowe jest wymienne. Wyboru dokonuje się zazwyczaj spośród następujących systemów operacyjnych: MS - DOS, MS - Windows, OS/2, QNX lub jednej z wersji systemu UNIX.

Ponieważ VMD stanowi część warstwy aplikacyjnej, więc do otrzymania pełnej zdolności implementowanego oprogramowania do pracy w sieci, należy to oprogramowanie uzupełnić o warstwy 1 - 6. Innymi słowy trzeba dokonać doboru kart sieciowych i odpowiedniego, współpracującego z nimi oprogramowania komunikacyjnego, realizującego tzw. stos sieciowy. Zespół realizatorów dysponował w PIAP kartami sieciowymi firmy Computrol (oddział firmy AEG/Modicon). Są one przystosowane do pracy na platformie sprzętowej komputera kompatybilnego z IBM-PC/AT z jednym z następujących systemów operacyjnych: DOS wer. 3.0 lub wyższa, OS/2 wersja 1.1 lub wyższa, SCO-UNIX System V rel. 3.2, wraz z odpowiednim oprogramowaniem. Wspomniane karty sieciowe skonstruowane są z wykorzystaniem procesora MC68020 firmy Motorola i procesora komunikacyjnego MC68824. Umożliwia to ich autonomiczną pracę na poziomie najniższych warstw stosu ISO/OSI. Komunikacja z komputerem odbywa się przerwaniowo, a dane do karty i z karty przesyłane są poprzez DMA.

Z kartami sieciowymi LP-25 firmy Computrol współpracuje pakiet oprogramowania noszący nazwę ISOcomm. W skład pakietu ISOcomm wchodzi oprogramowanie nadzorujące działanie sterownika specjalizowanej karty sieciowej: MAPSMC, MAPGO, PC8024.IMG i sterownik karty

(w przypadku wersji do pracy w systemie DOS nosi on nazwę ICPDOSDR, w przypadku OS/2 - ICPOS2DR). MAPSMC służy do konfigurowania parametrów karty (w tym również parametrów segmentu sieci). Są one potem przechowywane w pliku PC8024.IMG. Z MAPSMC można także wywoływać MAPGO. Zadaniem MAPGO jest inicjalizacja karty i załadowania parametrów z PC8024.IMG do pamięci NVRAM (Non Volatile RAM) karty sterownika. MAPGO służy także do zresetowania karty. Sterownik karty (a więc ICPDOSDR lub ICPOS2DR) realizuje protokoły warstw 1-6 modelu ISO/OSI i część warstwy 7, która jest niezależna od oprogramowania aplikacyjnego.

Poza wymienionymi wyżej plikami w pakiecie ISOcomm dostarczany jest zestaw plików nagłówkowych i dwie biblioteki o nazwach CI.LIB i RINGCTRL.LIB w formacie zgodnym z zestawem kompilatora Microsoft C wer. 6.00. Skorzystanie z nich pozwala utworzyć własne (prywatne) interfejsy aplikacyjne, które mogą realizować usługi sieciowe i współpracować z protokołem MMS.

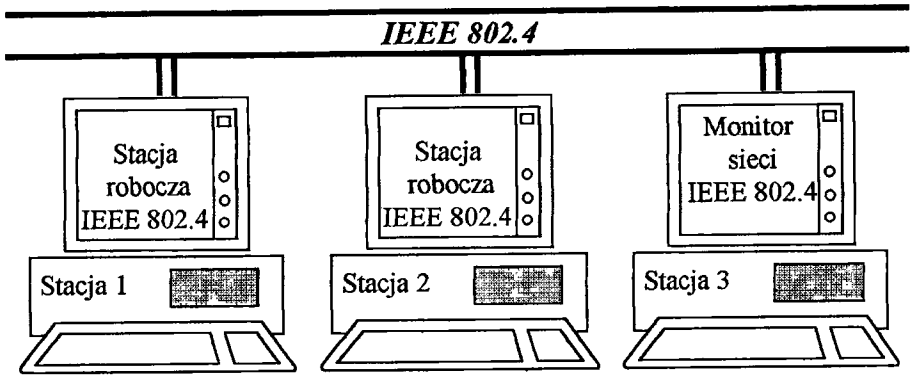
W skonstruowanym programie o nazwie MMS_PIAP wykorzystano wspomniane przed chwilą pliki nagłówkowe oraz biblioteki CI.LIB i RINGCTRL.LIB. Na poziomie warstwy 7 (a więc VMD - MMS) użyto struktur danych i funkcji opisanych w rozdziale 5 tego sprawozdania. Do testowania wprowadzono jeszcze dodatkową usługę (spoza MMS), informującą o stanie kanału logicznego.

Do komunikacji z otoczeniem programu MMS_PIAP służy konsola operatorska składająca się z klawiatury i monitora ekranowego. Na ekranie wyświetlane jest menu - poszczególne jego pozycje są wybierane z klawiatury klawiszami funkcyjnymi i klawiszem ENTER. Po wybraniu odpowiedniej pozycji menu użytkownik odpowiada na serię pojawiających się na ekranie pytań, po czym usługa wchodzi w etap realizacji.

6.2. Badania symulacyjne szczegółowego modelu VMD w sieci komputerów

Zestawiony do badań symulacyjnych segment sieci IEEE 802.4 składał się z 3 stacji. Stacje wyposażone były w karty sieciowe LP - 25 MAP Controller Board firmy AEG/Modicon/Computrol sprzężone z modemem AEG/Modicon/Computrol BK - 4 CM Carrierband Modem. Prędkość transmisji wynosiła 5 Mbit/s. Połączenia realizowane były kablem koncentrycznym głównym RG 11 i kablami koncentrycznymi odgałęzień stacyjnych RG 6, połączonymi przez rozgałęźniki i zaopatrzonymi w terminatory.

Stacje o numerach 1 i 2 pracowały jako stacje robocze - posadowiono w nich program MMS_PIAP (por. p. 6.1). Ponadto w skład segmentu IEEE 802.4 wchodził jeszcze monitor sieci IEEE 802.4, służący do śledzenia przebiegu procesów transmisyjnych, ich rejestracji i badania oraz nadzoru prawidłowości działania całego segmentu. Schemat ideowy segmentu przedstawiono na rys. 6.1. W tablicy. 6.1 zestawiono numery poszczególnych stacji, ich nazwy i przypisane im adresy sieciowe.



Rys. 6.1. Schemat ideowy segmentu sieci 802.4 do badań modelu VMD w sieci komputerów

Tablica 6.1. Przypisanie adresów stacjom segmentu sieci 802.4

Nazwa stacji	Nr stacji	Adres stacji w sieci 802.4 (wg. ISO 3166.DCC)
Stacja robocza IEEE 802.4	1	39840F454E45000170010001A02601
Stacja robocza IEEE 802.4	2	39840F454E45000170010001A02401
Monitor sieci IEEE 802.4	3	39840F454E45000170010001A02201

Konfiguracje sprzętowe i programowe stacji podano poniżej w odrębnych metryczkach.

1.	**Nr stacji :** 1			
2.	**Rodzaj (typ) sieci**: IEEE 802.4			
3.	**Nazwa stacji:** Stacja robocza IEEE 802.4			
4.	**Interfejs sieciowy:**			
	Karta sieciowa (typ): LP-25 firmy AEG/Modicon/Computrol z modemem BK-4 (Carrierband)			
	Interfejs programowy : DOS ISOcomm MAP 3.0 (802.4) Release 2.3 Stack for MS-DOS Software (SPA-35B), firmy AEG/Modicon/Computrol			
5.	**Komputer (specyfikacja):**			
	Procesor: 486DX /50MHz	***Dysk twardy:*** 120 Mbajtów +170 Mbajtów	***Karta graficzna:*** SVGA 512 kbajtów	***Monitor:*** 14" SVGA mono
	Pamięć RAM: 16 Mbajtów	***System operacyjny:*** MS-DOS 6.20		

1.	**Nr stacji :** 2			
2.	**Rodzaj (typ) sieci**: IEEE 802.4			
3.	**Nazwa stacji:** Stacja robocza IEEE 802.4			
4.	**Interfejs sieciowy:**			
	Karta sieciowa (typ): LP-25 firmy AEG/Modicon/Computrol z modemem BK-4 (Carrierband)			
	Interfejs programowy : DOS ISOcomm MAP 3.0 (802.4) Release 2.3 Stack for MS-DOS Software (SPA-35B), firmy AEG/Modicon/Computrol			
5.	**Komputer (specyfikacja):**			
	Procesor: 386DX/33MHz	***Dysk twardy:*** 100 Mbajtów +200 Mbajtów	***Karta graficzna:*** Hercules	***Monitor:*** 14" Hercules
	Pamięć RAM: 12 Mbajtów	***System operacyjny:*** MS-DOS 6.20		

1.	**Nr stacji :** 3			
2.	**Rodzaj (typ) sieci**: IEEE 802.4			
3.	**Nazwa stacji:** Monitor sieci IEEE 802.4			
4.	**Interfejs sieciowy:**			
	Karta sieciowa (typ): LP-25 firmy AEG/Modicon/Computrol z modemem BK-4 (Carrierband)			
	Interfejs programowy : firmy AEG/Modicon/Computrol			
5.	**Komputer (specyfikacja):**			
	Procesor: 486DX /50MHz	***Dysk twardy:*** 120 Mbajtów	***Karta graficzna:*** SVGA 512 kbajtów	***Monitor:*** 14" SVGA kolor
	Pamięć RAM: 16 Mbajtów	***System operacyjny:*** MS-DOS 6.20 + MS Windows 3.1		

W stacji o numerze 3 posadowiono NMP-35A Network Monitor Software for MS-DOS/Windows 3, #802.3/802.4 Ver 1.1, firmy AEG/Modicon/Computrol.

Po zainstalowaniu odpowiedniego oprogramowania dokonano zmian w plikach systemowych komputera. W przypadku stacji 1 i 2 do CONFIG.SYS wprowadzono dodatkowy wiersz o zawartości:

DEVICE=[*ścieżka_dostępu*]\ICPDOSDR.EXE 0X100/P 11/I 5/C

a do AUTOEXEC.BAT:

SET CONFIG_DIR=[*ścieżka_dostępu*]

gdzie [*ścieżka_dostępu*] wskazuje na katalog, w którym posadowiono plik ICPDOSDR.EXE i pliki konfiguracyjne, o których będzie mowa poniżej.

W przypadku stacji 3 (monitor IEEE 802.4) do CONFIG.SYS wprowadzono dodatkowy wiersz o zawartości:

DEVICE=[*ścieżka_dostępu*]\ICPDOSDR.EXE 0X100/P 11/I 5/C

a do AUTOEXEC.BAT:

SET NM=[*ścieżka_dostępu*]

gdzie [*ścieżka_dostępu*] wskazuje na katalog, w którym posadowiono plik ICPDOSDR.EXE i oprogramowanie monitora sieci (PCNM4.EXE i PCNM4.IMG).

Po dokonaniu zmian w plikach CONFIG.SYS i AUTOEXEC.BAT zresetowano komputer i odpowiednio skonfigurowano parametry karty sieciowej. W tym celu uruchamiono MAPGO.EXE, a następnie program konfiguracyjny MAPSMC.EXE. Przykładową konfigurację karty w stacji 1 (stacja o adresie sieciowym 39840F454E45000170010001A02601) przytoczono w tablicy. 6.2.

Tablica 6.2. Parametry konfiguracyjne stacji 1 z segmentu IEEE 802.4

I. Display/Modify Non-volatile RAM Parameters	
A. Station Address	0x70010001A026
B. Group Address Mask	0xFFFFFFFFFFFF
C. Individual Address Mask	0xFFFFFFFFFFFF
D. Slot Time Value	0x0100
E. Modem Selection Value	0x03
F. MAC Transmission Priority	0x07
G. No Stat Tracking (0=N,1=Y)	0x01
H. In Ring Desired (0=N,1=Y)	0x01
I. Prescalar Mode (0->6,1->3)	0x01
J. High Priority THT	0x03FF
K. TRT(4)	0x7FFF
L. TRT(2)	0x7FFF
M. TRT(0)	0x7FFF
N. TRT Ring Maintenance	0x7FFF
O. RM Initial Value	0x7FFF
P. Max Inter Solicit Count	0x10
R. PTP Register Value	0x9A55
A. Network PDU Flags	0x0001
B. Network Config Timer (Sec)	0x00FF
C. Max NPDU Segment Size	0x1FE0
D. NPDU Lifetime (500ms unts)	0x28
E. Subnet Size of NSAP	0x02
F. Network Link SAP Option	0xFE
G. Network QOS Option Value	0x00
H. Netwk Priority Option Val	0x00
I. Netwk Padding Option Size	0x00
J. Record of Rte Option Size	0x00
K. Intermediate Bcast Addr	0x09002B000005
L. End System Bcast Addr	0x09002B000004

A. FDB Pool Size	0x004B
B. BDB Pool Size	0x0096
C. Reserved	0x0096
D. Download Server MAC Addr	0x000000000000
E. Broadband Modem Channel	0x00
F. Broadband Modem Xmit Level	0x00
G. Broadband Modem User Data	0x000000000000

II. Display/Modify MAC Layer Parameters

Current TBC transmit priority: 7

III. Display/Modify Transport Layer Parameters

1. Maximum TPDU size (in bytes)
   Current maximum TPDU size: 8192
2. Maximum Number of Connections
   Current maximum number of connections: 64
3. Display/Modify Minimum Credit
   Current minimum credit: 1
4. Display/Modify Maximum Credit
   Current maximum credit: 8
5. Maximum Number of Retries
   Current maximum number of retries: 4
6. Retry Timer for Data (in seconds)
   Current retry timer (in seconds): 15
7. Retry Timer for Expedited Data (in seconds)
   Current expedited data retry timer (in seconds): 15
8. Window Timer (in seconds)
   Current window timer (in seconds): 12
9. Reference Wait Timer (in seconds)
   Current reference wait timer (in seconds): 30

IV. Display/Modify System Parameters

1. Display BDB Count
   Current BDB count: 123
2. Display/Modify User Limit
   Current system user limit: 75
3. Display/Modify Management Limit
   Current system management limit: 37
4. Display/Modify Network Limit
   Current system network limit: 112

Po skonfigurowaniu karty dołączono dwa tekstowe pliki konfiguracyjne o nazwach UMAP_2.DIR i SUIC.CFG, po czym stacja była gotowa do pracy (pliki konfiguracyjne są zbędne

w przypadku monitora IEEE 802.4). Przykłady zawartości wspomnianych plików konfiguracyjnych zamieszczono w tablicach 6.3 i 6.4.

