V Jubileuszowa Wyższa Górska Międzynarodowa Szkoła PTI

Szczyrk, 21 - 25 czerwca 1993

Polskie Towarzystwo Informatyczne
Oddział Górnośląski

Katowice 1993

Druk: PWR Sp. z o.o. W-wa, ul. Żwirki i Wigury 81, tel. 22 24 26

V JUBILEUSZOWA WYŻSZA GÓRSKA MIĘDZYNARODOWA SZKOŁA PTI

WSPÓŁCZESNE OBSZARY ZASTOSOWAŃ INFORMATYKI

Szczyrk 21-25 czerwca 1993

Program

PONIEDZIAŁEK 1993.06.21

9^{45} **Otwarcie Szkoły**

10^{00} - 11^{00} **mgr inż. Wojciech Głazek** (DOKP Katowice): *Założenia, projekt i realizacja sieci transmisji danych KOLPAK.*

przerwa na kawę

11^{30} - 12^{15} **mgr inż. Włodzimierz Adamski** (WSK OBR PZL - MIELEC) *Sieci komputerowe w polskim przemyśle lotniczym.*

12^{30} - 13^{15} **mgr inż. Włodzimierz Adamski** (WSK OBR PZL - MIELEC): *Elektroniczna postać dokumentacji konstrukcyjnej i technologicznej.*

15^{00} - 16^{00} **mgr inż. Mariusz Klapper** (Załad Systemów Komputerowych i Oprogramowania "K&MSOFT"): *Praktyczne doświadczenia opracowywania i wdrażania w przedsiębiorstwach kompleksowych systemów informatycznych.*

przerwa na kawę

16^{20} - 17^{20} **dr inż. Jacek Stochlak** (ICL POLAND): *Czy naprawdę dinozaury - rola i miejsce dużych systemów komputerowych w systemach informatycznych przedsiębiorstw.*

17^{45} - 18^{45} **Zbigniew A. Nowacki** (AKREM, Łódź): *Szablony klas i listy elastyczne w pakiecie narzędziowym POLY do języka C.*

WTOREK 1993.06.22

9^{00} - 10^{30} **dr Jerzy Skrzypek** (Akademia Ekonomiczna): *Metodologiczne aspekty modelowania procesów gospodarczych (w ujęciu Dynamiki Systemowej).*

przerwa na kawę

11^{00} - 12^{30} **dr inż. Marek Miłosz** (Politechnika Lubelska) *SYMULACJA SYSTEMÓW - Symulacja procesów dyskretnych.*

15^{00} - 16^{00} **Artur Kasprzyk** (Uniwersytet Wrocławski): *Obiektowo-zorientowane analiza i projektowanie.*

przerwa na kawę

16^{20} - 17^{20} **mgr inż. Zbigniew Benedykt** (BEFAMA Bielsko-Biała): *Co daje obiektowe podejście do analizy i projektowania systemów informatycznych.*

17^{45} - 18^{45} **dr inż. Stanisław Kędzierski** (Akademia Ekonomiczna Katowice): *Modelowanie obiektów dynamicznych.*

ŚRODA 1993.06.23

9^{00} - 10^{00} **Marek Wierzbicki** (Politechnika Łódzka): *Tworzenie obiektów poprzez dziedziczenie mono- i polimorficzne.*

10^{15} - 11^{15} **mgr inż. Mirosław Wieczorek** (Politechnika Śląska) *Podstawowe zagadnienia programowania zorientowanego obiektowo - polimorfizm i dziedziczenie klas w języku C++.*

przerwa na kawę

11^{40} - 13^{00} **dr inż. Marek Średniawa** (Politechnika Warszawska): *Podejście obiektowe do specyfikacji na przykładzie języka CCITT SDL'92.*

15^{00} - 16^{00} **mgr inż. Roman Siminski** (Uniwersytet Śląski): *Projekt i realizacja obiektowo zorientowanego podsystemu komunikacji z użytkownikiem.*

przerwa na kawę

16^{20} - 17^{20} **Borys Stokalski, Andrzej Maciej Wierzba** (InfoVIDE): *LBMS Systems Engineer. Zintegrowane środowisko do tworzenia systemów informatycznych.*

17^{45} - 18^{45} **mgr inż Piotr Woźniak** (SuperMemo World): *SuperMemo as a new tool increasing the productivity of a programmer. A case study - Programming in Object Windows.*

CZWARTEK 1993.06.24

9^{00} - 10^{20} **dr Joanna Kurzok-Derda** (Instytut Systemów Sterowania Katowice): *System ekspertowy EKSSTER dla wspomagania projektowania i dokumentowania układów automatyki przemysłowej na bazie sterownika STERISS-2.*

przerwa na kawę

10^{40} - 12^{00} **mgr Danuta Kajrunajtys** (Akademia Ekonomiczna, Kraków): *Komputerowe systemy wspomagania decyzji a modele sytuacji decyzyjnych.*

12^{15} - 13^{00} **Prezentacja firmy Siemens-Nixdorf**

15^{00} - 16^{00} **prof. Wojciech M. Jaworski** (Concordia University, Montreal) *InfoMAPY - inna metoda gromadzenia wiedzy.*

przerwa na kawę

16^{20} - 17^{20} **dr Marian Kuraś** (Akademia Ekonomiczna, Kraków): *Komunikacyjne aspekty tworzenia systemów informacyjnych.*

17^{45} - 18^{45} **dr Andrzej Zaliwski** (Akademia Ekonomiczna, Kraków): *CASAD - nowe podejście do modernizacji SI.*

PIĄTEK 1993.06.25

10^{00} - 11^{15} **dr inż. Jan Baranowski** (THOMSON POLKOLOR): *Wykorzystanie baz danych opartych na Btrievie i NetwareSQL w środowisku WINDOWS.*

przerwa na kawę

11^{30} - 13^{00} **dr inż. Jacek Stochlak** (ICL POLAND): *Systemy przetwarzania transakcyjnego OLTP. Modele i standardy.*

20^{00} - skutku **Bal Prezesów**

POSIŁKI

Śniadanie	-	8^{30}
Obiad	-	13^{30}
Kolacja	-	19^{00}

Wojciech Głazek
Ośrodek Informatyki Śląskiej DOKP

Założenia, projekt i realizacja sieci transmisji danych KOLPAK

1. Wstęp

Od roku 1989 nastąpiły w Polsce zasadnicze zmiany w zakresie inwestycji telekomunikacyjnych, w tym także dla potrzeb transmisji danych. Dla wszystkich stało się oczywiste, że dalszy rozwój zastosowań informatyki jest związany z rozpowszechnieniem technik transmisji danych. Obecnie wymienić można następujące bardziej znane inwestycje w zakresie transmisji danych będące w fazie realizacji lub projektowania:

a. Publiczna sieć POLPAK,
b. Sieć bankowa TELBANK,
c. Sieć akademicka NASK i sieć EARN oraz sieci poszczególnych środowisk akademickich,
d. Sieć prywatna banku PKO S.A. - PKONET,
e. Sieć prywatna Polskich Kolei Państwowych KOLPAK.

Polskie Koleje Państwowe (PKP) jako część infrastruktury państwa są modernizowane z kredytów Banku Światowego. Część tych kredytów przeznaczono na realizację inwestycji w zakresie informatyki, w tym na budowę prywatnej sieci transmisji danych PKP. W tym zakresie PKP idzie w ślady innych zarządów kolei europejskich, które z reguły zbudowały własne sieci transmisji danych. Także w krajach byłego RWPG, m.in. na Węgrzech i w byłej Czechosłowacji są budowane podobne sieci. Bazą dla budowy takich sieci są własne sieci kablowe zarządów kolei.

Decyzja o budowie sieci na PKP została poprzedzona pracami studialnymi firm konsultingowych. Były trzy takie firmy - po jednej z Kanady, Polski i Wlk. Brytanii. Analizowały one potrzeby PKP w zakresie informatyki, a jako konsekwencję - wymagania w zakresie transmisji danych. W rezultacie uzasadniono potrzebę budowy własnej sieci transmisji i wybrano standard X.25 styku użytkownika z siecią jako obowiązujący. Sieć tę nazwano KOLPAK.

PKP powołało komisję d/s przygotowania wymagań dla sieci. Komisja ta pracowała około 6 miesięcy i przygotowała wymagania techniczne. Po zatwierdzeniu tych wymagań (poszerzonych o część finansowo-prawną) przez Bank Światowy, ogłoszono przetarg.

2. Przebieg przetargu i kryteria techniczne wyboru ofert.

W przetargu wzięło udział jedenaście firm. Były to:

a. Alcatel (Francja),
b. Ascom (Szwajcaria),
c. AT&T (USA),
d. Bull (Francja),
e. Canac Telecom (Kanada),
f. I.T.S. (Niemcy),
g. Olivetti (Włochy),
h. Schrack Datacom (Austria),
i. Siemens AG (Niemcy),
j. Siemens Nixdorf (Niemcy),
k. US Sprint (USA).

Przedmiotem przetargu były:
a. węzły dla sieci,
b. centrum zarządzania,
c. poczta elektroniczna z oprogramowaniem EDIFACT,
d. modemy dla połączeń międzywęzłowych wraz z centrum zarządzania siecią modemową,
e. oferta na wykonawstwo sieci, sprzęt pomiarowy i szkolenia.

Komisja przetargowa nie dysponowała konkretnym projektem sieci transmisji. W okresie poprzedzającym przetarg prowadzono w PKP prace studialne w zakresie zapotrzebowania na terminale i hosty dla różnych, przewidywanych informatycznych systemów aplikacyjnych. Uzyskane dane, a także rozpoznanie podobnych systemów funkcjonujących w Europie Zachodniej pozwoliły określić rząd wielkości zapotrzebowania na poszczególne rodzaje usług w sieci transmisji danych. Pewne systemy już funkcjonowały, np. systemy rezerwacji miejsc w ruchu krajowym i międzynarodowym; niektóre stacje rozrządowe zostały wyposażone w tzw. System Kierowania Pracą Stacji, usprawniający obieg dokumentów technologicznych na stacji itd. W rezultacie tych prac studialnych ustalono, że w dającej się określić przyszłości 2 - 5 lat należy przewidywać następujące zapotrzebowanie na usługi w zakresie transmisji danych:

a. Liczba terminali typu IBM PC włączonych do sieci KOLPAK będzie wynosiła około 6 tysięcy.
b. Sieć KOLPAK musi stać się podstawą do połączenia około czterdziestu instalacji komputerowych (klasy np. podwojonego RISC 6000 prod. IBM) zlokalizowanych na terenie całej sieci PKP, z centralną instalacją komputerową znajdującą się w Warszawie.
c. PKP będzie modernizowało swoją sieć central telefonicznych w kierunku budowy central przystosowanych do ISDN (przetarg na siedem takich central jest aktualnie otwarty).
d. Sieć analogowych połączeń telefonii nośnej będzie zastępowana siecią kabli światłowodowych. W roku 1992 zakończono budowę kabla światłowodowego relacji Koszalin - Gdańsk - Warszawa - Kraków - Katowice - Cieszyn. Przewiduje się także rozpowszechnienie połączeń w technice PCM.
e. Poczta elektroniczna powinna zastąpić istniejącą sieć łączności telegraficznej, a na etapie przejściowym musi istnieć możliwość współpracy obu sieci.
f. Sieć transmisji powinna być użyta także do przekazywania telefaxów.
g. Powstająca sieć będzie eksploatowana bez większych zmian w zakresie stosowanego wyposażenia co najmniej do końca lat 90-tych.

Na bazie tych założeń sformułowano pewne wymagania - dotyczące stanu aktualnego oraz najbliższej przyszłości - które muszą być spełnione przez wszystkich biorących udział w przetargu. Wymagania te są następujące:

a. Podstawowym rodzajem łączności w połączeniach międzywęzłowych będzie łączność poprzez kanały telefonii nośnej, a praktycznie osiągalna prędkość transmisji w tych kanałach nie przekroczy 14.4 kbps dla transmisji synchronicznej i 19.6 kbps dla transmisji asynchronicznej. Nie wyklucza to stopniowego przechodzenia na transmisję w tzw. frame relay (2.048 Mbps), węzły sieci powinny być przystosowane do bezpośredniej współpracy z takimi kanałami, a producent powinien posiadać referencje w tym zakresie.
b. W węzłach sieci wszystkich rodzajów, w tym także najmniejszych należy stosować technologię 32 bitową.
c. Z punktu widzenia systemu zarządzania siecią niezbędne jest zrealizowanie następujących postulatów:
   - istnieje jedno centrum zarządzania dla wszystkich rodzajów węzłów,
   - węzły sieci powinny być widziane z centrum zarządzania jako logicz-

nie pojedyncze urządzenia a nie np. jako zespół odrębnych jednostek,
   - parametry węzłów będą ustawiane zdalnie z centrum.

d. Poczta elektroniczna musi posiadać możliwość współpracy z siecią telegraficzną, w tym także musi rozpoznawać telegraficzne adresy odbiorców.
e. Musi istnieć możliwość wymiany informacji nadawanych/odbieranych telefaxem z abonentami poczty elektronicznej.
f. Konstrukcja węzłów sieci musi przewidywać sprzętową redundancję pakietów. Nie określa się minimalnego MTBF dla węzłów i centrum zarządzania siecią.
g. Dostawca musi zapewnić możliwość współpracy z innymi sieciami X.25 poprzez dostarczenie oprogramowania realizującego wymagania standardów X.75 oraz X.121, a także oprogramowania dla taryfikacji.
h. Dostawca musi wyposażyć węzły w opcję X.32 tj. dostęp terminala X.25 do węzła poprzez sieć telefoniczną komutowaną.

Dla porównania warunków finansowych proponowanych przez różnych dostawców, do wymagań technicznych dołączono dane na temat tego, jakiej wielkości i ile węzłów przewiduje się w sieci, z zastrzeżeniem, że dane rzeczywiste mogą się różnić od określonych w materiale przetargowym o 25%.

Wyszczególniono trzy podstawowe grupy węzłów, dla których oferenci mieli określić koszt konfiguracji sieci. Obrazuje to tabela 1.

Określono również liczbę modemów oraz urządzeń dla pasma podstawowego *(base-band)*, przewidzianych dla realizacji połączeń międzywęzłowych. Założono, że będą one realizowały zalecenia V.22/V.22bis, V.32/V.32bis, V.54, V.42/V.42bis MNP5, będą pracować na łączach trwałych/komutowanych oraz w trybie synchronicznym/asynchronicznym. Liczbę modemów przyjętą dla oceny oferty od strony finansowej ustalono na 1200 sztuk, urządzeń base-band na 100 sztuk.

W wyniku pracy komisji przetargowej wybrana została oferta firmy US Sprint. Firma ta zaproponowała następujące rozwiązania techniczne jako realizację sieci KOLPAK:

a. Centrum zarządzania oparte na mikrokomputerze firmy Data General z systemem operacyjnym UNIX.
b. Jako węzły sieci zaproponowano urządzenia firmy US Sprint (patrz tabela 2).
c. Firma US Sprint zaproponowała własną pocztę elektroniczną o nazwie Sprintmail (wraz z oprogramowaniem EDIFACT) zrealizowaną na komputerze TANDEM CLX 820.

RODZAJ WĘZŁA (min. liczba portów)	LICZBA WĘZŁÓW	UWAGI
200	8	Istnieje opcja frame-relay Realizuje X.25, X.3/X.28/X.29,X.32,X.75
100	40	Nie przewiduje się frame-relay ani X.75
16	190	j.w.

TABELA 1: Grupy węzłów

TYP	RODZAJ WĘZŁA (max liczba portów)	LICZBA ZAKUPIONYCH WĘZŁÓW
TP4954	432 porty synchroniczne lub 736 asynchronicznych i 32 synchroniczne	10
TP4944	96 synchron./asynchron.	75
TP8010-16	16 synchron./asynchron.	212
	RAZEM	297

TABELA 2: Urządzenia zaproponowane prze US Sprint jako węzły sieci

d. Jako modemy zaproponowano urządzenia Penril Datalink 14,4 (DLX V.32 M), a jako urządzenia base-band urządzenia Racal Comlink 7.
e. Kontakt z siecią telegraficzną zapewnią urządzenia firmy Hasler typu Hasler Telex Unit (16 sztuk) podłączone do węzłów serii 4900.
f. Wymiana telefaxów między abonentami sieci KOLPAK a siecią telefoniczną będzie możliwa dzięki przyłączeniu do maszyny TANDEM CLX 820 ośmiu zestawów PC/Eicon.
g. Dla przygotowania aplikacji i testowania sprzętu oraz oprogramowania zakupiono odrębną sieć testową złożoną z dwóch węzłów.

3. Realizacja sieci KOLPAK

W momencie wyboru oferenta przez komisję przetargową rozpoczęto pracę nad szczegółową topologią sieci. Obecnie prace te trwają nadal w tym sensie, że w pewnych sytuacjach szczegółowa lokalizacja węzłów jest określana przez funkcjonujący w każdej DOKP Zespół Wykonawczy d/s Sieci KOLPAK.
Centralnie w Dyrekcji Generalnej PKP działa komisja uzgadniająca te zmiany i koordynująca realizację sieci. Pewne trudności spowodował fakt, że nie określono do końca, jakie systemy aplikacyjne będą funkcjonowały w przyszłości w oparciu o sieć. Z tego względu KOLPAK posiada wszystkie cechy sieci publicznej, a nie jest siecią zorientowaną na jakiś rodzaj użytkownika. Mimo to zasadniczy proces inwestycyjny budowy sieci KOLPAK rozpoczął się i obecnie został zrealizowany jego pierwszy etap. Cała budowa sieci została podzielona na siedem takich etapów, a jej koniec przewidziano na grudzień 1994 r.

Należy podkreślić, że przedstawiony tutaj zestaw urządzeń dostarczony przez firmę US Sprint nie wystarcza dla uruchomienia abonenta sieci. Aby uzyskać w pełni funkcjonującą sieć transmisji danych, potrzebne jest to wszystko, co służyć będzie dla połączenia procesu aplikacyjnego z najbliższym węzłem sieci. W najprostszej sytuacji będą to dwa modemy oraz dopasowanie oprogramowania aplikacyjnego do współpracy ze zdalnym partnerem poprzez sieć. Wszystko to trzeba będzie dopiero przygotować.

Wybór dostawcy sieci i takiej usługi jak poczta elektroniczna w znacznej mierze determinuje zakres wyposażenia dodatkowego, zapełniającego lukę między procesem aplikacyjnym a węzłem sieci.

4. Porównanie sposobu przeprowadzenia budowy sieci pakietowej przez PKP i Deutsche Bundesbahn.

Deutsche Bundesbahn (DB) rozpoczęło realizację swojej sieci X.25 (sieci IN) w latach 1986-1988, w zupełnie innych warunkach niż PKP. DB miało wcześniej sieć transmisji opartą o sprzęt firmy SIEMENS (system TRANSDATA) i działającą na zasadzie hierarchicznej sieci modemowej. Kłopoty, jakie wiązały się z jej stosowaniem, spowodowały, że DB zdecydowało się przejść na standard X.25. Finansowanie sieci IN odbywało się w pełni ze środków własnych DB lub budżetu państwa. DB znacznie wydłużyło (w stosunku do PKP) proces przygotowawczy, trwający w DB ponad 1 rok. Prace przygotowawcze obejmowały szczegółową inwentaryzację potrzeb DB w każdej Dyrekcji Okręgowej (DB ma organizację podobną do PKP). Podobnie wcześniej dokonano analizy sieci kablowej w każdej DOKP. Przeprowadzono osobno przetargi na węzły małe, duże i modemy. Nadrzędna sieć *(backbone)* dla publicznej sieci pocztowej i dla kolejowej jest zrealizowana na tym samym sprzęcie firmy SIEMENS AG. Ogólnie przyjęto takie założenie, że sieć IN realizuje usługi stanowiące podzbiór usług sieci pocztowej DATEX-P i w każdej chwili można zastąpić sieć kolejową poprzez sieć pocztową. Obecnie ta sytuacja zmieniła się, bo zmieniły się przepisy prawne funkcjonowania poczty w zakresie transmisji danych (został zlikwidowany monopol poczty). Sieć kolejowa IN może w przypadku prywatyzacji stać się konkurentem dla DATEX-P. W okresie rozpoczęcia budowy sieci IN podkreślano, że obie sieci są własnością instytucji rządowych i - jako takie - należą do państwa.

5. Wnioski

Realizacja sieci X.25, jak każdej dużej inwestycji, jest procesem wieloetapowym i skomplikowanym organizacyjnie. Wobec braku w Polsce doświadczenia w stosowaniu i eksploatacji tego rodzaju sieci, zadanie stojące przed władzami PKP, a później przed jednostkami wykonawczymi były i są szczególnie trudne.

Obecnie wydaje mi się, że następujące wnioski z dotychczasowego przebiegu zdarzeń należy uznać za szczególnie ważne:

a. Akcja zbierania ofert musi być poprzedzona uruchomieniem jakiejś własnej, nawet małej sieci, zbudowanej w tym samym standardzie jak sieć główna. PKP miało taką sieć, zakupioną dla systemu rezerwacji miejsc w ruchu międzynarodowym.
b. Wybór poszczególnych elementów sieci (centrum zarządzania, węzły, modemy, poczta elektroniczna) stwarza ogromne trudności w przypadku braku doświadczenia praktycznego, polegającego na wielomiesięcznym zapoznawaniu się z opisami technicznymi, próbami wdrożenia itd. Opis oferty danej firmy nie może być pełny z oczywistych względów, a z drugiej strony każdy producent ciągle modyfikuje swoje wyroby. W tej sytuacji przeprowadzenie właściwego wyboru jest szczególnie trudne. Przepisy Banku Światowego nie zezwalają na przetarg wieloetapowy co jeszcze bardziej komplikuje sprawę.
c. Administrator sieci transmisji danych musi, dla własnego bezpieczeństwa oraz wygody, testować wszystkie wyroby dopuszczone do stosowania na własnej sieci (sprzęt i oprogramowanie). Do tego celu powinien posiadać wyodrębnioną sieć testową.

Włodzimierz Adamski
WSK-OBR-PZL-Mielec

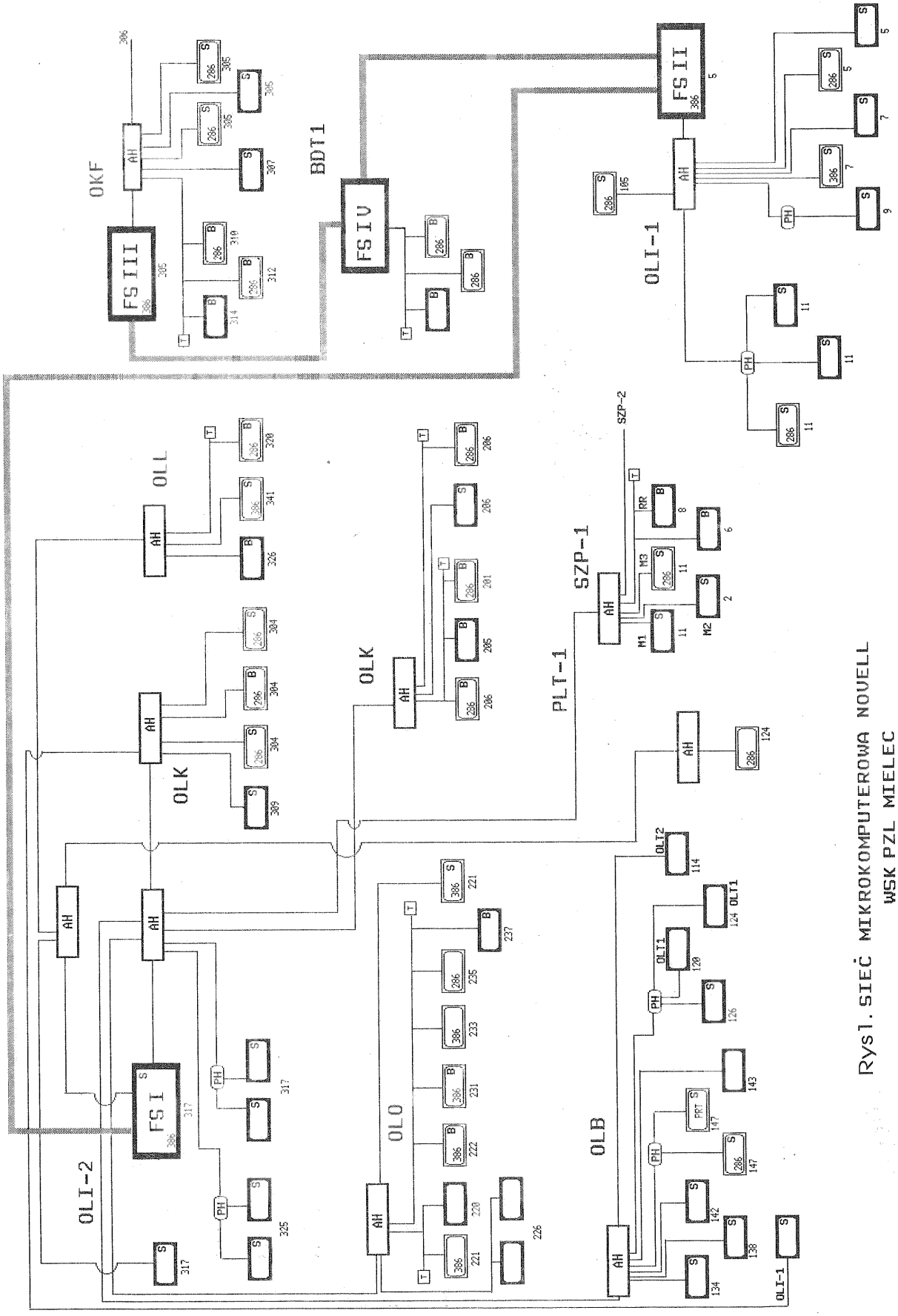
Sieci komputerowe w polskim przemyśle lotniczym

Znaczny rozwój mikrokomputerów, jaki nastąpił w ostatnich czasach, zmienił sposób podejścia do komputeryzacji przedsiębiorstw. W biurach instaluje się coraz więcej sieci lokalnych (NOVELL NETWARE) wypierających duże instalacje komputerowe. Przyjmuje się, że duży komputer jest wygodniejszy wówczas, gdy program sięga do bazy danych o wielkości co najmniej 40 MB. Budowę sieci mikrokomputerowej w WSK PZL Mielec rozpoczęto w 1987 roku. Obecnie w lokalnej sieci mikrokomputerowej pracuje 65 mikrokomputerów (od PC/AT do PC/486) - rys. 1. W skład sieci wchodzą cztery *file servery*. Pierwszy o nazwie BDK1 z dyskami o pojemności 2 x 320 MB i dyskiem roboczym 1 GB obsługuje bazy danych konstrukcyjnych, bazy danych administracyjnych (planowanie, rozliczanie z wykonanych prac, obsługa sekretariatów biur konstrukcyjnych), obliczenia wytrzymałościowe statyczne i zmęczeniowe, aeroelastyczności, aerodynamiczne i obciążenia oraz bazę danych z prób statycznych i dynamicznych, i pomiarową bazę danych z prób związanych z "oblotem" samolotów. Drugi serwer z dyskami o pojemności 2 x 550 MB, o nazwie BDT1, obsługuje biura technicznego przygotowania produkcji i biura technologiczne. Technologiczną bazą danych zarządza system DSS. Trzeci serwer GEO1 zawiera bazę danych związanych z numerycznym kształtem samolotu oraz dane dotyczące programowania obrabiarek sterowanych numerycznie. Pojemność dysku serwera wynosi 600 MB. Głównymi systemami na tym serwerze są DAMS i SPO. Ostatni, czwarty serwer FK1 zawiera bazę danych kadrowych, płacowych i finansowych, obsługuje księgowość, gospodarkę środkami trwałymi itp. Pojemność dysku serwera wynosi 150 MB. Wszystkie serwery są połączone między sobą kartami ETHERNET. W serwerach BDK1 i BDT1 drugi dysk jest "zwierciadlaną" kopią pierwszego, co stanowi zabezpieczenie niektórych ważniejszych danych. W pozostałych serwerach kopie są wykonywane na dyskach lokalnych, które posiadają wystarczającą pojemność (po 230 MB).

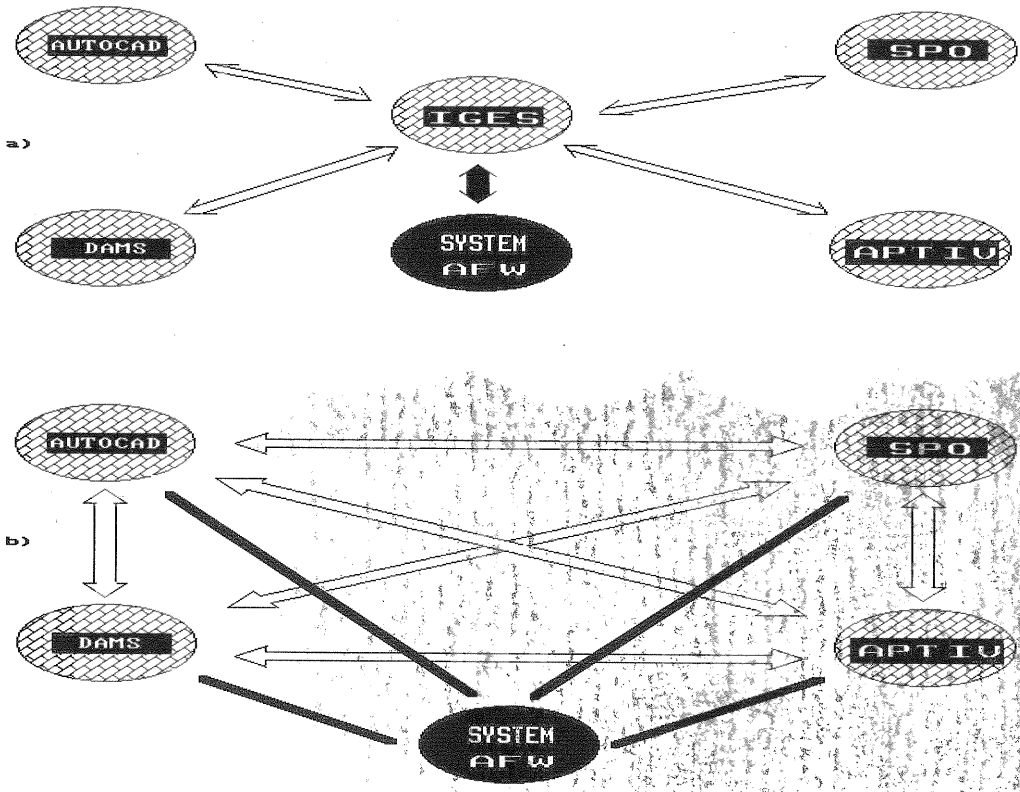
Nowoczesne komputery i ich oprogramowanie odgrywają dominującą rolę w rozwoju i wytwarzaniu w WSK-PZL Mielec. Od wielu już lat stosowana jest technologia CAD/CAM *(Computer Aided Design and Manufacturing)*.

Eksploatowane są następujące systemy (niektóre od 1977 roku): numeryczne odwzorowanie geometrii NMG *(Numerical Master Geometry)*, DAMS *(Design All Manufacturing Surface's)*, AUTOCAD (projektowanie konstrukcji wytworu), baza danych specyfikacji konstrukcyjnych DSS *(Design Specification System)*, automatyczne programowanie OSN do 2 1/2 osi SPO, automatyczne programowanie OSN do 5 osi APT IV, system obliczeń technicznych AFW (Aerodynamika, Flatter, Wytrzymałość).

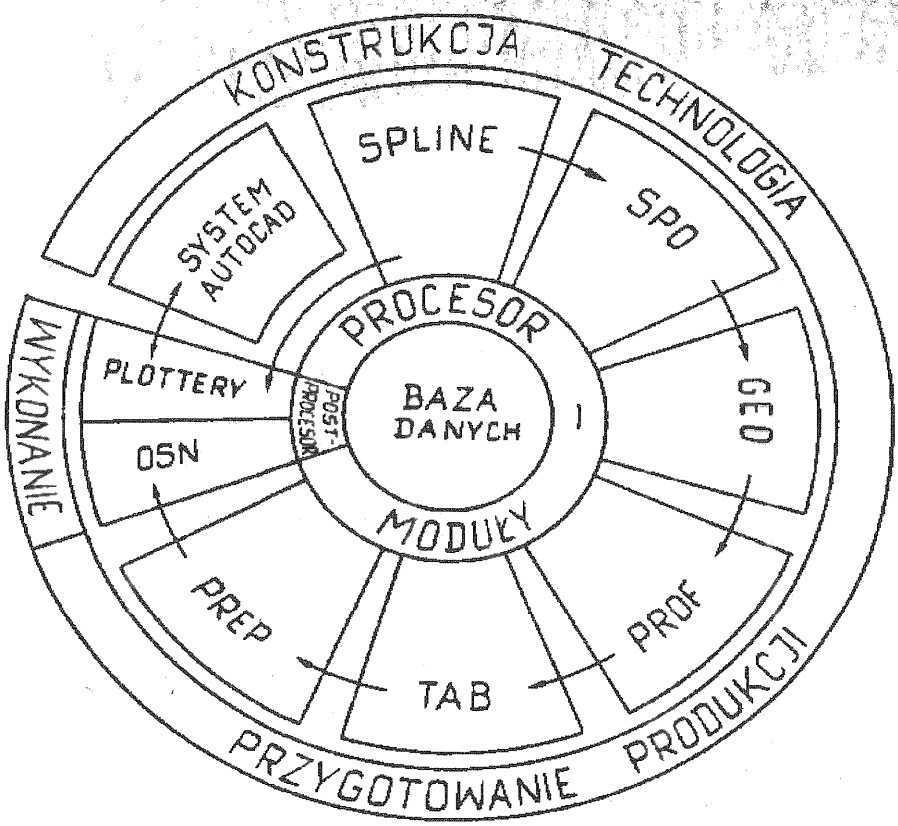
Systemy mogą być integrowane na różne sposoby. Najprostszą metodą jest wprowadzenie translatorów. I tak: dla czterech systemów (rys. 2b), zachodzi konieczność zbudowania dwunastu różnych translatorów. Wprowadzenie piątego systemu wymaga dołożenia dodatkowo ośmiu



Rys 1. SIEĆ MIKROKOMPUTEROWA NOVELL
WSK PZL MIELEC



Rys. 2 Polaczenie roznych systemow za pomoca translatorow:
a) z standardem za pomoca neutralnego zbioru IGES
b) bez standardu



Rys.3 System projektowania obrabiarek sterowanych numerycznie SPO

translatorów. Tak więc w miarę rozbudowy systemu liczba translatorów rośnie w sposób geometryczny. Dlatego lepszym rozwiązaniem jest wykorzystanie standardowego formatu wymiany danych, który umożliwia współpracę z wieloma innymi systemami. Do najbardziej znanych standardowych pakietów wymiany danych należą:

- IGES *(Initial Graphics Exchange Specification)* - W. Brytania,
- SET *(Specifications du Standard D'Exchange et de Transfert)* - Francja,
- VDA-FS - RFN.

Pakiet IGES umożliwia przenoszenie danych między dwoma różnymi systemami poprzez stworzenie neutralnego zbioru. Dla czterech systemów potrzeba ośmiu translatorów. Dołożenie piątego systemu wymaga tylko dwóch translatorów: służącego do odczytania - "preprocesora" i zapisywania - "postprocesora" (rys. 2a). Sposób zintegrowania systemów istniejących i eksploatowanych w sieci WSK PZL MIELEC przedstawia rys. 5. Przyszłościowym pakietem alternatywnym do IGES jest pakiet PDES *(Product Data Exchange Specification)*.

Obecnie pakiet IGES jest proponowany w większości systemów CAD. Jego ograniczenia są dobrze znane, np. w wersji 4 nie jest uwzględnione tzw. modelowanie bryłowe *(Solid model)*, które można znaleźć już w prawie każdym systemie CAD.

Problemy, jakie stwarzał standard IGES, przyczyniły się po międzynarodowej konsultacji do powstania i rozwoju standardu PDES *(Product Data Exchange Specifikation)*, który jest równoważny standardowi STEP *(Standard for the Exchange of Product Model Data)*. STEP używa specjalnego języka o nazwie Expass, który posiada niektóre cechy języka PASCAL. STEP jest obecnie proponowany przez Międzynarodową Organizację ISO jako standardowy pakiet wymiany danych.

Na rys. 6 przedstawiono najczęściej używane w przemyśle lotniczym i samochodowym systemy CAD/CAM. Jak widać, niekwestionowanym liderem na świecie jest system CATIA autorstwa lotniczej firmy francuskiej DASSAULT sprzedawany przez firmę IBM.

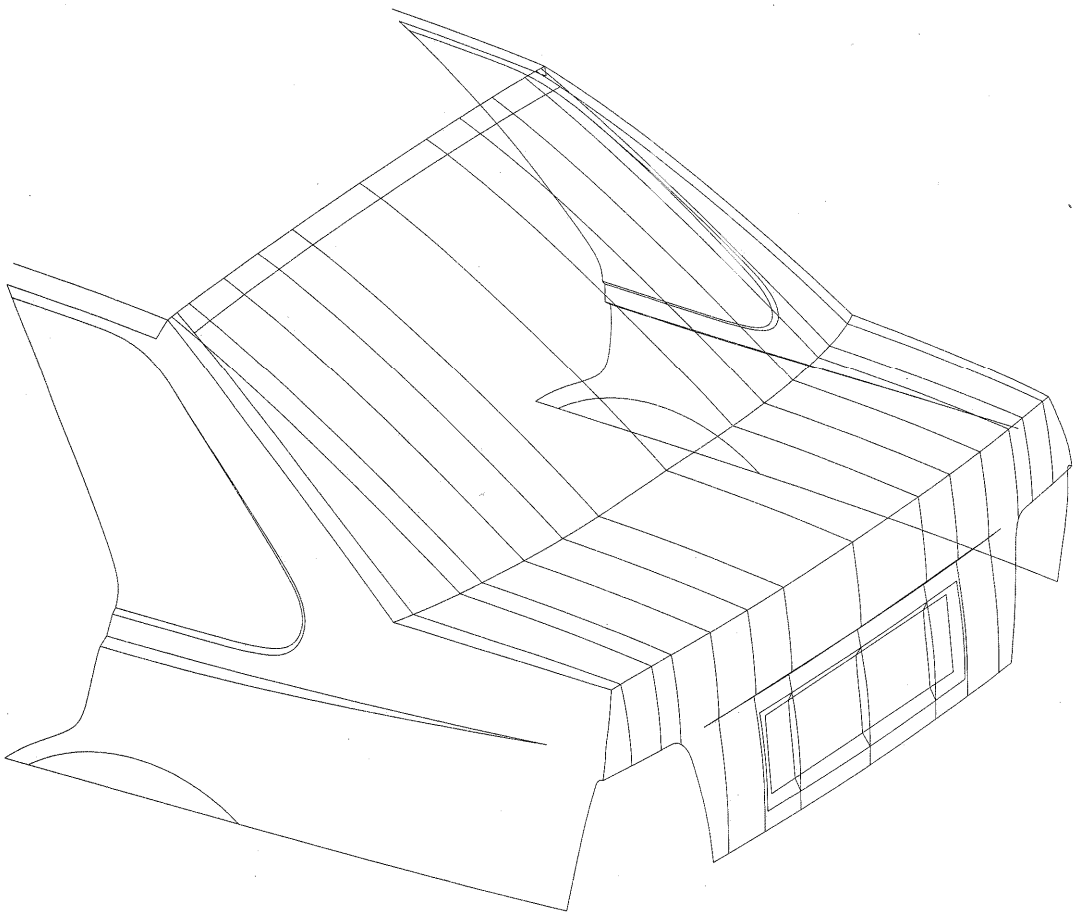
Przemysł lotniczy, bardziej niż inne gałęzie przemysłu, rozwinął szeroką kooperację między poszczególnymi zakładami. Różnorodność powstałych systemów CAD/CAM narzuciła od razu problem ich integracji i wymiany danych między tymi systemami.

Eksploatowane obecnie w przemyśle europejskim systemy, związane z numerycznym opisem kształtu, takie jak:

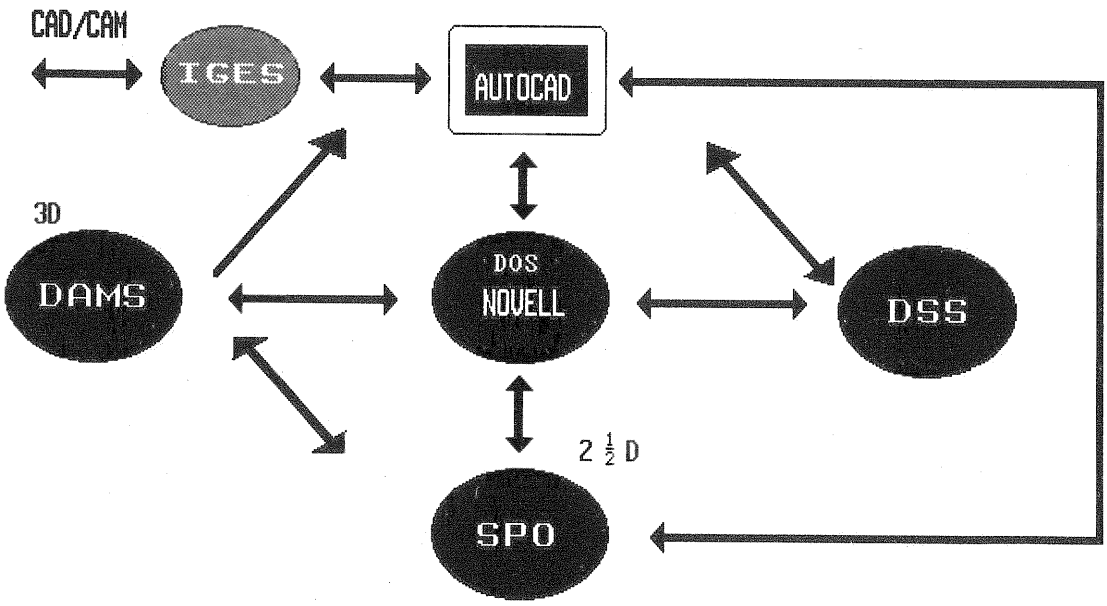
* MBB - Cadam,
* Aerospatiale - Computervision , STREAM 100,
* Aeritalia - Computervision,
* Dassault - Catia,
* British Aerospace - NMG, ANVIL 4000,
* Saab-Scania - NMG, Catia,
* PZL Mielec - NMG, DAMS,

umożliwiają zwiększenie wydajności konstruktora, uzyskanie polepszenia konstrukcji przez zbadanie alternatywnych rozwiązań, skrócenie czasu opracowań oraz opracowanie programów obróbczych na obrabiarkach sterowanych numerycznie.

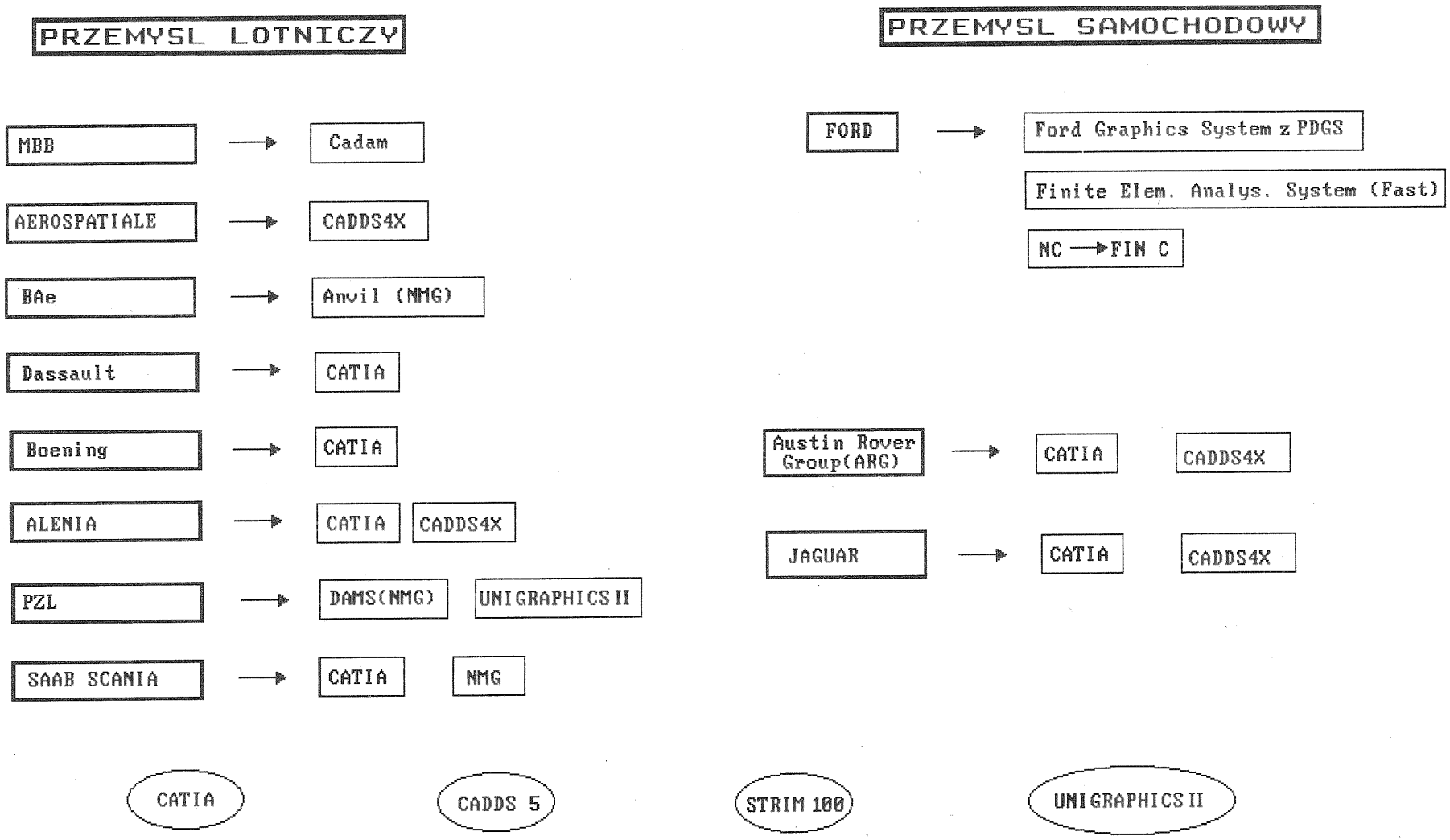
Dzięki wprowadzeniu sieci mikrokomputerowej bardzo rozwinął się system SPO automatycznego programowania OSN do 2 1/2 osi. Połączenie tego systemu z systemem AutoCad dało technologowi możliwości szybkiego i dobrego przygotowania programów obróbczych na OSN (rysunki na monitorze w kilku warstwach, razem lub osobno, np. kontur teoretyczny detalu, droga narzędzia w obróbce zgrubnej, wykańczającej, ruchy jałowe, powiększenia itp.). Na rys. 3 przedstawiono schemat przepływu informacji między poszczególnymi modułami systemu SPO . Stosowany w WSK PZL Mielec system CAD/CAM 3D/2D składa się z trzech głównych pakietów (rys. 5):



Rys.4 Geometria nadwozia samochodu



Rys.5 System CAD/CAM w sieci NOVELL



Rys. 6 Najczesciej stosowane systemy CAD/CAM

- DAMS *(Design All Manufacturing Surface's)*,
- SPO *(System Programing Operation)*,
- DSS *(Design Specification System)*.

System DSS stanowi uzupełnienie dwóch pierwszych pakietów i spełnia następujące funkcje:
- wprowadzanie i aktualizacje specyfikacji konstrukcyjnej,
- przeglądanie i edycja części, zespołów i podzespołów,
- obliczanie liczby części na zespół i wyrób oraz liczby części zapasowych,
- prowadzenie prac związanych z rozdziałem robót w technicznym przygotowaniu produkcji,
- prowadzenie prac związanych ze specyfikacjami materiałowymi,
- obliczanie ciężarów zespołów, wycena wyrobu,
- prowadzenie emisji dokumentacji uruchomieniowej,
- ewidencja wykazów materiałów, narzędzi, przyrządów i technologii,
- diagnozowanie poprawności kompletacji konstrukcyjnych.

System wdrożono przy produkcji samolotu AN-28, który składa się z 17500 części, 5100 zespołów i 4000 części znormalizowanych. Baza danych wszystkich wersji i kompletacji zajmuje 35 MB. Obecnie w bazie danych znajdują się kompletne informacje o produkowanych przez przedsiębiorstwo wyrobach. Samolot AN-28 jest produkowany w sześciu wersjach i czterech kompletacjach, samolot M-20 w siedmiu wersjach i czterech kompletacjach, samolot IRYDA I-22 w jednej wersji i dwóch kompletacjach, samolot SOCATA w czterech wersjach. Pozostałe wyroby, jak SATURN i M-26 są produkowane na razie w jednej wersji. Baza danych wyżej wymienionych wyrobów zajmuje łącznie około 84 MB. Większość stosowanych systemów jest eksploatowana zarówno na dużym komputerze IBM 4381 jak i w sieci mikrokomputerowej NOVELL (rys. 7). Dlatego też planuje się połączenie tych dwóch sieci w jedną globalną sieć przy wykorzystaniu protokołu SNA.

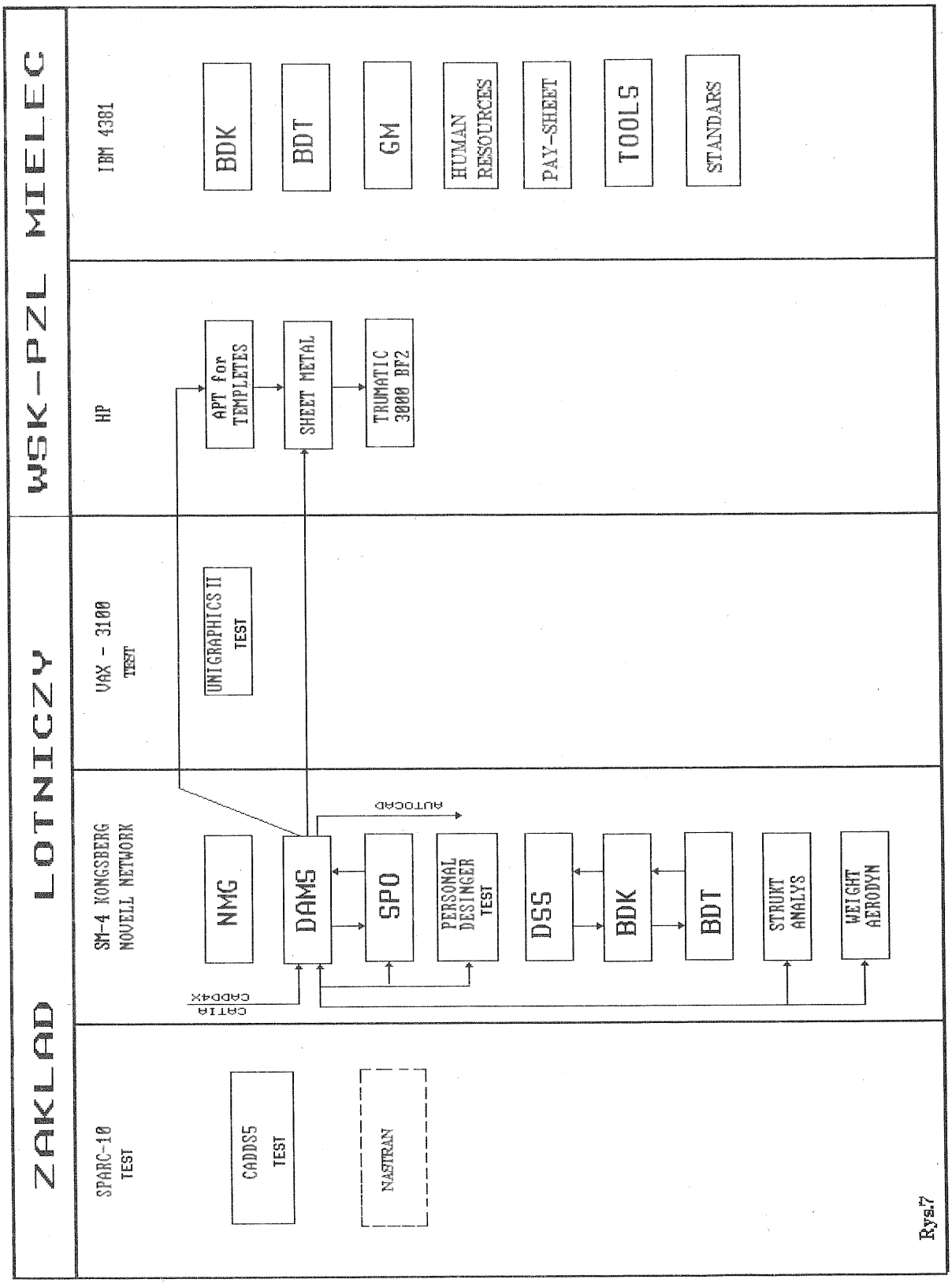
Rozwój informatyki w przedsiębiorstwach należy rozpatrywać w dwóch aspektach:
1. Sprzętowym, tj. komputerów, minikomputerów, mikrokomputerów oraz innych związanych z nimi maszyn i urządzeń niezbędnych do zapewnienia pracy komputera.
2. Programowym i systemowym, tj. dotyczącym używanego na komputerach oprogramowania i systemów oprogramowania.

Kompleksowa komputeryzacja przedsiębiorstwa jest niezwykle kosztowna i czasochłonna. Oprogramowanie użytkowe systemu projektowania i wytwarzania wspomaganego komputerem CAD/CAM można budować w różny sposób. Analiza istniejących systemów projektowo-produkcyjnych pokazuje, że jedynie 5% oprogramowania jest rzeczywiście oryginalnym wkładem autorów systemu danej generacji. Ponad 80% oprogramowania jest kopią oprogramowania poprzedniej wersji, a około 10% oprogramowania stanowi lekko zmodyfikowane opracowanie.

Tak więc szybkiemu rozwojowi komputeryzacji nie towarzyszą wielkie skoki technologiczne w strefie oprogramowania. Zaprojektowana sieć jest siecią otwartą, tzn. może być rozbudowywana o dodatkowe File Serwery, stacje robocze, zasoby dyskowe.
Rozmiary możliwości operacyjnych sieci są bardzo wielkie:
- maksymalna liczba czynnych stanowisk roboczych 250,
- maksymalna pojemność nośników danych 32000 GB,
- maksymalna wielkość pliku 4 GB.

Jak więc widać, pozwala to rozwiązać prawie każdy praktyczny problem z zakresu komputeryzacji przedsiębiorstwa, czy instytucji.

Zastosowanie odpowiedniej sieci mikrokomputerowej wraz z odpowiednim systemem projektowania geometrii nume-


ZAKŁAD LOTNICZY
WSK-PZL MIELEC
SPARC-10
TEST
SM-4 KONGSBERG
NOVELL NETWORK
VAX - 3100
TEST
HP
IBM 4381
CADDS5
TEST
NASTRAN
CATIA
CADDAX
NMG
DAMS
SPO
PERSONAL DESINGER
TEST
DSS
BDK
BDT
STRUKT ANALYS
WEIGHT AERODYN
AUTOCAD
UNIGRAPHICS II
TEST
APT for TEMPLETES
SHEET METAL
TRUMATIC 3000 BF2
BDK
BDT
GM
HUMAN RESOURCES
PAY-SHEET
TOOLS
STANDARS
Rys.7

rycznej i odpowiednio zorganizowaną centralną bazą danych geometrycznych, dało konstruktorom i technologom doskonałe i sprawne narzędzie w ich pracy. Sieć mikrokomputerowa z powodzeniem zastąpiła duże i drogie instalacje komputerowe, które są stosowane przez wszystkie wielkie firmy lotnicze. Umożliwia ona każdemu upoważnionemu użytkownikowi natychmiastowy dostęp do banku danych geometrycznych.

Wykaz literatury

[1] *Cesky Vybor Strojnicke Spolecnosti CSVTS. Stav a Vyuziti Vypocenti Techniky Pri Konstruovani a Vyrobe Letadel*, Duben 1984.

[2] Adamski W.: *Projektirowanie i praiswodstwo samoliotow s ispolzowaniem EWM w predprijatii WSK "PZL-Mielec"*, Konferencja naukowa RWPG, Moskwa, wrzesień 1986.

[3] Adamski W.: *Projektowanie i wytwarzanie samolotów wspomagane komputerem w WSK "PZL-Mielec"*, Konferencja Naukowa, Rydzyna 1988.

[4] Mayer R. J.: *One answer to the problems of CAD database exchange*, Byte, no. VI/1987.

[5] Adamski W.: *Rozwój komputeryzacji Ośrodka Badawczo-Rozwojowego OBR-SK Mielec*, Wrzesień 1987.

[6] Elliot W. S.: *Computer-aided mechanical engineering: 1958 to 1988*, Computer-Aided Design, Volume 21, Number 5, June 1989.

[7] Adamski W.: *Integracja systemów komputerowych w przemyśle lotniczym*, III Konferencja Naukowa aktualnych problemów lotnictwa polskiego, Warszawa, październik 1990.

[8] Saab Aicraft Division: *CAD/CAM and Geometry*, Saab Aicraft Division application of advanced technology. II 1991.

[9] Adamski W.: *Integracja systemów komputerowych w polskim przemyśle lotniczym*, VII Krajowa Konferencja Automatyzacji Procesów Przemysłowych, Kozubnik, wrzesień 1992.

[10] Adamski W.: *Stosowane systemy CAD/CAM w polskim przemyśle lotniczym*, XV Międzynarodowe Sympozjum Naukowe, Zielona Góra, IV 1993.

Włodzimierz Adamski
WSK-OBR-PZL-Mielec

Elektroniczna postać dokumentacji konstrukcyjnej i technologicznej

W systemie projektowania wspomaganego komputerem ważne jest jednoznaczne określenie zapisu konstrukcji. Dotyczy to opisu struktury zewnętrznej tj. geometrii zewnętrznej samolotu, kształtu nadwozia samochodu czy kształtu statku morskiego. Zagadnienie to rozwiązuje system definiowania powierzchni o podwójnej krzywiźnie **DAMS**. System numerycznego modelowania zewnętrznych kształtów samolotu **DAMS** (**D**esign **A**ll **M**anufacturing **S**urfaces) jest kompatybilny co do danych wejściowych i wyjściowych z systemem **NMG**, czy **ANVIL** stosowanymi w brytyjskim i szwedzkim przemyśle lotniczym (**British Aerospace** , **SAAB-SCANIA**).

System składa się z czterech głównych modułów i dwóch preprocesorów służących do przygotowania danych wejściowych. System jest systemem otwartym tzn. można go uzupełniać o swoje własne specjalistyczne moduły. System działa niezależnie na mikrokomputerze kompatybilnym z **IBM PC/AT** z kartą **Hercules**, **EGA**, **VGA** lub w wersji sieciowej (**NOVELL**), gdzie jest założona centralna baza danych geometrycznych na "file serwerze".

Ze względu na optymalne wykorzystanie pamięci operacyjnej komputera, system ten posiada budowę modułową (rys. 1). Geometrie zespołów wyrobu opisują powierzchnie z nazwami z rozszerzeniem **.srf**, które są tworzone przez moduł **DAMS0**. Dokładny opis wszystkich typów zbiorów przedstawiono na rys. 2. System ten umożliwia także obliczenie drogi narzędzia (freza kulistego) obrabiarki sterowanej numerycznie.

System składa się z następujących modułów:

-DAMS0: moduł definiowania powierzchni. W tym module konstruktor projektuje i tworzy numeryczny kształt powierzchni. Moduł ten posiada dodatkowe funkcje jak:

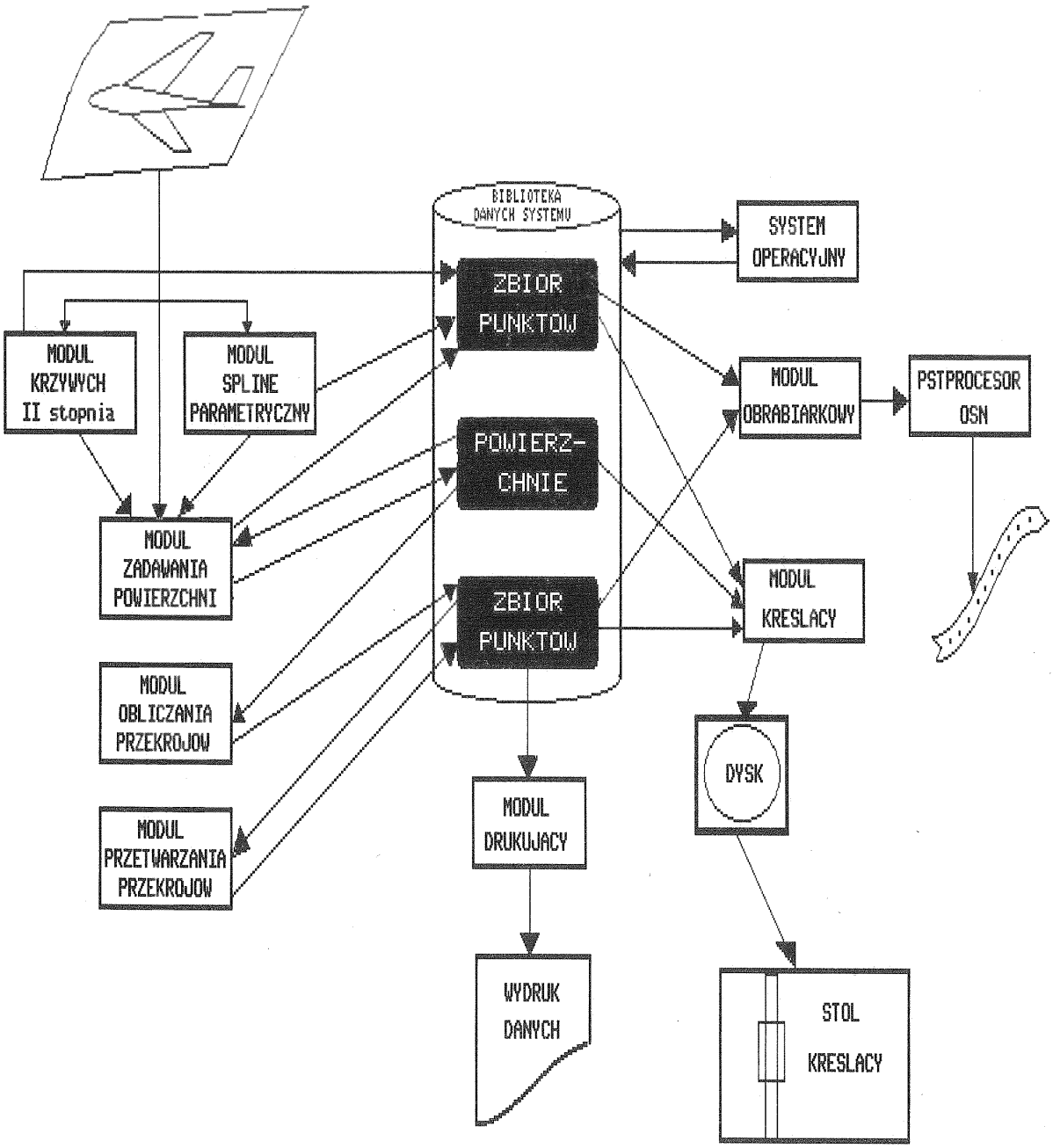
- transformacja powierzchni z jednego układu w drugi,
- łączenie kilku powierzchni w jedną,
- zmiana założonej gęstości siatki,
- wydzielanie określonego fragmentu powierzchni.

Powierzchnia jest określonym zbiorem dyskowym identyfikowanym przez nazwę danego zbioru dyskowego.

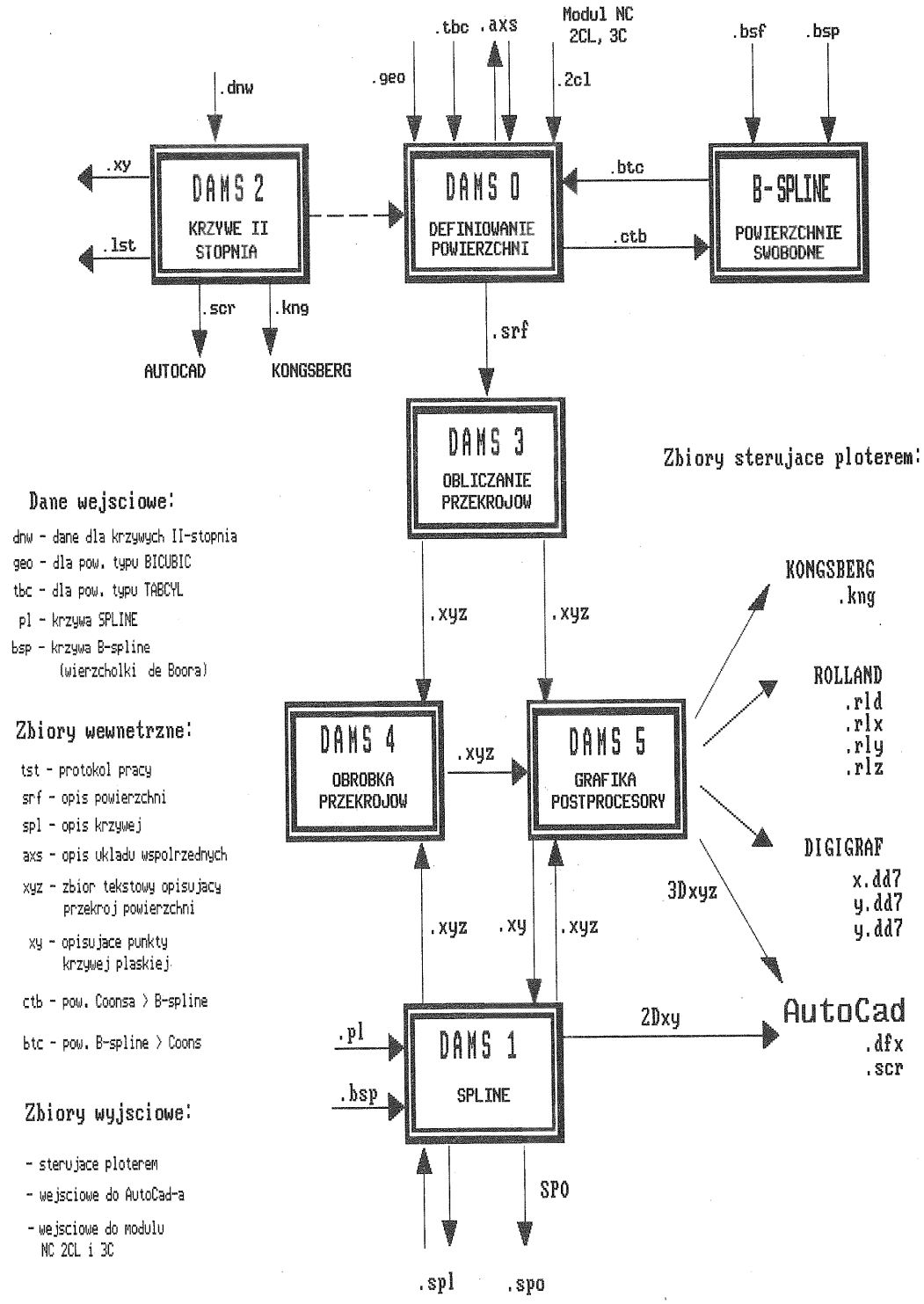
- DAMS1: moduł definiowania i obróbki krzywych. Moduł ten służy do definiowania krzywych gładkich zadawanych przy pomocy współrzędnych. Są dwa rodzaje krzywych typu "spline". Moduł ten przygotowuje dane do modułu **DAMS0**.

- DAMS2: moduł definiowania i obróbki krzywych II stopnia. Moduł przygotowuje dane wejściowe do **DAMS0**. Służy on do projektowania krzywych zadanych przy pomocy wyróżnika **f**.

- DAMS3: moduł obliczania przekrojów. Moduł ten służy do obliczania przekroju powierzchni dowolną płaszczyzną, w dowolnie założonej tolerancji lub do obliczania drogi narzędzia na obrabiarkach sterowanych numerycznie. W wyniku obliczeń przekroju otrzymujemy dyskowy zbiór punktów typu tekstowego co daje możliwość dalszej obróbki różnymi dowolnymi programami.



Rys.1 DAMS UNIWERSALNY, NUMERYCZNY SYSTEM MODELOWANIA ZEWNETRZNYCH KSZTALTOW SAMOLOTU, NADWOZIA SAMOCHODU, STATKU, SRUBY OKRETOWEJ.



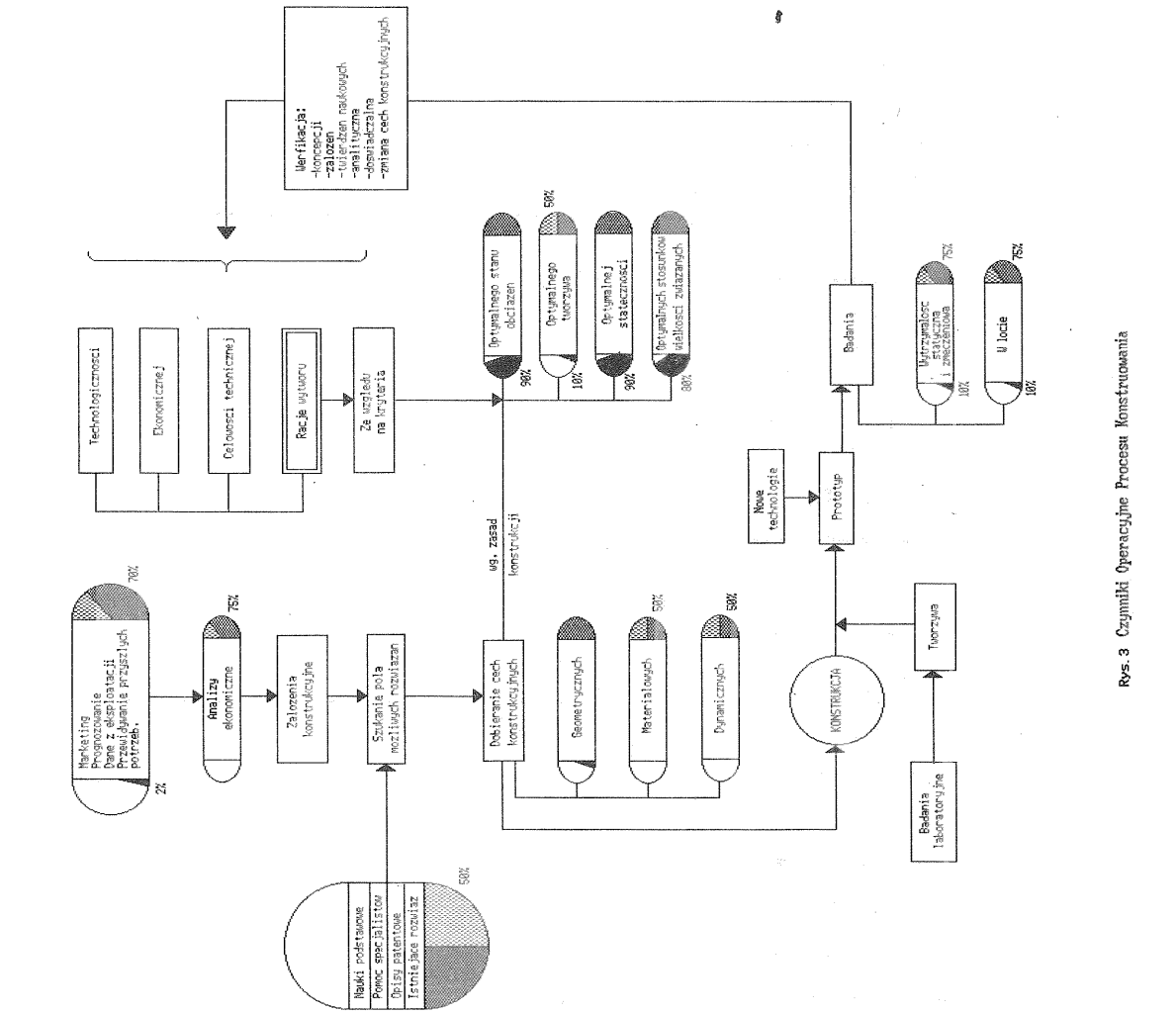
Rys 2 Oznaczenie zbiorow danych w systemie DAMS

- **DAMS4: moduł obróbki przekrojów.** Moduł służy do wykonywania różnych operacji na przekrojach jak skracanie, wybieranie fragmentów krzywej czy przeliczanie i wyliczanie innych wartości współrzędnych.
- **DAMS5: moduł graficzny.** Pozwala na graficzne przedstawienie powierzchni lub przekroju na monitorze lub ploterze (drukarce). Przy dużych ploterach jak **KONGSBERG** można kreślić przekroje w skali 1:1.

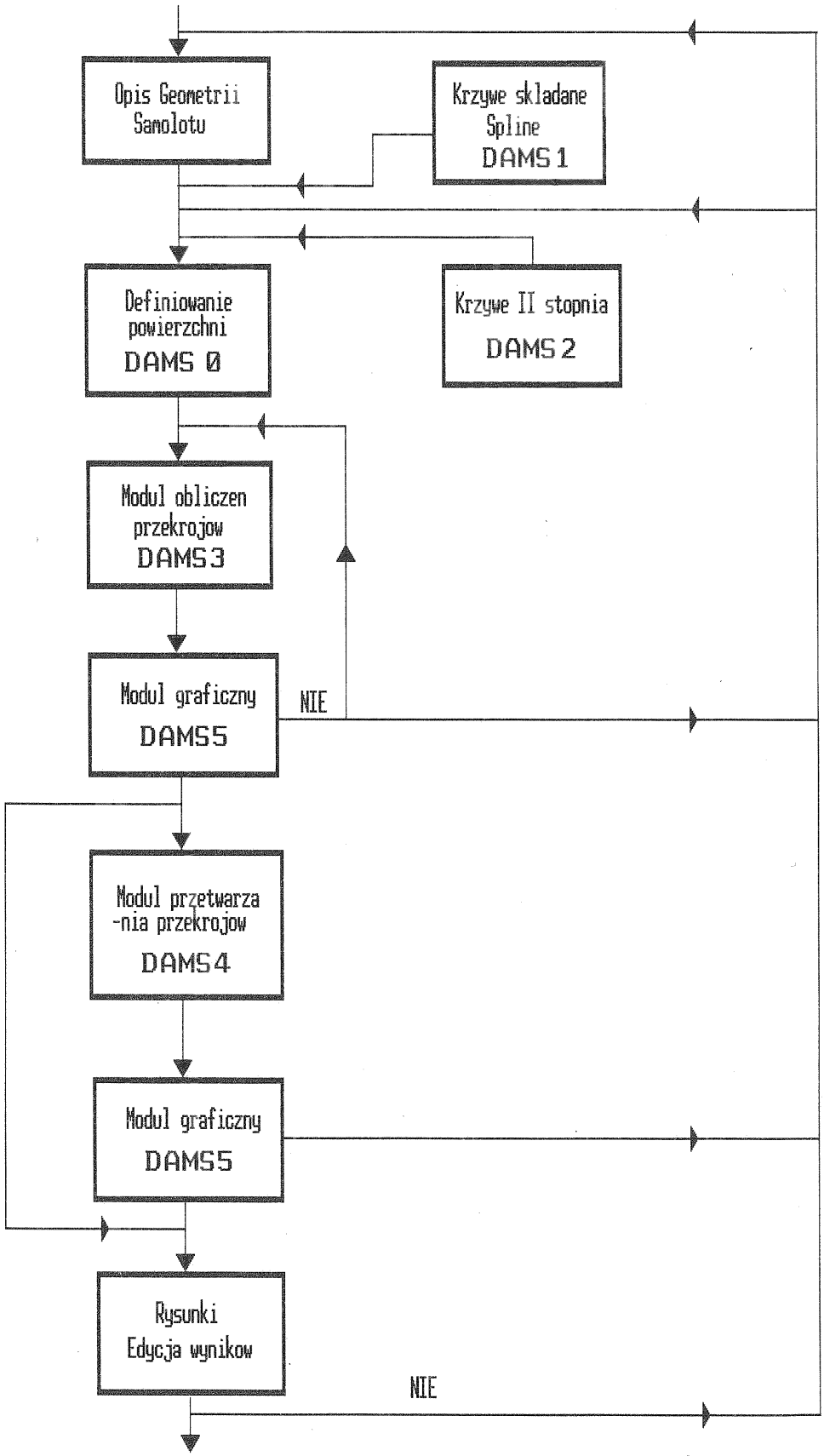
Proces konstruowania wymaga całościowego opisu, co jest równoznaczne z syntezą czynników operacyjnych. Na rys. 3 przedstawiono wyobrażenie całego procesu konstruowania z uwzględnieniem wykorzystania techniki komputerowej. Z lewej strony operatorów przedstawiono stan obecny a z prawej przewidywany stan wykorzystania komputerów przez autora. Wynika to przede wszystkim z bardzo szybkiego rozwoju sprzętu komputerowego i szerokiego wprowadzania sieci komputerowych w przedsiębiorstwach. Głównym czynnikiem tworzenia konstrukcji jest dobieranie cech konstrukcyjnych. Właściwą teorią tego procesu jest teoria konstrukcji, którą opieramy na zasadach konstrukcji i racjach istnienia wytworu. Racje te służą do określenia skutecznych kryteriów stanowiących podstawę oceny konstrukcji. Racjonalny proces konstruowania wymaga weryfikacji. Ze względów ekonomicznych należy położyć nacisk na weryfikację analityczną, bowiem jej narzędziami są jedynie zasady logiki i matematyki. Często weryfikacja prowadzi do zmiany cech konstrukcyjnych. Weryfikacja konstrukcji może stać się podstawą do weryfikacji założeń konstrukcyjnych oraz koncepcji a nawet twierdzeń nauk podstawowych.