Tablica 6.3. Plik konfiguracyjny UMAP_2.DIR stacji 1

```
% Section 1:  NODE NAME and MAPPING DIRECTIVE
init.\local
% Section 2:  LOCAL S-Selector Consumers
uMAP.pres\0001
% Section 3:  LOCAL T-Selector Consumers
uMAP.session\0001
% Section 4:  LOCAL N-Selector Consumers
uMAP.xport\39840f454e45000170010001A02601
% Section 5:  LOCAL L-Selector Consumers
uMAP.network\fe
% Section 6:  LOCAL & REMOTE Presentation Address and Application Entity Titles
#
MMSTEST1\00000040\\\\\
MMSTEST2\00000041\\\\\
MMSTEST3\00000042\\\\\
st16.XAR1\00000040\0001\0001\39840f454e45000170010001A02401\\
st16.XAR2\00000041\0001\0001\39840f454e45000170010001A02401\\
st16.XAR3\00000042\0001\0001\39840f454e45000170010001A02401\\
% END of FILE
```

Tablica. 6.4. Plik konfiguracyjny SUIC.CFG stacji 1

```
#1 Channel Configuration
TOTAL=4
BASE_ACSE=0
NUM_ACSE=4
#2 Names to be activated
MMSTEST1:ACSE30_LLP
MMSTEST2:ACSE30_LLP
MMSTEST3:ACSE30_LLP
#3 Names to be registered on channels (channels range from 0..n)
MMSTEST1
MMSTEST2
MMSTEST3
#4 Channels to be listening (called).
0
#5 End of File
```

Badania polegały na wielokrotnym (ponad 100 - krotnym), a następnie repetycyjnym wywoływaniu następujących usług: Initiate (Inicjacja), Abort (Przerwanie), Cancel (Kasowanie) oraz Conclude (Zakończenie), a ponadto: GetNameList (Pobranie_Listy_Nazw), Identify (Identyfikacja), Status (Status), UnsolicitedStatus (Status_Spontaniczny) oraz ObtainFile (Uzyskanie_Pliku), a wreszcie sekwencji tych usług. Nie stwierdzono przypadku wadliwej realizacji żadnej z nich. W badaniach wykorzystano również monitor sieci (stacja 3), który służył do podglądu bieżącej rejestracji zjawisk zachodzących w sieci i zbierania odpowiednich statystyk. Uzyskane w ten sposób w trakcie eksperymentu statystyki nie zawierały informacji o błędnym działaniu testowanego zestawu. Na tej podstawie przyjęto, że opracowane oprogramowanie zachowuje się poprawnie w sieci komputerów.

6.3. Lokalizacja implementacji modelu VMD na platformie sprzętowej sterownika robota URP i jej badania

Układ sterowania robota URP składa się z jednostki centralnej MV-52 i od jednego do ośmiu sterowników osi MV-20. W robotach przemysłowych URP sterowanie ruchem robota polega na odczycie stanu i zadawaniu przyrostów ruchu do sterowników osi przez autonomiczny program wykonywany w jednostce MV-52. Koncepcja sterowania przyjęta do badań dla robota URP zakłada, że trajektoria ruchu i związane z tym nastawy dla modułów MV-20 są obliczane przez program wykonywany przez zewnętrzny komputer IBM PC, a pakiet jednostki centralnej MV-52 odgrywa rolę przekaźnika pomiędzy komputerem a sterownikami osi.

Pakiet jednostki centralnej MV-52 jest przeznaczony do pracy w sterownikach przemysłowych, w tym w układach sterowania robotów. Pakiet pracuje na magistrali kasety 16-bitowej typu AMS-M. Jego najważniejsze dane techniczne są następujące:

- mikroprocesor typu 80186, 16-bitowy
- koprocesor numeryczny 8087-1
- pamięć danych RAM 256 KB, z podtrzymaniem bateryjnym
- pamięć programu EPROM 512 KB
- pamięć parametrów EEPROM 64 KB
- pojemność adresowania na magistrali kasety 16 MB
- interfejsy szeregowe: 2 interfejsy V24, z czego jeden dodatkowo może pracować jako prądowy 0..20 mA z oddzieleniem galwanicznym

Robot URP może być sterowany programem wykonywanym w zewnętrznym komputerze IBM PC, komunikującym się z układem sterowania robota przez interfejs szeregowy RS-232. Od strony jednostki centralnej układu sterowania robota transmisję realizuje kanał B układu Z8530, wyprowadzony na dolne, 25-pinowe złącze listwy czołowej pakietu MV-52. Kanał ten zostaje zaprogramowany do transmisji asynchronicznej w trybie generowania przerwań od pustego bufora nadawczego, pełnego bufora odbiorczego i zmiany stanu linii sterującej DCD (wyprowadzonej na pin 6 złącza zamiast linii DSR).

Restart programu jednostki centralnej MV-52, umożliwiającego sterowanie robotem URP z komputera zewnętrznego następuje po załączeniu zasilania części cyfrowej sterownika robota lub po wciśnięciu przycisku RESET na listwie czołowej pakietu MV-52, pod warunkiem założenia co

najmniej jednej zwory pomiędzy zaciskami 1-2 lub/i 3-4 na polu krosowym K7 tego pakietu. Załączenie zasilania lub użycie przycisku RESET bez żadnej zwory na polu krosowym K7 powoduje, że kontrolę nad układem sterowania robota przejmuje program MONITOR OPERATORSKI, skąd przejście w tryb zewnętrznego sterowania robotem URP można wykonać wykorzystując funkcję Z.

Podczas restartu programu umożliwiającego sterowanie robotem URP z komputera zewnętrznego programowany jest wewnętrzny sterownik przerwań i timer mikroprocesora 80186, oba zewnętrzne sterowniki przerwań PIC 8259A pakietu MV-52 i dwukanałowy układ transmisji szeregowej Z8530. Oba kanały układu Z8530 zostają zaprogramowane do pracy asynchronicznej z następującymi parametrami transmisji:

- prędkość 9600 bitów/s,
- kontrolą parzystości,
- długością słowa równą 8 bitom,
- dwoma bitami stopu.

przy czym kanał A jest ustawiany w tryb pracy z odczytem stanu układu, kanał B - w tryb pracy ze zgłaszaniem przerwań. Ponadto w trakcie restartu do wszystkich sterowników osi MV-20 robota URP zostają wysłane słowa sterujące zawierające zera na wszystkich pozycjach.

Sterowanie robotem URP z komputera zewnętrznego jest wykonywane przy pomocy kilku różnych przesyłek o ściśle określonych formatach. Program obsługujący transmisję od strony sterownika MV-52 sygnalizuje gotowość przyjęcia informacji z komputera zewnętrznego poprzez ustawienie linii DTR na pinie 20 kanału B układu Z8530 w stan wysoki (ok. +10V). Z kolei informacja ze sterownika może być wysłana tylko wtedy, gdy na pinie 6 tego kanału (linia DSR, dołączona do wyprowadzenia DCD kanału B) komputer współpracujący ustawił stan wysoki. Stan obu linii sterujących synchronizujących transmisję kanałem B układu Z8530 sygnalizują dodatkowo dwie zielone lampki LED na listwie czołowej pakietu MV-52: lampka lewa odpowiada linii DSR, lampka prawa - linii DTR, a świecenie danej lampki oznacza stan wysoki (ok. +10V) na odpowiadającej jej linii.

Program sterujący robotem URP, wykonywany w sterowniku MV-52, odczytuje dodatkowo stan odbiornika kanału A układu transmisji szeregowej Z8530. Odczyt znaku CTRL-Q (11H, inne znaki są ignorowane) powoduje wysłanie odpowiedniego komunikatu do kanału A tego układu, przekazanie sterowania do instrukcji pod adresem FFFF:0000H (FFFF0H) i w konsekwencji restart programu w sposób określony obecnością zwor na polu krosowym K7.

Sterując robotem URP przy pomocy programu wykonywanego przez zewnętrzny komputer IBM PC należy mieć świadomość, że przesłanie pojedynczego bajtu każdej przesyłki łączem szeregowym przy szybkości transmisji równej 9600 bodów trwa ponad 1 milisekundę. Oznacza to, że użytkownik w programie sterującym musi dodatkowo uwzględnić czas potrzebny na transmisję z/do sterownika, przy założeniu, że czas reakcji samego pakietu MV-52 jest pomijalny.

Przedstawione wyżej właściwości układu sterowania robota URP wykorzystano przy opracowaniu dla niego lokalizacji implementacji modelu VMD. Mianowicie poszczególne funkcje rozdzielono między komputer IBM PC oraz jednostkę centralną MV-52. Od strony sieci jest widoczny

komputer IBM PC, wyposażony i skonfigurowany jak to opisano w p. 6.2 (stacja 1). Posadowiono w nim wersję programu MMS_PIAP wzbogaconego o warstwę komunikacyjną po łączu szeregowym, umożliwiającą bezpośrednią ingerencję w działanie sterownika robota URP. Od strony robota widoczny jest układ sterowania URP z nieznacznie zmodyfikowanym oprogramowaniem. Oprogramowanie to umożliwia bezpośrednie sterowanie pracą robota z oddalonego komputera IBM PC. Schemat działania jest następujący. Po nawiązaniu połączenia sieciowego między dwoma komputerami (stacje 1 i 2 wg rys. 6.1) pracującymi w sieci z protokołem MMS (usługa Initiate), do komputera bezpośrednio współpracującego z robotem (stacja 1) przekazywany jest program działania robota. Program ten, odpowiednio zinterpretowany, jest zdalnie uruchamiany przy wykorzystaniu łącza szeregowego w w sterowniku robota URP. Szczegóły programowe (format przesyłek i zasady wzajemnej komunikacji między sterownikiem robota URP a komputerem) opisano dalej w p. 6.3.1 i 6.3.2.

W celu weryfikacji opisanej przed chwilą opracowano program testowy dla robota URP. Polega on na wstępnej synchronizacji osi robota, potem na wykonywaniu ruchów ramienia robota przy zmienianych cyklicznie inkrementach przesunięć po kolejnych osiach. Cztery pięciogodzinne testy układu robot - komputer IBM PC z wykorzystaniem wspomnianego programu nie wykazały wadliwej pracy, w związku z czym uznano przyjęte rozwiązanie za poprawne.

6.3.1. Specyfikacja przesyłek do sterowania robotem URP przy pomocy zewnętrznego komputera

Zapytanie o liczbę osi robota

Przesyłka z komputera (1-bajtowa):

7	6	5	4	x	x	x	x

- **bity 7, 6, 5, 4:** Kod przesyłki = 0001
- **bity 3, 2, 1, 0:** Bez znaczenia

Odpowiedź ze sterownika (1-bajtowa):

7	6	5	4	x	2	1	0

- **bity 7, 6, 5, 4:** Kod przesyłki = 0001
- **bit 3:** Bez znaczenia
- **bity 2, 1, 0:** Określają liczbę osi robota:

  000 - jedna oś,
  001 - dwie osie,
  010 - trzy osie,
  011 - cztery osie,
  100 - pięć osi
  101 - sześć osi,
  110 - siedem osi,
  111 - osiem osi.

Zapytanie o słowo stanu osi

Przesyłka z komputera (1-bajtowa):

7	6	5	4	x	2	1	0

- **bity 7, 6, 5, 4:** Kod przesyłki = 0010
- **bit 3:** Bez znaczenia

- **bity 2, 1, 0:** Określają numer osi robota:

000 - pierwsza oś,
001 - druga oś,
010 - trzecia oś,
011 - czwarta oś,
100 - piąta oś,
101 - szósta oś,
110 - siódma oś,
111 - ósma oś.

Odpowiedź ze sterownika (2-bajtowa):

1. Pierwszy bajt:

7	6	5	4	x	2	1	0

- **bity 7, 6, 5, 4:** Kod przesyłki = 0010
- **bit 3:** Bez znaczenia
- **bity 2, 1, 0:** Określają numer osi robota:

000 - pierwsza oś,
001 - druga oś,
010 - trzecia oś,
011 - czwarta oś,
100 - piąta oś,
101 - szósta oś,
110 - siódma oś,
111 - ósma oś.

2. Drugi bajt:

7	6	5	x	x	2	1	0

- **bit 7:** WRITEEN_IL - informacja, że jest zezwolenie na zapis nowych przyrostów ruchu dla wszystkich osi robota (jest to iloczyn logiczny bitów WRITEEN wszystkich osi robota):

0 - brak zezwolenia na zapis,
1 - jest zezwolenie na zapis.

- **bit 6:** WRITEEN

0 - brak zezwolenia na zapis,
1 - jest zezwolenie na zapis.

- **bit 5:** BUDZIK - informacja że zawiesił się program sterownika i wymagany jest restart sterownika ze strony programu głównego (PROGRST):

0 - program nie zawiesił się,
1 - program zawiesił się.

- **bit 4:** Bez znaczenia
- **bit 3:** Bez znaczenia
- **bit 2:** ERROR - informacja, że błąd położenia jest zbyt duży:

0 - brak błędu,
1 - jest błąd.

- **bit 1:** INPOS - informacja o błędzie położenia w strefie zerowej:

0 - brak błędu,
1 - jest błąd.

- **bit 0:** OSZSYNCHR: - informacja o zsynchronizowaniu osi osi:

0 - oś niezsynchronizowana
1 - 1 - oś zsynchronizowana

Zapytanie o zsynchronizowanie osi

Przesyłka z komputera (1-bajtowa):

7	6	5	4	x	x	x	x

- **bity 7, 6, 5, 4:** kod przesyłki = 0011
- **bity 3, 2, 1, 0:** Bez znaczenia

Odpowiedź ze sterownika (2-bajtowa):

1. Pierwszy bajt:

7	6	5	4	x	x	x	x

- **bity 7, 6, 5, 4:** Kod przesyłki = 0011
- **bity 3, 2, 1, 0:** Bez znaczenia

2. Drugi bajt:

7	6	5	4	x	2	1	0

- **bit 7:** Informacja o zsynchronizowaniu osi 8:
  0 - oś niezsynchronizowana,
  1 - oś zsynchronizowana.
- **bit 6:** Informacja o zsynchronizowaniu osi 7,
- **bit 5:** Informacja o zsynchronizowaniu osi 6,
- **bit 4:** Informacja o zsynchronizowaniu osi 5,
- **bit 3:** Informacja o zsynchronizowaniu osi 4,
- **bit 2:** Informacja o zsynchronizowaniu osi 3,
- **bit 1:** Informacja o zsynchronizowaniu osi 2,
- **bit 0:** Informacja o zsynchronizowaniu osi 1.

Zapytanie o zezwolenie na zapis nowych przyrostów ruchu

Przesyłka z komputera (1-bajtowa):

7	6	5	4	x	x	x	x

- **bity 7, 6, 5, 4:** Kod przesyłki = 0100
- **bity 3, 2, 1, 0:** Bez znaczenia

Odpowiedź ze sterownika (2-bajtowa):

1. Pierwszy bajt:

7	6	5	4	x	x	x	x

- **bity 7, 6, 5, 4:** Kod przesyłki = 0100
- **bity 3, 2, 1, 0:** Bez znaczenia

2. Drugi bajt:

7	6	5	4	3	2	1	0

- **bit 7:** Informacja o zezwoleniu na wykonanie zapisu nowego przyrostu ruchu dla osi 8:
  0 - brak zezwolenia,
  1 - jest zezwolenie.
- **bit 6:** Informacja o zezwoleniu na wykonanie zapisu nowego przyrostu ruchu dla osi 7,
- **bit 5:** Informacja o zezwoleniu na wykonanie zapisu nowego przyrostu ruchu dla osi 6,
- **bit 4:** Informacja o zezwoleniu na wykonanie zapisu nowego przyrostu ruchu dla osi 5,

- **bit 3:** Informacja o zezwoleniu na wykonanie zapisu nowego przyrostu ruchu dla osi 4,
- **bit 2:** Informacja o zezwoleniu na wykonanie zapisu nowego przyrostu ruchu dla osi 3,
- **bit 1:** Informacja o zezwoleniu na wykonanie zapisu nowego przyrostu ruchu dla osi 2,
- **bit 0:** Informacja o zezwoleniu na wykonanie zapisu nowego przyrostu ruchu dla osi 1.

Rozkaz synchronizacji wszystkich osi

Przesyłka z komputera (1-bajtowa):

7	6	5	4	x	x	x	x

- **bity 7, 6, 5, 4:** Kod przesyłki = 0101
- **bity 3, 2, 1, 0:** Bez znaczenia

Odpowiedź ze sterownika:

Brak odpowiedzi

Przesyłka ze słowem sterującym danej osi

Przesyłka z komputera (2-bajtowa):

1. Pierwszy bajt:

7	6	5	4	x	2	1	0

- **bity 7, 6, 5, 4:** Kod przesyłki = 0110
- **bit 3:** Bez znaczenia
- **bity 2, 1, 0:** Określają numer osi robota:

000 - pierwsza oś,
001 - druga oś,
010 - trzecia oś,
011 - czwarta oś,
100 - piąta oś,
101 - szósta oś,
110 - siódma oś,
111 - ósma oś.

2. Drugi bajt:

x	x	x	x	x	2	1	0

- **bity 7, 6, 5, 4, 3:** Bez znaczenia,
- **bit 2:** Rozkaz synchronizacji wszystkich osi:
  0 - nie synchronizuj wszystkich osi,
  1 - załączenie synchronizacji wszystkich osi.
- **bit 1:** Rozkaz STOP dla danej osi:
  0 - nie załączaj stopu,
  1 - załączenie stopu osi.
- **bit 0:** Rozkaz synchronizacji danej osi:
  0 - nie synchronizuj danej osi,
  1 - załączenie synchronizacji danej osi.

Działanie:

- jeśli bit 2 jest równy 1 (synchronizacja wszystkich osi), to do każdego sterownika osi jednostka MV-52 wysyła 1-bajtowy rozkaz nakazujący synchronizację tej osi (bez badania,

· **bity 7, 6, 5, 4, 3, 2, 1, 0:** LOW - określają położenie w ramach jednego obrotu osi silnika (rozdzielczość 1/256 obrotu).

Przesyłka z przyrostem zadanym

Przesyłka z komputera (3-bajtowa):

1. Pierwszy bajt:

7	6	5	4	x	2	1	0

· **bity 7, 6, 5, 4:** Kod przesyłki = 1000
· **bit 3:** Bez znaczenia
· **bity 2, 1, 0:** Określają numer osi robota:
000 - pierwsza oś,
001 - druga oś,
010 - trzecia oś,
011 - czwarta oś,
100 - piąta oś,
101 - szósta oś,
110 - siódma oś,
111 - ósma oś.

2. Drugi bajt:

7	6	5	4	3	x	1	0

· **bit 7:** Znak przyrostu ruchu:
0 - ruch ujemny, DO TYŁU, przeciwny do ruchu wskazówek zegara,
1 - ruch dodatni, DO PRZODU, zgodny z ruchem wskazówek zegara.
· **bity 6, 5, 4, 3:** Czas wykonania pojedynczego rozkazu ruchu:
1000 - czas 64 milisekundy,
0100 - czas 32 milisekundy,
0010 - czas 16 milisekundy,
0001 - czas 8 milisekund.
· **bit 2:** Bez znaczenia.
· **bity 1, 0:** Najstarsze dwa bity określające przyrost ruchu (bit 1 jest ignorowany jeśli czas wykonania pojedynczego rozkazu ruchu wynosi 32, 16 lub 8 milisekund, bit 1 i 0 są ignorowane jeśli czas wykonania pojedynczego rozkazu ruchu wynosi 16 lub 8 milisekund).

3. Trzeci bajt:

7	6	5	4	3	2	1	0

· **bity 7, 6, 5, 4, 3, 2, 1, 0:** Najmłodsze bity określające przyrost ruchu (bit 7 jest ignorowany jeśli czas wykonania pojedynczego rozkazu ruchu wynosi 8 milisekund).

Odpowiedź ze sterownika:

Brak odpowiedzi.

Przesyłka z poleceniem odczytania 8-bitowego portu

Przesyłka z komputera (3-bajtowa):

1. Pierwszy bajt:

7	6	5	4	x	x	x	x

· **bity 7, 6, 5, 4:** Kod przesyłki = 1001

1h0

czy jest ona zsynchronizowana, czy nie). Rozkaz ten nie zawiera polecenia STOP OSI nawet jeśli w przesyłce z komputera IBM PC był on wymieniony).

- jeśli bit 2 jest równy 0 (brak polecenia synchronizacji wszystkich osi), to do sterownika danej osi (o numerze zawartym w przesyłce) przesyłany jest rozkaz zawierający polecenia zawarte na dwóch ostatnich bajtach przesyłki z komputera IBM PC.

Odpowiedź ze sterownika:
Brak odpowiedzi

Przesyłka z pytaniem o pozycję osi
Przesyłka z komputera (1-bajtowa):

7	6	5	4	x	2	1	0

- **bity 7, 6, 5, 4:** Kod przesyłki = 0111
- **bit 3:** Bez znaczenia
- **bity 2, 1, 0:** Określają numer osi robota:

000 - pierwsza oś,
001 - druga oś,
010 - trzecia oś,
011 - czwarta oś,
100 - piąta oś,
101 - szósta oś,
110 - siódma oś,
111 - ósma oś.

Odpowiedź ze sterownika (przesyłka 3-bajtowa):
1. Pierwszy bajt:

7	6	5	4	x	2	1	0

- **bity 7, 6, 5, 4:** Kod przesyłki = 0111
- **bit 3:** Bez znaczenia
- **bity 2, 1, 0:** Określają numer osi robota:

000 - pierwsza oś,
001 - druga oś,
010 - trzecia oś,
011 - czwarta oś,
100 - piąta oś,
101 - szósta oś,
110 - siódma oś,
111 - ósma oś.

2. Drugi bajt:

7	6	5	4	3	2	1	0

- **bity 7, 6, 5, 4, 3, 2, 1, 0:** HIGH - określają numer kolejnego obrotu osi silnika (zakres ruchu 256 obrotów).

3. Trzeci bajt:

7	6	5	4	3	2	1	0

141

- **bity 3, 2, 1, 0:** Bez znaczenia

2. Drugi bajt:

7	6	5	4	3	2	1	0

- **bity 7, 6, 5, 4, 3, 2, 1, 0:** HIGH_ADDRESS - starszy bajt 16-bitowego adresu portu WE,

3. Trzeci bajt:

7	6	5	4	3	2	1	0

- **bity 7, 6, 5, 4, 3, 2, 1, 0:** LOW_ADDRESS - młodszy bajt 16-bitowego adresu portu WE.

Odpowiedź ze sterownika (2-bajtowa):

1. Pierwszy bajt:

7	6	5	4	x	x	x	x

- **bity 7, 6, 5, 4:** Kod przesyłki = 1001
- **bity 3, 2, 1, 0:** Bez znaczenia

2. Drugi bajt:

7	6	5	4	3	2	1	0

- **bity 7, 6, 5, 4, 3, 2, 1, 0:** Odczytana wartość z portu 8-bitowego.

Przesyłka z poleceniem odczytania 16-bitowego portu

Przesyłka z komputera (3-bajtowa):

1. Pierwszy bajt:

7	6	5	4	x	x	x	x

- **bity 7, 6, 5, 4:** Kod przesyłki = 1010
- **bity 3, 2, 1, 0:** Bez znaczenia

2. Drugi bajt:

7	6	5	4	3	2	1	0

- **bity 7, 6, 5, 4, 3, 2, 1, 0:** HIGH_ADDRESS - starszy bajt 16-bitowego adresu portu WE,

3. Trzeci bajt:

7	6	5	4	3	2	1	0

- **bity 7, 6, 5, 4, 3, 2, 1, 0:** LOW_ADDRESS - młodszy bajt 16-bitowego adresu portu WE.

Odpowiedź ze sterownika (3-bajtowa):

1. Pierwszy bajt:

7	6	5	4	x	x	x	x

- **bity 7, 6, 5, 4:** Kod przesyłki = 1010
- **bity 3, 2, 1, 0:** Bez znaczenia

2. Drugi bajt:

7	6	5	4	3	2	1	0

- **bity 7, 6, 5, 4, 3, 2, 1, 0:** HIGH_VALUE - starszy bajt odczytanej wartości 16-bitowego.

3. Trzeci bajt:

7	6	5	4	3	2	1	0

- **bity 7, 6, 5, 4, 3, 2, 1, 0:** LOW_VALUE - młodszy bajt odczytanej wartości 16-bitowego.

Przesyłka z poleceniem wysterowania 8-bitowego portu

Przesyłka z komputera (4-bajtowa):

1. Pierwszy bajt:

7	6	5	4	x	x	x	x

- **bity 7, 6, 5, 4:** Kod przesyłki = 1011
- **bity 3, 2, 1, 0:** Bez znaczenia

2. Drugi bajt:

7	6	5	4	3	2	1	0

- **bity 7, 6, 5, 4, 3, 2, 1, 0:** HIGH_ADDRESS - starszy bajt 16-bitowego adresu portu WY,

3. Trzeci bajt:

7	6	5	4	3	2	1	0

- **bity 7, 6, 5, 4, 3, 2, 1, 0:** LOW_ADDRESS - młodszy bajt 16-bitowego adresu portu WY.

4. Czwarty bajt:

7	6	5	4	3	2	1	0

- **bity 7, 6, 5, 4, 3, 2, 1, 0:** Wartość 1-bajtowa, która ma zostać wysłana przez port WY.

Odpowiedź ze sterownika:

Brak odpowiedzi.