Cały proces projektowo-wykonawczy wymaga następujących działań (rys. 4):
- sprawdzenie poprawności danych zawartych na rysunkach geometrii samolotu,
- obliczenie i wykreślenie kształtów opisanych krzywymi II stopnia, tzn. krzywe podane są za pomocą wyróżnika "f" **moduł krzywych II stopnia**,
- upłynnienie i sprawdzenie krzywych podanych w sposób dyskretny **moduł spline'a parametrycznego**,
- definiowanie powierzchni i ich kolejne modyfikacje **moduł zadawania powierzchni**,
- badanie i sprawdzenie powierzchni **moduł obliczania przekrojów**,
- obliczenie współrzędnych punktów, stycznych, normalnych do powierzchni itp., **moduł przetwarzania przekrojów**,
- wykreślenie rysunków powierzchni (w rzutach, w perspektywie) zadanych przekrojów, krzywych **moduł kreślący**,
- wydrukowanie wszystkich wyników obliczeń oraz protokołowanie pracy systemu **moduł drukujący**,
- sterowanie i zarządzanie biblioteką danych, archiwowanie **system operacyjny**,
- obliczanie drogi narzędzia na obrabiarkach sterowanych numerycznie dla płaskich i przestrzennych wzorników **moduł obrabiarkowy**.

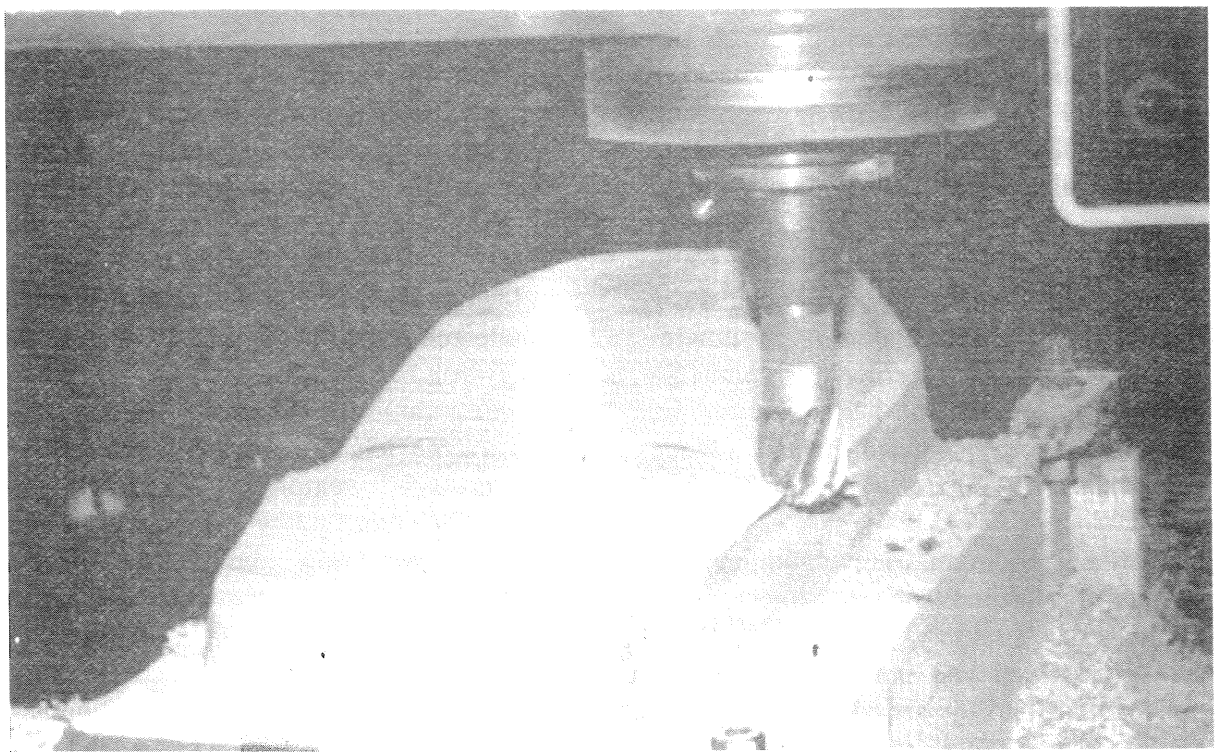
W systemie **DAMS** zaprojektowano kształty zewnętrzne, jak i wewnętrzne takich samolotów jak: **AN-28, I-22, M-26**, niektórych zespołów (usterzenie, klapy, wysięgniki) **IŁ-86, IŁ-96, M-18 Dromader, M-20 Seneca, ATR-42, ATR-72, BOEING-757, łopatki turbin, wózki golfowe, szybowce**. Opis numeryczny kształtu służy następnie do wykonania na komputerze **SM-4 Kongsberg** z ploterem o wym. 1.6m x 6m w skali 1:1 pełnego rozrysowania płazowo-konstrukcyjnego na specjalnej stabilnej folii. Z folii informacja przenoszona jest przez fotokopiarkę na blachę duralową o wym. 1m x 2m. Stanowi to tzw. "twardą kopię". Na podstawie tej kopii wykonuje się płaskie i przestrzenne wzorniki i inne pomoce warsztatowe. Oczywiście modele do dmuchań aerodynamicznych, wzorniki płaskie i przestrzenne robi się także bezpośrednio korzystając z **numerycznego zapisu kształtu** na **OSN**. Jest on podstawą do wykonania modelu (rys. 5), wzornika czy też samej części danego wyrobu (rys. 6).



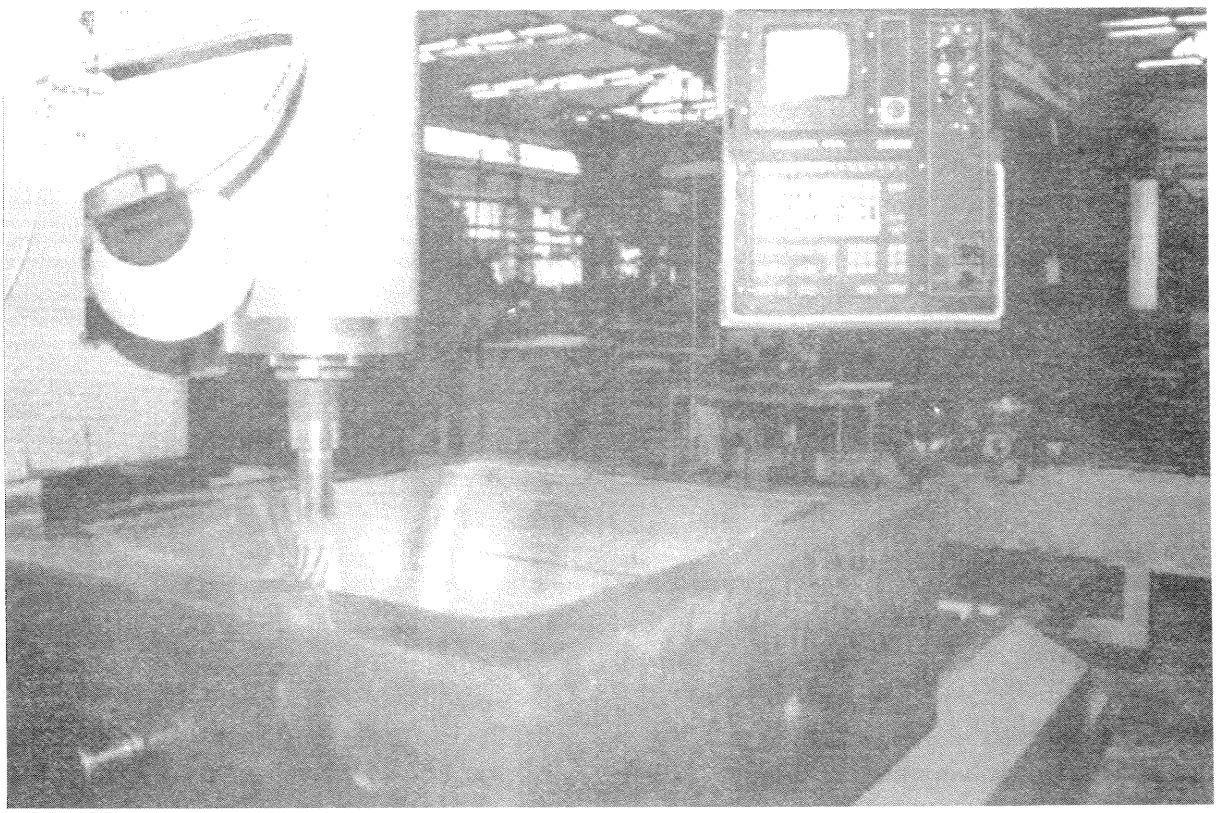
Rys. 3 Czynniki Operacyjne Procesu Konstruowania



Rys 4 Ogolny schemat uzywania systemu DAMS



Rys.5 Model samochodu podczas obróbki na OSN



Rys.6 Foremnik podczas obróbki na OSN

Najbardziej istotnym elementem procesu wytwarzania przedmiotów opisanych powierzchniami o podwójnej krzywiźnie, jest oczywiście frezowanie tych powierzchni. Do tego celu służą obrabiarki sterowane numerycznie **OSN**. Należy tutaj zwrócić uwagę na to, aby powierzchnie o podwójnej krzywiźnie były technologiczne i aby obróbka ich trwała jak najkrócej. Przy frezowaniu wykańczającym tych powierzchni na frezarkach sterowanych numerycznie, czas obróbki ma istotne znaczenie. Optymalny rodzaj obróbki, zależy przede wszystkim od typu frezarki **2CL**, **3C**, **5C**.

Najbardziej ogólnym i najwygodniejszym opisem do zdefiniowania powierzchni jak i obliczania charakterystycznych własności geometrycznych jest parametryczny opis powierzchni. Stosuje się go także w wielopłatkowych powierzchniach bikubicznych. Wektor elementarnego płatka powierzchni jest funkcją dwóch skalarnych zmiennych **u** i **v**, którymi są krzywoliniowe Gaussowskie parametry powierzchni.

Równanie ma postać:

$$P = P(u,v) = x(u,v)i + y(u,v)j + z(u,v)k$$

$$P_u = \frac{\partial P}{\partial u} = \left(\frac{\partial x}{\partial u}, \frac{\partial y}{\partial u}, \frac{\partial z}{\partial u} \right)$$

$$P_v = \frac{\partial P}{\partial v} = \left(\frac{\partial x}{\partial v}, \frac{\partial y}{\partial v}, \frac{\partial z}{\partial v} \right) \qquad (1)$$

$$P_{uv} = \frac{\partial^2 P}{\partial u\, \partial v} = \begin{vmatrix} \frac{\partial y}{\partial u} & \frac{\partial z}{\partial u} \\ \frac{\partial y}{\partial v} & \frac{\partial z}{\partial v} \end{vmatrix}, \begin{vmatrix} \frac{\partial z}{\partial u} & \frac{\partial x}{\partial u} \\ \frac{\partial z}{\partial v} & \frac{\partial x}{\partial v} \end{vmatrix}, \begin{vmatrix} \frac{\partial x}{\partial u} & \frac{\partial y}{\partial u} \\ \frac{\partial x}{\partial v} & \frac{\partial y}{\partial v} \end{vmatrix}$$

Linie o stałych parametrach **u** i **v** są krzywymi składanymi (spline). Dla różnych wartości parametru **v** odwzorowanie opisuje różne krzywe, przy czym zbiór tych krzywych tworzy powierzchnię. Kształt powierzchni określany jest przez definiowanie kształtu krzywych leżących na powierzchni, które tworzą zwykle na powierzchni siatkę (model drutowy). W większości praktycznych zastosowań siatka ta dzieli powierzchnię na czworokątne płatki powierzchni a więc zadanie polega na określeniu czworokątnego płatka powierzchni rozpiętego na danych czterech krzywych brzegowych.
Projektowanie powierzchni polega na utworzeniu ich numerycznej reprezentacji w postaci wektorów współrzędnych, wektorów stycznych w kierunku **u** i **v**, **Pu**, **Pv** oraz wektorów skrętu. Do opisania powierzchni wykorzystano metodę czworokątnych płatków profesora S. A. Coonsa.

Pojęcie geometrycznego modelowania pojawiło się w użyciu we wczesnych latach siedemdziesiątych wraz z rozwojem grafiki komputerowej jak i technologii **CAD/CAM**. W połowie lat sześćdziesiątych D. T. Ross (1967-MIT) rozwinął i unowocześnił komputer dla graficznego programowania. S. A. Coons (1963, 1965-MIT) i J. C. Ferguson (1964-Boeing) rozpoczęli w tym czasie prace nad powierzchnią o podwójnej krzywiźnie. Powierzchnie te zostały rozwinięte aby

zastąpić klasyczną technikę trasowania w skali 1:1 stosowaną w przemyśle samochodowym, okrętowym i lotniczym.

Do rozwoju analitycznych metod odwzorowania geometrii samolotu przyczyniły się nowe techniki wytwarzania, szczególnie zaś obrabiarki sterowane numerycznie **OSN**, które wymagają aby kształt obrabianych części był podany w sposób numeryczny. Nie jest to możliwe przy stosowaniu giętki traserskiej ze względu na małą dokładność odczytywanych współrzędnych punktów krzywej wykreślonej z planu warstwicowego.
Tak więc stosowanie **OSN** wymusiło tworzenie numerycznego opisu kształtu również dla obiektów, które konstruowano metodami tradycyjnymi.

System **DAMS** dostarcza wszystkich niezbędnych danych do wykonania:
- obliczeń technicznych;
- analizy konstrukcyjnej;
- wzorników płaskich (szablony) na OSN;
- wzorników przestrzennych (bałwanki) na OSN;
- elementów przyrządu montażowego na OSN;
- detali integralnych samolotu na OSN;
- modeli do dmuchań aerodynamicznych;
- narzędzi specjalnych.

Obecnie jednym z najważniejszych problemów, z jakim spotykają się konstruktorzy na całym świecie przy elektronicznym zapisie konstrukcji, jest graficzna wymiana informacji między kooperującymi zakładami, jak i też integracja między różnymi systemami komputerowymi.

Zastosowanie sieci komputerowej w pracy konstruktorów i technologów wraz z odpowiednim oprogramowaniem zmieniło jakość i metody ich pracy. Zamiast "papierowej" formy dokumentacji konstrukcyjnej coraz częściej pojawia się "elektroniczna" postać dokumentacji konstrukcyjnej i technologicznej. Coraz ważniejszą sprawą staje się problem standaryzacji danych przy wymianie dokumentacji konstrukcyjnej między partnerami stosującymi różne systemy **CAD/CAM**. Żadne przedsiębiorstwo nie utrzyma się dzisiaj na rynku bez stosowania odpowiedniej klasy systemów **CAD/CAM**.

Automatyzacja prac konstrukcyjnych i procesu wytwarzania samolotu, zmierza głównie do znacznego skrócenia cyklu produkcyjnego, obniżenia pracochłonności opracowania konstrukcyjnego i technologicznego oraz zapewnienia najwyższej jakości wykonania. Realizacja tych celów bezwzględnie wymaga zastosowania techniki komputerowej, opartej na rozwiązaniach sieciowych i obrabiarek sterowanych numerycznie.

Wykaz literatury

[1] Harder R. L. and Desmarais R. N.: *Interpolation Using Surface Splines*, Journal of Aircraft, Vol.9, No 2, Feb. 1972.
[2] *Tooling Built with computer graphics*, American Machinist, November 1972.
[3] Schaffer G.: *NC router and CAM cut cost*, American Machinist, December 1979.
[4] *How Aerospace Marries CAM for Drilling, Routing*, Production, May 1982.
[5] *Personal computers expand the CAM element of CAD/CAM*. American Machinist, no. VI/1987.
[6] Mayer R. J.: *One answer to the problems of CAD database exchange*, Byte, no. VI/1987.
[7] Adamski W., Rybak Cz., Gancarz T.: *"PZL-Mielec" Vallalatnal folyo szerkesztesi es technologiai munkak automati zalasa*, Konferencja Naukowa Mechatroninfo, Budapeszt, listopad 1988.
[8] Adamski W.: *Numeryczne odwzorowanie geometrii samolotu*, Konferencja Naukowa, Mielec 1989.
[9] Adamski W.: *Automatyzacja procesów projektowania CAD/CAM*, VI Międzynarodowe Sympozjum, Krasiczyn, październik 1989.

[10] Adamski W.: *Projektowanie i wytwarzanie samolotów wspomagane komputerem*, Aerotechnika Lotnicza, nr 11/91.

[11] Adamski W.: *Zasady numerycznego modelowania zewnętrznych kształtów obiektów*, Przegląd Mechaniczny, nr 3/92.

Mariusz Klapper
Zakład Systemów Komputerowych i Oprogramowania "K&MSOFT", Kraków

Praktyczne doświadczenia opracowywania i wdrażania w przedsiębiorstwach kompleksowych systemów informatycznych

W referacie zostały omówione praktyczne doświadczenia zebrane podczas kilkunastoletniej pracy związanej z projektowaniem, opracowywaniem i wdrażaniem użytkowych systemów przetwarzania danych dla przedsiębiorstw i organizacji gospodarczych. Ze względu na tematykę Szkoły główna uwaga zostanie poświęcona aspektom projektowania i programowania systemów związanym z obiektowym podejściem do struktur danych i procedur. Omawiane w referacie problemy dotyczą typowych sytuacji, z jakimi ma do czynienia przy opracowywaniu i uruchamianiu systemów komputerowego przetwarzania danych niewielka firma usługowa pracująca dla kilku, lub kilkunastu klientów. Omawiane rozwiązania wykorzystują możliwości narzędziowe istniejące w systemie programowania TURBO-PASCAL, oraz możliwości wynikające z hierarchicznej organizacji katalogu plików. Nie są to rozwiązania obiektowe w sensie dosłownym, gdyż nie są sformalizowane zgodnie z zasadami programowania obiektowego, ani nie mają wszystkich formalnych cech obiektów. Z merytorycznego punktu widzenia są to jednak techniki obiektowe, gdyż wywodzą się z obiektowego potraktowania struktur danych i procedur. Prezentowane koncepcje zostały wypracowane niezależnie od teoretycznych zasad programowania obiektowego, jako rezultat rozwiązywania napotykanych w praktyce problemów i uwarunkowań. Ich podobieństwo merytoryczne do zasad programowania obiektowego dowodzi, że uogólnienia formalne i nowe techniki programowania wynikają z rozwiązywania podobnych problemów praktycznych spotykanych w pracach informatycznych niezależnie od miejsca i tematyki tych prac. Osobiście nie sądzę, aby omawiane techniki były dla Czytelnika szczególną rewelacją; podobne rozwiązania są wykorzystywane w wielu zastosowaniach. Jest to jednak dobry przykład na to, że używanie takich technik do budowania kompleksowych systemów informatycznych jest możliwe i pożyteczne. Omawiane techniki i technologie są od kilku lat stosowane i wielokrotnie sprawdzone w praktyce, z bardzo pozytywnymi rezultatami.

Najważniejsze uwarunkowania prezentowanych w referacie koncepcji wynikają z następujących założeń i wymagań spotykanych w praktyce:

1. Użytkownik chce dysponować systemami kompleksowymi, obejmującymi wszystkie ważne dla niego obszary działalności. Jednak jego środki oraz możliwości finansowe i organizacyjne są z reguły ograniczone. Dlatego budowę kompleksowych systemów przetwarzania danych trzeba rozkładać na kolejne etapy, przy jednoczesnym zachowaniu wymogów

ich wzajemnej integralności. Dodatkowym uzasadnieniem dla rozkładania w czasie kolejnych wdrożeń jest bariera percepcyjna użytkowników, tzn. ich trudności z szybkim opanowaniem umiejętności obsługi i efektywnego wykorzystywania wielu różnych systemów informatycznych w krótkim okresie czasu.

2. Systemy komputerowego przetwarzania danych muszą mieć możliwości łatwego i niezawodnego dostosowywania do zmiennych warunków funkcjonowania. Często użytkownik nie potrafi precyzyjnie określić "z góry" swoich wymagań i oczekiwań związanych z przygotowywanymi systemami informatycznymi. Zazwyczaj rozwija on i uściśla swoje potrzeby w miarę wdrażania systemów, po stopniowym poznawaniu ich możliwości. Często wymagania stawiane systemom ulegają zmianom wynikającym ze zmienności warunków działania użytkownika (zmiany przepisów prawnych, zmiany koniunktury i wynikające z nich potrzeby informacyjne itp.). Na ogół wymaga to wprowadzania zmian i modyfikacji do systemów w trakcie ich wdrażania, lub eksploatacji. Uwarunkowania te są niezależne od starannego i kompetentnego przygotowania wdrożenia systemu: analizy funkcjonalnej, rzetelnego projektu systemu, wykorzystania innych doświadczeń itp. Opracowywane systemy muszą zatem mieć zmienność "wbudowaną" w ich strukturę.

3. Każdy użytkownik ma swoją specyfikę i indywidualne potrzeby. Dlatego nie można zakładać pełnej unifikacji systemów dla różnych użytkowników nawet wtedy, kiedy te systemy dotyczą identycznych problemów i obszarów zastosowań. Inaczej mówiąc: jeśli chcemy respektować faktyczne potrzeby użytkownika, to nie istnieją w pełni powielarne systemy przetwarzania danych. Stosowane rozwiązania technologiczne i metodyka programowania systemów muszą zatem stwarzać możliwości łatwego dopasowywania systemu do indywidualnych potrzeb użytkownika, przy jednoczesnej, możliwie daleko posuniętej unifikacji użytych narzędzi i technologii opracowywania systemu.

4. Konieczność równoczesnego opracowywania i serwisowania wdrożeń różnych systemów u różnych użytkowników stwarza potrzebę ujednolicenia technologii opracowywania systemów i technik programowania. Nawet staranne dokumentowanie prac projektowych i programowych (niestety dość rzadko spotykane w praktyce) nie gwarantuje bowiem sprawnego serwisowania systemów funkcjonujących u wielu użytkowników. Użytkownik z reguły woli przy rozwiązywaniu swoich problemów współpracować bezpośrednio z autorem systemu, niż studiować dokumentację systemu i uczyć się używania go we własnym zakresie. Autor systemu musi zatem umieć szybko i sprawnie konsultować problemy użytkownika, oraz lokalizować i modyfikować procedury lub uwarunkowania zastosowane w systemie.

5. Wdrażane systemy dziedzinowe muszą być przystosowane do łączenia ich w system kompleksowy. Niezbędnym tego warunkiem jest ujednolicenie stosowanych zasad kodowania danych, unifikacja metod i technik programowania, oraz stosowanie we wszystkich systemach wzajemnie dopasowanych technologii i narzędzi przetwarzania danych. Bardzo ważną sprawą jest również ujednolicenie (i związana z nim wymienność) algorytmów typowych procedur stosowanych w różnych systemach, pomimo ich zróżnicowania wynikającego z różnych potrzeb konkretnych użytkowników. Konieczne jest też opracowywanie systemów łatwo przystosowywalnych do zmiany techno-

logii i warunków ich eksploatacji (np. zmiana aktualnego użytkownika systemu i danych, przeniesienie do sieci systemów pracujących lokalnie itp.).

Dla zrealizowania powyższych wymagań podczas projektowania i uruchamiania systemów komputerowych zostały przyjęte następujące założenia dotyczące struktury systemu i organizacji danych:

1. Kompleksowy system informatyczny użytkownika jest podzielony na systemy dziedzinowe, obsługujące wydzielone i w miarę jednorodne obszary zastosowań. Każdy z wydzielonych systemów dziedzinowych jest identyfikowany unikalnym identyfikatorem (nazywanym przez nas umownie identyfikatorem MEMO), zapewniającym unikalność nazw i identyfikatorów używanych przez system. Dzięki stosowaniu identyfikatorów MEMO jest możliwe zapewnienie jednoznacznej identyfikacji zasobów związanych z używającym go systemem dziedzinowym, oraz ujednolicenie struktury identyfikatorów stosowanych w różnych systemach dziedzinowych, np. nazw procedur stosowanych w modułach procedur dostępu (nazwy te są budowane w identyczny sposób, a różnią się jedynie zawartym w nich identyfikatorem MEMO).
   Podstawowymi kryteriami określającymi zakres zasobów włączanych do wydzielonego systemu dziedzinowego są funkcje tego systemu, oraz struktury i rola danych dziedzinowych w kompleksowym systemie informatycznym użytkownika.

   Systemy dziedzinowe obejmują zazwyczaj dwie podstawowe grupy zastosowań: systemy obsługujące danę o charakterze indeksowym, (np. indeks kontrahentów, cennik wyrobów, indeks materiałowy, kartoteka osobowa, plan kont księgowych itp.), oraz systemy obsługujące ewidencję i przetwarzanie danych ewidencyjnych (np. sprzedaż wyrobów, ewidencja magazynowa, rozliczenia płac, system finansowo-księgowy itp.). Warto zwrócić uwagę, że opisane powyżej podejście do wyodrębniania obszarów zastosowań systemów dziedzinowych jest pierwszym, niejako "naturalnym" krokiem w kierunku obiektowego podejścia do organizacji systemu: cechami decydującymi o wyodrębnieniu obiektu są bowiem dane, oraz sposób ich używania, czyli związane z tymi danymi procedury i algorytmy.

2. Oprogramowanie kompleksowego systemu informatycznego jest organizowane na trzech poziomach. Na najwyższym poziomie znajduje się biblioteka procedur ogólnego użycia. Dotyczy ona wszystkich systemów i wszystkich użytkowników. Na poziomie wybranego użytkownika jest organizowana biblioteka procedur dedykowanych, która dotyczy wszystkich systemów tego użytkownika i biblioteki modułów procedur dostępu, obsługujących komunikację systemów z danymi użytkownika i zasobami pomocniczymi. Każdy użytkownik ma co najmniej jedną bibliotekę procedur dedykowanych i kilka bibliotek modułów dostępu. Na poziomie systemów dziedzinowych wybranego użytkownika są organizowane biblioteki oprogramowania użytkowego. Podstawowe zasady organizowania tych bibliotek są omówione poniżej.

Wszystkie definicje i procedury związane z obsługą zasobów wydzielonych dla systemu dziedzinowego są umieszczane w jednym pakiecie oprogramowania, nazywanym przez nas modułem procedur dostępu. Schemat struktury i zawartości typowego modułu procedur dostępu przedstawia rysunek 1.

Pakiet procedur dostępu jest organizowany w format wydzielonej dla używanego języka programowania jednostki kompilacyjnej (np. UNIT-u w języku

Definicje sprzęgów z bibliotekami procedur ogólnego użycia, bibliotekami procedur dedykowanych i innymi modułami dostępu
Definicje typów i struktur danych obsługiwanych przez moduł dostępu
Definicje głównych i pomocniczych plików danych modułu
Definicje procedur modułu dla bibliotek i oprogramowania użytkowego
Procedury podstawowe modułu dostępu
Procedury specjalizowane modułu dostępu

RYS. 1. Struktura typowego modułu procedur dostępu

TURBO-PASCAL). Praktycznie stanowi on zatem niesformalizowaną definicję obiektu dla kompleksowego systemu przetwarzania danych. Obowiązuje generalna zasada, że całe oprogramowanie w ramach systemu kompleksowego (a więc wszystkie programy stosowane przez użytkownika) musi korzystać z dostępu do danych przydzielonych dla modułu dostępu wyłącznie za pośrednictwem definicji i procedur zawartych w tym module. Struktura i zawartość wszystkich modułów dostępu w systemie kompleksowym jest identyczna, lub bardzo podobna. Całkowita unifikacja struktury modułu dostępu nie jest jednak sensowna, ani możliwa ze względu na zróżnicowanie struktur, funkcji i sposobów przetwarzania danych dziedzinowych.

Każdy moduł procedur dostępu zawiera następujące elementy składowe:

1. Definicje sprzęgów z bibliotekami procedur ogólnego użycia, bibliotekami procedur dedykowanych i innymi modułami procedur dostępu. Jeśli moduł procedur dostępu korzysta z zasobów, struktur lub procedur zdefiniowanych w innych bibliotekach i modułach procedur dostępu (np. system ewidencji sprzedaży korzysta z danych, struktur i procedur zdefiniowanych dla indeksu kontrahentów i dla cennika wyrobów), to dostęp do tych zasobów i struktur uzyskuje poprzez zdefiniowanie sprzęgu (odwołania) do odpowiedniej biblioteki, lub odpowiedniego modułu procedur dostępu. Zasada ta oznacza zatem faktyczne dziedziczenie przez moduł dostępu cech zasobów zdefiniowanych w innych bibliotekach, lub w innym module dostępu. Narzędziem dla realizacji sprzęgu jest zazwyczaj odpowiedni łącznik konsolidacyjny ("linkujący" wg określenia żargonowego) umieszczany w sprzęganym module i udostępniający mu globalne definicje wywołanego modułu.

2. Definicje typów i struktur danych, oraz głównych i pomocniczych plików danych obsługiwanych przez moduł dostępu (identyfikatorów, symboli, rekordów, kluczy, plików itp). Obowiązuje przy tym zasada unikalności takich samych definicji w ramach całej biblioteki, tzn. dziedziczenia dla identycznych definicji cech zdefiniowanych w innych modułach dostępu poprzez zdefiniowanie sprzęgów z tymi modułami (np. definicja struktury symbolu konta księgowego dla pliku ewidencyjnych dokumentów księgowych w module dostępu do tego pliku musi dziedziczyć cechy definicji

logii i warunków ich eksploatacji (np. zmiana aktualnego użytkownika systemu i danych, przeniesienie do sieci systemów pracujących lokalnie itp.).

Dla zrealizowania powyższych wymagań podczas projektowania i uruchamiania systemów komputerowych zostały przyjęte następujące założenia dotyczące struktury systemu i organizacji danych:

1. Kompleksowy system informatyczny użytkownika jest podzielony na systemy dziedzinowe, obsługujące wydzielone i w miarę jednorodne obszary zastosowań. Każdy z wydzielonych systemów dziedzinowych jest identyfikowany unikalnym identyfikatorem (nazywanym przez nas umownie identyfikatorem MEMO), zapewniającym unikalność nazw i identyfikatorów używanych przez system. Dzięki stosowaniu identyfikatorów MEMO jest możliwe zapewnienie jednoznacznej identyfikacji zasobów związanych z używającym go systemem dziedzinowym, oraz ujednolicenie struktury identyfikatorów stosowanych w różnych systemach dziedzinowych, np. nazw procedur stosowanych w modułach procedur dostępu (nazwy te są budowane w identyczny sposób, a różnią się jedynie zawartym w nich identyfikatorem MEMO).
   Podstawowymi kryteriami określającymi zakres zasobów włączanych do wydzielonego systemu dziedzinowego są funkcje tego systemu, oraz struktury i rola danych dziedzinowych w kompleksowym systemie informatycznym użytkownika.

   Systemy dziedzinowe obejmują zazwyczaj dwie podstawowe grupy zastosowań: systemy obsługujące danę o charakterze indeksowym, (np. indeks kontrahentów, cennik wyrobów, indeks materiałowy, kartoteka osobowa, plan kont księgowych itp.), oraz systemy obsługujące ewidencję i przetwarzanie danych ewidencyjnych (np. sprzedaż wyrobów, ewidencja magazynowa, rozliczenia płac, system finansowo-księgowy itp.). Warto zwrócić uwagę, że opisane powyżej podejście do wyodrębniania obszarów zastosowań systemów dziedzinowych jest pierwszym, niejako "naturalnym" krokiem w kierunku obiektowego podejścia do organizacji systemu: cechami decydującymi o wyodrębnieniu obiektu są bowiem dane, oraz sposób ich używania, czyli związane z tymi danymi procedury i algorytmy.

2. Oprogramowanie kompleksowego systemu informatycznego jest organizowane na trzech poziomach. Na najwyższym poziomie znajduje się biblioteka procedur ogólnego użycia. Dotyczy ona wszystkich systemów i wszystkich użytkowników. Na poziomie wybranego użytkownika jest organizowana biblioteka procedur dedykowanych, która dotyczy wszystkich systemów tego użytkownika i biblioteki modułów procedur dostępu, obsługujących komunikację systemów z danymi użytkownika i zasobami pomocniczymi. Każdy użytkownik ma co najmniej jedną bibliotekę procedur dedykowanych i kilka bibliotek modułów dostępu. Na poziomie systemów dziedzinowych wybranego użytkownika są organizowane biblioteki oprogramowania użytkowego. Podstawowe zasady organizowania tych bibliotek są omówione poniżej.

Wszystkie definicje i procedury związane z obsługą zasobów wydzielonych dla systemu dziedzinowego są umieszczane w jednym pakiecie oprogramowania, nazywanym przez nas modułem procedur dostępu. Schemat struktury i zawartości typowego modułu procedur dostępu przedstawia rysunek 1.

Pakiet procedur dostępu jest organizowany w format wydzielonej dla używanego języka programowania jednostki kompilacyjnej (np. UNIT-u w języku

Definicje sprzęgów z bibliotekami procedur ogólnego użycia, bibliotekami procedur dedykowanych i innymi modułami dostępu
Definicje typów i struktur danych obsługiwanych przez moduł dostępu
Definicje głównych i pomocniczych plików danych modułu
Definicje procedur modułu dla bibliotek i oprogramowania użytkowego
Procedury podstawowe modułu dostępu
Procedury specjalizowane modułu dostępu

RYS. 1. Struktura typowego modułu procedur dostępu

TURBO-PASCAL). Praktycznie stanowi on zatem niesformalizowaną definicję obiektu dla kompleksowego systemu przetwarzania danych. Obowiązuje generalna zasada, że całe oprogramowanie w ramach systemu kompleksowego (a więc wszystkie programy stosowane przez użytkownika) musi korzystać z dostępu do danych przydzielonych dla modułu dostępu wyłącznie za pośrednictwem definicji i procedur zawartych w tym module. Struktura i zawartość wszystkich modułów dostępu w systemie kompleksowym jest identyczna, lub bardzo podobna. Całkowita unifikacja struktury modułu dostępu nie jest jednak sensowna, ani możliwa ze względu na zróżnicowanie struktur, funkcji i sposobów przetwarzania danych dziedzinowych.

Każdy moduł procedur dostępu zawiera następujące elementy składowe:

1. Definicje sprzęgów z bibliotekami procedur ogólnego użycia, bibliotekami procedur dedykowanych i innymi modułami procedur dostępu. Jeśli moduł procedur dostępu korzysta z zasobów, struktur lub procedur zdefiniowanych w innych bibliotekach i modułach procedur dostępu (np. system ewidencji sprzedaży korzysta z danych, struktur i procedur zdefiniowanych dla indeksu kontrahentów i dla cennika wyrobów), to dostęp do tych zasobów i struktur uzyskuje poprzez zdefiniowanie sprzęgu (odwołania) do odpowiedniej biblioteki, lub odpowiedniego modułu procedur dostępu. Zasada ta oznacza zatem faktyczne dziedziczenie przez moduł dostępu cech zasobów zdefiniowanych w innych bibliotekach, lub w innym module dostępu. Narzędziem dla realizacji sprzęgu jest zazwyczaj odpowiedni łącznik konsolidacyjny ("linkujący" wg określenia żargonowego) umieszczany w sprzęganym module i udostępniający mu globalne definicje wywołanego modułu.

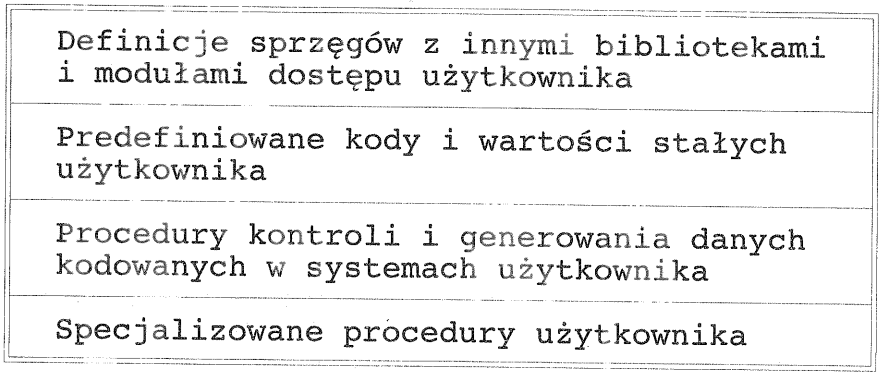
2. Definicje typów i struktur danych, oraz głównych i pomocniczych plików danych obsługiwanych przez moduł dostępu (identyfikatorów, symboli, rekordów, kluczy, plików itp). Obowiązuje przy tym zasada unikalności takich samych definicji w ramach całej biblioteki, tzn. dziedziczenia dla identycznych definicji cech zdefiniowanych w innych modułach dostępu poprzez zdefiniowanie sprzęgów z tymi modułami (np. definicja struktury symbolu konta księgowego dla pliku ewidencyjnych dokumentów księgowych w module dostępu do tego pliku musi dziedziczyć cechy definicji

struktury symbolu konta księgowego ustalonej w module dostępu do pliku planu kont księgowych). Identycznych typów i struktur dotyczących tych samych zasobów nie wolno definiować niezależnie w różnych modułach dostępu. Dzięki tej zasadzie zmiana definicji cech lub struktur w jednym module dostępu może być automatycznie przenoszona do wszystkich innych sprzęgniętych z nim modułów dostępu (wymaga to zazwyczaj jedynie ponownej kompilacji modułów i programów).

3. Definicje procedur modułu dla bibliotek i oprogramowania użytkowego.
Zawierają one zestaw typów, struktur i procedur udostępnianych przez moduł dedykowany dla innych procedur i modułów bibliotecznych, oraz dla oprogramowania użytkowego. Obowiązuje zasada, że program użytkowy musi wykorzystywać definicje określone przez moduł procedur dostępu i nie może ich dublować (np. wszystkie programy wykorzystujące dane z indeksu kontrahentów muszą wykorzystywać definicje typów i struktur danych indeksu kontrahentów ustalone w module dostępu do nich). Dla uzyskania dostępu do danych obsługiwanych przez moduł dostępu program użytkowy musi korzystać z pośrednictwa procedur zdefiniowanych przez ten moduł. Nie wolno tworzyć żadnych lokalnych procedur dostępu do tych danych bezpośrednio w programie użytkowym. Jeśli program użytkowy potrzebuje do obsługi danych procedury innej, niż dotychczas zdefiniowane w module dostępu, to potrzebną, nową procedurę wstawia się do modułu dostępu, a nie do programu (oczywiście obowiązuje tutaj nieoceniona zasada "brzytwy Ockhama").