Przesyłka z poleceniem wysterowania 16-bitowego portu

Przesyłka z komputera (5-bajtowa):

1. Pierwszy bajt:

7	6	5	4	x	x	x	x

- **bity 7, 6, 5, 4:** Kod przesyłki = 1100
- **bity 3, 2, 1, 0:** Bez znaczenia

2. Drugi bajt:

7	6	5	4	3	2	1	0

- **bity 7, 6, 5, 4, 3, 2, 1, 0:** HIGH_ADDRESS - starszy bajt 16-bitowego adresu portu WY,

3. Trzeci bajt:

7	6	5	4	3	2	1	0

- **bity 7, 6, 5, 4, 3, 2, 1, 0:** LOW_ADDRESS - młodszy bajt 16-bitowego adresu portu WY.

4. Czwarty bajt:

7	6	5	4	3	2	1	0

- **bity 7, 6, 5, 4, 3, 2, 1, 0:** HIGH_VALUE - starszy bajt 16-bitowej wartości, która ma zostać wysłana przez port WY.

5. Piąty bajt:

7	6	5	4	3	2	1	0

- **bity 7, 6, 5, 4, 3, 2, 1, 0:** LOW_VALUE - młodszy bajt 16-bitowej wartości, która ma zostać wysłana przez port WY.

Odpowiedź ze sterownika:

Brak odpowiedzi.

6.3.2. Pakiet do komunikacji sterownika robota URP z zewnętrznym komputerem

Przedstawiony tu pakiet oprogramowania umożliwia podłączenia do komputera IBM PC i obsługi urządzenia zewnętrznego (np. drugiego komputera, układu sterowania robotem URP), wyposażonego w port transmisji szeregowej. Przed wykorzystaniem pakietu użytkownik musi ustalić z którego portu będzie korzystał i przystosować wybrany port do pracy z parametrami transmisji urządzenia zewnętrznego. Wybór portu następuje poprzez ustalenie wartości zmiennej globalnej:
int serial_port;
zadeklarowanej w pliku COM.ASM (plik ten zawiera postać źródłową pakietu komunikacyjnego).

Ustalenie parametrów transmisji wybranego portu można wykonać:

- przy pomocy funkcji bibliotecznej system() dostępnej w bibliotekach dołączanych np. do każdego kompilatora języka C przeznaczonego do tworzenia programów pracujących w środowisku systemu operacyjnego MS DOS,

```
#include <stdlib.h>
extern int serial port;
.....
serial port = 1;
system ("mode com1:9600,e,8,2 > nul:");
```

- lub przy pomocy dołączonej do pakietu funkcji "set_transmission_parameters()" (patrz dalszy opis)

W obecnej wersji pakiet komunikacyjny zapewnia transmisję przez port COM1 lub COM2 komputera IBM PC (nie obsługuje portów COM3 i COM4).

W celu zapewnienia wymiany informacji za pośrednictwem wybranego portu COMx (x = 1, 2) komputera IBM PC kanałowi transmisji przyporządkowano w pamięci operacyjnej dwa bufory:

- programowy bufor odbiorczy, oznaczany w tekście źródłowym programu jako input_buffer,
- programowy bufor nadawczy, oznaczany w tekście źródłowym programu jako output_buffer,

Ponieważ zarówno urządzenie zewnętrzne jak i komputer IBM PC mogą nie nadążyć z interpretacją informacji odbieranej przez każde z nich, więc do podprogramów sterujących wysyłką danych z bufora nadawczego i wpisujących informację do bufora odbiorczego wbudowano dodatkowo mechanizm sterujący transmisją. Od strony komputera IBM PC polega on na wykorzystaniu linii sterujących RTS i CTS wybranego portu COMx (x = 1, 2) w ten sposób, że:

- do urządzenia zewnętrznego informacja jest wysyłana wtedy, gdy linia CTS danego portu COMx (x=1,2) znajduje się w stanie "1" (HIGH),
- linia RTS danego portu COMx (x=1,2) zostaje ustawiona w stan "1" (HIGH), gdy są spełnione warunki odbioru informacji z urządzenia zewnętrznego, a w stan "0" (LOW) - w przeciwnym przypadku. Bliższe informacje na temat sterowania transmisją stanem linii RTS podano dalej, w opisie w dotyczącym bufora odbiorczego.

AHH

Synchronizację transmisji linią CTS można programowo załączać/wyłączać korzystając z funkcji set_synchronization_by_cts() i reset_synchronization_by_cts() (patrz dalszy opis).

Przedstawiony pakiet oprogramowania nie zmienia parametrów transmisji (szybkości, długości znaku, liczby bitów stopu, kontroli parzystości), których ustalenie należy do użytkownika.

Kolejne znaki odbierane w wybranym kanale transmisyjnym COMx (x = 1 ,2) zostają wpisane bezpośrednio, bez interpretacji ich znaczenia do SIZE_INPUT_BUFFER, wymiarowego bufora odbiorczego. Każdemu elementowi tego bufora odpowiada pojedyncza komórka pamięci RAM. Bufor wejściowy jest buforem cyklicznym, tzn. że odbierane znaki są wstawiane do niego sekwencyjnie w kierunku rosnących adresów. Dojście do końca bufora powoduje automatyczne ustawienie indeksu sterującego jego wypełnianiem na początek.

Wypełnianiem i odczytem informacji z bufora odbiorczego sterują dwa następujące ideksy:

. Indeks odczytu z bufora odbiorczego, będący numerem komórki pamięci, liczonym względem początku bufora, zawierającej ostatni odczytany znak z tego bufora. Indeks ten jest zwiększany o jeden bezpośrednio przed odczytaniem znaku z bufora, a po dojściu do końca bufora - ustawiany do ponownego zliczania od zera. W tekstach programów źródłowych jest on oznaczany jako index_from_input_buffer.

. Indeks zapisu do bufora odbiorczego, będący numerem komórki pamięci, liczonym względem początku bufora, zawierającej ostatni odebrany znak z danego kanału transmisji szeregowej. Indeks ten jest zwiększany o jeden bezpośrednio przed wpisaniem do bufora odbiorczego odebranego znaku, a po dojściu do końca bufora - ustawiany do ponownego zliczania od zera. W tekstach programów źródłowych jest on oznaczany jako index_to_input_buffer.

Oprócz obu tych indeksów do obsługi bufora odbiorczego wykorzystano dodatkowo licznik obsługi tego bufora, oznaczany w tekstach źródłowych procedur jako counter_output_buffer który:

- jest zwiększany o jeden bezpośrednio przed wpisaniem do bufora odebranego znaku,
- jest zmniejszany o jeden bezpośrednio po odczytaniu znaku z bufora odbiorczego.

Licznik obsługi bufora odbiorczego, jest wykorzystywany do:

- określania momentu zakończenia czynności odczytu znaków z tego bufora (jeśli wartość licznika wynosi zero),
- określania sytuacji awaryjnej, w której bufor odbiorczy mógłby zostać przepełniony (wartość wpisana do licznika pozostaje wtedy równa rozmiarowi bufora odbiorczego, a stan awaryjny jest sygnalizowany przez ustawienie odpowiedniego wskaźnika błędów w programowym słowie stanu portu COMx (x=1,2). W takim przypadku informacja odczytana z kanału transmisyjnego nie jest wpisywana do tego bufora (zostaje zagubiona),
- określania sytuacji, w której w buforze odbiorczym pozostaje miejsce na wpisanie tylko dwóch 1-bajtowych danych. W takim przypadku linia sterująca RTS portu COMx (x=1,2) zostaje automatycznie ustawiona w stan "0" (LOW), a informacja o tym jest umieszczana w programowym słowie stanu tego portu,
- określania sytuacji, w której w buforze odbiorczym zwolniło się miejsce na wpisanie dziesięciu 1-bajtowych danych. W takim przypadku linia sterująca RTS portu COMx (x=1,2)

zostaje automatycznie ustawiona w stan "1" (HIGH), o ile poprzednio była w stanie "0" (LOW). Ponadto informacja o stanie linii jest umieszczana w programowym słowie stanu tego portu.

Użytkownik wpisuje przesyłkę w postaci ciągu kolejnych znaków stanowiących informację dla urządzenia zewnętrzego do SIZE_OUTPUT_BUFFER, wymiarowego bufora nadawczego danego kanału transmisyjnego. Wpisywana informacja ma już postać zakodowaną zgodnie z formatem przesyłek wymaganym przez urządzenie zewnętrzne. Każdemu elementowi tego bufora odpowiada pojedyncza komórka pamięci RAM. Bufor nadawczy jest buforem cyklicznym, tzn. że znaki są wpisywane do niego sekwencyjnie w kierunku rosnących adresów. Dojście do końca bufora powoduje automatyczne ustawienie indeksu sterującego jego wypełnianiem na początek.
Wypełnianiem i wysyłaniem informacji z bufora nadawczego sterują dwa następujące indeksy:

. Indeks zapisu do bufora nadawczego, będący numerem komórki pamięci, liczonym względem początku bufora, zawierającej ostatni zapisany znak do tego bufora. Indeks ten jest zwiększany o jeden bezpośrednio przed wstawieniem znaku do bufora, a po dojściu do końca bufora - ustawiany do ponownego zliczania od zera. W tekstach programów źródłowych jest on oznaczany jako index_to_output_buffer.

. Indeks wysłanych znaków z bufora nadawczego, będący numerem komórki pamięci, liczonym względem początku bufora, zawierającej ostatni wysłany znak. Indeks ten jest zwiększany o jeden bezpośrednio przed wysłaniem znaku z bufora nadawczego do urządzenia zewnętrznego, a po dojściu do końca bufora - ustawiany do ponownego zliczania od zera. W tekstach programów źródłowych jest on oznaczany jako index_from_output_buffer.

Oprócz obu tych indeksów do obsługi bufora nadawczego wykorzystano dodatkowo licznik obsługi tego bufora, oznaczany w tekstach źródłowych procedur jako counter_output_buffer, który:

- jest zwiększany o jeden bezpośrednio przed wpisaniem 1-bajtowej informacji do bufora nadawczego,
- jest zmniejszany o jeden bezpośrednio po odczytaniu 1-bajtowej informacji i wysłaniu jej do urządzenia zewnętrznego.

Licznik obsługi bufora nadawczego pełni podobną rolę jak jego odpowiednik dla bufora odbiorczego, tzn. określa stan zajętości tego bufora. W przypadku próby przepełnienia bufora nadawczego (wartość licznika jest wtedy równa rozmiarowi bufora nadawczego) informacja nie zostaje wpisana do niego, ale program ustawia jeden ze wskaźników w programowym słowie stanu portu.

Wszystkie istotne informacje na temat transmisji do/z urządzenia zewnętrzego przez dany port COMx (x = 1, 2) są zawarte w programowym słowie stanu tego portu. Poszczególnym wskaźnikom w mniej znaczącym bajcie tego słowa (bardziej znaczący bajt zawiera same zera) przyporządkowano nastęującą informację:

Deklaracja słowa wskaźnikow transmisji portu szeregowego COMx (bardziej znaczacy bajt jest wyzerowany) jest następująca:

7	6	5	4	3	2	1	0

- **bit 7:** Pusty bufor odbiorczy:
  0 - bufor odbiorczy zawiera nieodczytaną informację,
  1 - całą odebraną informację wpisaną do bufora odbiorczego już odczytano.
- **bit 6:** Przepełnienie bufora odbiorczego:
  0 - żadna informacja odebrana z kanału transmisyjnego COMx nie została zagubiona (bufor zawieral dość miejsca aby ją pomieścić),
  1 - informacja odebrana z kanału transmisyjnego COMx została zagubiona (bufor był przepelniony i nie zawieral miejsca aby ja pomiescic),
- **bit 5:** Pusty bufor nadawczy:
  0 - bufor nadawczy zawiera niewysłaną informację,
  1 - bufor nadawczy jest pusty (wszystko z niego wysłano)
- **bit 4:** Przepełnienie bufora nadawczego:
  0 - wpisano wartość do bufora nadawczego,
  1 - bufor nadawczy jest pełny, a więc nie przyjął wpisywanej wartości.
- **bit 3:** Framing Error:
  0 - brak błędu Framing Error,
  1 - wystapił błąd Framing Error.
- **bit 2:** Parity Error:
  0 - brak błędu Parity Error,
  1 - wystapił błąd Parity Error.
- **bit 1:** Overrun Error:
  0 - brak błędu Overrun Error,
  1 - wystąpił błąd Overrun Error.
- **bit 0:** Informacja o stanie linii RTS:
  0 - RTS = 0,
  1 - RTS = 1.

Pakiet zawiera następujące procedury obslugi przerwaniowej wybranego portu transmisji szeregowej COMx (x = 1, 2):
Procedury **niedostępne** dla użytkownika:

. **int_com()**
obsluga przerwania zglaszanego przez dany port transmisji szeregowej,

. **input()**
odczytanie znaku z kanału transmisji szeregowej i wpisanie go do programowego bufora odbiorczego,

. **output()**
odczytanie 1-bajtowego znaku z programowego bufora nadawczego i wpisanie go do sprzetowego bufora nadawczego danego portu szeregowego,

Procedury **dostępne** dla użytkownika:

. **read_com()**
odczytanie pojedynczego znaku (bajtu) z programowego bufora odbiorczego portu transmisji szeregowej,

. **write_com()**
wpisanie pojedynczego znaku (bajtu) do programowego bufora nadawczego portu trasnsmisji szeregowej oraz zainicjowanie transmisji,

147

. **open_com()**
wykonanie wszystkich czynnosci zwiazanych z przygotowaniem danego portu szeregowego do pracy w trybie przerwaniowym,
. **close_com()**
wykonanie zamiany wektora przerwania nr 12 (lub 11) będącego wskaźnikiem do podprogramu int_com() na oryginalny wektor, ustawiony przez system operacyjny MS DOS,
. **set_com_addresses()**
ustawienie zmiennych przechowujących adresy I/O rejestrow obslugujących dany port transmisji szeregowej,
. **clear_com()**
ustawienie w stan poczatkowy wszystkich zmiennych sterujących zapisem/odczytem do/z programowych buforow nadawczego i odbiorczego danego portu szeregowego,
. **clear_input_buffer()**
ustawienie wszystkich zmiennych sterujących odczytem/zapisem z/do programowego bufora odbiorczego danego portu szeregowego,
. **clear_output_buffer()**
ustawienie wszystkich zmiennych sterujących odczytem/zapisem z/do programowego bufora nadawczego danego portu szeregowego,
. **free_space_in_output_buffer()**
odczyt ilosci wolnego miejsca w programowym buforze nadawczym danego portu szeregowego,
. **contents_input_buffer()**
odczyt ilosci jeszcze nieodczytanych 1-bajtowych znakow wypelniających programowy bufor odbiorczy portu szeregowego,
. **set_rts()**
ustawienie w stan 1 (HIGH) linii RTS danego portu szeregowego,
. **reset_rts()**
ustawienie w stan 0 (LOW) linii RTS danego portu szeregowego,
. **read_status_word()**
odczytanie programowego słowa stanu danego portu szeregowego,
. **clear_transmission_errors()**
wyzerowanie w programowym słowie stanu danego portu szeregowego wskaźnikow bledow tranmisji,
. **read_modem_status_register()**
odczytanie rejestru zawierającego informacje o stanie linii modemowych danego portu szeregowego,
. **set_synchronization_by_cts()**
odblokowanie synchronizacji nadawania przez obslugiwany port szeregowy komputera. Od tego momentu port bedzie nadawal tylko wtedy, gdy na jego linii sterującej bedzie "1",
. **reset_synchronization_by_cts()**
zadblokowanie synchronizacji nadawania przez obslugiwany port szeregowy komputera. Od tego momentu port bedzie nadawal bez wzgledu na stan linii sterującej CTS,
. **read_synchronization_state()**
odczyt informacji o załączeniu badz wylaczeniu synchronizacji nadawania przez port szeregowy poprzez stan jego linii sterującej CTS,

set_transmission_parameters()
ustawienie parametrów transmisji danego portu transmisji szeregowej.

Poniżej podano szczegółowe opisy poszczególnych, wymienionych wyżej procedur.

Procedura int_com()
Procedura int_com() jest podprogramem obsługi przerwania zgłaszanego przez port transmisji szeregowej COMx (x = 1, 2) komputera IBM PC. W zależności od przyczyny przerwania wykonuje ona następujące czynności:

. Przyczyna: Błąd typu Overrun Error, Parity Error, Framing Error.
Wystąpienie jakiegokolwiek błędu transmisji (Overrun Error, Parity Error, Framing Error) powoduje sprzętowe ustawienie informacji o tym błędzie w rejestrze tzw. LINE CONTROL REGISTER danego portu, a następnie zgłoszenie przerwania. Obsługa tego przerwania polega na odczytaniu zawartości LINE CONTROL REGISTER (co zeruje wskaźniki błędów w tym rejestrze), a następnie obliczeniu sumy logicznej (OR) odczytanych wskaźników z ich odpowiednikami z programowego słowa stanu tego portu i umieszczeniu wyniku w słowie stanu.

. Przyczyna: Odebranie informacji z urządzenia zewnętrznego.
Obsługa przerwania spowodowanego tą przyczyną polega na wywołaniu podprogramu input().

. Przyczyna: Pusty bufor nadawczy portu COMx (x=1,2).
Obsługa przerwania spowodowanego tą przyczyną polega na wywołaniu podprogramu output().

. Przyczyna: Zmiana stanu linii sterujących portu COMx (x=1,2)
Jeśli zmiana stanu linii sterujących dotyczy linii DSR, DCD lub RI, to nie są podejmowane żadne czynności poza wysłaniem programowej instrukcji EOI do sterownika przerwań PIC 8259A. Jeśli zmiana stanu dotyczy linii CTS, to:

- w przypadku zmiany stanu z "1" (HIGH) na "0" (LOW) zablokowane zostają przerwania od pustego bufora nadawczego portu COMx (x=1,2), ponieważ obecny stan linii CTS nie pozwala na transmisję do urządzenia zewnętrznego,
- w przypadku zmiany stanu z "0" (LOW) na "1" (HIGH) odblokowane zostają przerwania od pustego bufora nadawczego portu COMx (x=1,2).

Wysłanie programowej instrukcji EOI do sterownika przerwań PIC 8259A nie następuje automatycznie po zakończeniu obsługi rzeczywistego źródła przerwania, lecz po stwierdzeniu, że w trakcie wykonywania tej obsługi port COMx (x = 1, 2) nie zgłosił przerwania spowodowanego inną przyczyną.
Wartość zwracana:
Procedura nie zwraca żadnej wartości.

Procedura input()
Procedura input() odczytuje znak z kanału transmisji szeregowej COMx (x = 1, 2) i umieszcza go w programowym buforze odbiorczym. Jeśli w buforze tym nie ma miejsca na odczytaną wartość, to zostaje ona zgubiona, ale informację o tym fakcie procedura zamieszcza w programowym słowie stanu portu COMx (x = 1, 2). Ponadto procedura modyfikuje wartości następujących zmiennych:

- index_to_input_buffer
- counter_input_buffer
- status_word

Procedura input() ustawia także linię RTS portu COMx w stan "0" (LOW), (zerując także wskaźnik stanu linii RTS w programowym słowie stanu portu COMx) w momencie, w którym programowy bufor odbiorczy może pomieścić nie więcej niż 2 bajty informacji, a linia ta znajduje się w stanie "1" (HIGH).

Ponadto procedura input() modyfikuje następujące wskaźniki w programowym słowie stanu portu COMx (x=1,2):

7	6	x	x	x	x	x	0

- **bit 7:** Pusty bufor odbiorczy:
  - 0 - bufor odbiorczy zawiera nieodczytaną informację,
  - 1 - całą odebraną informację wpisaną do bufora odbiorczego już odczytano.
- **bit 6:** Przepełnienie bufora odbiorczego:
  - 0 - żadna informacja odebrana z kanału transmisyjnego COMx nie została zagubiona (bufor zawieral dość miejsca aby ją pomieścić),
  - 1 - informacja odebrana z kanału transmisyjnego COMx została zagubiona (bufor był przepelniony i nie zawieral miejsca aby ja pomiescic),
- **bit 0:** Informacja o stanie linii RTS:
  - 0 - RTS = 0,
  - 1 - RTS = 1.

Wartość zwracana:
Procedura nie zwraca żadnej wartości.

Procedura output()

Procedura output() odczytuje 1-bajtowy znak z bufora nadawczego output__buffer i wpisuje go do bufora portu transmisji szeregowej COMx, a także modyfikuje wartości następujących zmiennych:

- index_to_output_buffer
- counter_output_buffer
- status_word

Ponadto procedura output() modyfikuje następujące wskaźniki w programowym słowie stanu portu COMx:

x	x	5	4	x	x	x	x

- **bit 5:** Pusty bufor nadawczy:
  - 0 - bufor nadawczy zawiera niewysłaną informację,
  - 1 - bufor nadawczy jest pusty (wszystko z niego wysłano)
- **bit 4:** Przepełnienie bufora nadawczego:
  - 0 - wpisano wartość do bufora nadawczego,
  - 1 - bufor nadawczy jest pełny, a więc nie przyjął wpisywanej wartości.

Wartość zwracana:
Procedura nie zwraca żadnej wartości.

Procedura read_com()

Procedura read_com() wykonuje odczyt pojedynczego znaku (wartości 1-bajtowej) z programowego bufora odbiorczego input_buffer obsługującego trasmisję z urządzenia zewnętrznego do komputera IBM PC przez wybrany port transmisji szeregowej COMx (x = 1, 2), a także modyfikuje wartości następujących zmiennych:

- index_from_input_buffer
- counter_input_buffer
- status_word

Ponadto procedura read_com() ustawia linię RTS portu COMx w stan "1" (HIGH) (ustawiając także wskaźnik stanu linii RTS w programowym słowie stanu portu COMx) w momencie, w którym w buforze odbiorczym input_buffer zwolniło się miejsce na co najmniej 10 bajtów, a linia RTS znajduje się w stanie "0" (LOW).

Procedura read_com() modyfikuje następujące wskaźniki w programowym słowie stanu portu COMx (x = 1, 2):

7	x	x	x	x	x	x	0

- **bit 7:** Pusty bufor odbiorczy:
  0 - bufor odbiorczy zawiera nieodczytaną informację,
  1 - całą odebraną informację wpisaną do bufora odbiorczego już odczytano.
- **bit 0:** Informacja o stanie linii RTS:
  0 - RTS = 0,
  1 - RTS = 1.

Wywołanie:

- z poziomu programów napisanych w języku asemblera MASM firmy MICROSOFT:

```
EXTRN READ_COM: FAR
CALL FAR PTR READ_COM
```

- z poziomu programów napisanych w języku C dla dużego (LARGE lub HUGE) modelu pamięci (kompilator firmy MICROSOFT wymaga użycia opcji /AL /Gx- (model LARGE) lub /AH /Gx- (model HUGE)):

```
extern unsigned int read_com (void);
unsigned int c;
c = read_com();
```

Wartość zwracana:

Procedura zwraca do programu nadrzędnego następujące wartości:

- w rejestrze AL - kod odczytanego znaku,
- w rejestrze AH - młodszy bajt programowego słowa stanu portu COMx (starszy bajt jest równy 00H)

Z punktu widzenia programu nadrzędnego wykorzystującego procedurę read_com() istotna jest wartość najstarszego bitu zwracanego w rejestrze AH. Jeśli bit ten jest równy "0" - oznacza to, że w rejestrze AL znajduje się wartość odczytana z bufora obsługującego transmisje przez COMx do komputera. Jeśli bit ten jest równy "1" - to oznacza, że bufor ten był pusty w momencie wywołania procedury read_com, a zatem w rejestrze AL znajduje się przypadkowa wartość. Dla programisty piszącego w języku C procedura read_com() zwraca wartość typu unsigned int, ktorej starszy bajt zawiera programowe słowo stanu portu COMx, a młodszy - odczytaną z bufora wartość.

Procedura write_com()

Procedura write_com wpisuje pojedynczy znak (wartość 1-bajtową) do bufora nadawczego output_buffer oraz inicjuje wysyłkę informacji z tego bufora do urządzenia zewnętrznego przez port transmisji szeregowej COMx (x = 1, 2) z wykorzystaniem systemu przerwań. Równocześnie zostają zmodyfikowane wartości następujących zmiennych:

- index_to_output_buffer
- counter_output_buffer
- status_word

Wywołanie:

- z poziomu programów napisanych w języku asemblera MASM firmy MICROSOFT:

```
EXTRN WRITE_COM: FAR
PUSH <-- słowo, ktorego młodszy bajt zawiera informacje do wyslania przez port COMx
CALL FAR PTR WRITE_COM
```

- z poziomu programów napisanych w języku C dla dużego (LARGE lub HUGE) modelu pamięci (kompilator firmy MICROSOFT wymaga użycia opcji /AL /Gx- (model LARGE) lub /AH /Gx- (model HUGE)):

```
extern unsigned int write_com (unsigned int);
unsigned int c;
write_com(c);
```

Wartość zwracana:

Procedura zwraca do programu nadrzędnego (w rejestrze AL) programowe słowo stanu portu COMx, zawierające m.in. następujące wskaźniki (wymieniono tylko te modyfikowane przez procedurę):

x	x	5	4	x	x	x	x

- **bit 5:** Pusty bufor nadawczy:
  - 0 - bufor nadawczy zawiera niewysłaną informację,
  - 1 - bufor nadawczy jest pusty (wszystko z niego wysłano)
- **bit 4:** Przepełnienie bufora nadawczego:
  - 0 - wpisano wartość do bufora nadawczego,
  - 1 - bufor nadawczy jest pełny, a więc nie przyjął wpisywanej wartości.

A zatem do bufora nadawczego wykonano prawidłowy zapis tylko wtedy, gdy wartość bitu nr 4 w programowym słowie stanu wynosi "0".

Procedura open_com()

Procedura open_com() wykonuje wszystkie czynności związane z przygotowaniem portu transmisji szeregowej COMx do pracy w trybie generowania przerwań spowodowanych:

- błędami transmisji z urządzenia zewnętrznego do komputera IBM PC,
- odebraniem informacji nadanej z urządzenia zewnętrznego do komputera IBM PC,
- nadaniem informacji do urządzenia zewnętrznego z komputera IBM PC,
- zmianą stanu linii sterującej CTS synchronizującej nadawanie informacji z komputera do urządzenia zewnęrznego.