4. Procedury podstawowe i specjalizowane modułu dostępu. Jest to pakiet procedur realizujących definicje opisane w punkcie 3, umożliwiających programom użytkowym dostęp do zasobów obsługiwanych przez moduł dostępu. Procedury podstawowe służą zazwyczaj do ustalania identyfikatorów danych, inicjacji i deinicjacji dostępu do zasobów, wyszukiwania informacji w plikach obsługiwanych przez moduł dostępu według wskazanych kluczy, przełączania wątka systemowego na podstawowe oprogramowanie obsługujące zasoby modułu dostępu itp. Procedury specjalizowane obejmują specyficzne funkcje i procedury związane z konkretnym modułem dostępu np. identyfikację cech lub zawartości wyróżnionych pól danych, systemy kodowania danych specyficzne dla zasobów modułu dostępu, standardową obróbkę danych obsługiwanych przez moduł dostępu itp. Procedury podstawowe i specjalizowane mogą być włączane do definicji opisanych w pkt. 3, lub mogą być lokalnymi procedurami modułu dostępu.

W kompleksowym systemie informatycznym oprócz modułów dostępu jest również organizowany oddzielny pakiet procedur, wyspecjalizowanych dla obsługi struktur danych i systemów kodowania danych użytkownika. Pakiet ten został przez nas nazwany biblioteką procedur dedykowanych użytkownikowi. Pakiet ten może być umieszczony w jednej, lub kilku wydzielonych bibliotekach procedur. Strukturę i zawartość biblioteki procedur dedykowanych przedstawia rysunek 2.
Biblioteka procedur dedykowanych zawiera wszystkie specyficzne dla użytkownika definicje typów, struktur, kodów, wartości i relacji danych. Może dotyczyć jednego, lub kilku systemów dziedzinowych. Jest ona organizowana w format jednostki kompilacyjnej wydzielonej dla używanego języka programowania (np. UNIT-u w języku TURBO-PASCAL). Stanowi więc również niesformalizowaną definicję cech obiektów (danych) dla kompleksowego systemu przetwarzania danych. W odniesieniu do biblioteki procedur



RYS. 2. Struktura biblioteki procedur dedykowanych użytkownikowi

dedykowanych obowiązuje również generalna zasada, że całe oprogramowanie użytkowe musi korzystać z biblioteki procedur dedykowanych dla używanych lokalnie typów, struktur i kodów danych, oraz dla realizacji wyspecjalizowanych algorytmów zdefiniowanych w tej bibliotece (zasada ta dotyczy również modułów dostępu !). W bibliotece procedur dedykowanych są umieszczane funkcje i procedury specyficzne dla konkretnego użytkownika. Jest ona zatem podstawowym narzędziem dopasowującym system do lokalnych potrzeb użytkownika. Biblioteka ta nie może zawierać procedur ogólnego użycia (stosowanych dla wielu użytkowników), ani ich substytutów.

W bibliotece procedur dedykowanych znajdują się następujące elementy:

1. Definicje sprzęgów z innymi bibliotekami i modułami dostępu użytkownika. Jeśli procedury dedykowane wykorzystują definicje związane z zasobami obsługiwanymi przez inne biblioteki dedykowane lub moduły dostępu, to dostęp do tych zasobów uzyskuje poprzez zdefiniowanie sprzęgu (odwołania) do odpowiedniej biblioteki dedykowanej lub modułu procedur dostępu.

2. Predefiniowane typy, kody i wartości stałych użytkownika. Są to typy, kody i wartości używane w kompleksowym systemie informatycznym, a nie ewidencjonowane w inny sposób (np. stosowane przez użytkownika kody dokumentów źródłowych). Predefiniowane kody i wartości stałych użytkownika są określane w taki sposób, aby były dostępne dla wszystkich programów użytkowych i modułów dostępu. Przyjęto zasadę, że kody i wartości predefiniowane w bibliotece procedur dedykowanych nie mogą być ponownie definiowane nigdzie indziej.

3. Procedury kontroli i generowania danych kodowanych w systemach użytkownika. Są to procedury i funkcje kontrolujące poprawność danych w systemach użytkownika (np. poprawność cyfr kontrolnych symboli), lub generujące opisy kodów stosowanych w systemach użytkownika (np. generujące nazwę magazynu dla jego kodu). Procedury te są bardzo użyteczne dla kontroli i ułatwienia wprowadzania przez użytkownika danych do systemów, zwłaszcza w połączeniu ze stosowaniem w programach użytkowych tzw. "menu generowanych", które przy pomocy bibliotecznej funkcji generującej wyświetlają na ekran menu zawierające stosowane kody i ich opisy.

4. Procedury specjalizowane. Są to funkcje i procedury realizujące specyficzne algorytmy użytkownika (np. funkcja zaokrąglania cen, przeliczniki wagowo-objętościowe, przekodowywanie danych itp.). Procedury te służą do ujednolicenia algorytmów używanych w kompleksowym systemie informatycznym

i zapewniają zgodność kodowania danych.

Zarówno moduły dostępu, jak i biblioteki procedur dedykowanych korzystają ze wspólnej biblioteki procedur ogólnego użycia. Biblioteka ta jest jedna dla wszystkich użytkowników. Zawiera ona definicje typów, kodów, struktur i procedur używanych we wszystkich programach, bibliotekach dedykowanych i modułach dostępu. Ze względu na swój charakter służy ona zatem przede wszystkim do unifikacji całego oprogramowania użytkowego. Najważniejsze grupy definicji i procedur wchodzących w skład biblioteki ogólnego użycia to:

a. Procedury obsługujące ekran monitora, oraz związane z nimi kody i definicje.
b. Procedury obsługujące klawiaturę, oraz związane z nimi kody i definicje.
c. Procedury obróbki elementarnych pól danych, oraz związane z nimi kody i definicje.
d. Procedury obsługujące przyjęty lokalnie standard organizacji ekranu monitora, oraz sposobów komunikacji programu z użytkownikiem i otoczeniem, oraz związane z nimi kody i definicje (standaryzacja tych procedur jest bardzo wygodna zarówno dla programisty, jak i dla użytkownika !).
e. Procedury obsługujące tworzenie tzw. wątków w systemie operacyjnym (zagnieżdżanie procesów) i przekazywanie parametrów pomiędzy procesami inicjowanymi przez programy jako wątki systemowe. Możliwość tworzenia wątków jest bardzo pożyteczną zaletą systemu operacyjnego, znacznie rozszerzającą możliwości użytkowe programów.
f. Procedury obsługujące lokalny zbiór tekstowy (tworzenie, otwieranie, zamykanie, kasowanie, zapis rekordu, wczytanie rekordu, wczytanie ciągu rekordów itp.). Standaryzacja tych procedur jest wygodna, gdyż w przetwarzaniu danych często są używane pomocnicze zbiory lokalne o organizacji tekstowej np. jako zestawy grup parametrów, pośrednie zbiory drukowania itp.
g. Procedury obsługujące konwersację z użytkownikiem realizowaną przy pomocy "menu" okienkowych, oraz związane z tym kody i definicje. Specyficznym problemem tej grupy procedur jest możliwość zagnieżdżania okienek, oraz generowanie przez programy użytkowe menu zawierających listy kodów i ich opisów, tworzone na podstawie odpowiednich procedur z bibliotek dedykowanych.
h. Procedury obsługujące kalendarz, oraz związane z tym kody i definicje. Pakiet procedur obsługujących kalendarz powinien obejmować daty z okresu co najmniej kilkudziesięciu lat (najlepiej całe stulecie), w tym również zmianę daty związaną ze zmianą stulecia w roku 2000. Musi on m. in. umożliwiać poprawne wykonywanie działań arytmetycznych na datach kalendarzowych.
i. Procedury obsługujące systemową organizację danych w pamięci zewnętrznej.
j. Procedury obsługujące drukowanie danych, oraz związane z nimi kody i definicje. Pakiet procedur obsługujących drukowanie danych powinien zapewniać odporność programu na zakłócenia technicznego stanu procesu drukowania (np. nie zabezpieczone odpowiednio próby drukowania na niesprawnej drukarce mogą powodować utratę dużej ilości danych), powinien dawać użytkownikowi możliwość przerwania wydruku w dowolnym momencie, oraz powinien umożliwiać wyprowadzanie wydruku na lokalny zbiór pośredni w pamięci zewnętrznej. Pakiet procedur obsługi drukowania powinien być uzupełniony oprogramowaniem pomocniczym obsługującym różne opcje związane z drukowaniem danych zapisanych do zbiorów pośrednich w pamięci zewnętrznej.
k. Procedury organizowania dostępu do danych w plikach (np. obsługujący

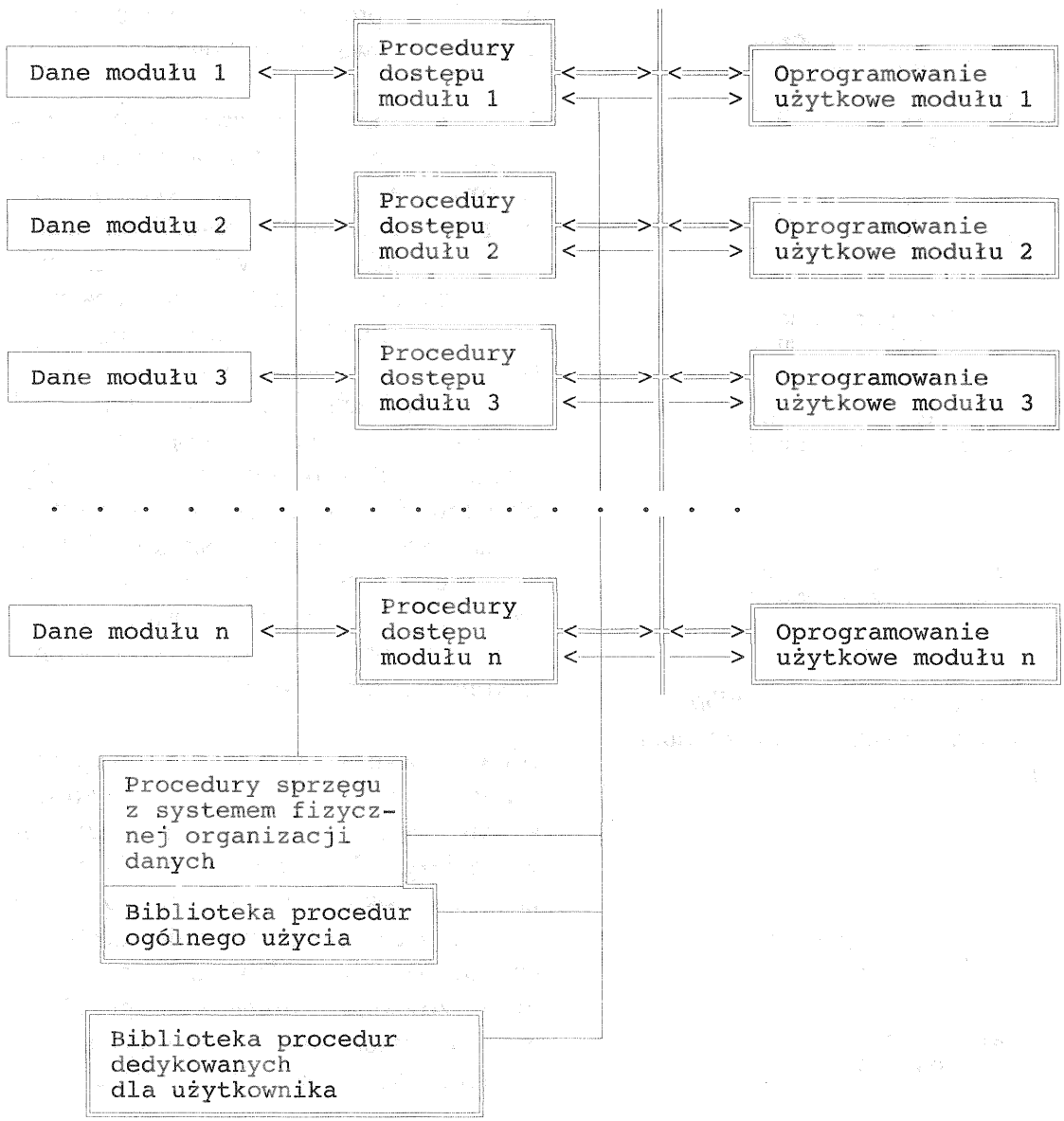
organizację danych w struktury B-drzewa). Jest to fundamentalny pakiet procedur dla całego oprogramowania. Na ogół procedury te nie są wbudowane w używany system kompilacji. Niektóre systemy programowania (np. BTRIEVE NOVELLa) zawierają taki pakiet procedur, ale zazwyczaj wymaga on dostosowania do używanego lokalnie standardu programowania. Jeśli chcemy używać lokalnego standardu organizacji danych, to dla jego obsługi należy przygotować pakiet procedur realizujących wszystkie niezbędne operacje na danych i ich strukturze. Pakiet ten powinien zapewniać łatwą, niezawodną i efektywną współpracę programu z danymi. Struktury i organizacja danych powinny zostać ujednolicone dla wszystkich zbiorów i programów. Dużą uwagę należy poświęcić dobremu "dostrojeniu" parametrów systemu organizowania danych (rozmiar bloku i rekordu, organizacja indeksów, rozmiar, ilość i typy kluczy, parametry indeksów itp.), gdyż ma to decydujący wpływ na efektywność systemów i programów. Procedury organizowania dostępu do danych w plikach powinny być opracowywane wariantowo, umożliwiając dostosowywanie oprogramowania do współpracy z różnymi systemami fizycznej organizacji danych (np. lokalne B-drzewo, siec NOVELL, sieć UNIX itp.). Takie wariantowe zestawy procedur nazywamy sprzęgami z systemem fizycznej organizacji danych (bliższy opis takich sprzęgów jest umieszczony poniżej).

l. Pakiet procedur pomocniczych dla obsługi standardów wymaganych w bankowości.

m. Inne pakiety pomocnicze np. dla obsługi wyświetlania na ekranie i drukowania formularzy z rubrykami, nagłówków programów, objaśnień dla użytkownika programu itp.

Jednym z istotnych problemów związanych z organizacją biblioteki procedur ogólnego użycia jest umożliwienie łatwego dostosowywania programów użytkowych do pracy w różnych warunkach technologicznych np. z różnymi systemami operacyjnymi, lub do pracy pracy lokalnej albo w sieci komputerowej itp. Dostosowywanie procedur i programów do pracy w różnych warunkach technologicznych powinno wymagać jak najmniejszych zmian w kodzie źródłowym programów i procedur. Można to osiągnąć wydzielając specjalne pakiety procedur pośredniczących pomiędzy procedurami i programami, a ich otoczeniem technologicznym. Pakiety te nazywamy procedurami sprzęgu ze środowiskiem. Szczególnym przypadkiem takich pakietów są sprzęgi z systemem fizycznej organizacji danych. Idea takich sprzęgów polega na tym, że dla obsługi dostępu do plików danych używane są w całym oprogramowaniu (również w modułach dostępu !) ujednolicone formalnie i syntaktycznie funkcje i procedury. Definicje tych funkcji i procedur są umieszczane w odpowiednich pakietach biblioteki procedur ogólnego użycia (patrz powyżej pkt. k.), obsługujących określoną fizyczną organizację danych (np. lokalne struktury B-drzewa, system BTRIEVE NOVELLa, itp.). Sprzęgnięcie programu użytkowego lub modułu dostępu z wymaganą fizyczną strukturą danych uzyskuje się poprzez umieszczenia w module lub programie odpowiedniej definicji sprzęgu z biblioteką procedur ogólnego użycia, bez konieczności wykonywania jakichkolwiek zmian w kodzie źródłowym programu. Dogodnym narzędziem do tego są dyrektywy przełącznikowe kompilatora. Dostosowanie oprogramowania do pracy w innym środowisku jest uzyskiwane poprzez zmianę odpowiedniej dyrektywy przełącznikowej kompilatora i rekompilację oprogramowania. Zalety praktyczne takiego rozwiązania są oczywiste.

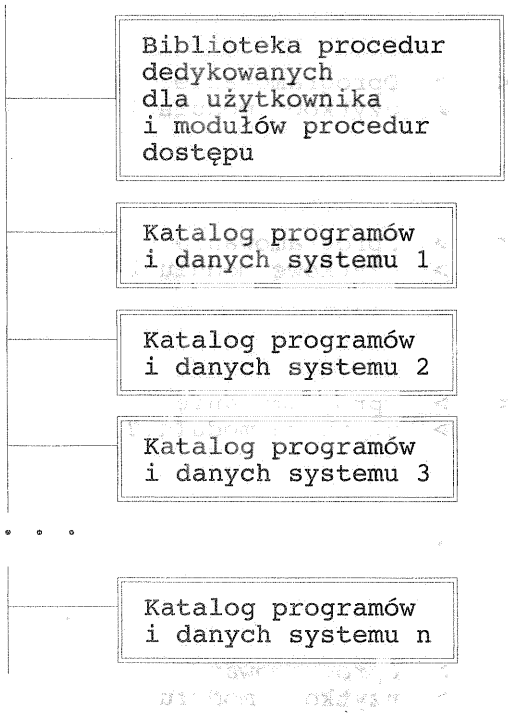
Ograniczone ramy referatu nie pozwalają niestety na omawianie wielu interesujących problemów szczegółowych związanych z opisaną organizacją systemu oprogramowania.



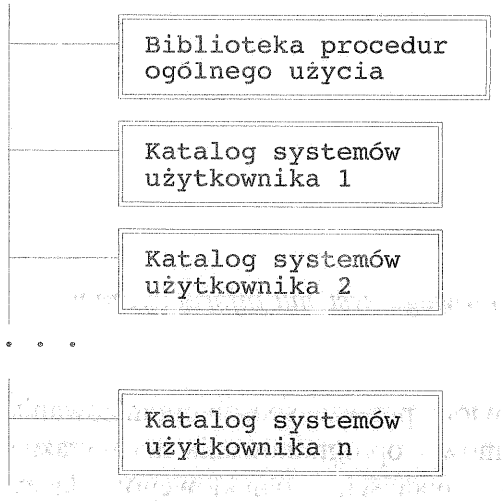
RYS. 3. Schemat organizacji oprogramowania kompleksowego systemu informatycznego

Przykładową organizację oprogramowania kompleksowego systemu informatycznego dla wybranego użytkownika, odpowiadającą opisanym powyżej zasadom przedstawia rysunek 3. Schemat organizacji bibliotek programowych i oprogramowania użytkowego dla tego użytkownika w katalogu dyskowym ilustruje rysunek 4. Schemat organizacji bibliotek programowych i oprogramowania w katalogu dyskowym dla różnych użytkowników ilustruje rysunek 5. Przykład organizacji kompleksowego oprogramowania dla użytkownika w katalogu dyskowym pokazuje rysunek 6.

Omówiony powyżej sposób organizowania systemów i oprogramowania ma wyraźne cechy podejścia obiektowego: łączy definicje typów i struktur danych z ich cechami i związanymi z nimi procedurami, oraz umożliwia dziedziczenie tych definicji poprzez określanie ich wzajemnych powiązań. Ma on bardzo ważną zaletę praktyczną: określone definicje o zasięgu globalnym znajdują się tylko w jednym miejscu systemu i nie są nigdzie powielane. Pozwala to uniknąć kłopotów i pomyłek związanych ze zmianą tych definicji. Ułatwia też wprowadzanie zmian zarówno w strukturach danych, jak i w związanych



RYS .4. Struktura rozmieszczenia bibliotek, systemów i danych użytkownika w katalogu dyskowym



RYS. 5. Struktura rozmieszczenia systemów w katalogu dyskowym

z nimi procedurach. Dzięki takiemu podejściu dostosowywanie istniejącego systemu do lokalnych potrzeb użytkownika jest stosunkowo łatwe, gdyż polega zazwyczaj na odpowiednich zmianach w bibliotece procedur dedykowanych i w modułach dostępu, nie wymaga natomiast wprowadzania istotnych zmian w oprogramowaniu użytkowym.

Opisane metody organizowania oprogramowania nie zostały sformalizowane, nie wykorzystują też formalnych technik programowania obiektowego. Formalizacja taka jest możliwa, ale w praktyce nie była konieczna, ani celowa. Ogranicza ona m.in. przejrzystość kodu źródłowego programów (cechy struktur i obiektów zostają "ukryte" w ich definicji i nie są "widoczne" bezpośrednio w kodzie programu), oraz utrudnia dostosowywanie definicji obiektów do aktualnych potrzeb użytkownika (różnice typów i struktur danych mogą tu być dość duże). Poprzestaliśmy zatem na praktycznym zastosowaniu idei podejścia obiektowego, pragmatycznie traktując jej formalizację.

Niewątpliwym ułatwieniem przy wypracowywaniu omówionych rozwiązań było stosowane przez nas narzędzie programowania: język TURBO-PASCAL. Zachowując wszystkie zalety precyzyjnego i eleganckiego programowania, dostarcza on dodatkowych ułatwień: możliwości organizowania bibliotek procedur w wyodrębnione moduły, oraz wygodnego kojarzenia takich modułów bibliotecznych.

Opisane rozwiązania są przez nas od kilku lat stosowane w praktyce z dużym powodzeniem. Dzięki nim możliwe było m.in. "bezpieczne" wykonywanie w dużych systemach bardzo istotnych zmian w trakcie ich eksploatacji (np. zmiany zasad i algorytmów naliczania podatku obrotowego od sprzedaży, zmiany struktury planu kont księgowych, zmiana struktury symbolu REGON itp.). Umożliwiają one dobrą kontrolę i skuteczne serwisowanie bogatego oprogramowania pracującego u różnych klientów i w różnych warunkach (lokalnie lub w sieciach). Pozwalają też na integrację wdrażanych stopniowo systemów dziedzinowych w system kompleksowy, przy minimalizacji kosztów i nakładu pracy. Metody te są przez nas stale doskonalone i rozwijane, w miarę napoty-

```
BIBL          (biblioteka procedur ogólnego użycia)
    uvideo.pas          (procedury obsługujące ekran monitora)
    ubasea.pas          (procedury sprzęgu lokalnego B-drzewa)
    ubaseb.pas          (procedury sprzęgu z siecią NOVELL)
  . . .
    . . .  (inne procedury ogólnego użycia)

AGRS        (ścieżka katalogowa użytkownika)
  BIBL            (biblioteka procedur użytkownika)
    aproagrs.pas  (biblioteka procedur dedykowanych)
    adoinod.pas   (moduł dostępu do kart. kontrahentów)
    adoceny.pas   (moduł dostępu do cennika wyrobów)
    adoewst.pas   (moduł dostępu do stanów mag. wyr.)
    adoewdo.pas   (moduł dostępu do obrotów mag. wyr.)
    adospwf.pas   (moduł dostępu do ewid. sprzedaży)
  CENY            (system dziedzinowy cennika wyrobów)
    ceny.exe    (program obsługi cennika wyrobów)
    cenydane.bdb    (dane cennika wyrobów)
    ceny.pas      (program źródłowy)
  INOD            (system dziedzinowy kartoteki kontrahentów)
    inod.exe    (program obsługi kartot. kontrahentów)
    inoddane.bdb    (dane kartoteki kontrahentów)
    inod.pas      (program źródłowy)
  EWWG            (system dziedzinowy ewidencji magaz. wyrobów)
    ewst.exe    (program stanów magaz. wyrobów)
    ewdo.exe    (program ewidencji magaz. wyrobów)
    ewkm.exe    (program kartoteki magaz. wyrobów)
    ewstdane.bdb    (dane stanów magaz. wyrobów)
    ewdodane.bdb    (dane ewidencji magaz. wyrobów)
    ewdo.pas      (program źródłowy)
    ewst.pas      (program źródłowy)
    ewkm.pas      (program źródłowy)
  ZBYT            (system dziedzinowy ewidencji sprzedaży)
    spwf.exe    (program dysponowania sprzedaży)
    spfa.exe    (program emisji faktur sprzedaży)
    spwz.exe    (program emisji dokumentów WZ)
    spwfda93.bdb    (dane dyspozycji sprzedaży)
    spwfop93.bdb    (dane opisów dla dysp. sprzedaży)
    spwf.pas      (program źródłowy)
    spfa.pas      (program źródłowy)
    spwz.pas      (program źródłowy)
```

RYS. 6. Przykład rozmieszczenia danych i oprogramowania użytkownika w katalogu dyskowym (dla uproszczenia przedstawiono wyłącznie oprogramowanie źródłowe)

kania i rozwiązywania nowych problemów praktycznych, oraz powstawania nowych, coraz doskonalszych narzędzi programowania.

Na zakończenie referatu warto podać kilka doświadczeń praktycznych uzyskanych podczas opracowywania i wdrażania złożonych systemów informatycznych. Nie są one co prawda bezpośrednio związane z tematem referatu, ale mogą być interesujące dla ludzi. którzy zetknęli się (lub mogą się zetknąć) z podobnymi problemami w swojej praktyce.

1. Jeśli użytkownik ma ograniczone środki, lub ograniczone możliwości percepcyjne (np. nie dysponuje własnym, wysoko kwalifikowanym personelem informatycznym), to do budowania swoich kompleksowych systemów informatycznych nie powinien wykorzystywać dużych systemów uniwersalnych baz danych. Systemy takie mają oczywiste i nie kwestionowane zalety, ale dla wielu użytkowników mają też poważne wady: wysoki koszt, czasochłonność i pracochłonność wdrożenia i opanowania obsługi, brak możliwości kontrolowania

działania systemu na "niskim poziomie" zastosowań (jest to szczególnie kłopotliwe w sytuacji, kiedy np. system "robi" nie to, co powinien i co wynika z jego dokumentacji, w zastosowaniu nie widać błędów, a serwis autorski systemu jest na drugiej półkuli). Często nakład pracy i środków koniecznych dla opanowania i uruchomienia uniwersalnego systemu baz danych jest wyższy, lub porównywalny z nakładami na opracowanie potrzebnego oprogramowania od nowa, a efekty niekoniecznie są lepsze. Użytkownik rzadko przy tym wykorzystuje wszystkie możliwości uniwersalnego oprogramowania, a musi za nie zapłacić w cenie jego zakupu. Użytkownik nie powinien się też nadmiernie sugerować entuzjastycznymi opiniami oferentów takich systemów. Opinie takie często, niestety, wynikają z braku doświadczenia, lenistwa, lub nie dysponowania ofertą alternatywną. Nierzadko mają one też podłoże czysto merkantylne. Dlatego decyzja o ewentualnym zastosowaniu uniwersalnego systemu baz danych powinna zawsze być poprzedzona staranną analizą jego możliwości, własnych potrzeb, oraz rozwiązań alternatywnych.

2. Użytkownicy kompleksowych systemów informatycznych nie powinni dążyć do nadmiernej integracji zasobów systemu. W szczególności powinni oni ograniczać wzajemną interaktywność danych i procesów w systemie do rozsądnego minimum. Dotyczy to zwłaszcza systemów przetwarzających duże i bardzo duże ilości danych. Użytkownik rzadko naprawdę potrzebuje bieżącej i pełnej informacji z danych rejestrowanych bezpośrednio w kompleksowym systemie. Na ogół wystarcza mu wyciąg z tych danych, lub informacja syntetyczna opóźniona o parę minut, godzin, a nawet dni. Dotyczy to zwłaszcza systemów pracujących w czasie rzeczywistym (np. w dużych sieciach). Nadmierna interaktywność danych zawsze zmniejsza sprawność systemu i zwiększa narzut niezbędny dla jej uzyskania przy zachowaniem warunków integralności i bezpieczeństwa danych. Komplikuje też procedury korygowania błędnie zarejestrowanych danych (co przy wielu operatorach współpracujących z systemem wcale nie jest tak rzadkie i mało prawdopodobne, jak zapewniają zazwyczaj twórcy systemu, lub jego dokumentacja). Dlatego wszędzie tam, gdzie nie jest to absolutnie konieczne należy pozostawić do uznania operatora uruchamianie procedur interaktywnych (np. rejestrowanie w obrocie magazynowym danych sprzedaży), lub tworzenie danych syntetycznych (np. księgowanie dokumentów obrotowych na stany kont księgowych). Oczywiście system musi wtedy umożliwiać kontrolę integralności danych i gwarantować jej odzyskanie w wypadku stwierdzenia błędów operatora, oraz umożliwiać restrukturyzację danych po awariach. Takie rozwiązanie upraszcza zazwyczaj strukturę systemu, ułatwia jego obsługę i dobrze zabezpiecza go przed awariami.

3. Użytkownicy kompleksowych systemów informatycznych powinni dążyć do zapewnienia sobie maksymalnego wglądu w strukturę, technologię, oprogramowanie i funkcjonowanie systemu. Dostarczana z systemem dokumentacja eksploatacyjna nie zawsze jest do tego wystarczająca. Nasza praktyka dowodzi ponadto, że objętość dostarczonej dokumentacji systemu bywa odwrotnie proporcjonalna do chęci korzystania z niej przez użytkowników. Dlatego użytkownicy kompleksowych systemów powinni zapewnić sobie własną wykwalifikowaną obsługę tych systemów, a od twórców żądać dostarczenia wszelkich informacji niezbędnych dla tej obsługi (bezpośredniego szkolenia obsługi systemu, serwisowania wdrożeń i eksploatacji, konsultacji, a nawet kodów źródłowych programów). Wszystkie nasze dotychczasowe niepowodzenia wdrożeń dużych systemów (na szczęście niezbyt liczne), miały zawsze podłoże w przeświadczeniu użytkowni-

ka, że komputer, lub autorzy systemu sami za niego wszystko "załatwią", a on nie musi się niczego nowego uczyć.

4. Kompleksowych systemów nie udaje się na ogół wdrożyć szybko i bez problemów. Jest to z reguły proces wymagający sporych nakładów pracy i środków zarówno ze strony autorów, jak również (a może nawet przede wszystkim) ze strony użytkowników. Nie należy mieć złudzeń, że duże systemy profesjonalne można uruchomić tanio i bez wysiłku. Zasada ta dotyczy przede wszystkim użytkowników mających duże chęci i ambicje wobec komputeryzacji, ale niewystarczające przygotowanie merytoryczne. Nie wolno forsować złudnych oszczędności na kosztach zakupu i wdrażania systemu. Za marną cenę z reguły otrzymuje się marny system. Niewątpliwym pocieszeniem jest fakt, że wysiłek włożony we wdrożenie i opanowanie systemu często przynosi spore, czasami nieoczekiwane korzyści. Znamy przypadki, kiedy bezpośrednie nakłady na uruchomienie systemu (często niemałe) zwracały się kilkunastokrotnie już w początkowym okresie jego użytkowania, dzięki uzyskaniu przez użytkownika danych, o których w tradycyjnym systemie nie miał tzw. zielonego pojęcia.

5. Systemy powinny być wyposażane w możliwości odtwarzania danych na każdym poziomie ich organizacji. Nie wolno przyjmować założeń, że "coś w systemie jest niemożliwe". Użytkownicy (nawet ci przeszkoleni) potrafią czasami robić rzeczy, którym sami się potem dziwią (znamy przypadek, kiedy superoperator dużej sieci, po awarii zasilania zewnętrznego, rozpoczął "zwijanie" sieci od wyłączenia lokalnego podtrzymywacza napięcia). Należy również przewidywać możliwość odtwarzania w systemie danych i przebiegów teoretycznie niepowtarzalnych (np. przeksięgowanie danych zaksięgowanych).

6. Systemy powinny być odporne na wszelkie nieracjonalne zachowania użytkowników, oraz wyposażane w staranne i "szczelne" zabezpieczania danych. Dotyczy to przede wszystkim systemów sieciowych (ale nie tylko). Typowe rozwiązania stosowane w oprogramowaniu zarządzającym sieciami nie zawsze jest w tych sprawach wystarczające. Ciekawość użytkowników, ich pomysłowość w docieraniu do danych nie dla nich przeznaczonych, oraz skłonność do "zabawy" systemami komputerowymi bywa bowiem zaskakująca.

7. W poważnych zastosowaniach nie należy wykorzystywać oprogramowania nielicencjonowanego. Zasada, że kradzione nie tuczy, w informatyce sprawdza się bowiem wyjątkowo często i zazwyczaj bardzo boleśnie.

Jestem świadomy faktu, że powyższe opinie mogą się wydawać trywialne, kontrowersyjne, lub zachowawcze. Nic jednak nie poradzę na to, że takie właśnie są moje doświadczenia praktyczne. W razie potrzeby potrafię je uzasadnić licznymi przykładami i opowieściami.

Zbigniew A. Nowacki
AKREM, Łódź

Szablony klas i listy elastyczne w pakiecie narzędziowym POLY do języka C

W pierwszej części tego wykładu chciałbym zaprezentować zarys pakietu narzędziowego POLY do języka C. Natomiast w drugiej części zajmę się bardziej szczegółowo zagadnieniami zasygnalizowanymi w tytule, tj. przedstawię jedno z najważniejszych ostatnio wprowadzonych rozszerzeń języka C - **szablony** oraz pokażę, w jaki sposób zostały one wykorzystane w pakiecie POLY do implementacji tzw. **list elastycznych**, czyli nowej metody przetwarzania struktur listowych.

1. Pakiet narzędziowy POLY-DOS.

Pakiet POLY do języka C składa się z dwóch zasadniczych członów: biblioteki funkcji POLY oraz zestawu programów określanych wspólnym mianem POLY-DOS. Rozpoczniemy od omówienia tej drugiej składowej.
Pakiet POLY-DOS został wyodrębniony z opracowanego wcześniej pakietu POLY5+87 (dla języka CLIPPER), do którego były sukcesywnie włączane programy rezydentne i narzędziowe o uniwersalnym zakresie działania. Jądro POLY-DOS-u stanowią trzy programy rezydentne: Video Manager POLYFONT, Keyboard Manager POLPRESS i Printing Manager POLPRINT. Uzupełniają je programy narzędziowe: edytor fontów dla monitora i drukarki PFONTED, edytor makrodefinicji klawiatury KEYEDIT, program definicji "download" POLDRUK i inne.
Pakiet POLY-DOS rozwiązuje, rzecz jasna, wszelkie problemy związane z polskimi literami (dowolny system kodowania, opcja zmiany systemu w "locie", rozmaite polskie klawiatury, nieznikające polskie znaki na ekranie i drukarce), ale daje też, a może przede wszystkim, szereg udogodnień rozszerzających możliwości standardowego PC-ta. Przykładowo, Video Manager pozwala pracować w trybie tekstowym z zestawem 512 fontów, z których wszystkie lub niektóre mogą być zdefiniowane przez użytkownika, umożliwia natychmiastowe przejście z 25 wierszy ekranu na 28, 43 lub 50 wierszy i odwrotnie, zaś Keyboard Manager jest w stanie przyporządkować ponad 1000 makrodefinicji o dowolnej długości do poszczególnych klawiszy (modyfikowanych, być może, klawiszami sterującymi).
Programy POLPRINT i POLDRUK pozwalają drukować dowolne fonty w trybie tekstowym na drukarce posiadającej opcję "download" (wg standardu Epson lub IBM). POLDRUK ma bardzo szerokie możliwości włącznie z trybem NLQ, podczas gdy POLPRINT posiada niezwykłą zdolność automatycznego definiowania fontów po każdym włączeniu drukarki. W pakiecie dostarczane są również pliki zawierające gotowe fonty możliwe do wykorzystania zarówno na drukarce, jak i na ekranie.
POLY-DOS posiada ponadto pewną szczególną cechę wyraźnie odróżniającą go od innych nakładek polonizacyjnych i usprawiedliwiającą użycie nazwy "POLY-DOS". Są to mianowicie tzw. *funkcje POLY-DOS-u* wywoływane podobnie jak standardowe funkcje DOS-u lub BIOS-u. Ponieważ

każdy praktycznie używany język programowania (a zwłaszcza język C) ma udogodnienia umożliwiające korzystanie z przerwań, więc funkcje POLY-DOS-u mogą być stosowane w dowolnych programach użytkowych.
W ten sposób programista pracujący pod POLY-DOS-em uzyskuje szereg niedostępnych nigdzie indziej możliwości. Może np. wymieniać fonty, makrodefinicje klawiszy czy systemy kodowania polskich liter bez potrzeby reinstalacji rezydujących Managerów, może zlecać im wykonanie pewnych zadań (zmiana ilości wierszy na ekranie, tłumaczenie systemów kodowania między komputerem i drukarką, itp.), czy też po prostu okresowo wstrzymywać ich pracę. Oczywiście stosowanie funkcji POLY-DOS-u nie jest obowiązkowe; w istocie rzeczy programy wchodzące w skład pakietu same wykonują je często w sposób automatyczny.
Rezydentne programy pakietu POLY-DOS zostały napisane w języku ASSEMBLER, zaś nierezydentne - w C. Zapewniło to odpowiednią efektywność systemu, a w szczególności stosunkowo niewielką zajętość pamięci. Przykładowo, jeśli oszczędzanie pamięci jest konieczne, to rezygnując z niektórych mniej ważnych funkcji można spowodować, iż POLYFONT po instalacji zajmie tylko ok. 4 KB, zaś POLPRESS i POLPRINT po 2 KB pamięci.

2. Biblioteka funkcji POLY dla C

Biblioteka POLY dla C powstała z opracowanej wcześniej biblioteki POLY dla CLIPPERA, której większość funkcji była napisana właśnie w języku C. Zostały one zmodyfikowane zgodnie z filozofią C, a ponadto dołączono do nich szereg dodatkowych funkcji, bądź niepotrzebnych w CLIPPERZE, jako że wchodzących tam do podstawowego języka, bądź też niemożliwych w nim do wykorzystania.
Biblioteka może być wykorzystywana zarówno w tradycyjnym C wg standardu ANSI, jak i w C++. W tym drugim przypadku programista ma do dyspozycji wiele dodatkowych udogodnień, których zestaw określany jest mianem POLY++. Wszystkie funkcje dostarczane są dla użytkownika również w postaci źródłowej. Ponad 300 funkcji biblioteki POLY można podzielić na następujące zasadnicze moduły:

I. Realizacja przerwań BIOS-u.
II. Obsługa ekranu.
III. Obsługa klawiatury.
IV. Obsługa drukarki.
V. Przetwarzanie polskich znaków diakrytycznych.
VI. Komunikacja z rezydentami pakietu.
VII. Manipulacje na łańcuchach znaków.
VIII. Pakowanie i szyfrowanie danych.
IX. Przetwarzanie dat.
X. Przetwarzanie list.
XI. Pełnoekranowe wprowadzanie danych.
XII. Organizacja menu.
XIII. Funkcje wywoływane przez makra języka NewOrder.
XIV. Obsługa okienek.
XV. Obsługa baz danych (upgrade w przygotowaniu).

Nazwa każdej funkcji biblioteki POLY tradycyjnie zaczyna się od znaku podkreślenia _. Dzięki temu nazwy niektórych spolonizowanych funkcji różnią się od nazw swych standardowych odpowiedników właśnie tylko kreseczką, np. `_strcmp` i `strcmp`.
Funkcje modułów II, III oraz IV obsługują urządzenia zewnętrzne przy pomocy przerwań BIOS-u wywoływanych za pośrednictwem należącej do modułu I funkcji `_intpoly`, bądź też odwołując się bezpośrednio do zmiennych BIOS-u przy użyciu makrodefinicji `FARPOINT`. Ta pierwsza realizuje swoje zadania albo przy pomocy standardowej funkcji `int86`, albo (po wywołaniu funkcji `_vdirect`) poprzez symulację operacji BIOS-u bezpośrednim działaniem na pamięci ekranu. Ponadto `_intpoly` jest jedyną funkcją z biblioteki POLY odwołującą się do funkcji niezdefiniowanych w standardzie ANSI i niedo-

stępnych w systemie UNIX (konkretnie do `int86`). Zatem w celu przeniesienia biblioteki na komputery lub systemy operacyjne różne od IBM PC i DOS, wystarczy zmodyfikować funkcję `_intpoly` i przedefiniować `FARPOINT`. (Oczywiście potrzebna jest w tym celu dobra znajomość BIOS-u dla IBM PC oraz BIOS-u i systemu operacyjnego komputera docelowego).

Moduł V nie jest określony w sposób "czysty"; wiele znajdujących się tu funkcji można by zaliczyć także do innych modułów, a w szczególności do modułu VI i VII. Funkcje z modułu V zostały opracowane w ten sposób, aby przetwarzanie z polskimi literami maksymalnie upodobnić do zwykłego wykorzystującego alfabet międzynarodowy. Np. funkcje tradycyjnie realizowane jako makra mają w POLY spolonizowane odpowiedniki również zrealizowane jako makra.

Warto zaznaczyć, iż biblioteka POLY jest dostosowana do dowolnego systemu kodowania polskich znaków, nawet wymyślonego *ad hoc* przez użytkownika. Poszczególne standardy są definiowane bądź przez zmienne środowiskowe DOS-u o nazwach POL, MAZ, DHN, itd., bądź przez zbiory dyskowe zakładane pomocniczym programem POLZNAK. Zmienne te lub zbiory zawierają tylko 18 polskich liter; wszystkie inne tablice tworzone są automatycznie przez funkcję `_pldef`, która musi wystąpić na początku każdego programu korzystającego z polskich liter.

Funkcje z biblioteki POLY umożliwiają tłumaczenie danych z jednego systemu na drugi, a także pozwalają przenosić programy z komputera wykorzystującego, dajmy na to, system DHN na komputer z systemem Mazovii i odwrotnie. Co więcej, funkcja `_p` umożliwia pisanie programów z polskojęzycznymi komunikatami na komputerze nie posiadającym polskich liter. Zresztą użytkownik pakietu POLY może pracować z polskimi literami na dowolnym komputerze bez jakichkolwiek przeróbek sprzętu (ekran, klawiatura, drukarka) wykorzystując zawarte w pakiecie programy POLY-DOS-u. Funkcje modułu VI umożliwiają sterowanie rezydentami z programów użytkowych pisanych w C.

Nazwy funkcji modułu VII zazwyczaj zaczynają się od `_str`. Nie jest to jednak regułą, gdyż nazwy wielu funkcji z tego modułu zostały zapożyczone z CLIPPERA. Dotyczy to funkcji "kombajn" `_stuff` oraz takich funkcji jak `_pad`, `_empty` czy też `_trim`.

Moduł IX zawiera funkcje, które w CLIPPERZE są wbudowane w standardowy język, a więc nie było potrzeby włączania ich do biblioteki POLY dla CLIPPERA. W module zrealizowany został pełny algorytm gregoriański umożliwiający wykonywanie dowolnych operacji na datach i przechowywanie ich w postaci liczb całkowitych.

Dla potrzeb biblioteki POLY zostało zdefiniowanych pięć typów listowych (patrz wydruk 1), a funkcje je obsługujące zostały zebrane w module X. Programista może deklarować własne listy powyższych typów, a w przypadku konieczności użycia innych typów listowych może wzorując się na wersjach źródłowych napisać własne wersje funkcji je obsługujących. Nie jest to jednak, jak zobaczymy dalej, potrzebne

```
typedef struct anylist
 {struct anylist *prev; void *any; } *anylist;                    (1)
typedef struct wordlist
 {struct wordlist *prev; int word;} *wordlist;
typedef struct numlist
 {struct numlist *prev;long num;} *numlist;
typedef struct namelist
 {struct namelist *prev; char* name;} *namelist;
typedef struct keylist
 {struct keylist *prev;int key;intfun funkcja;} *keylist;
```

WYDRUK 1: Typy listowe zdefiniowane dla potrzeb biblioteki POLY

w POLY++ z szablonami.
Warto dodać, że przetwarzanie list w pakiecie POLY zostało zrealizowane w oparciu o pewną sztuczkę programistyczną, dzięki czemu listy te dają znacznie większe możliwości niż listy implementowane w standardowych bibliotekach. Oczywiście istnieje tu również ściśle określony algorytm, w pełni widoczny dla użytkownika POLY, ale algorytm ten posiada pewną charakterystyczną cechę: dość trudno go sformułować zajmując się listami od strony teoretycznej, zaś znacznie łatwiej pisząc programy przetwarzające listy, najlepiej w języku C.
Pełnoekranowe wprowadzanie danych wykorzystujące obiekty GET i tzw. *formaty izometryczne* zrealizowane jest w module XI. Obiekty GET dostępne są już w bibliotece POLY używanej w zwykłym C, ale POLY++ umożliwia realizację pełnoekranowego strumienia danych przy pomocy obiektów klasy `Getstream`, wg przykładowego schematu:

```
Getstream gs;
gs << Get(...)
<< Get(...)
 ...
<< Get(...)
>> SCREEN;
```

Funkcje `Get` zwracają wskaźniki do obiektów GET podających dla każdej danej następujące informacje: komentarz wprowadzający, format, nazwa funkcji sprawdzającej poprawność, nazwa funkcji zezwolenia na wprowadzanie i inne.

Ważne miejsce w bibliotece POLY zajmują funkcje z modułu XII umożliwiające zorganizowanie menu dwuwymiarowego, rolowanego lub nie, z dowolną liczbą kolumn i wierszy. Dwuwymiarowość jest często naturalną strukturą opcji, a zawsze pozwala zaoszczędzić powierzchnię ekranu. Dzięki wprowadzeniu obiektów PROMPT, menu może mieć w razie konieczności dowolny nieregularny kształt. Istnieje również możliwość konstrukcji menu logicznego polegającego na wyborze nie jednej opcji, lecz pewnego podzbioru zbioru opcji prezentowanych. Termin "menu logiczne" pochodzi stąd, że taki wybór musi być zapisany w tablicy o elementach traktowanych jako wartości logiczne i funkcja `_logmenu` zwraca tablicę określającą dokonany wybór. Ponadto w POLY++ zdefiniowany jest operator << umożliwiający konstrukcję menu strumieniowego.

W module XIII zostały zgrupowane funkcje, które nie powinny być przez programistę wywoływane bezpośrednio, a jedynie poprzez makra należące do tzw. **języka NewOrder**. Język ten wykorzystuje preprocesor C w celu maksymalnego ułatwienia procesów tworzenia helpu kontekstowego, sformatowanego wprowadzania danych oraz wielopoziomowych menu powiązanych ze strukturą programu. Jest on trochę podobny do języka NewOrder dla CLIPPERA 5, aczkolwiek wiele różnic zostało wymuszonych przez odmienne możliwości preprocesorów.
Wydruk 2 zawiera przykładowy fragment programu w języku NewOrder (a właściwie NewOrder++ ze względu na użycie operatora <<). Jak widać, komendy języka NewOrder mogą dowolnie przeplatać się z instrukcjami i deklaracjami C.

Podejście okienkowe pozwala na tworzenie programów i procedur niezależnych od parametrów aktualnie stosowanego okna. Przy pomocy funkcji `_okno` i pokrewnych (należących do modułu XIV) możliwe jest utworzenie listy okien, przy czym okno znajdujące się w danym momencie na końcu listy nazywane jest oknem aktywnym. Podejście okienkowe nie ogranicza się tylko do funkcji biblioteki POLY, ale może być wykorzystywane w wywołaniach funkcji użytkownika oraz funkcji z innych bibliotek; wszędzie tam, gdzie jako parametry stosuje się współrzędne okna lub kursora.

W czasie przygotowywania tego wykładu funkcje modułu XV były testowane. Przewiduje się, że upgrade pakietu POLY zawierający moduł obsługi baz danych wejdzie do sprzedaży na jesieni 1993 r.

```
#include "NEWORDER.H"
void PROTOTYPE(help,(void))
PROTOTYPE(fun,(char* par))
void cfun(char*);
GENESIS
   _pldef(0);
   ASSIGN(EXTEND(F1),help)
   fun(ARGUMENT(1));
STOP(0)
FUNCTION(fun,(char* par))
   static DATE dat={1978,4,23};
   int log=TAK;
   double d=-1;
   static char t[]="ALFA21";
   MAINMENU(2,2)
      THISMENU << Prompt(2,4,"Kopiowanie")
      << Prompt(2,24,"baza Danych")
      << Prompt(4,4,"Indeksy")
      << Prompt(4,24,"kaSowanie");
      SELECT
      SUBMENU(3,2,1,1)
         THISMENU << Prompt(12,15,"Przeszukiwanie")
         << "Testowanie"
         << "Kopiowanie"
         << FREE
         << par
         << "Wyniki";
         SELECT
         BEGIN(WITHOUT,CHOICE)
            cout << "BEZ WYBORU\n";
         END
      ENDMENU
      BEGIN(0,0)
         cout << _p("WYBRANO OPCJE' 0,0\n");
      END
      BEGIN(OTHER,WISE)
         cout << "WYBRANO INNE\n";
      END
   ENDMENU
   GETSTREAM
      << Get(4,0,"Kalendarz",it,CALENDAR,&dat,"DD/MM/RRRR",
         INVERSE,sprawdz_dat,0,TAK)
      << Get(6,40,"Warunek",it,LOGICAL,&log,"......T......")
      << Get(12,0,_p("Wielkos'c'"),it,NUMERIC,&d,
         "-9,999,990.09")
      << Get(12,40,"Tekst",it,TEXT,&t[0],"XxxxCC");
   READ
STOP(1)
void FUNCTION(help,(void))
   SAVE_SCREEN
   cfun(CONTEXT(1));
   REST_SCREEN
FINISH
```

WYDRUK 2: Przykładowy fragment programu w języku NewOrder

3. Szablony - najnowsze rozszerzenie języka C.

Pojęcie **szablonów** zwanych również *sparametryzowanymi typami* zostało wprowadzone do standardu Cfront 3.0, czyli najnowszego wydania obiektowo zorientowanego nadzbioru języka C opracowanego przez Bjarne Stroustrupa z firmy AT&T Bell Laboratories. Pierwsza kompletna komercyjna implementacja szablonów pojawiła się w kompilatorze Borland C++ 3.1 dostępnym również w Polsce.

Najogólniej mówiąc, jeśli tylko stwierdzimy konieczność napisania szeregu bardzo podobnych procedur, powinniśmy pomyśleć o szablonach. Jednym z najprostszych, lecz mimo to interesujących przy-

kładów jest funkcja `swap(x,y)`, której zadaniem jest zamiana zawartości dwóch parametrów `x` i `y` identycznego typu. Konieczność wykonania operacji tego rodzaju jest często spotykana w informatyce, wiadomo jak ją przeprowadzić, ale dotychczas trzeba było taką funkcję pisać oddzielnie dla każdego typu.
Koncepcja szablonów rozwiązuje powyższy problem raz na zawsze. Aby ją zastosować, trzeba przede wszystkim zadeklarować szablon przy pomocy deklaracji postaci

```
template <argument_1,
          ...,argument_k>    (2)
```

gdzie `k` >= 1, a każdy argument szablonu może być jednej z następujących postaci:

```
class nazwa      (3)
nazwa_typu nazwa1    (4)
nazwa_typu nazwa1 = wyrażenie stałe
                                (5)
```

Bezpośrednio po deklaracji szablonu powinna wystąpić deklaracja lub prototyp funkcji, bądź też definicja klasy, w których można używać identyfikatorów `nazwa` z argumentów postaci (3) dla oznaczenia dowolnych typów (oczywiście ten sam identyfikator użyty wielokrotnie reprezentuje ten sam typ), zaś identyfikatorów `nazwa1` z argumentów (4) lub (5) dla oznaczenia wielkości stałych (argumenty te są nazywane *pozatypowymi*).
Deklaracja funkcji poprzedzona deklaracją szablonu określana jest mianem *szablonu funkcyjnego* lub *funkcji ogólnej (ang. generic function)*. W szablonie funkcyjnym nie wolno stosować argumentów pozatypowych. Natomiast konkretny przypadek użycia szablonu funkcyjnego nazywa się *funkcją szablonową* i formalnie rzecz biorąc nie różni się niczym od wywołania zwykłej funkcji; kompilator sam zadba o to, aby został wykorzystany odpowiedni szablon.
Powracając do naszego przykładu funkcji `swap`, możemy zastosować następujący szablon funkcyjny:

```
template <class T>
void swap(T & x, T & y)
{
 T robocza;
 robocza=x;
 x=y;
 y=robocza;
};
```

Dla każdego typu `T` takiego, że w programie znajduje się przynajmniej jedno wywołanie postaci `swap(x,y)`, gdzie `x` i `y` są typu `T`, kompilator wygeneruje dokładnie jeden egzemplarz funkcji szablonowej przeprowadzającej żądaną wymianę. Zatem jest to dokładnie to, co musieliśmy do tej pory robić "ręcznie". Widać też, że bez straty efektywności można teraz wykonać w C wiele rzeczy możliwych dotąd tylko w językach takich jak CLIPPER, gdzie typ zmiennej jest znany dopiero w czasie realizacji programu.

Klasa, której definicja poprzedzona jest deklaracją szablonu nazywa się *szablonem klas* albo *generatorem klas* albo też *klasą ogólną (ang. generic class)*. Jeśli `x` jest nazwą takiej klasy, to użycie jej konkretnej wersji musi mieć postać:

```
X<T1, ..., Tm>                  (6)
```

gdzie `T1, ...., Tm` powinny być aktualnymi argumentami odpowiadającymi formalnym argumentom z (2). Dokładniej mówiąc, pod argumenty postaci (3) należy podstawić konkretne typy, zaś pod argumenty pozatypowe - stałe wyrażenia typu zgodnego z podanym w (4) lub (5). Ilość argumentów aktualnych może być mniejsza od ilości argumentów formalnych tylko wtedy, gdy wszystkie brakujące argumenty były postaci (5); występujące w nich wyrażenia są traktowane jako wartości domyślne. Zaleca się zastępowanie (6) odpowiednią nazwą wprowadzoną przez `typedef`.

4. Listy elastyczne.

Aczkolwiek nie jestem w stanie zaprezentować tu całego algorytmu przetwarzania list w bibliotece POLY, to jednak przedstawiam jego najistotniejszą część, a mianowicie strukturę danych tworzących listę. Niektórzy czytelnicy mogą poczuć się rozczarowani, ponieważ struktura ta jest

niezwykle prosta (ale właśnie dlatego to działa). Definicja podstawowego szablonu klas o nazwie `polylist` w pliku POLYTEMP.H zawiera tylko dwie składowe danych i zaczyna się następującym fragmentem:

```
template <class T>
class polylist
{
anylist lista;
static T* (*copy)(T*);
 ...
```

Dalej występują już tylko definicje operatorów i składowych funkcji. Najistotniejszą składową danych jest oczywiście `lista`, która jest typu `anylist` - dokładnie takiego, jaki został zdefiniowany przez (1), z tym że w C++ musimy użyć:

```
typedef
 struct Anylist {Anylist
       *prev; void *any; } Anylist;
typedef Anylist *anylist;
```

Pomimo formalnych różnic powyższa definicja znaczy to samo co (1), a zatem funkcje obsługi list typu `anylist` zawarte w zwykłym POLY są również wykorzystywane w POLY++.

Aby przetwarzać listy w POLY++, należy włączyć do programu plik POLYTEMP.H oraz zadeklarować używane typy listowe przy pomocy makro

```
POLYLIST(typ_listy,
                typ_obiektu, kopia)
```

gdzie `typ_listy` jest unikalną (dotychczas nieużywaną) nazwą typu, `typ_obiektu` jest nazwą dowolnego już istniejącego typu, zaś `kopia` jest nazwą funkcji o podanym wcześniej prototypie:

```
*typ_obiektu kopia(*typ_obiektu);
```

lub nazwą specjalną `NOCOPY`. Od tego momentu w programie mogą występować deklaracje takie jak, przykładowo:

```
typ_listy a, b, c;
```

która powoduje utworzenie trzech początkowo pustych list `a`, `b` i `c`, których elementami mogą być wskaźniki do obiektów typu `typ_obiektu`. Parametr `kopia` określa, czy obiekty dołączane do listy mają być kopiowane (wyjaśniamy to dokładniej dalej).

W programie makro `POLYLIST` może wystąpić dowolnie wiele razy i utworzone w ten sposób różne typy listowe mogą być wykorzystywane jednocześnie. Jak łatwo można się domyślić, każde makro `POLYLIST(x,y,z)` m.in. nadaje przy pomocy `TYPEDEF` nazwę `x` dla typu `polylist<y>` (tj. konkretyzacji typu `polylist` wg (6)), zaś `z` służy do inicjacji członu statycznego `copy`. Po zadeklarowaniu zmiennych listowych można poddawać je działaniu kilku operatorów. I tak operator

```
list <= pointer
```

dodaje wskaźnik `pointer` na końcu listy `list` zwracając referencję do `list`, co umożliwia wykonywanie operacji wielokrotnych, np.

```
POLYLIST(textlist, char, NOCOPY)
 ...
textlist li;
li <= "CD" <= "EF" <= "GH" <= "IJ";
```

W przypadku, gdy parametr `kopia` makra `POLYLIST` jest różny od `NOCOPY`, to do listy `list` dodawany jest nie `pointer`, ale `kopia(pointer)`. Na ogół oczekuje się przy tym, że funkcja `kopia` skopiuje obiekt wskazywany przez `pointer` do nowego obszaru przydzielonego przy pomocy standardowego operatora `new` i zwróci adres kopii. Zatem ten wariant może być interpretowany jako wstawianie do listy obiektów zamiast wskaźników.

Należy podkreślić, iż dzięki wykorzystaniu szablonów istnieje tu pełna kontrola typów. Na przykład sekwencja

```
POLYLIST(intlist, int, NOCOPY)
 ...
intlist inli;
inli <= "CD";
```

spowoduje zasygnalizowanie błędu już na etapie kompilacji. Zatem chociaż listy te sprowadzają się, jak widzieliśmy, do list

typu `anylist`, programista może uważać, iż dysponuje listami nieskończenie wielu rodzajów.
Unarne operatory + i -- zastosowane do listy zwracają ostatni (czyli ostatnio wprowadzony) wskaźnik na liście. Jedyną różnicą między nimi jest to, że + pozostawia listę bez zmiany, a -- usuwa ostatni wskaźnik z listy. W przypadku, gdy lista jest pusta, oba operatory zwracają 0.
Również i w tym przypadku użycie szablonów daje nam istotne korzyści polegające na tym, iż zwracane wskaźniki są odpowiedniego typu. Przykładowo, sekwencja:

```
POLYLIST(textlist, char, kopiuj)
 ...
textlist li;
 ...
cout << --li;
```

spowoduje wyprowadzenie tekstu, podczas gdy przy bezpośrednim korzystaniu z list typu `anylist` niezbędny byłby dodatkowy operator rzutowania.
Usuwanie wskaźników z listy może być także przeprowadzone przy pomocy operatora -=. Wyrażenie:

```
list -= number
```

kasuje `number` ostatnich wskaźników z listy `list`. Dodatkowo, jeśli została określona funkcja kopiująca (parametr `kopia` był różny od NOCOPY), to do wszystkich usuwanych wskaźników stosowany jest operator `delete`. Operator -= zwraca referencję do `list`.
Operator = służy do kopiowania list. Ściśle mówiąc, wyrażenie:

```
list1 = list2
```

powoduje skasowanie wszystkich wskaźników listy `list1`, a następnie przyporządkowanie do zmiennej `list1` kopii listy `list2`. Jeśli funkcja kopiująca została określona, to będzie ona wykorzystana do kopiowania obiektów wskazywanych przez wskaźniki listy `list2`, a ponadto obiekty wskazywane przez pierwotne wskaźniki z `list1` będą kasowane operatorem `delete`.
Zachowywane są wszystkie intuicyjne aspekty operatora =. Przykładowo, we fragmencie programu:

```
POLYLIST(textlist,char,copy)
 ...
textlist a;
 ...
textlist b = a, c;
a = 0;
c = a = b;
```

lista `b` już w momencie zadeklarowania zostaje utworzona jako dokładna kopia listy `a`, następnie lista `a` jest czyszczona, aż wreszcie listy `c` i `a` stają się dokładnymi acz różnymi kopiami listy `b`.
Chociaż funkcja `Len` zwraca długość listy, nie powinno się jej używać do testów na to, czy lista jest pusta. Wynika to z faktu, iż `Len` po prostu czyta całą listę, zliczając jej elementy. Jednakże operator ! został zdefiniowany tak, aby wyrażenie `!list` było różne od zera tylko wtedy, gdy `list` jest puste. Tym samym w warunkach można stosować wyrażenie `!!list`, ale nie można równoważnego pod względem logicznym wyrażenia `list` (dlaczego?).
Omówimy teraz ostatni, ale najważniejszy operator list elastycznych, oznaczany symbolem |. Warto w tym miejscu przypomnieć, że oparte na teoretycznych algorytmach funkcje zawarte w bibliotekach standardowych w celu przetworzenia elementów znajdujących się wewnątrz listy wykorzystują tzw. iteratory. Aczkolwiek producenci oprogramowania starają się pisać funkcje obsługujące iteratory w sposób jak najbardziej ogólny, to jednak zawsze muszą wystąpić pewne ograniczenia wynikające z faktu, iż iteratory są dodatkowymi obiektami nie dającymi się opisać w terminach samej listy. Otóż operator | umożliwia przetwarzanie list bez pośrednictwa iteratorów, dając programiście możliwość pełnej kontroli i optymalizacji tego procesu.
Wyrażenie:

```
list | number
```

gdzie `list` jest listą, a `number` liczbą całkowitą nieujemną, definiuje się jako adres podlisty listy `list` powstałej

przez pominięcie number ostatnich (ostatnio wprowadzonych) elementów. Należy podkreślić, że operator | zwraca tylko adres, a więc żadna nowa lista nie jest tworzona.

Skoro mamy już adres, to natychmiast przychodzi na myśl operator wyłuskania. Oczywiście, zgodnie z podstawowymi regułami C, również unarny operator * nie tworzy żadnego nowego bytu. Zatem *(list|number) jest listą będącą fizyczną częścią listy list (identyczną z list w przypadku number = 0). W konsekwencji widzimy natychmiast, w jaki sposób odczytać zawartość elementu listy, za którym znajduje się number innych elementów; wystarczy ewaluować wyrażenie

```
+ * (list | number)                    (7)
```

Rzecz jasna, czas ewaluacji zależy od parametru number. Jednak podróżując po liście od elementu do elementu nie musimy za każdym razem odwoływać się do wyrażenia (7). Możemy natomiast znacznie zoptymalizować przetwarzanie, jeśli zauważymy, iż podlista podlisty jest znów podlistą całej listy, a ściśle mówiąc zachodzi następujący wzór:

```
*(list | number1) | number2 =
          list | number1 + number2
```

Wynika stąd, że optymalna pętla wyprowadzająca elementy listy na standardowe wyjście może wyglądać następująco:

```
textlist li, *a;
  ...
for(a = li|0; !! *a; a = *a|1)
                    cout << + *a ;
                                    (8)
```

Tutaj wyrażenia z operatorem | wykonywane są z maksymalną prędkością (oczywiście zamiast li|0 można by było napisać &li). Ponieważ zmiany adresu o 1 element zdarzają się bardzo często, wprowadzono specjalną funkcję Inc działającą na adresach list. Jeśli adr jest zmienną typu "wskaźnik do listy", to Inc(adr) wykonuje podstawienie:

```
adr = *adr | 1
```

oraz wyłuskuje i zwraca listę poprzednio wskazywaną przez adr. Stosując Inc możemy zapisać pętlę (8) następująco:

```
a = li|0;
while(!! *a) cout << +Inc(a);    (9)
```

Odczytywanie elementów jest proste z intuicyjnego punktu widzenia. Jednak skoro podlista jest normalną listą, to przy pomocy operatorów -- i -= można kasować elementy znajdujące się na końcu podlisty. Powstaje zatem pytanie: Co w tym przypadku stanie się z całą listą? Otóż nie ma powodów do obaw; w liście nie powstanie dziura. Po prostu można w ten sposób kasować środkowe elementy listy. Podobne uwagi dotyczą dodawania elementów do podlisty. Przykładowa instrukcja

```
*(li | k) <= "A" <= "B" <= "C"
```

spowoduje wstawienie trzech elementów do środka listy li (po elemencie "C" będzie k już przedtem istniejących elementów). Zatem lista li zostanie, obrazowo mówiąc, rozepchnięta niczym rajstopy elastyczne w celu przyjęcia nowych elementów. Ta właśnie cecha list implementowanych w pakiecie POLY uzasadnia nazwę *listy elastyczne*.

Oczywiście w tym samym wyrażeniu można usunąć pewne elementy, a na ich miejsce wstawić nowe. Taki sposób pracy jest nawet zalecany, ponieważ pozycja wprowadzenia zmian znajdowana jest tylko raz. W razie potrzeby adres odpowiedniej podlisty może być zapamiętany w zmiennej wskaźnikowej.

Warto tu zauważyć, że listy wykorzystujące pojęcie iteratora (implementowane w standardowych bibliotekach) nie są elastyczne. W przypadku list jednokierunkowych nowe elementy mogą być dodawane na tylko końcu listy, zaś dla list dwukierunkowych - na końcu lub początku.

Na podlistach list elastycznych można wykonywać naprawdę dowolne operacje. Wszystkie poniższe przykłady są poprawne

i mogą być stosowane w sposób celowy.

```
*(li|k) = 0;
// lista li będzie zawierać k
// elementów
*(li|n) = li2; // konkatenacja list
*(li|k) = *(li|n);
li <= (--*(li|3));
```

Ostatnia operacja powoduje przestawienie elementów na liście. Jeśli została określona funkcja kopiująca, to takie wyrażenie dobrze jest umieszczać wewnątrz makra REFER, jak to pokazuje poniższy przykład.

```
REFER(textlist, li <=
                  (-- *(li | 3) ) )
```

Wszystkie operatory umieszczone tym sposobem w makro REFER wykonywane są bez kopiowania i kasowania obiektów, a więc przestawienie zostanie wykonane szybciej.

Jak już stwierdziliśmy wcześniej, wyniki działania operatora | mogą być przechowywane w zmiennych wskaźnikowych. Natomiast po wyłuskaniu nie powinny być one przesyłane operatorem = do innych zmiennych listowych, ponieważ w ten sposób otrzymujemy kopie podlist (chyba, że jest to specjalnie wymagane). Niemniej jednak można napisać, przykładowo:

```
{
 textlist &b = *(li | 4);
 ...
}
```

Tutaj `b` będzie w dalszym ciągu podlistą listy `li` i wszystkie zmiany `b` zostaną odzwierciedlone w `li`.

Pętle (8) i (9) odczytywały elementy listy w porządku odwrotnym do tego, w jakim elementy te zostały do listy wstawione. Nie znaczy to jednak wcale, że listy elastyczne są tylko listami typu LIFO. Poniższy przykład wskazuje, jak należy wstawiać elementy do listy, aby takie pętle jak (8) i (9) odczytały elementy listy w tej samej kolejności, w jakiej te elementy zostały do listy wstawione.

```
textlist li, *a;
a = li|0;
 ...
a= *a <= p | 1;                (10)
 ...
```

Dla list FIFO, wyrażenie (10) powinno być stosowane zamiast zwykłego:

```
li <= p;                       (11)
```

Oczywiście to działa dlatego, że (10) powoduje zawsze wstawienie nowego elementu na początku listy. Zauważmy, iż dzięki wykorzystaniu zmiennej adresowej `a`, (10) działa niemal tak samo szybko jak (11).

Listy elastyczne wydają się być, na pierwszy rzut oka, listami jednokierunkowymi. Prawdą jest (zob. (1)), iż struktury list elastycznych zawierają tylko jeden wskaźnik `(prev)` do innej struktury listy, co rzekomo wyklucza możliwość poruszania się w dwóch kierunkach. Dlatego jedną z najciekawszych własności list elastycznych jest fakt, że to wrażenie jest fałszywe. Pokażemy bowiem za chwilę, w jaki sposób zorganizować poruszanie się w dwóch kierunkach na liście elastycznej. Ważną rolę, przynajmniej dla elegancji wyrażenia algorytmu, odgrywają tu znów szablony klas.

Poglądowo mówiąc, zastosowaną tu metodę można porównać ze strategią wytrawnego trapera, który poruszając się po prerii zostawia sobie tylko wiadome znaki ułatwiające powrót. Skoro w makro `POLYLIST` można wstawiać dowolne typy, więc nic nie stoi na przeszkodzie, aby jako argument aktualny szablonu klas wykorzystać typ, który sam został utworzony przy pomocy szablonu klas. Rzeczywiście, kompilator Borlanda przyjmuje to bez zastrzeżeń i możemy napisać przykładowe deklaracje:

```
POLYLIST(textlist, char, kopia)
POLYLIST(overlist,   textlist,
NOCOPY)
textlist li,*a;
overlist ov,*av;
```

Zatem lista `ov` może być nazwana "listą list" (określenie to brzmi groźnie, ale w rzeczywistości `ov` jest zwykłą listą wskaźników). Teraz wystarczy tylko poruszając się po liście `li`, przy użyciu pętelek takich jak (8) lub (9), zapełniać listę `ov` instrukcją `ov <= a`. To zabezpieczy nam powrót przy pomocy takich pętelek jak:

```
// w przeciwnym kierunku
for(av = ov[0; !! *av; Inc(av))
cout << +* +* av ;
// kasowanie w przeciwnym kierunku
while(!! ov) cout << + * -- ov ;
```

Należy podkreślić, że lista ov nie może być zastąpiona listą typu textlist, ale odwrotnie uporządkowaną, bowiem wtedy poruszając się do tyłu nie wiedzielibyśmy, jak wrócić na listę podstawową. Innymi słowy, zastosowane tu podejście wymaga posiadania umiejętności stosowania adresów podlist. Z drugiej strony, jest ono bardziej efektywne od klasycznych list dwukierunkowych, ponieważ nie jest potrzebna ciągła aktualizacja dwóch wskaźników (tworzenie listy list inicjujemy dopiero wtedy, gdy może być potrzebne poruszanie się w dwóch kierunkach.

5. Zakończenie

Istnieje szeroki wachlarz problemów, w których zastosowanie list wydaje się być celowe. Listy mogą zmniejszyć wykorzystanie pamięci, a jednocześnie pozwalają wyeliminować ograniczenia nakładane w wyniku beztroskiego stosowania tablic. Sądzę, że listy elastyczne z pakietu POLY tworzą bardzo efektywny i wygodny w użyciu mechanizm, który może być z powodzeniem wykorzystany w takich przypadkach.

Jerzy Skrzypek
Akademia Ekonomiczna w Krakowie, Katedra Ekonometrii

Metodologiczne aspekty modelowania procesów gospodarczych (w ujęciu Dynamiki Systemowej)

1. Wstęp

Wraz z rosnącym stopniem komplikacji systemów zarządzania nowoczesnymi firmami rysuje się wyraźnie potrzeba stosowania metod, pozwalających na opis funkcjonowania obiektów ekonomicznych w formie modeli matematycznych, a w konsekwencji na wspomaganie procesu podejmowania decyzji, dotyczących dalszej ich działalności. Wymaga to więc z jednej strony wypracowania zaplecza metodologicznego, służącego do budowy modeli, a z drugiej strony zaproponowania konkretnych technik umożliwiających rozwiązywanie problemów dotyczących podejmowania decyzji w organizacjach gospodarczych.

Można przy tym wyodrębnić kilka różnych szkół metodologicznych bazujących, w trakcie budowy modeli, na odmiennych procedurach i technikach postępowania. Do najbardziej rozpowszechnionych należą tu metody wypracowane na gruncie ekonometrii, badań operacyjnych, teorii sterowania oraz symulacji[1]. Przedstawiciele wspomnianych wyżej szkół metodologicznych wykazują wiele wspólnych poglądów dotyczących konieczności modelowania procesów gospodarczych, korzystają z tych samych teorii ekonomicznych, często stosują te same techniki matematyczne oraz języki programowania. Oprócz tego każda ze szkół posługuje się jednak własnymi teoriami dotyczącymi filozofii modelowania a w konsekwencji wyspecjalizowanymi, właściwymi danej metodzie, narzędziami.

Ponieważ oczywistym jest stwierdzenie, że każde ze wspomnianych uprzednio podejść ma silne i słabe strony, bardzo atrakcyjną - z teoretycznego punktu widzenia - wydaje się więc być propozycja wykorzystania zalet każdego z wymienionych uprzednio podejść, w celu stworzenia zaplecza teoretycznego służącego do budowy oraz optymalizacji modeli procesów gospodarczych.

Analizując potencjalne możliwości każdego ze wspomnianych podejść metodologicznych można stwierdzić, że najbardziej elastyczną i podatną na zapożyczenia z innych dziedzin jest metoda Dynamiki Systemowej. Ponadto obiecująco wygląda możliwość wzbogacenia tradycyjnego podejścia metodologicznego o możliwość optymalizacji (metoda Relativity Dynamics), proponowaną przez Keloharju [5].

Koncepcja Keloharju bazuje na założeniu, że przynajmniej niektóre etapy postępowania w trakcie procesu modelowania można zautomatyzować poprzez połączenie symulacji z heurystyczną metodą optymalizacji. Warto podkreślić, że stosowany tu algorytm optymalizacyjny (SDR) nie stwarza żadnych ograniczeń dotyczących posta-

[1] Różnice w paradygmatach rządzących ekonometrią, Dynamiką Systemową oraz teorią sterowania jest przedmiotem rozważań w pracach [5], (Appel, Fassing, 1978) oraz (Bell, Bell, 1980) i (Meadows, 1980). [*Prace wymienione w nawiasach () nie występują w bibliografii. Przyp. red.*]

ci analitycznej funkcji kryterium oraz równań modelu. Należy się jednak przy tym liczyć z faktem, że tak jak zresztą większość algorytmów heurystycznych, nie gwarantuje on uzyskiwania rozwiązań optymalnych w sensie optimum globalnego. Nic nie stoi jednak na przeszkodzie zastosowania oprogramowania, z innym, nowocześniejszym algorytmem optymalizacji[2].

W związku z powyższym, problemy metodologiczne, powstające w trakcie modelowania procesów gospodarczych w konwencji Dynamiki Systemowej, ze szczególnym uwzględnieniem zadań, które należy wykonać, aby w rezultacie otrzymać symulacyjny model, umożliwiający optymalizację decyzji, stanowią zasadniczą część niniejszej pracy.

2. Proces modelowania zjawisk gospodarczych

Przystępując do budowy modeli procesów gospodarczych należy wziąć pod uwagę fakt, że sytuacja badacza procesów gospodarczych jest często dużo trudniejsza niż przedstawiciela nauk ścisłych, który zwykle ma do czynienia z obiektywnymi prawami. Prawa te stosunkowo łatwo można opisać w formie równań modelu matematycznego. Ekonomista natomiast, często napotyka na niejednolitą i niekompletną teorię, a nie istnieje przecież laboratorium, w którym mógłby wykonywać eksperymenty na badanym obiekcie. Ponadto struktura procesów gospodarczych jest zwykle zmienna w czasie, co wymaga odpowiednich - bieżąco dokonywanych w trakcie badania - zmian w równaniach modelu.

Kolejny problem stwarzają dane empiryczne, często niekompletne, natomiast obarczone trudnymi do oszacowania błędami pomiaru. Pomijając już kłopoty związane z estymacją i weryfikacją modeli na podstawie danych empirycznych, należy zasygnalizować problemy związane z zastosowaniem modeli procesów gospodarczych do poszukiwania rozwiązań optymalnych. Kłopoty może tu sprawiać konieczność sformułowania a następnie sformalizowania celów działania, ustalenia wag przypisanych realizacji poszczególnych celów, a także dobór odpowiednich instrumentów ekonomicznych i zastosowanie ich we właściwy sposób.

Nasuwa się więc pytanie: czy procesy gospodarcze poddają się modelowaniu i optymalizacji? Pozytywna odpowiedź na tak postawione pytanie musi być poprzedzona próbą sformułowania zaplecza metodologicznego służącego do budowy oraz aplikacji modeli procesów gospodarczych. Zdaniem autora niniejszej pracy, dobrym punktem wyjścia jest tu stosowanie Dynamiki Systemowej.

Metoda ta będzie w dalszych rozważaniach stosowana niejako w podwójnej roli. Z jednej strony ma ona stanowić metodę opisu badanego systemu - ze szczególnym uwzględnieniem problemów związanych z modelowaniem na szczeblu mikroekonomicznym - a z drugiej strony będzie wykorzystywana jako technika umożliwiająca przeprowadzenie eksperymentów symulacyjnych i optymalizacyjnych. Na poparcie celów i zadań stawianych metodzie Dynamiki Systemowej można przytoczyć jej następującą definicję: "jest to ścisła metoda opisu systemu, ułatwiająca analizę - zwykle poprzez użycie ciągłych modeli symulacyjnych - efektów oddziaływania alternatywnych struktur i polityk decyzyjnych na funkcjonowanie badanego systemu" (Wolstenholme, 1982, str. 382).

Tradycyjne podejście metodologiczne proponowane na gruncie Dynamiki Systemowej obejmuje przynajmniej dwie fazy (Wolstenholme, 1982):

a) fazę opisu badanego obiektu wraz z analizą jakościową i weryfikacją;

[2] Autor niniejszej pracy zebrał już pozytywne doświadczenia, przy zastosowaniu algorytmu MINUIT z biblioteki CERN [11]

b) fazę analizy ilościowej alternatywnych sposobów postępowania.

Autor niniejszej pracy postanowił zaproponować nieco inny sposób postępowania. Mianowicie proces budowy modelu oraz jego aplikacji postanowiono podzielić na trzy fazy:

a) fazę opisu badanego obiektu wraz z analizą jakościową;
b) fazę testowania wiarygodności modelu *(credibility testing)*;
c) fazę analizy ilościowej modelu przy wykorzystaniu symulacji komputerowej.

Warto przy tym zwrócić uwagę na fakt, że proces budowy modeli symulacyjnych nosi w sobie znamiona swego rodzaju "sztuki". Wszelkie próby usystematyzowania kolejnych czynności, które należy wykonać aby zbudować model nadający się do zastosowań praktycznych, mogą prowadzić na manowce. Wymienione uprzednio fazy modelowania przenikają się nawzajem, a więc ścisłe ich wyspecyfikowanie ma służyć bardziej zaznaczeniu konieczności ich wykonania niż ustaleniu kolejności ich występowania. Przykładowo testowanie wiarygodności modelu jest procesem ciągłym, przy czym niektóre z testów weryfikacyjnych wykonywane są na etapie konstrukcji modelu, inne po jego zbudowaniu a jeszcze inne w fazie aplikacji.