Procedura ta wykonuje następujące czynności:

- wywołuje podprogram clear_com() (patrz dalszy opis),
- ustawia linię RTS portu COMx (x = 1, 2) w stan "1" (HIGH),

- ustawia maskę przerwań programowalnego sterownika przerwań 8259 tak, że odmaskowane zostają przerwania zgłaszane przez port COMx,
- ustawia maskę przerwań portu COMx umożliwiając zgłaszanie przerwań przez ten port spowodowanych przyczynami opisanymi powyżej,
- wymienia aktualny wektor przerwania nr 12 lub 11 (przerwanie zgłaszane przez port COM1 lub COM2) na wskaźnik do procedury INT_COM,
- ustawia flagę IF=1 (INTERRUPT FLAG) mikroprocesora 8086.

Wywołanie:

- z poziomu programów napisanych w języku asemblera MASM firmy MICROSOFT:

```
EXTRN OPEN_COM: FAR
PUSH <-- słowo zawierające numer portu do transmisji (1 lub 2)
CALL FAR PTR OPEN_COM
```

- z poziomu programów napisanych w języku C dla dużego (LARGE lub HUGE) modelu pamięci (kompilator firmy MICROSOFT wymaga użycia opcji /AL /Gx- (model LARGE) lub /AH /Gx- (model HUGE)):

```
extern void open_com (int);
int numer_portu;
open_com (numer_portu);
```

Wartość zwracana:

Procedura nie zwraca żadnej wartości.

Procedura close_com()

Procedura close_com() wykonuje następujące czynności związane z zakończeniem wykonywania transmisji przez port COMx (x = 1, 2) w trybie przerwaniowym:

- wymienia aktualny wektor przerwania nr 12 (w przypadku portu COM1) lub przerwania nr 11 (w przypadku portu COM2) będący wskaźnikiem do procedury INT_COM na stary wektor ustawiony przez system operacyny MS DOS przed uruchomieniem podprogramu open_com(),
- ustawia obie linie sterujące: RTS i DTR portu COMx (x = 1, 2) w stan "1" (HIGH).

Wywołanie:

- z poziomu programów napisanych w języku asemblera MASM firmy MICROSOFT:

```
EXTRN CLOSE_COM: FAR
PUSH <-- słowo zawierające numer portu do transmisji (1 lub 2)
CALL FAR PTR CLOSE_COM
```

- z poziomu programów napisanych w języku C dla dużego (LARGE lub HUGE) modelu pamięci (kompilator firmy MICROSOFT wymaga użycia opcji /AL /Gx- (model LARGE) lub /AH /Gx- (model HUGE)):

```
extern void close_com (int);
int numer_portu;
close_com (numer_portu);
```

Wartość zwracana:

Procedura nie zwraca żadnej wartości.

Procedura set_com_addresses()

Procedura set_com_addresses() ustawia zmienne przechowujące adresy I/O rejestrow obslugujących dany port transmisji szeregowej. Jest to procedura pomocnicza, wykorzystywana przez podprogram "open_com()".

Wywołanie:

- z poziomu programów napisanych w języku asemblera MASM firmy MICROSOFT:

```
EXTRN SET_COM_ADRESSES: FAR
PUSH <-- słowo zawierające numer portu do transmisji (1 lub 2)
CALL FAR PTR SET_COM_ADRESSES
```

- z poziomu programów napisanych w języku C dla dużego (LARGE lub HUGE) modelu pamięci (kompilator firmy MICROSOFT wymaga użycia opcji /AL /Gx- (model LARGE) lub /AH /Gx- (model HUGE)):

```
extern void set_com_addresses (int);
int numer_portu;
set_com_addresses (numer_portu);
```

Wartość zwracana:

Procedura nie zwraca żadnej wartości.

Procedura clear_com()

Procedura clear_com() ustawia w stan początkowy wszystkie zmienne sterujące zapisem/odczytem do/z buforow nadawczego i odbiorczego (odpowiednio output_buffer, input_buffer) poprzez wywołanie podprogramów clear_input_buffer() i clear_output_buffer() (patrz dalszy opis), a także zeruje wskaźniki zmiany stanu linii sterujących portu COMx (x = 1, 2).

Wywołanie:

- z poziomu programów napisanych w języku asemblera MASM firmy MICROSOFT:

```
EXTRN CLEAR_COM: FAR
CALL FAR PTR CLEAR_COM
```

- z poziomu programów napisanych w języku C dla dużego (LARGE lub HUGE) modelu pamięci (kompilator firmy MICROSOFT wymaga użycia opcji /AL /Gx- (model LARGE) lub /AH /Gx- (model HUGE)):

```
extern void clear_com (void);
clear_com();
```

Wartość zwracana:

Procedura nie zwraca żadnej wartości.

Procedura clear_input_buffer()

Procedura clear_input_buffer() ustawia wszystkie zmienne sterujące odczytem/zapisem z/do bufora odbiorczego input_buffer w stan początkowy, a także ustawia następujące wskaźniki w programowym słowie stanu portu COMx:

7	6	x	x	3	2	1	x

- **bit 7:** Pusty bufor odbiorczy:
  1 - całą odebraną informację wpisaną do bufora odbiorczego już odczytano.
- **bit 6:** Przepełnienie bufora odbiorczego:
  0 - żadna informacja odebrana z kanału transmisyjnego COMx nie została zagubiona (bufor zawieral dość miejsca aby ją pomieścić),

- **bit 3:** Framing Error:
  0 - brak błędu Framing Error,
- **bit 2:** Parity Error:
  0 - brak błędu Parity Error,
- **bit 1:** Overrun Error:
  0 - brak błędu Overrun Error,

Wywołanie:

- z poziomu programów napisanych w języku asemblera MASM firmy MICROSOFT:

```
EXTRN CLEAR_INPUT_BUFFER: FAR
CALL FAR PTR CLEAR_INPUT_BUFFER
```

- z poziomu programów napisanych w języku C dla dużego (LARGE lub HUGE) modelu pamięci (kompilator firmy MICROSOFT wymaga użycia opcji /AL /Gx- (model LARGE) lub /AH /Gx- (model HUGE)):

```
extern void clear_input_buffer (void);
clear_input_buffer();
```

Wartość zwracana:
Procedura nie zwraca żadnej wartości.

Procedura clear_output_buffer()
Procedura clear_output_buffer() ustawia wszystkie zmienne sterujące odczytem/zapisem z/do bufora nadawczego output_buffer w stan początkowy, a także ustawia następujące wskaźniki w programowym słowie stanu portu COMx:

x	x	5	4	x	x	x	x

- **bit 5:** Pusty bufor nadawczy:
  1 - bufor nadawczy jest pusty (wszystko z niego wysłano)
- **bit 4:** Przepełnienie bufora nadawczego:
  0 - wpisano wartość do bufora nadawczego,

Wywołanie:

- z poziomu programów napisanych w języku asemblera MASM firmy MICROSOFT:

```
EXTRN CLEAR_OUTPUT_BUFFER: FAR
CALL FAR PTR CLEAR_OUTPUT_BUFFER
```

- z poziomu programów napisanych w języku C dla dużego (LARGE lub HUGE) modelu pamięci (kompilator firmy MICROSOFT wymaga użycia opcji /AL /Gx- (model LARGE) lub /AH /Gx- (model HUGE)):

```
extern void clear_output_buffer (void);
clear_output_buffer();
```

Wartość zwracana:
Procedura nie zwraca żadnej wartości.

Procedura free_space_in_output_buffer()
Procedura free_space_in_output_buffer() podaje ilość wolnego miejsca (w bajtach) w programowym buforze nadawczym output_buffer związanym z portem COMx (x = 1, 2).
Wywołanie:

- z poziomu programów napisanych w języku asemblera MASM firmy MICROSOFT:

```
EXTRN FREE_SPACE_IN_OUTPUT_BUFFER: FAR
```

CALL FAR PTR FREE_SPACE_IN_OUTPUT_BUFFER

- z poziomu programów napisanych w języku C dla dużego (LARGE lub HUGE) modelu pamięci (kompilator firmy MICROSOFT wymaga użycia opcji /AL /Gx- (model LARGE) lub /AH /Gx- (model HUGE)):

```
extern int free_space_in_output_buffer (void);
free_space_in_output_buffer();
```

Wartość zwracana:
Ilość wolnego miejsca w buforze nadawczym output_buffer jest zwracana w rejestrze AX.

Procedura contents_input_buffer()
Procedura contents_input_buffer() podaje ile nieodczytanych 1-bajtowych wartości znajduje się w programowym buforze odbiorczym input_buffer.
Wywołanie:

- z poziomu programów napisanych w języku asemblera MASM firmy MICROSOFT:

```
EXTRN CONTENTS_INPUT_BUFFER: FAR
CALL FAR PTR CONTENTS_INPUT_BUFFER
```

- z poziomu programów napisanych w języku C dla dużego (LARGE lub HUGE) modelu pamięci (kompilator firmy MICROSOFT wymaga użycia opcji /AL /Gx- (model LARGE) lub /AH /Gx- (model HUGE)):

```
extern int contents_input_buffer (void);
int liczba_nieodczytanych_bajtow;
.....
liczba_nieodczytanych_bajtow = contents_input_buffer();
```

Wartość zwracana:
Liczba nieodczytanych bajtow znajdujących się w buforze odbiorczym jest zwracana w rejestrze AX.

Procedura set_rts()
Procedura set_rts() ustawia w stan "1" (HIGH) linię sterującą RTS portu COMx (x = 1, 2), a także ustawia wskaźnik stanu tej linii w programowym słowie stanu tego portu:

x	x	x	x	x	x	x	0

- **bit 0:** Informacja o stanie linii RTS:
  0 - RTS = 1

Uwaga:
Ponieważ linie sterujące RTS i DTR portu COMx (x = 1, 2) są ustawiane przez zmianę zawartości tego samego rejestru (tzw. MODEM CONTROL REGISTER), więc jednocześnie ze zmianą stanu linii RTS należy ustalać stan linii DTR. Linia DTR jest zawsze ustawana w stan "1" (HIGH).
Wywołanie:

- z poziomu programów napisanych w języku asemblera MASM firmy MICROSOFT:

```
EXTRN SET_RTS: FAR
CALL FAR PTR SET_RTS
```

- z poziomu programów napisanych w języku C dla dużego (LARGE lub HUGE) modelu pamięci (kompilator firmy MICROSOFT wymaga użycia opcji /AL /Gx- (model LARGE) lub /AH /Gx- (model HUGE)):

```
extern void set_rts (void);
```

set_rts();

Wartość zwracana:

Procedura nie zwraca żadnej wartości.

Procedura reset_rts()

Procedura reset_rts() ustawia w stan "0" (LOW) linie RTS portu COMx (x = 1, 2), a także zeruje wskaźnik stanu tej linii w programowym słowie stanu tego portu:

x	x	x	x	x	x	x	0

- **bit 0:** Informacja o stanie linii RTS:

  0 - RTS = 0

Uwaga:

Ponieważ linie sterujące RTS i DTR portu COMx (x = 1, 2) są ustawiane przez zmianę zawartości tego samego rejestru (tzw. MODEM CONTROL REGISTER), więc jednocześnie ze zmianą stanu linii RTS należy ustalać stan linii DTR. Linia DTR jest zawsze ustawiana w stan "1" (HIGH).

Wywołanie:

- z poziomu programów napisanych w języku asemblera MASM firmy MICROSOFT:

  EXTRN RESET_RTS: FAR
  CALL FAR PTR RESET_RTS

- z poziomu programów napisanych w języku C dla dużego (LARGE lub HUGE) modelu pamięci (kompilator firmy MICROSOFT wymaga użycia opcji /AL /Gx- (model LARGE) lub /AH /Gx- (model HUGE)):

  extern void reset_rts (void);
  reset_rts();

Wartość zwracana:

Procedura nie zwraca żadnej wartości.

Procedura read_status_word()

Procedura read_status_word() odczytuje programowe słowo stanu portu COMx i zwraca go do podprogramu nadrzędnego w rejestrze AX (dla programu nadrzędnego napisanego w języku C - zwraca poprzez nazwę procedury).

Wywołanie:

- z poziomu programów napisanych w języku asemblera MASM firmy MICROSOFT:

  EXTRN READ_STATUS_WORD: FAR
  CALL FAR PTR READ_STATUS_WORD

- z poziomu programów napisanych w języku C dla dużego (LARGE lub HUGE) modelu pamięci (kompilator firmy MICROSOFT wymaga użycia opcji /AL /Gx- (model LARGE) lub /AH /Gx- (model HUGE)):

  extern unsigned int read_status_word (void);
  unsigned int i;
  i = read_status_word();

Wartość zwracana:

Patrz wyżej.

Procedura clear_transmission_errors()

Procedura clear_transmission_errors() zeruje w programowym słowie stanu portu COMx następujące wskaźniki błędów transmisji:

x	x	x	x	3	2	1	x

- **bit 3:** Framing Error:
  0 - brak błędu Framing Error,
- **bit 2:** Parity Error:
  0 - brak błędu Parity Error,
- **bit 1:** Overrun Error:
  0 - brak błędu Overrun Error,

Wywołanie:

- z poziomu programów napisanych w języku asemblera MASM firmy MICROSOFT:
  EXTRN CLEAR_TRANSMISSION_ERRORS: FAR
  CALL FAR PTR CLEAR_TRANSMISSION_ERRORS
- z poziomu programów napisanych w języku C dla dużego (LARGE lub HUGE) modelu pamięci (kompilator firmy MICROSOFT wymaga użycia opcji /AL /Gx- (model LARGE) lub /AH /Gx- (model HUGE)):
  extern void clear_transmission_errors (void);
  clear_transmission_errors();

Wartość zwracana:

Procedura nie zwraca żadnej wartości.