Faza "A".	Opis badanego obiektu wraz z analizą jakościową
Zadanie A1:	Definicja celów modelowania
Zadanie A2:	Tworzenie koncepcji modelu
Zadanie A3:	Konstrukcja modelu graficznego
Zadanie A4:	Analiza jakościowa modelu graficznego
Zadanie A5:	Ustalenie wartości początkowych zmiennych i parametrów modelu
Zadanie A6:	Konstrukcja modelu matematycznego
Zadanie A7:	Zapisanie modelu komputerowego
Faza "B".	Testowanie wiarygodności modelu
Zadanie B1:	Przeprowadzenie wybranych testów weryfikacyjnych *(verification)*
Zadanie B2:	Testowanie zgodności zachowania modelu z zachowaniem modelowanego obiektu *(validation)*
Zadanie B3:	Testowanie zgodności struktur modelu ze strukturą rzeczywistego obiektu *(legitimation)*
Faza "C".	Analiza ilościowa modelu przy wykorzystaniu symulacji komputerowej
Zadanie C1:	Analiza i synteza modelu na drodze intuicyjnych eksperymentów symulacyjnych
Zadanie C2:	Zastosowanie algorytmów sterowania optymalnego opracowanych na gruncie teorii sterowania
Zadanie C3:	Stosowanie heurystycznych algorytmów optymalizacyjnych
Uwaga:	Zadania pozostające do wykonania w fazie "C" należy traktować alternatywnie

TABLICA 1.1. Fazy modelowania procesów gospodarczych
Źródło: opracowanie własne

3. Faza opisu badanego obiektu wraz z analizą jakościową

3.1. Definicja celów modelowania

Podstawowym celem modelowania jest zawsze studiowanie zachowania się modelowanego obiektu w trakcie eksperymentów symulacyjnych. Różny może być jednak sposób wykorzystywania modelu. Model może przecież służyć jako (Wahlstrom, 1986) środek komunikacji, ułatwiający prezentację i zrozumienie badanego obiektu lub stanowić narzędzie analizy, swego rodzaju laboratorium badawcze ekonomisty a także służyć wyłącznie celom dydaktycznym.

W zależności od funkcji jakie ma więc spełniać model sensowne może być użycie mniej lub bardziej szczegółowego modelu. Niestety, w badaniach ekonomicznych trudno jest stworzyć obiektywną miarę niezbędnej dokładności modelu, jaką na przykład w fizyce stanowi dokładność urządzeń pomiarowych. Właściwe określenie celów budowy modelu i funkcji jakie ma on spełniać jest więc jednym z krytycznych punktów w procesie modelowania. Rozstrzyga ono bowiem między innymi o stopniu szczegółowości modelu a także przesądza o stosowanych sposobach testowania jego wiarygodności.

3.2. Tworzenie koncepcji modelu

Koncepcja modelu powstaje zwykle na drodze identyfikacji badanego procesu, rozumianej jako określenie obiektu identyfikacji, wyodrębnienie jego podstawowych cech oraz ustalenie związków przyczynowo - skutkowych pomiędzy nimi.

Jeśli model budowany jest z myślą o poddaniu go eksperymentom optymalizacyjnym to już w trakcie tworzenia jego koncepcji należy wziąć pod uwagę konieczność wyodrębnienia reguł decyzyjnych oraz sformułowania funkcji kryterium optymalizacji. Model taki powinien więc reprezentować nie tylko " fizykę" badanego obiektu ale również sposób podejmowania decyzji. W przypadku modelu przedsiębiorstwa reguły te mogą dotyczyć na przykład określania planów produkcji, ustalania cen produktów lub zapotrzebowania na materiały i surowce.

O ile jednak odwzorowanie relacji fizycznych jest stosunkowo łatwe, to modelowanie procesu podejmowania decyzji nastręcza już poważne trudności. Problem ten rozwiązywany jest zwykle na drodze zastosowanie jednego z dwu następujących sposobów. Pierwszy z nich, tradycyjnie stosowany w modelach Dynamiki Systemowej, polega na włączeniu do modelu reguł decyzyjnych opisujących explicite informacje brane pod uwagę przy podejmowaniu określonych decyzji. W tego typach regułach znaczny wysiłek poświęcany jest na wyspecyfikowanie zmiennych, objaśniających - zdaniem modelującego - sposób ustalania wartości zmiennych decyzyjnych. Tym samym, jak zauważa Wąsik ([22], str. 4) "badacz zostaje zobowiązany do odzwierciedlenia w modelu także i takich składników systemu, które są z natury słabo zdefiniowane i tylko intuicyjnie pojmowane przez samych decydentów".

Alternatywny sposób postępowania został rozwinięty na gruncie ekonomii eksperymentalnej. W modelach tego typu tradycyjne reguły decyzyjne zostały zastąpione przez rzeczywistych decydentów, których zadaniem jest nadanie wartości wszystkim zmiennym decyzyjnym. Sposób podejmowania przez nich decyzji jest następnie obserwowany i analizowany[3].

Pozostawiając do wyboru modelującego, sposób postępowania przy konstrukcji tych fragmentów modelu, które dotyczą procesu decyzyjnego, należy stwierdzić, że z pun-

[3] Modele tego typu zostały zapoczątkowane w pracach Smith'a (1979, 1982a) oraz Plott'a (1982). Również modele KRAEK-1 oraz MSP, opisane w rozdziale trzecim niniejszej pracy, zostały zbudowane zgodnie z powyższymi zasadami

ktu widzenia możliwości optymalizacji, oba wymienione uprzednio podejścia metodologiczne są równoprawne. Oznacza to, że w przypadku wyposażenia modelu w reguły decyzyjne skonstruowane w tradycyjny sposób, optymalizacji poddane mogą być jedynie parametry wchodzące w ich skład.

Jeśli budowany model ma służyć do przeprowadzania eksperymentów optymalizacyjnych, to już w fazie koncepcyjnej należy wziąć pod uwagę konieczność zaproponowania postaci funkcji kryterium optymalizacji[4].

3.3. Konstrukcja modelu graficznego

Koncepcja modelu, wypracowana w poprzednim kroku, zostaje następnie przetworzona w model graficzny. Korzyści płynące z przedstawienia strukturalnej postaci modelu w formie graficznej doceniają w zasadzie wszyscy autorzy budujący swe modele w konwencji Dynamiki Systemowej. Przeważa przy tym pogląd, że rola schematu graficznego sprowadza się w zasadzie do funkcji komunikacyjnych, to znaczy, że dzięki niemu można w łatwy sposób prezentować i objaśniać cechy konstrukcyjne modeli symulacyjnych. Wydaje się jednak, że schematy graficzne mają kapitalne znaczenie w fazie konstrukcji i weryfikacji modeli. Przede wszystkim pozwalają one lepiej poznać zależności strukturalne modelu, a przez to stanowią inspirację do zmian struktury modeli. Umożliwiają one również weryfikację podstawowych założeń dotyczących struktury modelu. Na koniec poprawnie zbudowany schemat graficzny (to znaczy zgodnie z zasadami Dynamiki Systemowej) umożliwia w zasadzie automatyczne przejście do modelu matematycznego.

Diagramom graficznym stawiane są liczne zarzuty. przede wszystkim dotyczą one niepełnej odpowiedniości pomiędzy schematami graficznymi a modelami matematycznymi. Schematy graficzne nie pokazują bowiem precyzyjnie wszystkich zależności matematycznych, zwykle nie pokazują też wartości początkowych zmiennych i parametrów modelu. Zarzuty te mogłyby mieć jednak rację bytu w przypadku gdyby diagramy przepływów były jedyną formą prezentacji modeli. Są one jednak przecież tylko elementem składowym opisu modelu.

Na gruncie Dynamiki Systemowej stosowane są bardzo różnorodne typy schematów graficznych (począwszy od forresterowskich flow graphs), oparte w głównej mierze na zasadach wypracowanych na gruncie teorii sterowania oraz teorii grafów[5]. Istnieje w związku z tym konieczność wyboru takiego typu prezentacji graficznej, która byłaby wygodna również w przypadku modeli optymalizacyjnych.

Ponieważ diagramy przepływów zaproponowane przez Forrestera (1961) są niewygodne, szczególnie w przypadku modeli o dużych rozmiarach powstała konieczność zastąpienia ich przez inną formułę. Spośród bardzo wielu propozycji, zdecydowano się na zastosowanie w niniejszej pracy, diagramów zależności zaproponowanych przez Coyle'a [4] oraz Wolstenholme'a (1982), a w niektórych szczególnych przypadkach diagramów opcjonalnych *(option diagrams)* lansowanych przez Keloharju [5].

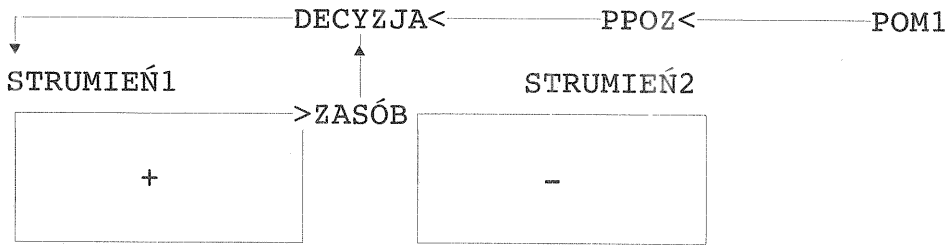
Załóżmy przez chwilę, że mamy skonstruować model graficzny dla pewnej, bardzo uproszczonej, sytuacji decyzyjnej. Sytuacja ta jest opisana przy pomocy następującego modelu werbalnego: zmienna decyzyjna (DECYZJA) oddziałuje na stan zmiennej ZASÓB poprzez zmiany natężenia zmiennej STRUMIEŃ 1. Wartość zmiennej decyzyjnej ustalana jest na podstawie informacji o aktualnej wartości zmiennej

[4] Sposoby konstrukcji funkcji kryterium w przypadku modeli procesów gospodarczych zostaną omówione w dalszej części pracy

[5] Stosunkowo najszerszy przegląd typów modeli graficznych, stosowanych na gruncie Dynamiki Systemowej zawiera praca (Morecroft, 1982)

ZASÓB oraz wartości pożądanego poziomu danej zmiennej (PPOZ), który z kolei definiowany jest przy pomocy dwóch zmiennych pomocniczych POM1 i POM2.

Odpowiedni model graficzny, skonstruowany w konwencji Wolstenholme'a-Coyle'a przedstawiono na rysunku 1.1.



RYS. 1.1.Konstrukcja schematu graficznego według koncepcji Wolstenholme'a-Coyle'a

Reguły rządzące diagramami tego typu sprowadzają się do następujących zasad:

a) przepływy materialne zaznaczane są przy pomocy linii ciągłych;
b) przepływy informacyjne zaznaczane są przy pomocy linii przerywanych;
c) kierunek pętli sprzężenia zwrotnego zaznaczany jest przy pomocy znaku + lub - .

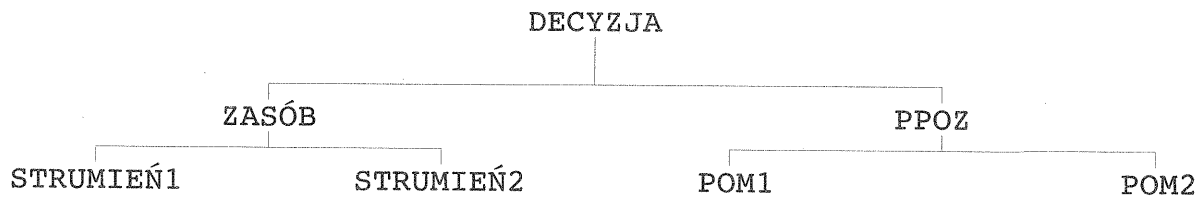
Diagramy Wolstenholme'a-Coyle'a nadają się świetnie do prezentacji koncepcji poszczególnych sektorów modelu. Natomiast w przypadku gdy model ma służyć do badania alternatywnych struktur reguł decyzyjnych, to niezwykle przydatne mogą się okazać diagramy opcjonalne *(option diagrams)*. Diagram opcjonalny opisujący wspomnianą uprzednio sytuację decyzyjną przedstawiono na rysunku 1.2.

Diagramy te prezentują więc nieco inne spojrzenie na model, eksponując sytuację decyzyjną oraz hierarchiczność poszczególnych zmiennych. Są one szczególnie przydatne do prezentowania konstrukcji reguł decyzyjnych oraz tych fragmentów modelu, w których testowane będą alternatywne rozwiązania strukturalne.

3.4. Analiza jakościowa modelu graficznego

Schemat graficzny należy poddać następnie analizie jakościowej, a więc swego rodzaju weryfikacji. Powinno się tu wykorzystywać na szeroką skalę osiągnięcia teorii sterowania w zakresie analizy modeli graficznych.

Jednym z rozwiązań może tu być zastosowanie systematycznego sposobu analizy pętli sprzężeń zwrotnych oraz projektowanych polityk decyzyjnych, zaproponowanego przez Coyle'a [1]. Nie jest to jednak sformalizowana procedura, polega ona bowiem na obserwacji zachowania się efektu działania modeli i wyjaśniania go poprzez zastosowanie pewnych ogólnych reguł działania pętli sprzężeń zwrotnych (reguła kciuka).Mimo to dostarcza jednak pewnych ogólnych wskazówek dotyczą-



RYS. 1.2. Konstrukcja diagramu opcjonalnego

cych analizy modeli graficznych. Przede wszystkim jak zauważa Coyle [2]:

1) nie powiązane ze sobą pętle sprzężeń zwrotnych, to znaczy na przykład takie w których zmienne definiowane w pętli numer 1 są jednocześnie zmiennymi objaśniającymi dla zmiennych objaśnianych w pętli numer 2, a związek o przeciwnym zwrocie nie istnieje, powodują słabą sterowalność modelu,

2) przyczyną niesterowalności modelu mogą być również te ze zmiennych typu strumień, które nie są objaśniane przy pomocy przynajmniej jednej zmiennej decyzyjnej.

Analiza jakościowa modelu graficznego może również stać się bardzo użytecznym narzędziem analizy wiarygodności modeli symulacyjnych. Pozwala ona bowiem weryfikować, już na tym wczesnym etapie modelowania, charakter związków pomiędzy zmiennymi modelu.

3.5. Ustalanie warunków początkowych

Ustalenie wartości początkowych dla zmiennych i parametrów modelu jest w zasadzie warunkiem niezbędnym do uruchomienia modelu. Sposób ich doboru zależy natomiast od tego czy budowany model będzie pracował na danych rzeczywistych czy na hipotetycznych. W pierwszym przypadku powstaje problem oszacowania wartości parametrów modelu, który może być rozwiązany na drodze zastosowania, któregoś z algorytmów estymacji opracowanych na gruncie ekonometrii lub teorii sterowania[6]. Warto również wspomnieć o możliwości takiego sformułowania zadania sterowania optymalnego, aby w rezultacie można było znaleźć wartości parametrów modelu.

[6] Godną polecenia wydaje się tu być metoda filtracji kalmanowskiej [11]

3.6. Konstrukcja modelu matematycznego i komputerowego

Problemy konstrukcji modelu matematycznego i komputerowego postanowiono połączyć z kilku względów. Przede wszystkim istnieją już liczne języki symulacyjne[7], będące jednocześnie językami publikacyjnymi, pozwalające przez to na bezpośrednie przejście od schematu graficznego do modelu komputerowego z pominięciem modelu matematycznego. Unika się w ten sposób dodatkowego źródła bardzo trudnych do wykrycia błędów, powstających w trakcie przekształcania modelu matematycznego w model komputerowy. Problem weryfikacji dotyczy więc wyłącznie modelu graficznego i komputerowego. Natomiast korzystanie z języków ogólnego przeznaczenia wymaga przeprowadzania specjalnych testów weryfikujących zgodność modelu matematycznego z komputerowym. Nie jest to zadanie łatwe, jeśli weźmiemy pod uwagę chociażby problem ustalania kolejności równań w programie komputerowym. Jeśli model zbudowany został zgodnie z zasadami Dynamiki Systemowej i korzystamy z języka symulacyjnego to możemy być pewni, że równania dadzą się posortować we właściwej kolejności (Lyneis, 1980).

Modele budowane w konwencji Dynamiki Systemowej zapisywane są zwykle przy pomocy notacji zaproponowanej przez Forrester'a[8], która swe odbicie znalazła w konstrukcji języków symulacyjnych typu DYNAMO czy DYSMAP. Sposób ten, mimo swej przejrzystości, jest niewygodny w przypadku modeli o dużych rozmiarach, w których można by z powodzeniem zastosować zapis macierzowy, którego jednak wspomniana notacja nie dopuszcza.

Alternatywny sposób polega na wykorzy-

[7] Na przykład DYNAMO, DYSMAP czy powstający w Akademii Ekonomicznej w Krakowie MDL (Model Description Language, Bocheński, 1988)

[8] Por. praca: (Forrester, 1961)

staniu konwencji równania stanu, która jest podstawowym sposobem opisu modeli konstruowanych na gruncie teorii sterowania. Nie istnieją przy tym żadne techniczne problemy przejścia od jednej konwencji do drugiej[9]. Niestety, zapis modeli skonstruowanych w konwencji Dynamiki Systemowej, w postaci równania stanu wymaga zredukowania modelu wyłącznie do równań opisujących zasoby, poprzez włączenie do nich równań opisujących pozostałe typy zmiennych.

W związku z tym podjęto próbę stworzenia teoriomnogościowej koncepcji modeli Dynamiko-Systemowych[10], wolnej od wyżej wspomnianych wad, a jednocześnie dostosowanej do wymogów Dynamiki Systemowej. Opracowany, przy okazji, sposób notacji MDS umożliwia zapis modeli o strukturze zmiennej w czasie, zapewnia obliczalność modelu a w przypadku liniowym pozwala na stosowanie zapisu macierzowego. Nie ma przy tym żadnych przeszkód przy przechodzeniu do notacji przestrzenio-stanowej, zatem w łatwy sposób, można korzystać (o ile to możliwe ze względu na typ modelu) z algorytmów sterowania optymalnego opracowanych na gruncie teorii sterowania.

W modelach skonstruowanych w konwencji Dynamiki Systemowej (MDS) wyróżnia się przynajmniej trzy podstawowe typy zmiennych: zasoby, strumienie oraz zmienne pomocnicze[11].

Zasoby są to zmienne akumulujące media przepływające wewnątrz systemu. Opisują one stan modelowanego obiektu i w związku z tym ich wartości można obserwować również wtedy gdy natężenie zmiennych typu strumień przyjmuje wartości zerowe.

W przyjętej notacji, każdy ze strumieni musi mieć swe źródło oraz akumulator (zasób akumulacyjny). Powyższa zasada nie zawsze jest stosowana *explicite* w MDS. Szczególnie dotyczy to strumieni, których źródło bądź akumulator znajdują się w otoczeniu modelu. W takich przypadkach odpowiednie zasoby nie są definiowane, a jedynie zaznaczane jest ich obecność przy pomocy umownych symboli graficznych.

Zmienne typu zasób zostaną zatem określone jako:

zasoby:

$$z_i: TIME \rightarrow \mathbb{R}, \quad i=0,1,2,\ldots,n,k;$$

przy czym n oznacza ilość rozważanych zasobów, nie należących jednak do otoczenia.

Zmienne typu strumień opisują natężenie przepływów mediów pomiędzy zasobami. Generalnie rzecz biorąc, określają więc one sposób w jaki informacje o stanach zasobów przetwarzane są na decyzje.
Zmienne typu strumień są definiowane jako:

strumienie:

$$s(i_1,i_2): TIME \rightarrow \mathbb{R}, \quad i_1 \in \{0,1,2,\ldots,n\} \quad i_2 \in \{1,2,\ldots,n,k\}$$

gdzie i_1 oraz i_2 oznaczają odpowiednio: numer zasobu, dla którego $s(i_1,i_2)$ jest strumieniem wyjściowym, oraz numer zasobu, dla którego jest on strumieniem wejściowym.

Równania opisujące natężenie zmiennych typu strumień mogą być niekiedy bardzo złożonymi wyrażeniami. Aby móc je zapisać w postaci prostych równań, poprawiając tym samym czytelność modelu, używa się zmiennych pomocniczych. Zmienne te pokazują więc te elementy, które składają

[9] Por. np. [21]

[10] Por. praca (Skrzypek, Tatar, 1989)

[11] Szerszą charakterystykę podstawowych typów zmiennych występujących w modelach Dynamiki Systemowej można znaleźć np. w pracach: (Forrester, 1961), (Coyle, 1978), [19]

się na reguły decyzyjne, określane przy pomocy odpowiadających im strumieni.

Niech więc:

zmienne pomocnicze:

$$a_j : TIME \rightarrow \mathbb{R}, \quad j = 1,2,\ldots,m;$$

gdzie m oznacza liczbę zmiennych pomocniczych.

Zdefiniowane uprzednio trzy podstawowe typy zmiennych wystarczają do opisu powiązań strukturalnych dowolnego modelu, zbudowanego w konwencji Dynamiki Systemowej.

Związki modelu z otoczeniem opisywane są przy pomocy zmiennych egzogenicznych. Są to zmienne, które nie będąc objaśniane przez jakąkolwiek inną zmienną określoną w modelu, mają jednak istotny wpływ na niektóre zmienne objaśniane.

Niech więc:

zmienne egzogeniczne:

$$e_l : TIME \rightarrow \mathbb{R}, \quad l = 1,2,\ldots,p;$$

gdzie p oznacza liczbę zmiennych egzogenicznych.

Te ze zmiennych egzogenicznych, których wartości mogą być przedmiotem wyboru decydentów, będą dalej nazywane zmiennymi decyzyjnymi. Nie jest przy tym istotne, przynajmniej na obecnym etapie rozważań, czy wartości te będą znalezione przy pomocy komputera, ekspertów lub może zgadnięte. Warto jednak przy tym zasygnalizować, że niektóre ze zmiennych decyzyjnych, mogą nabierać charakteru zmiennych egzogenicznych przy przejściu na inny szczebel w strukturze modelu i *vice versa*. Przykład mogą tu stanowić decyzje strategiczne, zastrzeżone dla dyrekcji przedsiębiorstwa. Z punktu widzenia niższego personelu mają one bowiem charakter zmiennych egzogenicznych.

Bez straty ogólności rozważań można, dla wygody, przyjąć, że spośród p zmiennych egzogenicznych q pierwszych jest zmiennymi decyzyjnymi. Z uwagi na szczególne znaczenie zmiennych decyzyjnych, w modelu rezerwowany jest dla nich osobny symbol d_l.

Zatem:

$$d_l \overset{ozn.}{=} e_l : TIME \rightarrow \mathbb{R}, \quad \text{dla } l=1,2,\ldots,q,$$

gdzie oczywiście: $0 \leq q \leq p$.

Warto przy tym wyraźnie podkreślić, że przez d_l oznaczane są przede wszystkim decyzje strumieniowe oraz te z decyzji strukturalnych, które mogą być realizowane poprzez zmiany wartości parametrów modelu[12].

Oprócz zmiennych w modelu mogą występować również parametry, czyli te z elementów modelu, których wartości są stałe w czasie trwania eksperymentu symulacyjnego. Będą one oznaczane przez c_h, gdzie $c_h \in \mathbb{R}$, $(h=1,2,3,\ldots,w)$, a w oznacza liczbę parametrów modelu.

Zbiór
$\mathbb{C} = \{c_h \in \mathbb{R} : h=1,2,3,\ldots,w\}$,
nazywany będzie zbiorem parametrów modelu. Warto przy tym zwrócić uwagę na fakt, że w MDS często mianem parametrów określa się również te elementy modelu, których wartości zmieniane są w trakcie eksperymentów symulacyjnych (tzw. parametry aktywne). W myśl notacji, przyjętej na użytek niniejszej pracy, muszą one być jednak traktowane jako zmienne decyzyjne. Wobec powyższych ustaleń, mamy do czynienia z wektorami odpowie-

[12] Definicje decyzji strukturalnych i strumieniowych można znaleźć w pracy [18]

dniego typu zmiennych:

$$Z = (z_0, z_1, \ldots, z_n, z_k) : TIME \rightarrow \mathbb{R}^{n+2} \quad (1)$$

$$A = (a_1, a_2, \ldots, a_m) : TIME \rightarrow \mathbb{R}^m \quad (2)$$

$$E = (e_1, e_2, \ldots, e_p) : TIME \rightarrow \mathbb{R}^p \quad (3)$$

oraz z kwadratową macierzą zmiennych typu strumień o postaci:

c) natężenie dowolnego strumienia w każdym czasie t (a więc zgodnie z przyjętymi ustaleniami także w przedziale $<t-1, t>$) może zależeć od wartości zasobów, zmiennych pomocniczych, decyzyjnych i egzogenicznych w dowolnym momencie $\bar{t} \leq t-1$.

Dla zadośćuczynienia dalszym wymaganiom, jakie na gruncie Dynamiki Systemowej, przyjmuje się w stosunku do związków pomiędzy poszczególnymi typami zmiennych, założono, że każda funkcja $f^{i,t}$ jest liniowa ze względu na zasoby oraz wyrażenia postaci. Oznacza to, że wartość natężenia strumienia może być

$$\hat{S} = \begin{bmatrix} s(0,0), & s(0,1), & s(0,2), & \ldots & ,s(0,n), & s(0,k) \\ s(1,0), & s(1,1), & s(1,2), & \ldots & ,s(1,n), & s(1,k) \\ \cdot & \cdot & \cdot & \ldots & \cdot & \cdot \\ s(n,0), & s(n,1), & s(n,2), & \ldots & ,s(n,n), & s(n,k) \\ s(k,0), & s(k,1), & s(k,2), & \ldots & ,s(k,n), & s(k,k) \end{bmatrix} : TIME \rightarrow \mathbb{R}^{n+2} \times \mathbb{R}^{n+2} \quad (4)$$

W ramach wektora E można oczywiście wyodrębnić podwektor zmiennych decyzyjnych:

$$D = (d_1, d_2, \ldots, d_q) \overset{ozn.}{=} (e_1, e_2, \ldots, e_q) : TIME \rightarrow \mathbb{R}^q \quad (4)$$

Przyjęcie tak określonych funkcji oznacza, że:

a) wartość każdego zasobu, w dowolnej chwili czasowej t, zależy tylko od jego wielkości w chwili poprzedniej $(t-1)$ oraz od wartości strumieni wypływających oraz wpływających do niego w czasie $(t-1, t)$;
b) wartość każdej zmiennej pomocniczej w dowolnym momencie czasowym może być funkcją wyłącznie wielkości zasobów oraz wartości zmiennych pomocniczych obliczonych w chwili $\bar{t} \leq t$, przy czym - wobec określenia 2.10. - nie może ona zależeć od samej siebie w rozważanym czasie;

w szczególności:
- decyzją podjętą w dowolnym (wcześniejszym) momencie czasowym,
- wartością zmiennej egzogenicznej (nie będącej jednak zmienną decyzyjną) określoną w dowolnym momencie czasowym (wcześniejszym)
- funkcją zmiennych typu: zasób, pomocnicza i decyzyjna,
- lub funkcją zmiennych typu zasób i pomocnicza.

Powyższe założenie oznacza, że wyklucza się możliwość istnienia dwóch zmiennych pomocniczych, których wartości obliczane w tym samym momencie czasowym t będą od siebie wzajemnie zależeć bezpośrednio lub pośrednio, poprzez inne zmienne pomocnicze.

Model matematyczny lub komputerowy należy następnie poddać procedurze sprawdzania sterowalności, obserwowalności i stabilności. Dysponując modelem matematycznym lub komputero-

wym warto się bowiem upewnić (*explicite*, bez jego rozwiązania) czy zapewnia on możliwość osiągnięcia założonych celów. Odpowiedź na tak postawione pytanie można uzyskać na drodze badania sterowalności modelu[13]. Z kolei dla systemów dynamicznych interesującym wydaje się być pytanie dotyczące możliwości utrzymania na stabilnym poziomie raz osiągniętych celów, czy też nadążania przy pomocy stabilnych instrumentów za zmieniającymi się celami. Warto więc również badać stabilność modeli. Trzecią z pożądanych cech modeli jest ich obserwowalność. Cecha ta dotyczy możliwości znalezienia wartości zmiennych nie obserwowanych bezpośrednio.

Własności sterowalności, obserwowalności i stabilności są nawzajem ze sobą powiązane. Aoki (1976) pokazuje, że jeśli model jest sterowalny i obserwowalny, to przy spełnieniu, pewnych, wyłącznie technicznych warunków, można znaleźć stabilne rozwiązania zadania sterowania optymalnego.

Sposoby badania wymienionych uprzednio własności modeli zostały szczegółowo opracowane na gruncie teorii sterowania (Neck, 1984). Wymagają one jednak sprowadzenia modeli Dynamiki Systemowej do postaci równań stanu [21], a następnie wykonania serii rachunków macierzowych. Jedyną wadą wspomnianego sposobu postępowania jest fakt, że dla systemów nieliniowych nie da się w ścisły sposób sformułować warunków koniecznych i wystarczających sterowalności, obserwowalności i stabilności modelu. Alternatywnym sposobem postępowania może być badanie wrażliwości funkcji kryterium optymalizacji na zmiany wartości parametrów. Jeśli okaże się, że funkcja kryterium nie jest wrażliwa na duże zmiany wartości parametrów to oznacza, że dany człon nie ma na nią wpływu a więc jest niesterowalny ze względu na użyte instrumenty.

4. Faza testowania wiarygodności modeli symulacyjnych

Faza testowania wiarygodności modelu ma szczególnie duże znaczenie w przypadku modeli o nastawieniu aplikacyjnym. Bez zapewnienia bowiem odpowiedniego poziomu zaufania do modelu nie ma co marzyć o możliwości wnioskowania na jego podstawie.

Proces testowania wiarygodności modelu jest procesem ciągłym, wykonywanym zarówno w procesie budowy modelu jak i jego eksploatacji. Dodatkowe komplikacje spowodowane są brakiem zgodności większości autorów co do rozumienia powyższego procesu jak i sposobu jego przeprowadzenia[14].

Przede wszystkim wiarygodność *(credibility)* modeli symulacyjnych często mierzona jest stopniem zaufania osób użytkujących dany model do informacji uzyskanych przy jego pomocy (OSSIM, 1977). Oznacza to, że nie można w ogóle poddać ocenie wiarygodności modelu, który nie znalazł klientów. Podobny zarzut można by skierować pod adresem Maisel'a (1976), który twierdzi, że symulacja byłaby wiarygodna, gdyby menadżerowie (użytkownicy modelu symulacyjnego), opierając się na jej wynikach, podejmowali te same działania, co i w przypadku opierania się na danych pochodzących z obserwacji rzeczywistego systemu. Zapewne chodzi tu o krytyczne spojrzenie na model, które nie jest zwykle właściwe autorom modeli, zafascynowanym swoimi modelami.

Autor niniejszej pracy uważa jednak, że

[13] Sterowalność rozumiana jest tu (w stosunku do systemu deterministycznego, liniowego) jako możliwość przeprowadzenia systemu ze stanu Z_o w momencie t_o do dowolnego pożądanego stanu Z_* w skończonym czasie (Aoki, 1976).

[14] Przegląd poglądów dotyczących weryfikacji modeli symulacyjnych można znaleźć w pracach: (Rzońca, 1981), (Maciaszek, 1988) oraz (Kotva, 1988)

testowanie wiarygodności modelu powinno przebiegać w ten sam sposób bez względu na to czy model jest przeznaczony do prezentacji "klientom" czy nie. Innymi słowy badacz, oceniając wiarygodność swego modelu jest niejako klientem samego siebie. Natomiast powodzenie modelu u "klientów" związane jest z jego innymi cechami. Jak twierdzi Randers (1980) model przeznaczony do sprzedaży musi mieć tak dobrane założenia wejściowe, aby były one łatwo zrozumiane i zgodne z wyobrażeniami klientów *(descriptive realism)*. Ponadto wynikiem eksperymentów symulacyjnych powinny być rozwiązania, których nie można by osiągnąć w inny sposób[15].

W związku z tym w dalszych rozważaniach, ogół czynności zmierzających do uzyskania zaufania do modelu, będzie określany mianem testowania jego wiarygodności. Termin ten jest przy tym bardziej zgodny z duchem języka polskiego niż "testowanie ogólne" *(general testing)*, proponowane przez Randersa (1980).

Ponadto w związku z różnicami poglądów co do sposobu podejścia do procesu testowania wiarygodności modeli[16], postanowiono przyjąć (po uprzednim zmodyfikowaniu) propozycję Coyle'a (1984). W myśl tej propozycji faza testowania wiarygodności modelu będzie realizowana poprzez następujące czynności:

a) weryfikację stwierdzenia, czy model właściwie odwzorowuje mechanizm rządzący modelowanym obiektem *(verification)*;
b) testowanie adekwatności zachowania się modelu w stosunku do obiektu rzeczywistego *(validation)*[17];
c) testowanie adekwatności odwzorowania struktury badanego obiektu przez model *(legitimation)*[18].

Warto obecnie zastanowić się nad sposobem wykonania powyższych czynności a także ich umiejscowieniem w zaprezentowanym uprzednio schemacie budowy i aplikacji modeli symulacyjnych (Tablica 1.1.). Stosunkowo najprostszym rozwiązaniem jest tu prześledzenie faz budowy i aplikacji modeli, w celu znalezienia newralgicznych miejsc, z punktu widzenia możliwości popełnienia błędów, a następnie zastosowanie odpowiednich testów, z których większość zaproponował Coyle [4].

W fazie budowy modelu symulacyjnego należy zwrócić szczególną uwagę na wnioski płynące z analizy modelu graficznego. Przede wszystkim chodzi tu o odpowiedź na pytanie czy model jest spójny, czy jego struktura odpowiada strukturze badanego obiektu *(legitimation)*. Należy również zweryfikować jego zgodność z przyjętymi wcześniej założeniami. Wszelkie uproszczenia powinny bowiem wynikać z przyjętego systemu założeń a nie na przykład z niedostatków metodologicznych. Zdaniem autora niniejszej pracy, w przypadku modeli symulacyjnych o dużej skali, należy stosować budowę modułową, co w znacznym stopniu może ułatwić analizę modelu graficznego. Jednak w takim przypadku oprócz badania pojedynczych modułów należy zbadać również ich wzajemne powiązania. Nie do odrzucenia są również, wspomniane uprzednio sposoby analizy modeli graficznych opracowane na gruncie teorii sterowania.

Warunkiem przeprowadzenia pozostałych testów jest dysponowanie modelem komputerowym, oczywiście uprzednio zweryfikowanym pod kątem zgodności z modelem konceptualnym. Właściwie pierwszym testem, często niedocenianym,

[15] Więcej rad można znaleźć w pracy (Weil, 1980)

[16] Na przykład stanowisko racjonalistyczne, empiryczne czy oparte na założeniach ekonomii pozytywistycznej [9]

[17] Polscy autorzy próbują tu używać niezbyt fortunnie spolszczonego terminu: "walidacja"

[18] Zeigler (1984) używa tu terminu: "zasadność strukturalna"

powinno być zbadanie fizycznych wymiarów w każdym z równań. Stwierdzenie niezgodności wymiarów obu stron równania jest przecież stosunkowo łatwe technicznie[19]. Przy okazji można zweryfikować sposób modelowania pojedynczych relacji modelu *(verification)*. Jakże bowiem często nawet prawidłowe zachowanie się modelu jako całości nie przesądza o prawidłowości odwzorowania pojedynczych relacji modelu.

Pierwsze eksperymenty symulacyjne powinny mieć za zadanie stwierdzenie czy model nie produkuje absurdalnych wartości, którejś ze zmiennych *(validation)*. Stosunkowo popularne są tu również tak zwane testy bilansowe, przy pomocy których łatwo można wykryć te błędy, które powodują niezbilansowanie się relacji, wyliczanych na podstawie informacji pochodzących z różnych źródeł *(legitimation)*.

W modelach wykorzystujących dane rzeczywiste można stosować test zgodności z danymi historycznymi *(validation)*. Ma on jednak ograniczone zdarzenie w przypadku modeli, które będą wykorzystywane w sytuacjach w których możliwe są zmiany struktury modelu. Wtedy bowiem wiarygodność modeli może być stwierdzona wyłącznie w stosunku do założeń, które były aktualne w momencie budowy modelu. Przy tym od modeli symulacyjnych powinno się wymagać raczej projekcji niż predykcji (Meadows, 1978). Natomiast w przypadku modeli pracujących na danych hipotetycznych pozostaje właściwie test polegający na symulacyjnym badaniu wpływu nagłych zmian niektórych zmiennych modelu na jego zachowanie. Jeśli model zachowuje się zgodnie z przewidywaniami modelarza, to może to świadczyć o poprawności jego budowy. W przeciwnym przypadku stanowi natomiast źródło informacji o popełnionych błędach. Wykonanie całego kompletu powyższych testów oczywiście nie zapewnia automatycznej poprawności modelu.

5. Faza analizy ilościowej modelu przy wykorzystaniu symulacji komputerowej

5.1. Założenia

Dysponując modelem, przetestowanym pod względem jego wiarygodności, można przystąpić do prowadzenia eksperymentów symulacyjnych. Stosowane tu najczęściej sposoby pracy z modelem noszą nazwę analizy i syntezy [19]. O ile jednak istnieje wypracowany sposób postępowania z tego typu zadaniami, to sytuacja komplikuje się w przypadku próby przeprowadzenia eksperymentów optymalizacyjnych, bowiem Dynamika Systemowa nie dostarcza tu odpowiednich narzędzi. Wyjściem z tej sytuacji może być zastosowanie algorytmów opracowanych na gruncie teorii sterowania. Wymagają one jednak przekształcenia modelu skonstruowanego w konwencji Dynamiki Systemowej do postaci równań stanu. Jak już wspomniano, zadanie to jest wykonalne od strony technicznej [21], jednak w przypadku modeli o dużych rozmiarach nieefektywne, poprzez złe uwarunkowanie powstających macierzy. Ponadto istnieje konieczność dostosowania postaci analitycznej samego modelu oraz funkcji celu do wymagań stawianych przez algorytmy teorii sterowania (zwykle liniowość modelu oraz kwadratowa postać funkcji kryterium). W tej sytuacji rozsądną alternatywą może być stosowanie metody proponowanej przez Keloharju [5]. Posiada ona liczne zalety, z których na czoło wysuwa się fakt, że model skonstruowany w konwencji Dynamiki Systemowej może być poddany optymalizacji bez jakichkolwiek zmian. Dodatkowo, liczne ułatwienia techniczne, dostępne za pośrednictwem specjalnego oprogramowania (DYSMOD), powodują, że prowadzenia procesu optymalizacji jest rzeczywiście bardzo proste. Konsekwentne stosowanie specjalnego algorytmu optyma-

[19] Język DYSMAP zapewnia automatyczne wykonanie tej czynności *(legitimation)*

lizacji (SDR) powoduje, że nie istnieją żadne ograniczenia co do postaci analitycznej równań modelu, w tym funkcji kryterium optymalizacji. Ceną płaconą za powyższe ułatwienia jest fakt, że nie ma pewności co osiągnięcia rozwiązania optymalnego (w sensie ekstremum globalnego).

5.2. Sformułowanie zadania decyzyjnego

5.2.1. Jednoetapowe zadanie decyzyjne

Zdefiniowanie zadania decyzyjnego wymaga przyjęcia określonych założeń dotyczących opisu dynamiki układu, ewentualnych ograniczeń nakładanych na wartości, które mogą przyjmować zmienne decyzyjne oraz zdefiniowania funkcji kryterium optymalizacji.

Niech więc będą dane:

a) przedział czasu $<T_1, T_s>$, zwany horyzontem symulacji, przy czym:
   $T_1 \in TIME$ oznacza chwilę początkową,
   $T_s \in TIME$ oznacza chwilę końcową;
b) model zbudowany w konwencji Dynamiki Systemowej (MDS), a więc tym samym muszą być spełnione następujące warunki:
   - znane są wartości wszystkich zmiennych typu zasób z_i w chwili T_0,
   - znane są wartości zmiennych egzogenicznych, nie będących zmiennymi decyzyjnymi $(e_{q+1}, e_{q+2}, \ldots, e_p)$, w przedziale czasowym $<T_0, T_s>$;
   - znane są postaci analityczne funkcji opisujących zasoby i strumienie;
c) na wartości zmiennych decyzyjnych nałożone są ograniczenia postaci:

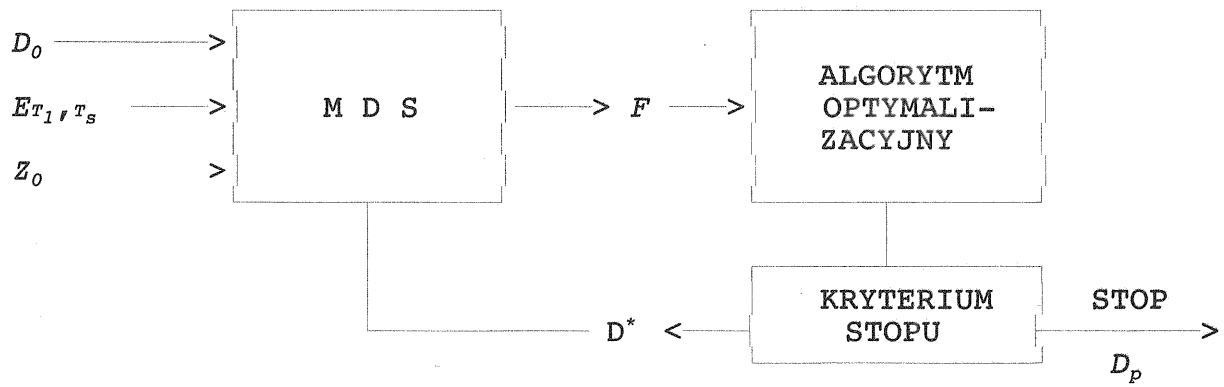
$$DL_{l,t} \leq d_{l,t} \leq DH_{l,t} \qquad (6)$$

   przy czym: $DL_{l,t}$ oraz $DH_{l,t}$ oznaczają odpowiednio dolne i górne ograniczenie nałożone na l-tą zmienną decyzyjną w chwili t,
d) określona jest postać analityczna funkcji kryterium optymalizacji f_k:

$$F = \sum_{T_1}^{T_s} f_k(Z_t, A_t, D_t) \qquad (7)$$

   przy czym:
   - F oznacza wartość funkcji kryterium w przedziale czasowym $<T_1, T_s>$,
   - Z_t oznacza wektor zmiennych typu zasób,
   - A_t oznacza wektor zmiennych pomocniczych,
   - D_t oznacza wektor zmiennych decyzyjnych,
   - f_k oznacza postać analityczną funkcji kryterium optymalizacji.

Przy powyższych założeniach (a-d), należy znaleźć wartości zmiennych decyzyjnych D_1, stałych w przedziale $<T_1, T_s>$, w taki sposób aby wartość funkcji kryterium osią-



RYS. 2.3. Algorytm optymalizacji jednoetapowej

gnęła ekstremum.

Jest to tak zwane jednoetapowe zadanie decyzyjne (JZD). Algorytm rozwiązania tak postawionego zadania rozbija się na dwie wyraźnie odrębne fazy: symulacji oraz optymalizacji. W fazie symulacji, na podstawie informacji o kształtowaniu się wartości zmiennych egzogenicznych E w przedziale czasowym $<T_1, T_s>$, znajomości wartości początkowych zmiennych typu zasób Z, wartości parametrów oraz dowolnie przyjętych wartości początkowych zmiennych decyzyjnych D_0, (spełniających jednak warunek "c"), przeprowadza się eksperyment symulacyjny w przedziale czasowym $<T_1, T_s>$. Jednym z jego rezultatów jest obliczenie wartości wskaźnika jakości optymalizacji F. W fazie optymalizacji, przy pomocy dowolnego algorytmu optymalizacyjnego[20], generowane są nowe wartości zmiennych decyzyjnych D, zastępujące dotychczasowe decyzje, po czym następuje powrót do fazy symulacji. To iteracyjne postępowanie jest prowadzone aż do spełnienia kryterium stopu, kiedy to otrzymuje się decyzje do zastosowania w danym etapie D_p. Niezbędny jest tu krótki komentarz. Mianowicie decyzje do zastosowania wcale nie muszą być decyzjami optymalnymi. Odpowiednie określenie liczby iteracji oraz kryterium stopu może bowiem ograniczyć się do decyzji zadowalających.

5.2.2. Wieloetapowe zadanie decyzyjne

Istotę zadania optymalizacji jednoetapowej najlepiej określa założenie dotyczące stałości decyzji w czasie trwania eksperymentu symulacyjnego, to znaczy w przedziale czasowym $<T_1, T_s>$. Założenia tego nie da się jednak utrzymać w większości zastosowań praktycznych. Trudno bowiem żądać aby raz ustalone decyzje obowiązywały w długim okresie czasu, pomimo zmian warunków zewnętrznych i wreszcie zmian samego modelowanego obiektu. Ponadto rozwiązanie zadania optymalizacji jednoetapowej nie daje możliwości uwzględnienia skutków decyzji długookresowych. Jest przecież rzeczą oczywistą, że wartość funkcji kryterium z m następujących po sobie etapów (liczona narastająco), nie będzie zwykle równa wartości funkcji kryterium, otrzymanej w wyniku rozwiązania analogicznego zadania optymalizacji jednoetapowej.

Wymienione względy przemawiają za stosowaniem podejścia właściwego dla algorytmu optymalizacji wieloetapowej.

W związku z tym horyzont symulacji $<T_1, T_s>$, postanowiono podzielić na krótsze odcinki czasu, zwane dalej:
- okresem w którym podejmowane decyzje mogą być traktowane jako stałe *(action time)*,
- horyzontem planowania *(planning horizon)*, a więc przedziałem czasowym, który bierzemy pod uwagę przy ocenie długookresowych skutków podejmowanych decyzji.

Niech więc:
- k oznacza liczbę kroków symulacji, które składają się na pojedynczy okres podejmowania decyzji,
- h oznacza liczbę kroków symulacji, składającą się na horyzont planowania.

Zatem:
przedział czasu $<T_i, T_{i+k-1}>$ nazywany będzie okresem podejmowania decyzji $(O.P.D.)$, a przedział $<T_i, T_{i+h-1}>$ horyzontem planowania $(H.P.)$. Oczywiście musi być przy tym spełniony warunek:

$$O.P.D. \leq H.P. \leq H.S. \qquad (8)$$

Oczywiście w przypadku gdy:
$k = h = H.S.$, mamy do czynienia z jednoetapowym zadaniem optymalizacji. Niech więc będą dane:

a) przedział czasu $<T_1, T_s>$, zwany

[20] Autor zebrał pewne doświadczenia również z innymi algorytmami, szczególnie chodzi tu o procedurę MINUIT

horyzontem symulacji, przy czym:
- $T_1 \in TIME$ oznacza chwilę początkową,
- $T_s \in TIME$ oznacza chwilę końcową,

b) przedział czasu $<T_i, T_{i+k-1}>$, zwany okresem podejmowania decyzji *(O.P.D.)*,

c) przedział czasu $<T_i, T_{h+k-1}>$, (przy czym: $k \leq h \leq T_s - T_i$), zwany dalej horyzontem planowania,

d) model zbudowany w konwencji Dynamiki Systemowej (MDS), a więc zgodnie z (2.20. - 2.23.):
- znane są wartości wszystkich zmiennych typu zasób z_i w chwili T_0,
- znane są wartości zmiennych egzogenicznych, nie będących zmiennymi decyzyjnymi $(e_{q+1}, e_{q+2}, \ldots, e_p)$, w przedziale czasowym $<T_0, T_s>$
- znane są postaci analityczne funkcji (2.20. - 2.22.)

e) na wartości zmiennych decyzyjnych nałożone są ograniczenia postaci:

$$DL_{l,t} \leq d_{l,t} \leq DH_{l,t} \qquad (9)$$

przy czym: $DL_{l,t}$ oraz $DH_{l,t}$ oznaczają odpowiednio dolne i górne ograniczenie nałożone na l-tą zmienną decyzyjną w chwili t,

f) określona jest postać analityczna funkcji kryterium optymalizacji f_k:

$$F = \sum_{T_0}^{T_s} f_k\,(Z_t, A_t, D_t) \qquad (10)$$

przy czym:
- F oznacza wartość funkcji kryterium w przedziale czasowym $<T_0, T_s>$,
- Z_t oznacza wektor zmiennych typu zasób,
- A_t oznacza wektor zmiennych pomocniczych,
- D_t oznacza wektor zmiennych decyzyjnych,
- f_k oznacza postać analityczną funkcji kryterium optymalizacji.

Przy powyższych założeniach (a-f), należy znaleźć wartości zmiennych decyzyjnych D_l, stałych w przedziale $<T_i, T_{i+k-1}>$, w taki sposób aby wartość funkcji kryterium osiągnęła ekstremum.

Zadanie to będzie nazywane dalej wieloetapowym zadaniem decyzyjnym (WZO). Algorytm jego rozwiązania wykorzystuje ideę JZD.

Reasumując, rozwiązanie WZO wymaga rozwiązania $n : k$ zadań jednoetapowych w następującej kolejności:

a) rozwiązania zadania jednoetapowego w przedziale czasu $<T_i, T_{i+h-1}>$,

b) przyjęcia wartości zmiennych decyzyjnych otrzymanych w punkcie "a", jako obowiązujących w przedziale czasu $<T_i, T_{i+k-1}>$ i jednocześnie będących wartościami początkowymi w okresie $<T_{i+k}, T_{i+k+h-1}>$,

c) przyjęcia, że $i = i + k$,

d) powrotu do punktu "b" aż do momentu gdy $i = T_s$.

Uzyskane w ten sposób rozwiązanie jest optymalne raczej z punktu widzenia całego zadania niż poszczególnych etapów. Algorytm optymalizacyjny poszukuje bowiem rozwiązania w przestrzeni $q \times n$, gdzie:
- q oznacza liczbę zmiennych decyzyjnych,
- n liczbę etapów.

Powstają przy tym czysto praktyczne problemy dotyczące:
- częstotliwości z jaką należy modyfikować decyzje *(O.P.D.)*,
- długości horyzontu planowania *(H.P.)*.

Niestety, istnieją tu tylko pewne wskazówki teoretyczne, a więc odpowiedź na tak postawione pytania musi zostać znaleziona (dla konkretnej sytuacji decyzyjnej) na drodze eksperymentalnej, uwzględniając przy tym względy merytoryczne oraz własności stosowanego algorytmu optymalizacyjnego.

6. Uwagi końcowe

Na koniec kilka uwag o praktycznym wykorzystaniu koncepcji metodologicznych zaprezentowanych w niniejszej pracy. Zdaniem jej autora, obszar potencjalnych zastosowań można podzielić na dwie sfery:
a) zarządzanie firmami;
b) dydaktyka przedmiotów ekonomicznych.

Jeśli chodzi o sferę zarządzania firmami to modele symulacyjne znajdują swe zastosowania jako istotne elementy wszelkiego typu systemów wspomagania decyzji (SWD)[21].

W dydaktyce przedmiotów ekonomicznych stosuje się z kolei symulacyjne gry decyzyjne oraz opisy przypadków. W obu sytuacjach jądro stanowi odpowiednio przygotowany model symulacyjny[22].

[21] Przykłady tego typu modeli można znaleźć w pracach [9] oraz (Wąsik, 1988b)

[22] Opis postępowania można znaleźć w pracy: [16]

Bibliografia

[1] Coyle R. G.: *Managament System Dynamics*, John Wiley and Sons, 1977.

[2] Coyle R. G.: *The Use of Optimization Methods for Policy Design in a System Dynamics Models*, System Dynamics Review, no. 1, 1985.

[3] Coyle R. G.: *The Technical Elements of the System Dynamics Approach*, System Research, vol. 2, 1985.

[4] Coyle R. G.: *A System Description of Counter Insurgency Warfare*, Policy Sciences 18, pg. 55-78, Elsevier Science Publishers, Amsterdam 1985.

[5] Keloharju R.: *The Relativity Dynamics*, Helsinki School of Economics, 1983.

[6] Keloharju R., Wolstenholme E.: *A Case Study in System Dynamics Optimization*, The 1986 InternationalConference of the System Dynamics Society, Sevilla 1986.

[7] Keloharju R.: *Simplification by Planning; The Third Way*, Conference on Dynamic Simulation Society, Shanghai 1987.

[8] Keloharju R., Kuusi O.: *The Modelling of Intuition in Futures Research*, Acta Academiae Oeconomicae Helsingensis, Helsinki 1987.

[9] Naylor T. H.: *Modelowanie cyfrowe systemów ekonomicznych*, PWN, Warszawa 1975.

[10] Skrzypek J.: *Filtracja kalmanowska jako metoda estymacji modeli procesów gospodarczych*, praca doktorska, AE Kraków 1982.

[11] Skrzypek J.: *Próba oceny przydatności filtracji kalmanowskiej do estymacji modeli procesów gospodarczych*, "Zeszyty Naukowe" AE w Krakowie, nr 169, str. 185-201, 1984.

[12] Skrzypek J.: *Zastosowanie filtracji kalmanowskiej do estymacji parametrów modeli procesów gospodarczych*, Prace Naukowe AE we Wrocławiu, nr 311, str. 165-173, 1985.

[13] Skrzypek J.: *Zastosowanie filtracji kalmanowskiej do estymacji łącznej w modelach procesów gospodarczych*, Zeszyty naukowe AE w Krakowie, nr 222, str. 63-80, 1986.

[14] Skrzypek J.: *The Parameter Estimation in Dynamic Economic Systems via Kalman Filtering*, Proceedings of the V-th IFAC/IFORS on Dynamic Modelling and Control of National Economies, Pergamon Press, 1987.

[15] Skrzypek J.: *Optymalizacja w modelach Dynamiki Systemowej*, Prace Naukowe AE we Wrocławiu, nr. 404, str. 151-156, 1988.

[16] Skrzypek J., Szubra M.: *Symulacyjne gry decyzyjne jako narzędzie aktywnego kształcenia menedżerów*, Informatyka nr 9, 1992.

[17] Szumilak J.: *Dynamika procesów realnych w przedsiębiorstwie handlowym*, Akademia Ekonomiczna w Krakowie 1987.

[18] Świtalski W.: *Symulacyjne modele systemów ekonomicznych*, Ekonomista nr 6, 1987.

[19] Wąsik B.: *Elementy dynamiki systemowej dla ekonomistów*, Akademia Ekonomiczna, Kraków 1983.

[20] Wąsik B.: *Komputerowa gra kierownicza KRAEK-1*, materiał powielony, AE w Krakowie, 1986.

[21] Wąsik B., Litwa A., Skrzypek J.: *Równanie stanu w badaniu dynamiki obiektów ekonomicznych*, Akademia Ekonomiczna w Krakowie, 1986.

[22] Wąsik B.: *Koncepcja budowy i wykorzystanie modułowego symulatora przedsiębiorstwa przemysłowego*, 1988.

Wykaz prac nie umieszczonych w bibliografii, do których odwołuje się Autor (opr. red.)

1. Aoki, 1976
2. Appel, Fassing, 1978
3. Bell, Bell, 1980
4. Bocheński, 1988
5. Coyle, 1978
6. Coyle, 1984
7. Forrester, 1961
8. Kotva, 1988
9. Lyneis, 1980
10. Maciaszek, 1988
11. Maiser, 1976
12. Meadows, 1978
13. Meadows, 1980
14. Morecroft, 1982
15. Neck, 1984
16. OSSIM, 1977
17. Plott, 1982
18. Randers, 1980
19. Rzońca, 1981
20. Skrzypek, Tatar, 1989
21. Smith, 1979
22. Smith, 1982a
23. Wahlstrom,1986
24. Wąsik, 1988b
25. Weil, 1980
26. Wolstenholme, 1982
27. Zeigler, 1984

Marek Miłosz
Politechnika Lubelska, Katedra Informatyki

SYMULACJA SYSTEMÓW - Symulacja procesów dyskretnych

1. Wstęp

W teorii systemów jednym z najważniejszych kryteriów podziału systemów jest charakter zachodzących w nich procesów. Analogicznie, dynamiczne modele istniejących lub projektowanych systemów dzielą się na ciągłe, dyskretne lub mieszane (ciągło-dyskretne) [1].
W zasadzie mało jest systemów, które są *sensu stricto* wyłącznie ciągłe lub dyskretne. Jednak w większości wypadków albo w systemie dominuje jeden typ zmian, albo cel badań w sposób jednoznaczny określa typ stworzonego modelu [2]. W przypadku rozwiązywania modeli ciągłych metodami numerycznymi powstaje nowy typ: modele dyskretne systemów ciągłych.

Systemy dyskretne charakteryzują się skokowymi zmianami stanu. Ich opis związany jest zatem z opisem ciągu zdarzeń wywołujących te zmiany - tj. z opisem procesu dyskretnego [3].

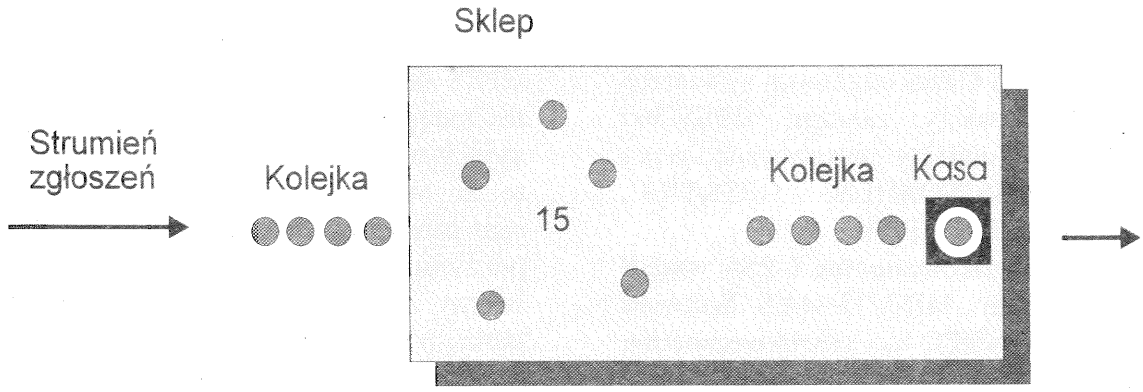
Symulacja systemów jest szczególną metodą badania systemów i procesów w nich przebiegających. Polega ona na śledzeniu zmian, zachodzących w dynamicznym modelu systemu. Jest więc **eksperymentem** z tym modelem. Zastosowanie komputerów do symulacji umożliwia badanie modeli o praktycznie dowolnej dokładności odwzorowania rzeczywistości (w pewnych przypadkach aż do symulacji tożsamościowej włącznie). Eksperyment symulacyjny ma tak wiele zalet, że stał się podstawowym narzędziem badawczym w wielu dziedzinach [2]. Zmniejsza on koszty badań i skraca ich czas. Umożliwia pełną, pozbawioną błędów pomiarowych kontrolę stanu modelu i dowolne sterowanie przebiegiem symulacji (np. zatrzymywanie i wznawianie badań w praktycznie dowolnym momencie). Zapewnia także stuprocentową powtarzalność eksperymentu. Poprzez właściwość kondensowania lub rozciągania czasu modelowego w stosunku do rzeczywistego eksperyment symulacyjny umożliwia w wielu przypadkach jakiekolwiek badanie systemów.

Cyfrowa symulacja systemów, będąc bardzo atrakcyjną metodą badawczą, pozostaje w chwili obecnej sztuką [2]. Sztuką właściwego wytyczenia celu badań, sztuką budowy modelu, doboru stopnia jego szczegółowości i wartości parametrów, sztuką komputerowej realizacji (oprogramowania) i weryfikacji modelu, sztuką przeprowadzenia ciągu eksperymentów symulacyjnych i wreszcie sztuką interpretacji ich rezultatów. Brak jest bowiem (poza wytycznymi dla poszczególnych typowych przypadków i pewnymi wskazówkami metodologicznymi) ogólnych i uniwersalnych prawideł, umożliwiających konstruowanie i weryfikację programów symulujących procesy rzeczywiste. Pomimo tych problemów warto opanować sztukę symulacji systemów i stosować ją w praktyce badawczej, projektowej i eksploatacyjnej różnorodnych systemów.

2. Procesy dyskretne i ich symulacja

Aby scharakteryzować elementy służące do opisu procesu dyskretnego, najlepiej będzie posłużyć się klasycznym i ogólnie zrozumiałym przykładem funkcjonowania sklepu samoobsługowego. Do sklepu klient może wejść wyłącznie z koszykiem. Liczba koszyków jest ograniczona do 15. Wszystkie koszyki są identyczne. W sklepie klient przez pewien czas dokonuje wyboru towaru, po czym płaci w kasie (sklep jest niewielki, więc kasa jest jedna) i opuszcza sklep, zwalniając jednocześnie koszyk.
Wszystkie przebiegające w takim systemie procesy są w zasadzie ciągłe, ale do badania charakterystyk systemu można je interpretować jako dyskretne. Zdarzenia (wejście klienta do sklepu, pobranie koszyka, podejście do kasy itd.) zachodzą tylko w pewnych momentach czasu (nie rozpatruje się w modelu np. przemieszczania się klienta po sklepie). Pomiędzy zdarzeniami system nie zmienia swojego stanu. Nie ma więc potrzeby rozpatrywania procesu w momencie braku jego aktywności.
W systemie dyskretnym występują zdarzenia dwóch typów: **bezwzględne** i **warunkowe**. Zdarzeniem bezwzględnym (jego wystąpienie zależy tylko od upływu czasu) jest w sklepie przyjście kolejnego klienta, natomiast warunkowym - jego wejście do sklepu (zależy ono bowiem od dostępności koszyka).
Przybywający do sklepu klienci są **zgłoszeniami** (zadaniami), które są obsługiwane przez **urządzenia** (stanowiska obsługi) lub ich grupy. W sytuacji, gdy **strumień zgłoszeń** przewyższa w pewnym momencie wydajność procesu obsługi, w systemie tworzą się **kolejki**.
Statystycznie prosty strumień zgłoszeń opisuje się przy pomocy rozkładu odstępów czasowych pomiędzy kolejnymi zgłoszeniami.
Proces obsługi opisywany jest przez parametry statystyczne czasu obsługi jednego zgłoszenia, krotność (pojemność) urządzenia obsługującego i jego dostępność w konkretnym momencie czasu. W rezultacie symulacji uzyskuje się **obciążenia** urządzeń i ich **zajętości**. Obciążenie urządzenia jest to stosunek czasu, w którym urządzenie było zajęte obsługą zgłoszeń, do czasu symulacji (w jednostkach czasu modelowego). Zajętość natomiast jest równa stosunkowi przeciętnej liczby jednocześnie obsługiwanych zgłoszeń do krotności urządzenia.
Kolejki powstają przed urządzeniami w wyniku braku możliwości obsługi części zgłoszeń w danym momencie czasu. Proces tworzenia się kolejek może być skomplikowany. Przykładowo klienci mogą rezygnować z zajęcia miejsca w kolejce (i z robienia zakupów), jeśli kolejka jest zbyt długa lub nawet opuszczać kolejkę po bezskutecznym odstaniu w niej pewnego (stałego lub losowego) czasu. Zgłoszenia z kolejek wybierane są do obsługi według różnych **strategii**: FIFO *(First-In, First-Out)*, LIFO *(Last-In, First-Out)*, losowej, z priorytetem zgłoszenia, wg czasu obsługi itp.
Statystyczne parametry kolejek, dotyczące ich długości, czasów oczekiwania czy też prawdopodobieństwa oczekiwania, są podstawowymi wartościami wyznaczanymi podczas symulacji procesów dyskretnych i określającymi jakość ich funkcjonowania.

W przykładowym sklepie (rys. 1) urządzeniami obsługującymi zgłoszenia są koszyki i kasa. Koszyki mogą jednocześnie obsługiwać 15 klientów. Są więc one urządzeniem o określonej pojemności (lub inaczej mówiąc grupą identycznych, nierozróżnialnych urządzeń). Kasa natomiast jest pojedynczym urządzeniem, w którym może przebywać tylko jeden klient (zgłoszenie). Zarówno przed sklepem (grupa urządzeń: koszyki) jak i przed kasą mogą się tworzyć kolejki. Przeważnie są to kolejki o strategii wyboru typu FIFO, ale mogą być również wyjątki (np. uprzywilejowani klienci). Klient zajmuje koszyk przez cały czas pobytu w sklepie, natomiast kasę - tylko na czas potrzebny na zapłacenie. Do konkretyzacji modelu symulacyjnego sklepu samoobsługowego niezbędne są jeszcze trzy charakterystyki: strumienia klientów,



RYS. 1. Schemat funkcjonowania sklepu samoobsługowego

czasu traconego przez klienta na wybór towaru wewnątrz sklepu i czasu pobytu klienta w kasie.

Możliwa jest oczywiście większa komplikacja modelu, bardziej zbliżająca go do rzeczywistości. Wymaga ona jednak dodatkowych danych o klientach i ich zachowaniu, a także zachowaniu się urządzeń obsługujących. Można symulować różne kategorie klientów: zwykłych i uprzywilejowanych, bardziej lub mniej niecierpliwych, dokonujących zakupów dużych i małych, itd. Można symulować kradzieże koszyków (zmniejszenie pojemności urządzenia koszyki), ich zwrot lub uzupełnienie, przerwy w pracy całego sklepu czy też kasy itp. Wprowadzając wszelkie tego typu uściślenia należy ciągle pamiętać o celu symulacji, możliwości uzyskania i stopniu dokładności danych wejściowych, a także przeanalizować sens rozbudowy modelu. Przykładowo: jeśli kradzieże koszyków są zjawiskiem marginalnym i obsługa szybko uzupełnia braki w koszykach, to zbędne jest uwzględnianie tego zjawiska w modelu symulacyjnym.

Przytoczony przykład systemu dyskretnego jest charakterystyczny dla zastosowań teorii obsługi masowej. Wiele systemów organizacyjnych, produkcyjnych i technicznych funkcjonujących w warunkach losowych daje się opisać w analogicznych kategoriach.

Do symulacji procesów dyskretnych niezbędny jest system, który będzie odmierzał upływ czasu modelowego (umowny czas symulacyjny, wyznaczający następstwo zdarzeń) i analizował stany modelu w pewnych, określonych momentach czasu. Sterowanie przebiegiem symulacji może odbywać się przy pomocy metody **planowania zdarzeń**, **przeglądu działań** lub **interakcji procesów** [3].
Systemy symulujące procesy dyskretne można w zasadzie tworzyć przy pomocy języków algorytmicznych ogólnego przeznaczenia (np. FORTRAN, Pascal lub C). Znacznie efektywniejszą metodą przekształcenia modelu w program komputerowy jest wykorzystanie któregoś z gotowych systemów - **języków symulacyjnych**. Poza pewnymi szczególnymi przypadkami (np. symulacja bardzo dużych systemów) są one doskonałym narzędziem badawczym. Jednym z nich jest **GPSS** *(General Purpose Simulation System)*, który jest dostępny również w wersji dla mikrokomputerów typu IBM PC [4].

GPSS jest językiem o strukturze blokowej pozwalającym opisywać i symulować dynamiczne systemy dyskretne (stochastyczne lub deterministyczne) w postaci procesów quasi-równoległych. Wykorzystuje on metodę interakcji procesów [3]. Pod nazwą GPSS, oprócz języka do opisu modeli systemów, kryje się translator i program sterujący procesem symulacji.

GPSS powstał w 1961 roku w laboratoriach firmy IBM. Jego twórcą był Geoffrey Gordon. Do najważniejszych implementacji GPSS na dużych systemach cyfrowych należy zaliczyć GPSS/360 i GPSS V (dialekty opracowane przez IBM), GPSS/NORDEN, GPSS/UCC, GPSSTS i GPSS-10 [5].

W świat mikrokomputerów GPSS wszedł dzięki firmom: MINUTEMAN Software (wersja : GPSS/PC dla mikrokomputerów typu IBM PC) [6], Simulation Software (wersja: GPSSR/PC dla mikrokomputerów typu IBM PC i różnych minikomputerów, a także stacji roboczych z systemem operacyjnym UNIX) i Wolverine Software Corporation (wersja: GPSS/H dla mikrokomputerów typu IBM/PC i stacji roboczych VAX/VMS i UNIX) [7]. Do interesujących implementacji języka należy zaliczyć mikro-GPSS, opracowany przez profesora I. Ståhla ze Sztokholmskiej Akademii Ekonomicznej [8]. Mikro-GPSS dostępny jest w wersjach dla IBM PC, Macintosh i VAX/VMS.

Większość wersji GPSS posiada interpreter języka. Wyjątkiem jest GPSS/H, który został zaopatrzony w kompilator, zwiększający szybkość symulacji [7]. GPSS/H nie posiada jednak możliwości pracy z modelem w trybie interakcyjnym, co znacznie utrudnia uruchamianie programów.

3. Podstawy języka GPSS[1]

Zdarzenia dyskretne, zachodzące w realnym systemie, w języku GPSS są odwzorowywane za pomocą ruchu obiektów-zadań *(transaction)* między blokami w symulowanym czasie (czasie modelowym). Każdy z bloków wpływa na zmianę stanu określonego zadania lub grupy zadań. Kolejność przepływu zadań jest wyznaczana przez naturalne uporządkowanie bloków. Istnieją bloki kierujące zadania w inne miejsce modelu w zależności od spełnienia się określonego warunku lub też przypadkowo (z zadanym prawdopodobieństwem).

System GPSS posiada wbudowany zegar, który odmierza upływ czasu modelowego. Model (a właściwie program) w języku GPSS składa się z segmentów, w których zachodzą równolegle (mierząc wg czasu modelowego) określone procesy. Oznacza to, że w każdej części programu jest ten sam czas modelowy. Każdy segment rozpoczyna się blokiem GENERATE, który tworzy zadania, a kończy się przeważnie blokiem TERMINATE, usuwającym je z modelu. Poszczególne segmenty modelu mogą imitować różne zjawiska, a zadania w nich - różne obiekty rzeczywiste.

Przy pomocy parametrów bloku GENERATE można określać:

- rozkład statystyczny odstępów czasowych pomiędzy tworzonymi zadaniami;
- opóźnienie generowania pierwszego zadania;
- maksymalną liczbę tworzonych zadań;
- priorytet zadania.

Oprócz poziomu priorytetu z każdym zadaniem może być związany zbiór parametrów numerycznych.

Czas przejścia zadania przez blok (oprócz specjalnego bloku opóźniającego ADVANCE) jest równy zero.

W swojej drodze zadanie może zostać zatrzymane przez:

- blok opóźniający ADVANCE (na ściśle określony czas);
- wystąpienie braku możliwości wejścia zadania do następnego bloku (np. blokada lub zajęcie urządzenia, niespełnienie warunku itp.);
- czasowe usunięcie zadania z modelu.

Do najważniejszych bloków sterujących przepływem zadań w GPSS należą:

TRANSFER - blok skoku, który może pracować w trybie skoku bezwzględnego, statystycznego, warunkowego, przełącznikowego, funkcjonalnego, parametrycznego, do podprogramu itd.;

LOOP - blok organizacji pętli;

TEST - blok warunkowy, testujący warunek arytmetyczny;

GATE - blok przesunięcia zadania w zależności od wartości określonego wskaź-

[1] ten i następny rozdział pochodzą z [4]

nika logicznego, określającego stan poszczególnych elementów modelu.

Podstawowymi elementami języka GPSS, służącymi do odwzorowywania systemów obsługi są: urządzenia *(facility)* i ich grupy, zwane magazynami *(storage)*.

Urządzenie jest elementem, w którym w danym momencie czasu może znajdować się tylko jedno zadanie. Wejście do urządzenia (zajęcie urządzenia) odbywa się przy pomocy bloku SEIZE. Wejście innego zadania do zajętego już urządzenia jest niemożliwe. Próba dokonania tego spowoduje zatrzymanie (zawieszenie) zadania w ruchu w poprzednim bloku. Po zwolnieniu urządzenia (blok RELEASE) ruch zadania zawieszonego w ten sposób jest wznawiany automatycznie. Urządzenia mogą być także blokowane (blokowanie: FUNAVAIL, zwalnianie: FAVAIL) lub okupowane (okupowanie: PREEMPT, uwolnienie: RETURN).

Magazyn jest grupą urządzeń o określonej liczności (pojemności). Miejsca w magazynie (urządzenia w grupie) są identyczne i nierozróżnialne. Zadanie może zająć przy pomocy bloku ENTER określoną liczbę miejsc w magazynie (zwolnienie: blok LEAVE), pod warunkiem, że żądana liczba miejsc jest wolna. W przeciwnym przypadku zadanie oczekuje w bloku poprzednim na zmniejszenie zajętości magazynu. W dowolnym momencie czasu modelowego zajętość magazynu nie może bowiem przekroczyć jego pojemności. Dostępne jest blokowanie (SUNAVAIL) i zwalnianie blokady (SAVAIL) magazynu.

W miejscach warunkowego zatrzymania zadań tworzą się ich kolejki. Są one obsługiwane zgodnie ze standardową procedurą FIFO w ramach każdego poziomu priorytetów zadań. Zmiana strategii obsługi kolejki możliwa jest poprzez przeniesienie go na listę użytkownika *(user chain entity)* przy pomocy bloku LINK i wprowadzenie do modelu przez blok UNLINK. Lista użytkownika może być uporządkowana wg strategii FIFO, LIFO lub wg wartości określonego parametru zadania. Umożliwia to realizację praktycznie dowolnych strategii obsługi kolejek.

W GPSS istnieje pojęcie rodziny zadań *(related transaction)*. Rozmnożone w bloku SPLIT zadania mogą być łączone w jedno (blok ASSEMBLE), zbierane w jednym miejscu (blok GATHER) lub synchronizowane w różnych miejscach modelu (MATCH). Nieco bardziej skomplikowanych operacji można dokonywać na zadaniach należących do tzw. grup *(transaction group)*.

GPSS jest wyposażony w cały arsenał środków służących do rejestracji rezultatów symulacji. Niektóre dane (np. parametry pracy urządzeń, magazynów czy też poszczególnych bloków) zbierane są w sposób automatyczny. W celu rejestracji parametrów kolejek tworzone są specjalne obiekty: kolejki *(queue)*. Blok QUEUE określa miejsce wejścia zadania do kolejki (z zajęciem w niej określonej liczby miejsc), natomiast DEPART - punkt opuszczenia kolejki. Obiekty zwane kolejkami zbierają uśrednione dane (np. maksymalna, bieżąca i średnia długość kolejki, średnie czasy pobytu zadań w kolejce). Przy pomocy deklaracji QTABLE można zadeklarować dla danej kolejki tabelę, w której będzie automatycznie tworzony histogram czasu pobytu zadań w kolejce. Należy podkreślić, że obiekty zwane kolejkami w GPSS służą wyłącznie do rejestracji parametrów pobytu zadań w określonych miejscach w modelu. W miejscach tych mogą (ale nie muszą) tworzyć się kolejki zadań, oczekujących na zajście pewnych zdarzeń. Podobnie kolejki zadań mogą powstawać w miejscach modelu, w których nie ma rejestracji ich parametrów.

Dla dowolnego systemowego atrybutu numerycznego *(System Numerical Attribute)* można tworzyć tabele-histogramy (deklaracja TABLE i blok rejestracji danych TABULATE). Atrybuty te są dostępne w każdym miejscu modelu i zawierają dane o praktycznie wszystkich jego elementach: od czasu modelowego, poprzez parametry zadań, kolejek, urządzeń i magazynów, aż do funkcji, generatorów liczb losowych, zmiennych prostych (arytmetycznych i logicznych) i tablicowych, czy też przełączników logicznych

i zmiennych pamięciowych. Systemowe atrybuty numeryczne (SAN) wykorzystywane są do budowy modeli, sterowania przebiegiem symulacji i zbierania jej rezultatów.

Blokowy charakter języka GPSS znalazł odbicie w bardzo przejrzystym sposobie odwzorowania modeli systemów przy pomocy schematów blokowych. Dla każdego bloku języka opracowany został specjalny symbol graficzny [9].
Ten krótki przegląd elementów języka GPSS nie wyczerpuje oczywiście wszystkich jego możliwości w zakresie budowy modeli symulacyjnych. Łatwość w interpretacji funkcji poszczególnych bloków i sposobu symulacji pozwala tworzyć dość skomplikowane modele już po poznaniu kilku-kilkunastu bloków. Najprostszy model jednokolejkowego systemu obsługi masowej wymaga znajomości zaledwie pięciu bloków: GENERATE, SEIZE, ADVANCE, RELEASE i TERMINATE. Sklep samoobsługowy można natomiast symulować przy pomocy siedmiu typów bloków.