Procedura read_modem_status_register()

Odczytanie rejestru zawierającego informacje o stanie linii modemowych:

7	6	5	4	3	2	1	0

bit 7: DCD (Data Carrier Detect)
bit 6: RI (Ring Indicator)
bit 5: DSR (Data Set Ready)
bit 4: CTS (Clear To Send)
bit 3: Delta DCD (zmiana stanu linii DCD od ostatniego odczytu rejestru MODEM STATUS REGISTER)
bit 2: Trailing-Edge RI (zmiana stanu linii CTS od ostatniego odczytu rejestru MODEM STATUS REGISTER)
bit 1: Delta DSR (zmiana stanu linii DSR od ostatniego odczytu rejestru MODEM STATUS REGISTER)
bit 0: Delta CTS (zmiana stanu linii CTS od ostatniego odczytu rejestru MODEM STATUS REGISTER)

Wywołanie:

- z poziomu programów napisanych w języku asemblera MASM firmy MICROSOFT:
  EXTRN READ_MODEM_STATUS_REGISTER: FAR
  CALL FAR PTR READ_MODEM_STATUS_REGISTER
- z poziomu programów napisanych w języku C dla dużego (LARGE lub HUGE) modelu pamięci (kompilator firmy MICROSOFT wymaga użycia opcji /AL /Gx- (model LARGE) lub /AH /Gx- (model HUGE)):

extern unsigned char read_modem_status_register (void);
unsigned char c;
c = read_modem_status_register();

Wartość zwracana:
Odczytana wartość jest zwracana w rejestrze AX (AL).

Procedura set_synchronization_by_cts()
Procedura set_synchronization_by_cts() odblokowuje synchronizację nadawania przez obsługiwany port szeregowy komputera. Od tego momentu port będzie wysyłał informację tylko wtedy, gdy na jego linii sterującej CTS będzie "1".
Wywołanie:

- z poziomu programów napisanych w języku asemblera MASM firmy MICROSOFT:
  EXTRN SET_SYNCHRONIZATION_BY_CTS: FAR
  CALL FAR PTR SET_SYNCHRONIZATION_BY_CTS
- z poziomu programów napisanych w języku C dla dużego (LARGE lub HUGE) modelu pamięci (kompilator firmy MICROSOFT wymaga użycia opcji /AL /Gx- (model LARGE) lub /AH /Gx- (model HUGE)):
  extern void set_synchronization_by_cts (void);
  set_synchronization_by_cts();

Wartość zwracana:
Procedura nie zwraca żadnej wartości.

Procedura reset_synchronization_by_cts()
Procedura reset_synchronization_by_cts() blokuje synchronizację nadawania przez obsługiwany port szeregowy komputera. Od tego momentu port będzie wysyłał informację bez względu na stan jego linii sterującej CTS.
Wywołanie:

- z poziomu programów napisanych w języku asemblera MASM firmy MICROSOFT:
  EXTRN RESET_SYNCHRONIZATION_BY_CTS: FAR
  CALL FAR PTR RESET_SYNCHRONIZATION_BY_CTS
- z poziomu programów napisanych w języku C dla dużego (LARGE lub HUGE) modelu pamięci (kompilator firmy MICROSOFT wymaga użycia opcji /AL /Gx- (model LARGE) lub /AH /Gx- (model HUGE)):
  extern void reset_synchronization_by_cts (void);
  reset_synchronization_by_cts();

Wartość zwracana:
Procedura nie zwraca żadnej wartości.

Procedura read_synchronization_state()
Procedura read_synchronization_state() przekazuje do programu nadrzędnego informację o załączeniu badź wyłączeniu synchronizacji nadawania przez port szeregowy stanem jego linii sterującej CTS.
Wywołanie:

- z poziomu programów napisanych w języku asemblera MASM firmy MICROSOFT:
  EXTRN READ_SYNCHRONIZATION_STATE: FAR

CALL FAR PTR READ_SYNCHRONIZATION_STATE

- z poziomu programów napisanych w języku C dla dużego (LARGE lub HUGE) modelu pamięci (kompilator firmy MICROSOFT wymaga użycia opcji /AL /Gx- (model LARGE) lub /AH /Gx- (model HUGE)):

```
extern int read_synchronization_state (void);
read_synchronization_state();
```

Wartość zwracana:

Procedura zwraca do programu nadrzednego (w rejestrze AX) następujące wartości:

- jesli synchronizacja jest załączona: 00000H
- jesli synchronizacja jest wyłączona: 0FFFFH (wartość różna od zera).

Procedura set_transmission_parameters()

Ustawienie parametrów transmisji danego portu transmisji szeregowej. Dopuszczalne wartości poszczególnych parametrów to:

. Parametr port określający numer portu szeregowego:

port=1	-	określa port COM1,
port=2	-	określa port COM2,
port=3	-	określa port COM3,
port=4	-	określa port COM4.

. Parametr baud określający prędkość transmisji:

baud=5	-	prędkość 50 bitów/s,
baud=50	-	prędkość 50 bitów/s,
baud=11	-	prędkość 110 bitów/,s
baud=110	-	prędkość 110 bitów/s,
baud=15	-	prędkość 150 bitów/s,
baud=150	-	prędkość 150 bitów/s,
baud=30	-	prędkość 300 bitów/s,
baud=300	-	prędkość 300 bitów/s,
baud=60	-	prędkość 600 bitów/s,
baud=600	-	prędkość 600 bitów/s,
baud=12	-	prędkość 1200 bitów/,s
baud=1200	-	prędkość 1200 bitów/s,
baud=24	-	prędkość 2400 bitów/s,
baud=2400	-	prędkość 2400 bitów/s,
baud=48	-	prędkość 4800 bitów/s,
baud=4800	-	prędkość 4800 bitów/s,
baud=72	-	prędkość 7200 bitów/s,
baud=7200	-	prędkość 7200 bitów/s,
baud=96	-	prędkość 9600 bitów/s,
baud=9600	-	prędkość 9600 bitów/s,
baud=19	-	prędkość 19200 bitów/s,
baud=19200	-	prędkość 19200 bitów/s,
baud=38	-	prędkość 38400 bitów/s,
baud=38400	-	prędkość 38400 bitów/s,
baud=57	-	prędkość 57600 bitów/s,

baud=57600 - prędkość 57600 bitów/s,
baud=115 - prędkość 115200 bitów/s.

. Parametr parity określający kontrole parzystosci/nieparzystosci:

parity=n - brak kontroli parzystości/nieparzystości
parity=N - brak kontroli parzystości/nieparzystości
parity=e - kontrola parzystości
parity=E - kontrola parzystości
parity=o - kontrola nieparzystości
parity=O - kontrola nieparzystości

. Parametr length określający dlugosc slowa:

length=5 - słowo o długości 5 bitów,
length=6 - słowo o długości 6 bitów,
length=7 - słowo o długości 7 bitów,
length=8 - słowo o długości 8 bitów.

. . Parametr stop_bits określający liczbę bitów stopu:

stop_bits=1 - 1 bit stopu,
stop_bits=5 - 1.5 bita stopu,
stop_bits=2 - 2 bity stopu.

Wywołanie:

- z poziomu programów napisanych w języku asemblera MASM firmy MICROSOFT:

```
EXTRN SET_TRANSMISSION_PARAMETERS: FAR
.....
PUSH <-- numer portu
PUSH <-- określenie prędkośći transmisji
PUSH <-- określenie kontroli parzystosci/nieparzystosci
PUSH <-- określenie dlugosci slowa
PUSH <-- określenie liczby bitów stopu
CALL FAR PTR SET_TRANSMISSION_PARAMETERS
```

- z poziomu programów napisanych w języku C dla dużego (LARGE lub HUGE) modelu pamięci (kompilator firmy MICROSOFT wymaga użycia opcji /AL /Gx- (model LARGE) lub /AH /Gx- (model HUGE)):

```
extern unsigned int set_transmission_parameters (int, unsigned int, char, int, int)
int port;
unsigned int baud, wynik;
char parity;
int length;
int stop_bits;
.....
i = set_transmission_parameters (port, baud, parity, length, stop_bits);
```

Wartość zwracana:

Procedura zwraca (rejestr AL) wartość określająca powodzenie wykonania ustawienia parametrów transmisji portu (rejestr AH jest zawsze wyzerowany). Ustawienie parametrów transmisji portu jest zależne od podanych parametrów, a zatem zwracana wartość zawiera ocenę ich poprawności. Wystąpienie jakiegokolwiek błędu oznacza, że port nie został przeprogramowany:

x	x	5	4	3	2	1	0

- **bit 5:** Inne błędy (np. niewłaściwa kombinacja parametrów transmisji):
  0 - brak błędów,
  1 - wystąpiły inne błędy.
- **bit 4:** Poprawność parametru określającego liczbę bitów stopu:
  0 - parametr poprawny,
  1 - parametr niepoprawny.
- **bit 3:** Poprawność parametru określającego długość słowa:
  0 - parametr poprawny,
  1 - parametr niepoprawny.
- **bit 2:** Poprawność parametru określającego kontrolę parzystości/nieparzystości:
  0 - parametr poprawny,
  1 - parametr niepoprawny.
- **bit 1:** Poprawność parametru określającego prędkość transmisji:
  0 - parametr poprawny,
  1 - parametr niepoprawny (użytkownik zamierza ustawić niestandardową szybkość transmisji..
- **bit 0:** Poprawność parametru określającego port szeregowy:
  0 - komputer zawiera dany port szeregowy
  1 - komputer nie zawiera danego portu szeregowego.

6.4. Sprawdzenie spełnienia wymagań bezpieczeństwa

Wymagania dotyczące bezpieczeństwa pracy robotów przemysłowych są, formalnie rzecz biorąc, sformułowane w:

- PN-M-42087: 1995 (idt. ISO 10218: 1992, eqv. EN 775:1982) - Roboty przemysłowe - Bezpieczeństwo i powołanej w niej:
- PN-EN 60204-1: 1997 - Bezpieczeństwo maszyn - Wyposażenie elektryczne maszyn - Wymagania ogólne.

Jednakże ze względu na to, że przedmiotem pracy jest oprogramowanie, obsługujące m.in. funkcje związane z bezpieczeństwem, konieczne jest też uwzględnienie wymagań:

- EN 954-1: 1996 - Maszyny - Bezpieczeństwo - Elementy systemu sterowania związane z bezpieczeństwem - Wymagania ogólne.
- IEC DIS 61508-3:1994 - Bezpieczeństwo funkcjonalne: układy związane z bezpieczeństwem - Wymagania dotyczące oprogramowania.

6.4.1. PN-M-42087

6.4.1.1. Odporność na zakłócenia elektromagnetyczne (w rozumieniu Dyrektywy 89/336/ECC) - p. 4.2.1.h.1. i p. 6.9.- została zapewniona przez wykorzystanie do badań robota i układu sterowania uprzednio sprawdzonych w laboratorium akredytowanym PIAP-LAB.

6.4.1.2. Odporność na zakłócenia mechaniczne (drgania, uderzenia) - p. 4.2.1.h.2. - została zapewniona też w sposób wyżej podany.

6.4.1.3. Pozostałe wymagania dotyczące wyposażenia elektrycznego są formułowane przez powołanie PN-EN 60204-1.

6.4.2. PN-EN 60204-1

6.4.2.1. Wymagane jest wprowadzenie środków do sprawdzania, czy oprogramowanie jest zgodne z dokumentacją programów.
Wymaganie to jest spełnione.

6.4.2.2. Pozostałe wymagania dotyczące wyposażenia elektronicznego i wyposażenia programowalnego dotyczą konstrukcji i wykonania układu sterowania, które nie są przedmiotem niniejszego projektu.

6.4.3. EN 954-1

6.4.3.1. Funkcje związane z bezpieczeństwem i ich relacje do oprogramowania VMD zestawiono w tablicy 6.5.

Tablica 6.5. Funkcje związane z bezpieczeństwem

l.p.	Funkcja	Realizacja przez VMD	Wynik sprawdzenia
1.	Stop funkcjonalny	tak	dodatni
2.	Stop awaryjny	nie dotyczy	ditto
3.	Nastawianie ręczne	nie dotyczy	ditto
4.	Start i restart	tak	dodatni
5.	Czas odpowiedzi systemu sterowania	tak	dodatni
6.	Parametry związane z bezpieczeństwem - działanie w przypadku przekroczenia dozwolonych granic	nie dotyczy	ditto

6.4.3.2. Zestawienie kategorii związanych z bezpieczeństwem i ich relacje do oprogramowania VMD zestawiono w tablicy 6.6.

Tablica 6.6. Kategorie bezpieczeństwa

l.p.	Kategoria bezpieczeństwa	Realizacja przez VMD	Wynik sprawdzenia
1.	B - podstawowa	nie dotyczy oprogramowania, zapewniona przez użytego robota	oprogramowanie - dodatni
2.	1 - kategoria B i ponadto stosowanie wypróbowanych części składowych i wypróbowanych zasad bezpieczeństwa	spełnione w zakesie oprogramownia, w zakresie sprzętu spełnione przez użytego robota	dodatni
3.	2 - kategoria 1 i ponadto okresowe sprawdzanie funkcji bezpieczeństwa	nie wbudowane	nie dotyczy
4.	3 - kategoria 1 i ponadto pojedynczy defekt nie prowadzi do utraty funkcji bezpieczeństwa	nie wbudowane	nie dotyczy
5.	4 - kategoria 1 i ponadto: · pojedynczy defekt nie prowadzi do utraty funkcji bezpieczeństwa; · pojedynczy defekt jest rozpoznawany przed kolejnym wykonaniem funkcji bezpieczeństwa lub gdy nie, to kumulacja defektów nie prowadzi do utraty funkcji bezpieczeństwa.	nie wbudowane	nie dotyczy

Oprogramowanie VMD spełnia wymagania dotyczące kategorii 1. Jest to wystarczające do przewidywanych zastosowań przemysłowych.

6.4.4. IEC DIS 61508-3

Z analizy zastosowanych technik i języków programowania wynika, że oprogramowanie VMD spełnia wymagania poziomu nienaruszalności 1, tj. zapewnia prawdopodobieństwo wystąpienia uszkodzenia w granicach od 1 na 100 lat do 1 na 10 lat. To prawdopodobieństwo jest wystarczające przy pracy robota w normalnych warunkach przemysłowych, do jakich VMD był projektowany.

LITERATURA

[1] Christofides N.: Graph Theory. An Algorithmic Approach. New York. Academic Press. 1975.

[2] Deo N.: Teoria grafów i jej zastosowania w technice i informatyce. Warszawa. PWN. 1980.

[3] Dijkstra E. A.: Note on Two Problems in Connection with Graphs. Numerische Matematik, 1 (1959) 1, 269-271.

[4] Dinic E. A.: Algorithm for solution a problem of maximal flow in a network with power estimation, Soviet Math. Dokl., 11 (1970) 5, 1277-1280.

[5] Dreyfus S. E.: An Appraisal of Some Shortest Path Algorithms. Operations Research, 17 (1969) 3, 395-412.

[6] Edmonds J., Karp R. M.: Theoretical Improvements in Algorithmic Efficiency for Network Flow Problem. J. of Association for Computing Machinery, 19 (1972) 2, 248-264.

[7] EN 29506-1: 1993 (ISO/IEC 9506-1: 1990) Industrial Automation Systems-Manufacturing Message Specification-Part 1-Service Definition.
PN-ISO/IEC 9506-1: 1994 Systemy automatyki przemysłowej-Specyfikacja Komunikatów w Procesie Wytwarzania. Definicja usługi.

[8] EN 29506-2: 1993 (ISO/IEC 9506-2: 1990) Industrial Automation Systems-Manufacturing Message Specification-Part 2-Protocol Specification.
PN-ISO/IEC 9506-2: 1994 Systemy automatyki przemysłowej-Specyfikacja Komunikatów w Procesie Wytwarzania. Specyfikacja protokołu.

[9] Even S.: Algorithmic Combinatorics. New York. The Macmillan Comp. 1973.

[10] Floyd R. W.: Algorithm 97-Shortest Path. Comm of Association for Computing Machinery, 5 (1962) 6, 345.

[11] Ford L. R., Fulkerson D. R.: Przepływy w sieciach. Warszawa. PWN. 1969.

[12] Galil Z. An $O(V^{5/3}E^{2/3})$ Algorithm for the Maximal Flow Problem. Acta Informatica, 14 (1980) 3, 221-242.

[13] Gomory R. E., Hu T. C.: Multi-terminal Network Flows. SIAM J. Appl. Math., 9 (1961), 551-570.

[14] Hu T. C.: The Maximum Capacity Route Problem. Operations Research, 9 (1961) 6, 898-900.

[15] I'Anson C.I,. Pell A., Understanding OSI Applications, Prentice Hall, 1993.

[16] ISO/IEC 9506-1/DAM 1: 1992 Industrial Automation Systems-Manufacturing Message Specification-Service Definition, Amendment 1: Data Exchange.

[17] ISO/IEC 9506-1: 1990/AC1:1995 + AC2:1995 Industrial Automation Systems-Manufacturing Message Specification-Part 1-Service Definition. TECHNICAL CORRIGENDUM 1+2.
PrPN-ISO/IEC 9506-1: 1995 Systemy automatyki przemysłowej-Specyfikacja Komunikatów w Procesie Wytwarzania. Definicja usługi. (Poprawka AC1+AC2).

[18] ISO/IEC 9506-2/DAM 1: 1992 Industrial Automation Systems-Manufacturing Message Specification-Protocol Specification, Amendment 1: Data Exchange.

[19] ISO/IEC 9506-2: 1990/AC1:1995 + AC2:1995 Industrial Automation Systems-Manufacturing Message Specification-Part 2-Protocol Specification. TECHNICAL CORRIGENDUM 1+2.
PrPN-ISO/IEC 9506-2: 1995 Systemy automatyki przemysłowej-Specyfikacja Komunikatów w Procesie Wytwarzania. Specyfikacja protokołu. (Poprawka AC1+AC2).
[20] ISO/IEC 9506-3: 1990 Industrial Automation Systems-Manufacturing Message Specification-Part 3-Companion Standard for Robotics.
[21] ISO 7498-1. OSI-Basic Reference Model.
[22] Kacprzyk J., Stańczak W.: Zastosowanie metody zespołów minimalnych do podziału grupy przedsiębiorstw na podgrupy. Archiwum Automatyki i Telemechaniki, 20 (1975) 4, 513-526.
[23] Kacprzyk J., Stańczak W.: On an Extension of the Method of Minimally Interconnected Subnetworks. Control a. Cybernetics, 5 (1976) 4, 61-77.
[24] Kacprzyk J., Stańczak W.: On a Further Extension of the Method of Minimally Interconnected Subnetworks. Control a. Cybernetics, 7 (1978) 2, 17-31.
[25] Kacprzyk J., Stańczak W.: Partitioning a Computer Network into Subnetworks and Allocation of Distributed Data Bases. [w:] Optimization Techniques, J. Stoer (ed.). Berlin-Heidelberg-New York, Springer Verlag. 1978, 464-472.
[26] Kacprzyk J., Stańczak W.: Some Contribution to the Method of Minimal Sets for Large Graphs. Control a. Cybernetics, 10 (1981) 1-2, 89-100.
[27] Kaliszewski I., Nowicki T., Stańczak W.: O dekompozycji struktury sieci telefonicznej międzycentralowej metodą zespołów minimalnych. Rozprawy Elektrotechniczne, 21 (1975) 2, 573-580.
[28] Kaufman L., Rousseeuw P. J.: Finding Groups in Data. An Introduction to Cluster Analysis. New York. John Wiley. 1990.
[29] Kevin V., Whitney M.: Algorithm 422-Minimal Spanning Tree. Comm of Association for Computing Machinery, 15 (1972) 4, 273-274.
[30] Kulikowski J. L.: Zarys teorii grafów-zastosowania w technice. Warszawa. PWN. 1986.
[31] Luccio F., Sami M.: On the Decomposition of Networks in Minimally Interconnected Subnetworks. IEEE Trans. Circuit Theory, CT-16 (1969) 1-2, 47-52.
[32] MAP 3.0 Specification. 1993 Release, World Federation of MAP/TOP Users Groups, New York 1993.
[33] Mayeda W. Graph theory. New York. John Wiley. 1972.
[34] Missala T.: Robot IRP-6/60 w elastycznym systemie produkcyjnym. Biuletyn MERA-PIAP, (1987) 1/120, 5-12.
[35] Missala T.: System MAP-geneza i cele. Biuletyn MERA-PIAP, (1987) 2/121, .3-10.
[36] Missala T.: Koncepcja realizacji systemu MAP w Polsce. Mat. X Krajowej Konf. Automatyki, Lublin 1988 t. 3., 210-220.
[37] Missala T.: Robot positioning error due to resolvers error. Mat. 33 Internationales Wissenschaftliches Kollokwium, Ilmenau 1988, t. A1, 229-232.
[38] Missala T.: Stan obecny systemu MAP (raport na podstawie literatury zagranicznej). Biuletyn MERA-PIAP, (1989) 6/146, 3-47.
[39] Missala T.: Geneza i stan obecny systemu MAP. Mat. Konf. "Lokalne sieci komputerowe dla automatyzacji przemysłu", NOT oddz.Częstochowa 1989, 1-17.

-Literatura/strona2-

[40] Missala T.: Znaczenie systemu MAP w komputerowo zintegrowanym wytwarzaniu CIM. Mat. XI Krajowej Konf. Automatyki, Białystok-Białowieża 1991 t.1, 36-42.

[41] Missala T.: Struktura zadaniowa automatyzacji przedsiębiorstwa przemysłowego z punktu widzenia CIM. Biuletyn PIAP, (1992) 3/161, 3-16.

[42] Missala T.: Sieci pracujące w systemach CIM. Przykłady aplikacji. Biuletyn PIAP, (1992) 3/161, 17-47.

[43] Missala T.: Bezpieczeństwo robotów przemysłowych. Aspekty systemowe. Prace Naukowe Instytutu Cybernetyki Stosowanej Politechniki Wrocławskiej nr 94. Prace IV Krajowej Konf. Robotyki, , Wrocław 1993, t. 2. 374-380.

[44] Missala T.: Aspekty bezpieczeństwa funkcjonalnego w projektowaniu serwomechanizmów. Mat. IX Sympozjum "Mikromaszyny i serwonapędy", Kraków 1994, 49-54.

[45] Missala T.: Bezpieczeństwo funkcjonalne układów sterowania. Politechnika Warszawska. Prace naukowe. Konferencje, 3. II Krajowa Konferencja Naukowo-Techniczna "Mechatronika' 94, Warszawa 1994, 43-46

[46] Missala T.: Robot path and positioning error caused by sensor error. Prace Naukowe Instytutu Cybernetyki Stosowanej Politechniki Wrocławskiej nr 95. Prace V Krajowej Konf. Robotyki „ Wrocław 1996, t.1. 370-378.

[47] Missala T.:Bezpieczeństwo funkcjonalne urządzeń automatyki i robotyki. Mat. Konf. Automation 1997, Warszawa 1997, t. I, 113-125, oraz Pomiary, Automatyka, Robotyka, 1 (1997) 3, 5-8.

[48] Missala T: Kompatybilność elektromagnetyczna urządzeń energoelektroniki. Wymagania dotyczące odporności na zakłócenia elektromagnetyczne. Przegląd Elektrotechniczny, (1997) 7, 169-173.

[49] Missala T.: Bezpieczeństwo funkcjonalne w procesie projektowania urządzeń mechatroniki. Politechnika Warszawska, Prace naukowe. Konferencje, 14. III Krajowa Konf. Naukowo-Techniczna "Mechatronika' 97, Warszawa 1997, 541-546.

[50] Murchland J. D.: A New Method for Finding All Elementary Paths in a Complete Directed Graph. London, School of Economics, Report LSE-TNT-22. 1965

[51] Nieminen J.: On Minimally Interconnected Subnetworks of a Network. Control a. Cybernetics, 9 (1980) 1-2, 47-52.

[52] Nowicki T., Stańczak W.: O metodzie wstępnego projektowania struktur topologicznych i konfiguracji sieci teleprzetwarzania. Warszawa-Łódź. PWN. 1980.

[53] Nowicki T., Stańczak W.: Partitioning a Set of Elements into Subsets due to their Similarity. [w:] Data Analysis and Informatics, E. Diday et al. (eds.), Amsterdam, Horth-Holland. 1980, 583-591.

[54] Pollack M.: The Maximum Capacity Route through a Network. Operations Research, 8 (1960) 5, 733-736.

[55] Prim R. C.: Shortest Connection Networks and Some Generalizations. Bell System Techn. J., 36 (1957) 6, 1389-1401.

[56] Rasiowa H.: Wstęp do matematyki współczesnej. Warszawa. PWN. 1969.

[57] Späth H. Cluster Analysis Algorithms for Data Reduction and Classification of Objects. New York. Ellis Horwood Ltd.-John Wiley. 1982.

[58] Stańczak W.: Minimal sets. Theory and applications. ICS PAS Reports, Warszawa 1981.

[59] Stańczak W.: An Efficient Algorithm for Partitioning a Network into Minimally Interconnected Subnetworks. Control a. Cybernetics, 13 (1984) 1-2, 97-112.

[60] Stańczak W.: An Introduction to Max-minimal Sets. Control a. Cybernetics, 15 (1986) 1, 83-99.

[61] Stańczak W.: Some General Structure Implied by the Idea of Minimal Sets. Control a. Cybernetics, 15 (1986) 2, 233-243.

[62] Stańczak W.: Max-minimal sets and cut-sets in edge weighted graphs. Biuletyn PIAP, (1991) 5/157, 5-36.

[63] Stańczak W.: A Polynomial-type algorithm for finding Max-minimal sets. Biuletyn PIAP, (1991) 5/157, 37-68.

[64] Stańczak W.: An $O(n^2)$ algorithm for finding Max-minimal sets. Biuletyn PIAP, (1995) 5-6/181-182, 3-19.

[65] Stańczak W., Hosza J.: On Some Clustering Technique for Inexact Data. Mat. 1st Joint IFSA-EC and EURO-WG Workshop on Progress in Fuzzy Sets in Europe, Warszawa 1986.

[66] Syryczyński A.: Trendy rozwojowe układów sterowania robotów, na przykładzie opracowań PIAP. Prace Naukowe Instytutu Cybernetyki Stosowanej Politechniki Wrocławskiej nr 94. Prace IV Krajowej Konf. Robotyki, , Wrocław 1993, t. 2. 65-72.

[67] Warshall A.: A Theorem on Boolean Matrices. J. of Association for Computing Machinery, 9 (1962) 1, 11-12.