4. Implementacja GPSS na mikrokomputerach - GPSS/PC

Amerykańska firma MINUTEMAN Software opracowała w 1985 roku jedną z pierwszych wersji języka GPSS na mikrokomputery typu IBM PC (pod handlową nazwą GPSS/PC) [6], [9]. GPSS/PC jest środowiskiem pozwalającym na przygotowywanie, uruchamianie i przeprowadzanie obliczeń symulacyjnych modeli w języku GPSS. Oprócz edytora, interpreter języka, symulatora i programu wspomagającego uruchamianie modeli w trybie interakcyjnym, posiada on rozbudowane możliwości animacji, czyli graficznego ilustrowania procesu symulacji. W GPSS/PC zaimplementowano ponad 70 bloków i poleceń języka GPSS. Stanowi to ponad 95% bloków jednej z najbardziej rozbudowanej wersji, którą jest GPSS V. Ponadto GPSS/PC został wyposażony w szereg bloków i poleceń nieznanych we wcześniejszych wersjach języka. Jest to jedyny system z pełną interakcyjną edycją modelu w czasie chwilowych przerw w symulacji. Efektywność systemu wynosi wg danych producenta ok. 0.001 sek./blok (dla standardowego IBM PC/AT). Prosta wersja GPSS/PC (nr 2.36 z listopada 1992r) wykorzystuje maksymalnie 640 KB pamięci RAM (minimalne wymagania to około 350 KB) i umożliwia uruchamianie modeli zawierających parę tysięcy bloków i parę tysięcy jednocześnie aktywnych zadań. Wersja GPSS/PC EMS może wykorzystywać pamięć rozszerzoną typu LIM EMS wersja 3.2 lub późniejsza i umożliwia budowę znacznie większych modeli.
GPSS/PC wyposażony został w możliwość dołączania programów w języku FORTRAN (kompilator Microsoft), wywoływanych w trakcie symulacji przez blok HELP. Do i z programu w języku FORTRAN można przekazywać parametry. Przy pomocy FORTRANU można realizować dialog z użytkownikiem, grafikę wysokiej rozdzielczości, komunikację z systemami czasu rzeczywistego, symulację modeli mieszanych, optymalizację parametrów systemu itd.
Edytor, wbudowany w system GPSS/PC, jest wysoko wyspecjalizowanym narzędziem, które na bieżąco podpowiada format wprowadzanych informacji, przeprowadza kontrolę poprawności linii programu i udostępnia pomoc kontekstową. Edytor ten uniemożliwia wprowadzenie niepoprawnych składniowo poleceń i instrukcji. Modele mogą być tworzone także przy pomocy dowolnego innego edytora - są one przechowywane w standardowych plikach tekstowych.
Interakcyjne przeprowadzanie procesu symulacji umożliwia:

- przerwanie symulacji w dowolnym momencie (klawisz ESC lub HOME);
- wprowadzenie modyfikacji do modelu i wznowienie symulacji (polecenie CONTINUE);
- bezpośrednie wprowadzanie i natychmiastowe wykonywanie bloków języka

GPSS;
- dostęp do wartości wszystkich SAN, zmiennych modelu i list zdarzeń;
- krokowe wykonywanie symulacji z zadaną liczbą zdarzeń (polecenie STEP);
- użycie kalkulatora do wyznaczania wartości wyrażeń (polecenie SHOW);
- wyświetlanie wykresu zmian wartości dowolnego SAN w trakcie symulacji (polecenie PLOT);
- śledzenie przebiegu symulacji (polecenia TRACE i UNTRACE);
- dynamiczne przełączanie się pomiędzy oknami graficznymi.

Okna graficzne zawierają wizualizację procesu symulacji. Poza ekranem z programem, w którym jest także wyświetlany wykres dowolnego SAN (polecenie PLOT), dostępne są okna z obrazami urządzeń, magazynów, tabel statystycznych (histogramów) i macierzy. Istnieje także okno z automatycznie tworzonym schematem blokowym modelu. Zawartość okien jest aktualizowana w trakcie symulacji, umożliwiając obserwację jej rezultatów. Specjalne okno służy do tworzenia dwuwymiarowych ruchomych modeli. Ruch zadań w trakcie symulacji powoduje przesuwanie się po ekranie symboli związanych z nimi, ilustrując w realistyczny sposób działanie symulowanego systemu. Okno animacji jest doskonałym elementem do prezentowania na pokazach zasad działania modelu. Okna wizualizujące proces symulacji są sterowane przy pomocy klawiatury lub urządzenia wskazującego (myszy, pióra świetlnego).

Standardowe okna mogą być uzupełnione specjalnymi obiektami: mikrookami. W maksymalnie czterech mikrooknach można wyświetlać bieżące wartości dowolnych SAN z odpowiednim komentarzem.

Do tworzenia bardziej wyrafinowanych ruchomych rysunków służy GPSS/PC Animator. Jest to program wykorzystujący rysunki stworzone w systemie AutoCAD do animacji działania modelu symulacyjnego. W programie tym dostępna jest trójwymiarowa grafika i pełna kontrola obrazu (powiększanie, przesuwanie, cofanie animacji, zmiana kolorów itp.). Rysunki animacyjne mogą być łączone z wyświetlaniem wartości SAN (np. czasu modelowego) i wykresów (np. histogramów).

GPSS/PC umieszcza rezultaty symulacji w specjalnym pliku tekstowym. Program generowania raportów tworzy na jego podstawie standardowy wydruk (lub wyświetla wyniki na ekranie). Zawartość raportu jest kształtowana przez zespół parametrów konfigurujących. Przejrzysta struktura pliku z rezultatami symulacji umożliwia napisanie własnych programów generujących raporty w żądanej postaci.

5. Symulacja sklepu samoobsługowego

Na podstawie obserwacji istniejącego sklepu ustalono, że:
- klienci przychodzą do sklepu średnio co 10 min. (rozkład Poissona);
- średni czasu wybierania towaru przez klienta wynosi 15±5 min. (rozkład równomierny);
- średni czas pobytu klienta w kasie: 3±2 min. (rozkład równomierny).

Informacje te wystarczają do budowy modelu symulacyjnego - programu w języku GPSS/PC (wydruk 1). W programie tym występują: urządzenie KASA, grupa urządzeń KOSZYKI (pojemność: 15), dwie kolejki: SKLEP i PLAC. Kolejki te zostały użyte do rejestracji parametrów: łącznego czasu pobytu klienta w sklepie (kolejka SKLEP i tabela KOL1) i czasu traconego na płacenie (PLAC i KOL2). Nie są więc rejestrowane dane dotyczące parametrów tworzących się w sklepie kolejek, chociaż czas pobytu klienta w kolejkach jest częścią składową zbieranych informacji.

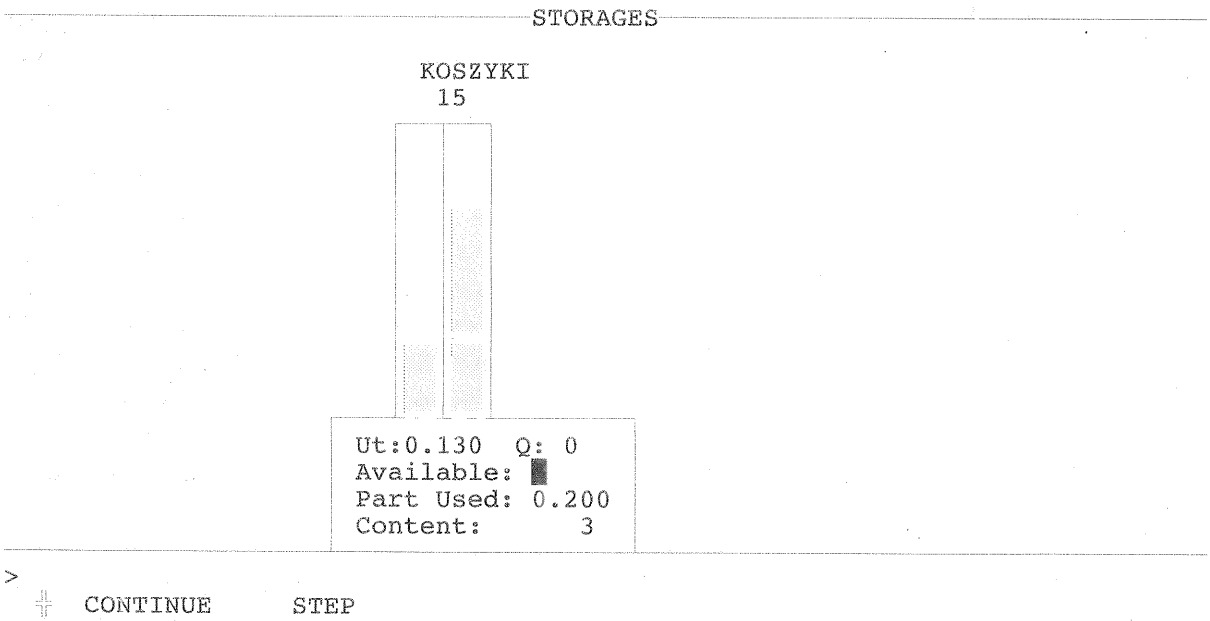
Struktura pierwszego segmentu modelu odwzorowuje logikę funkcjonowania sklepu.

Segment drugi odlicza natomiast ośmiogodzinne odcinki czasu modelowego (jednostką czasu modelowego jest 1 min.).

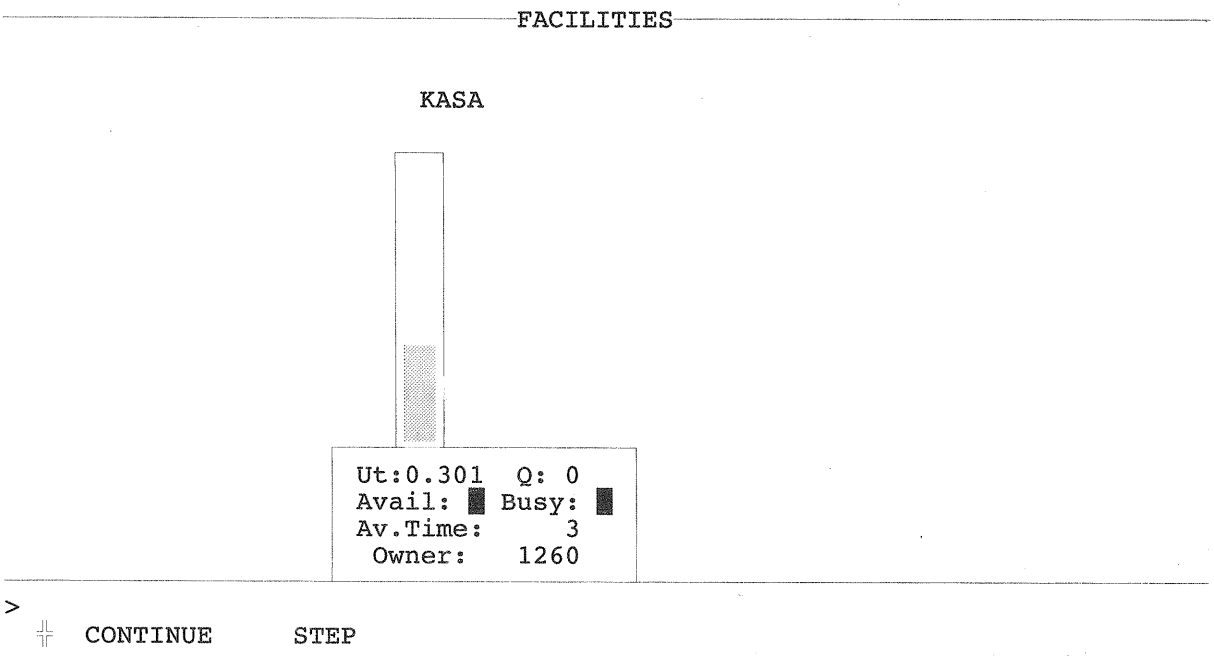
```
10 *                    Sklep samoobsługowy
20 *  Segment 1
30 *
40 XPDIS    FUNCTION    RN3,C24              ;Exponential distribution function
0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.75,1.38
.8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2
.97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9998,8
50 *
60 KOSZYKI  STORAGE     15                   ;liczba koszyków
65 *
70 KOL1     QTABLE      SKLEP,10,2,17        ;czas pobytu w sklepie
80 KOL2     QTABLE      PLAC,0,1,20          ;czas w kolejce do kasy
90 *
100         GENERATE    10,FN$XPDIS          ;strumień klientów
110         QUEUE       SKLEP
120         ENTER       KOSZYKI              ;pobranie koszyka
130         ADVANCE     15,5                 ;pobyt w sklepie
140         QUEUE       PLAC
150         SEIZE       KASA                 ;podejście do kasy
160         ADVANCE     3,2                  ;obsługa w kasie
170         RELEASE     KASA                 ;opuszczenie kasy
180         LEAVE       KOSZYKI              ;oddanie koszyka
190         DEPART      PLAC
200         DEPART      SKLEP
210         TERMINATE                        ;opuszczenie sklepu
220 *
230 *   Segment 2 - timer
240         GENERATE    480                  ;1 zadanie na 8 godz.
250         TERMINATE   1
```

WYDRUK 1. Program - model sklepu samoobsługowego

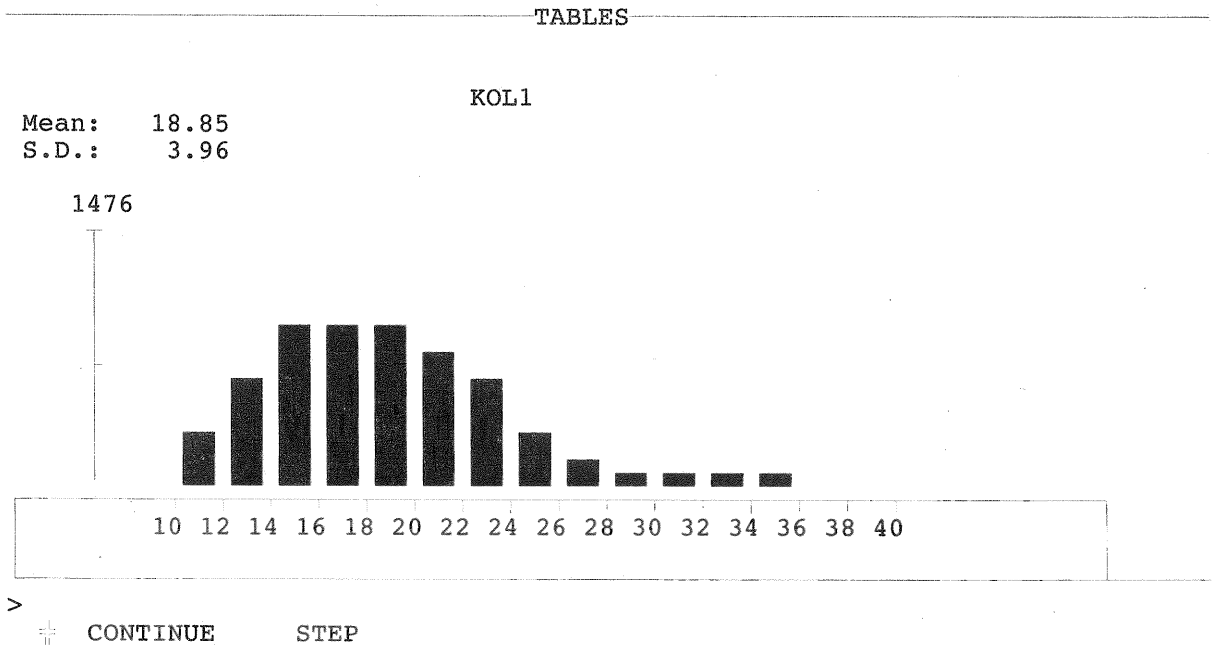
Uruchomienie programu przy pomocy polecenia START 100 symuluje działanie sklepu przez 100 dni.
W trakcie trwania symulacji GPSS/PC umożliwia obserwację rezultatów: parametrów grupy urządzeń (rys. 2), urządzenia (rys. 3) i tabel - histogramów (rys. 4 i 5).



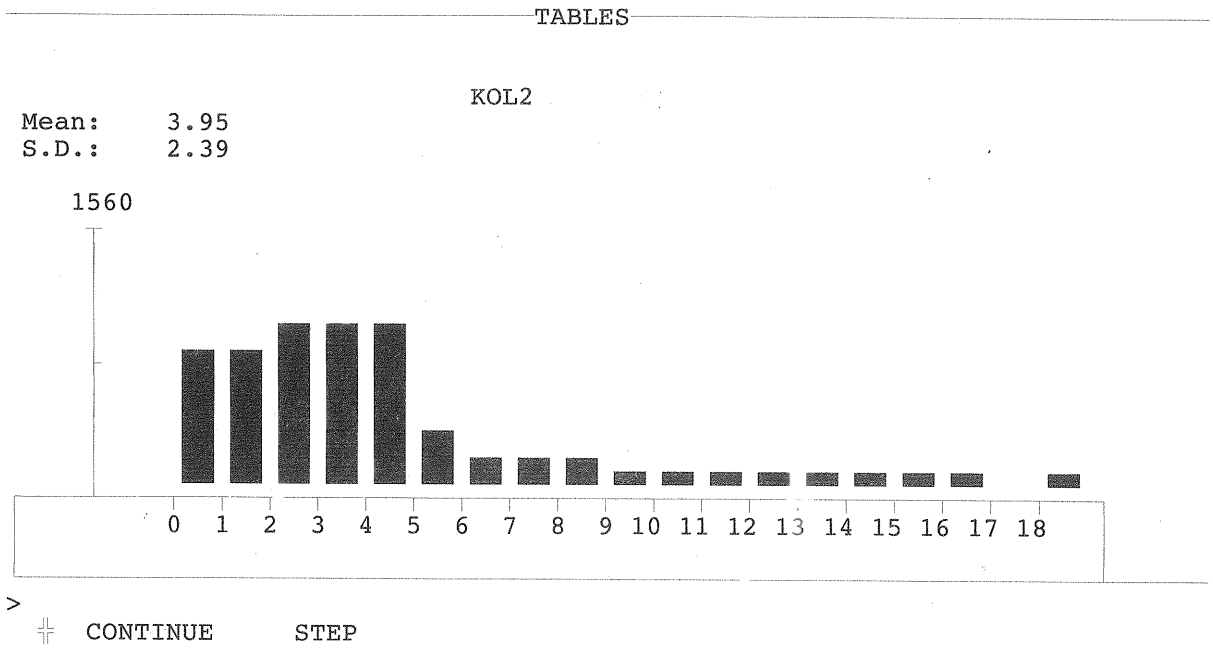
RYS. 2. Parametry grupy urządzeń KOSZYKI



RYS. 3. Parametry urządzenia KASA



RYS. 4. Histogram czasu traconego przez klienta na zakupy



RYS. 5. Histogram czasu traconego przez klienta na zapłatę

Opracowany model symulacyjny sklepu można w prosty sposób rozbudowywać. Przykładowo, niewielkich zmian w programie wymaga uwzględnienie przerw w pracy kasy sklepowej. Segment symulujący te przerwy został przedstawiony na wydruku 2. Kasa średnio co 60±5 minut jest zamykana na 10±2 min. (oba rozkłady równomierne). Po dołączeniu segmentu trzeciego do dwóch pozostałych model będzie uwzględniać opisane przerwy.

Analiza rezultatów symulacji obu modeli (tj. bez i z przerwami w pracy kasy) pokazuje, że pomimo krótkiego średniego czasu pobytu w sklepie (ok. 19 min. dla sklepu bez przerw w pracy kasy i ok. 21 min. - z przerwami) w pewnych przypadkach może on przekraczać 40 min. ("ogon" rozkładu na rys. 4). Analogiczna uwaga dotyczy również czasu traconego na zapłatę w kasie (czasy średnie: 4 i 6 min., a maksymalne: powyżej 18 min.). Zjawisko to występuje przy stosunkowo niewielkich obciążeniach urządzeń obsługujących. Obciążenie koszyków wynosi bowiem 0.132 (w drugim przypadku wzrasta do 0.145), natomiast kasy: 0.314 (i 0.480). Dokładna analiza histogramów (rys. 4 i 5) pozwala określić, jaki procent klientów pozostaje w sklepie np. dłużej niż 30 min.

```
260 *
270 *   Segment 3 - przerwy w pracy kasy
280         GENERATE     60,5,,,1              ;przerwy co 1 h
290         SEIZE        KASA                  ;zamknięcie kasy
300         ADVANCE      10,2                  ;na 10 min.
310         RELEASE      KASA                  ;otwarcie kasy
320         TERMINATE
```

WYDRUK 2. Segment programu - model przerw w pracy kasy sklepu

6. Obszary zastosowań symulacji procesów dyskretnych

Symulacja procesów dyskretnych może być użyta do rozwiązania całego szeregu problemów rzeczywistych, związanych z:
- funkcjonowaniem obiektów technicznych i technologicznych;
- analizą niezawodności strukturalnej systemów;
- badaniem systemów organizacyjnych i ekonomicznych.

Ponadto metoda ta może być użyta do prezentacji różnych praw statystyki i takich specyficznych zastosowań jak całkowanie metodą Monte Carlo czy też analiza systemu pasowań w budowie maszyn.

Do ciekawszych problemów technicznych rozwiązywanych metodą symulacji należą następujące przykłady *(case studies)*:
- badania obciążenia systemu telefonicznego [10];
- wybór optymalnego sposobu funkcjonowania systemu pomp zasilających miasto w wodę [11];
- symulacja systemów komputerowych w celu określenia ich optymalnych parametrów [3], [9], [12], [13];
- analiza funkcjonowania systemu kontroli jakości i sortowania układów scalonych [5];
- optymalizacja struktury systemu obsługi technicznej parku maszyn budowlanych [14] i innych [5];
- optymalizacja schematów eksploatacji maszyn [5], [15], [16], [17];
- organizacja remontów i przeglądów technicznych maszyn, samochodów itp. [9], [5];
- badanie charakterystyk automatycznych linii produkcyjnych, robotów przemysłowych, montażu elementów itp. [18].

Ocena niezawodności realnych systemów maszyn i urządzeń na podstawie niezawodności ich elementów składowych jest zadaniem rozwiązywanym analitycznie tylko dla szczególnych, prostych przypadków. Przypadki te dotyczą systemów o całkowicie niezależnych elementach, o nieskomplikowanych strukturach niezawodnościowych i standardowych rozkładach czasów trwania stanów zdatności (pracy) i niezdatności (awarii) poszczególnych elementów składowych. Symulacja procesów dyskretnych jest jedyną metodą pozwalającą na analizę niezawodności strukturalnej systemów z uwzględnieniem:
- rezerwowania elementów w systemie [9], [19];
- skomplikowanej, mostkowej struktury niezawodnościowej systemu [5], [9];
- elementów buforujących wpływ awarii elementów systemu na stan niezawodnościowy systemu [19].

Obszary zastosowań symulacji procesów dyskretnych do analizy zjawisk i wyznaczania parametrów funkcjonowania różnorodnych systemów to przede wszystkim:
- wyznaczanie parametrów funkcjonowania sklepów, banków, stacji benzynowych, zakładów usługowych, bufetów itp. systemów masowej obsługi [5], [8], [9], [10], [12];
- analiza systemów sterowania zapasami materiałów i części zamiennych [5], [17], [20], [21];
- badanie systemów transportu (drogowego, kolejowego, morskiego i lotniczego) [5], [8], [22];
- analiza stochastycznych sieci czynności dla celów planowania i kontroli realizacji przedsięwzięć w warunkach niepewności czasów wykonania czynności, zakresów robót i ograniczoności zasobów [5], [12], [13], [17], [21];
- dobór optymalnej strategii utrzymania sprawności samochodów, budynków, oświetlenia itp. [5], [14], [20].

7. Banki i generatory modeli symulacyjnych

Efektywne wykorzystanie metod symulacyjnych jest procesem twórczym na praktycznie wszystkich etapach badań. Wymaga zatem z jednej strony dość wysokich kwalifikacji i doświadczenia osób zajmujących się tą dziedziną, a z drugiej

strony jest bardzo pracochłonne. Istnieją grupy problemów dla których można albo zbudować sparametryzowane, typowe modele symulacyjne, albo też generatory modeli-programów.

Pierwsza alternatywa doprowadza do powstania tzw. banku modeli. Jest to przeważnie zbiór określonej liczby programów symulacyjnych (tzw. symulatorów) wyposażony w język manipulacji metodami i język opisu wyników (tekstowych i graficznych). Symulatory są zwykle silnie sparametryzowane. Rola użytkownika sprowadza się do wyboru metody, jej konkretyzacji (określenia wartości parametrów) i zdefiniowania postaci rezultatów. Przykładem banku modeli jest system opracowany na komputerze ODRA-1305 w Poznaniu [23].

W pewnych przypadkach nie jest możliwe użycie gotowych programów symulacyjnych. Ma to miejsce wszędzie tam, gdzie parametrem modelu jest jego struktura. Typowym przykładem takiego systemu jest analiza stochastycznych sieci czynności lub też elastycznych systemów produkcyjnych. Poza tym opracowanie modelu symulacyjnego np. dla dużej sieci czynności jest bardzo pracochłonne i monotonne. Łatwo jest przy tym popełnić błąd bardzo trudny do wykrycia. Generatory modeli symulacyjnych na podstawie opisu struktury systemu (np. przy pomocy sieci GAN lub Petriego) tworzą w sposób automatyczny model-program. Przykładem takich rozwiązań jest generator modeli symulacyjnych stochastycznych sieci planowania przedsięwzięć z ograniczonymi zasobami - GMSSP, opracowany w Katedrze Informatyki Politechniki Lubelskiej lub generator modeli systemów produkcyjnych - AMMG [24]. Rezultatami pracy generatorów są kompletne programy symulacyjne gotowe do bezpośredniego użycia.

8. Uwagi końcowe

Problemy symulacji cyfrowej procesów dyskretnych nie są czymś nowym. Ich rozwój przebiega równolegle z rozwojem bazy technicznej - sprzętu komputerowego. Wraz z rozwojem i upowszechnieniem mikrokomputerów stosowanie tej techniki badania systemów stało się tańsze i prostsze. Totalna informatyzacja niemalże wszystkich dziedzin działalności człowieka zlikwiduje największy problem wiążący się z symulacją systemów: zdobycie danych statystycznych o badanych systemach i ich elementach. Dane te będą gromadzone i przechowywane w systemach komputerowych w ramach funkcjonowania systemów ewidencji, rozliczeń itp. Trzeba je będzie tylko z nich wydobyć.

W dobie zachłyśnięcia się procesorami tekstów, programami graficznymi i bazami danych warto, moim zdaniem, przypomnieć sobie (lub poznać) symulację procesów dyskretnych i jej możliwości.

Literatura

[1] *Analiza systemowa - podstawy i metodologia*, pod red. Wł. Findeisena, PWN, Warszawa 1985.

[2] Kącki E.: *Symulacja komputerowa - metody, środki, zastosowania*, Materiały V Krajowej Konferencji Informatyków, Poznań 1985.

[3] Tyszer J.: *Symulacja cyfrowa*, WNT, Warszawa 1990.

[4] Miłosz M.: *Mikrokomputerowa symulacja systemów - GPSS/PC*, Informatyka, 3 , 1993, str. 11-16.

[5] Schriber T. J.: *Simulation Using GPSS*, John Wiley & Sons, New York 1974.

[6] *GPSS/PC. Reference Manual*, MINUTEMAN Software, Stow, 1992.

[7] Henriksen J. O., Crain R. C.: *GPSS/H Reference Manual*, Wolverine Software Corporation, Annandale 1989.

[8] Ståhl I.: *Introduction to Simulation with GPSS on the PC, Macintosh and VAX*, Prentice Hall, London 1990.

[9] Chisman J. A.: *Introduction to Simulation Modeling Using GPSS/PC*, Prentice Hall, New

Jersey 1992.

[10] Gordon G.: *Symulacja systemów.* WNT, Warszawa 1974.

[11] Hyden J.W., Chisman J. A., Macaulay H. H.: *Simulation of Water Quality Management Policies*, Journal of The Environmental Engineering Division, ASCE , Vol. 101, No. 4, pp. 623-641, 1975.

[12] Thesen A., Travis L. E.: *Simulation for Decision Making*, West Publishing Company, St. Paul 1992.

[13] Karian Z. A., Dudewicz E. J.: *Modern Statistical, Systems, and GPSS Simulation. The First Course*, Computer Science Press, New York, 1991.

[14] Kudriawcew E. M.: *Kompleksnaja mechanizacja, awtomatizacja i mechanowoorużonnost stroitelstwa*, Strojizdat, Moskwa 1989.

[15] Miłosz M.: *Symulacyjna optymalizacja schematów eksploatacji maszyn budowlanych*, Mat. III Konferencji Towarzystwa Naukowego Inżynierii Procesów Budowlanych, t. 3, Warszawa 1992.

[16] Kapliński O., Kamiński R., Pawlak G.: *Komputer IBM PC w przedsiębiorstwie budowlanym i na budowie*, TNOiK, Poznań 1991.

[17] Miłosz M., Sobotka A.: *Modelowanie symulacyjne procesów budowlanych*, Politechnika Lubelska, Lublin 1992.

[18] Chisman J. A.: *Simulation Modeling for Automated Manufacturing*, Proceedings of AM-86. Greenville 1986.

[19] Korbel L., Miłosz M.: *Model symulacyjny systemu maszyn o strukturze szeregowej do oceny niezawodności*, Materiały sympozjum naukowego: Kierunki Modernizacji Górnictwa, t. 2, Kraków 1991.

[20] Hillier F. S., Lieberman G. J.: *Introduction to Operations Research*, McGraw-Hill Publishing Company, 1986.

[21] Radzikowski Wł.: *Komputerowe systemy wspomagania decyzji*, PWE, Warszawa 1990.

[22] Chisman J. A.: *Traffic Planning Simulation Training Model*, ORSA/TIMS Joint Meeting, Denver 1988.

[23] Waćkowski K.: *Bank metod dla potrzeb zarządzania*, Akademia Ekonomiczna w Poznaniu, Poznań 1985.

[24] *Automatic Model Generator*, The MINUTEMAN Messenger, Stow, number 7, 1992.

Artur Kasprzyk
Student V roku informatyki
na Uniwersytecie Wrocławskim

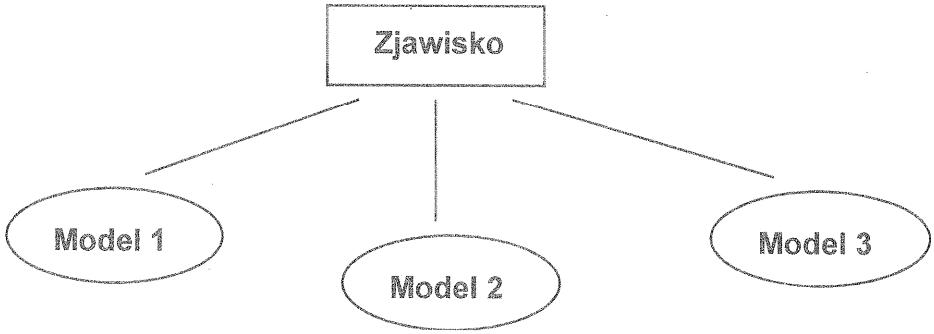
Obiektowo-zorientowane analiza i projektowanie

Cykl życia systemu informatycznego rozpoczyna się od momentu sformułowania problemu. Następnymi etapami są: analiza, projektowanie, implementacja, testowanie. W tym momencie gotowy produkt zostaje oddany do rąk użytkownika. Nie kończy się jednak cykl jego życia. Rozpoczyna się bowiem etap jego pielęgnacji i rozszerzania (może to być, i z reguły jest, proces cykliczny).

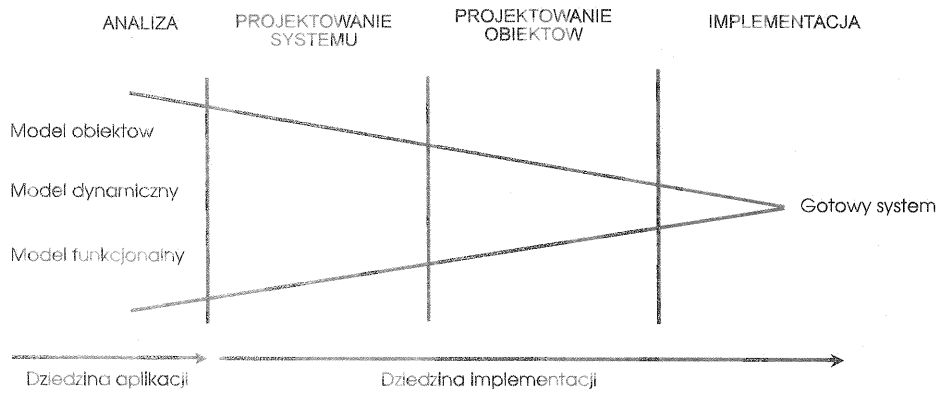
Celem rozważań niniejszej pracy będą początkowe z wymienionych etapów - obiektowo-zorientowane analiza i projektowanie[1]. Jest to ta część cyklu życia systemu, która rzutuje na całą przyszłą jego strukturę oraz zachowanie. Jej efektem jest gotowy model systemu. Modelowanie problemu nie jest rzeczą nową. Od stuleci inżynierowie i naukowcy najpierw budowali modele, by po ich akceptacji rozpocząć konstrukcję konkretnej rzeczy. Testowanie modelu jest o wiele prostsze i *mniej kosztowne* niż testowanie samego produktu końcowego. Nie ma już zakładów motoryzacyjnych, które najpierw budowałyby samochód, by dopiero potem testować jego działanie. Do tego służą komputerowe symulacje różnorodnych procesów. Co więcej, dzięki modelom możliwe jest zmniejszenie złożoności całego przedsięwzięcia poprzez podział na mniejsze części. Ta właściwość jest jednak odczuwalna dopiero przy projektowaniu większych systemów i być może stąd wynika powszechne lekceważenie tych etapów.

Każdy model jest pewną abstrakcją opisywanego wycinka rzeczywistości. Abstrakcje w swojej naturze są niekompletne, opisują jedynie pewien sposób widzenia zjawiska. Dzięki temu w znacznym stopniu je upraszczają. Nie oznacza to, iż przez to tracona jest jakaś wiedza o zjawisku, gdyż można skonstruować kilka jego różnych modeli (rys. 1). Przykładem na to może być instrukcja obsługi radioodbiornika, w której jest on opisany za pomocą schematu, rysunku poglądowego oraz słownego opisu (zauważmy, że słowa też są jedynie abstrakcjami,

[1] W dalszej części pracy będą one nazywane modelowaniem.



RYS 1. Każdy model dotyczy jednego aspektu rozpatrywanego zjawiska



RYS. 2. Kolejne składowe metodologii OMT

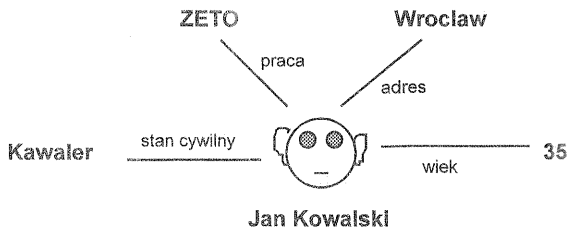
a jednak spełniają swoje zadanie znakomicie).

Object Modeling Technique (OMT)

Jedną z obiektowo-zorientowanych metodologii tworzenia systemów informatycznych jest Object Modelling Technique. Jak można się domyślić, podstawowym elementem tej techniki jest obiekt, na który składają się struktury danych oraz operacje, a jej celem jest identyfikacja i organizacja obiektów występujących w dziedzinie aplikacji (nie implementacji !!). Cały proces modelowania jest przeprowadzany niezależnie od jakiegokolwiek języka programowania. Co ciekawsze, nie zakłada on, iż system będzie zaimplementowany w języku obiektowym. Można z powodzeniem wykorzystać do tego celu języki proceduralne oraz systemy baz danych.
Metodologia OMT składa się z czterech podstawowych etapów: analizy, projektowania systemu, projektowania obiektów oraz implementacji[2] (rys. 2). W każdym z nich stworzone na etapie analizy modele obiektów, dynamiczny oraz funkcjonalny przechodzą swoistą ewolucję. Modele odzwierciedlają konstruowany system widziany na trzy różne, choć powiązane ze sobą, sposoby. Model obiektów odpowiada statycznej strukturze aplikacji. Dostarczaon informacji na temat "wydobytych" z rzeczywistości obiektów, ich atrybutów i operacji, a także wzajemnych powiązań. Model ten powinien zawierać tylko te obiekty, które są ważne z punktu widzenia aplikacji. Model obiektów w metodologii OMT jest reprezentowany graficznie za pomocą diagramów, składających się z klas (ich atrybutów oraz operacji) tworzących hierarchie oraz powiązanych ze sobą wiązaniami. Model obiektów jest strukturą, nad którą konstruowane są dwa pozostałe modele.
Model dynamiczny odpowiada zmianom stanu aplikacji w czasie. Podstawowymi pojęciami związanymi z nim są zdarzenie *(ang. event)* oraz stan *(ang. state)* - rozumiany jako kolekcja powiązań obiektu[3] z innymi obiektami. Sposób reakcji na

[2] Metodologia OMT została zaproponowana przez J. Rumbaugh'a, Michaela Blaha, Williama Premerlani, Fredericka Eddy'ego oraz Williama Lorensena (General Electric Research and Development Center Schenectady, New York). Składa się z czterech etapów. Ostatni z nich - implementacja - został w pracy pominięty z racji tego, iż przejście od projektowania do implementacji wydaje się być bardzo naturalne.

[3] Rysunek poniżej opisuje obiekt Jan Kowalski w postaci jego powiązań z innymi obiektami (inaczej mówiąc - wartościami jego atrybutów). Stanem obiektu jest w tym przypadku kolekcja (35,Wrocław,ZETO,Kawaler).



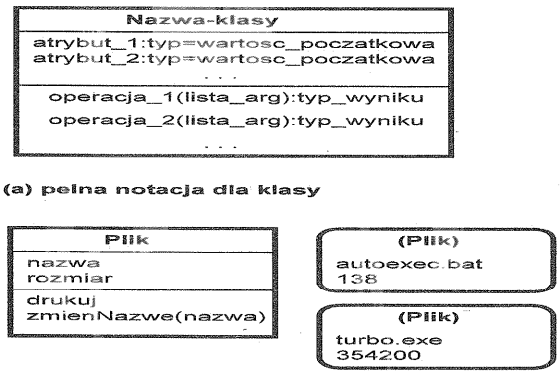
zdarzenie zależy od stanu, w jakim obiekt się znajduje. Efektem zdarzenia może być (choć nie musi) zmiana stanu obiektu. Graficzną reprezentacją modelu dynamicznego jest zbiór diagramów stanów *(ang. state diagram)* -skończonych automatów deterministycznych, działających niezależnie od siebie. Każdy diagram odpowiada klasie, której zachowanie w czasie jest istotne z punktu widzenia analizy systemu. Ostatnim modelem jest model funkcjonalny. Przedstawia on obliczenia dokonywane w systemie (kolejne przekształcenia wartości wejściowych) bez względu na to, kiedy są one wykonywane. Graficznym przedstawieniem modelu funkcjonalnego są diagramy przepływu danych *(ang. data flow diagram)*.
Jak widać, z każdym modelem jest związana jego graficzna reprezentacja. Poniżej zostaną przybliżone ich podstawowe składowe.

Model obiektów

Jak wspomniano wcześniej, na model obiektów składają się obiekty wraz z atrybutami i operacjami oraz powiązania między nimi.

(1) Obiekty i klasy.

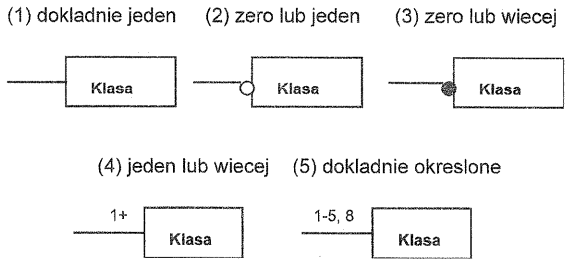
Graficzną reprezentację pojęć pokazuje rys. 3.



RYS. 3. Notacja dla klas i obiektów

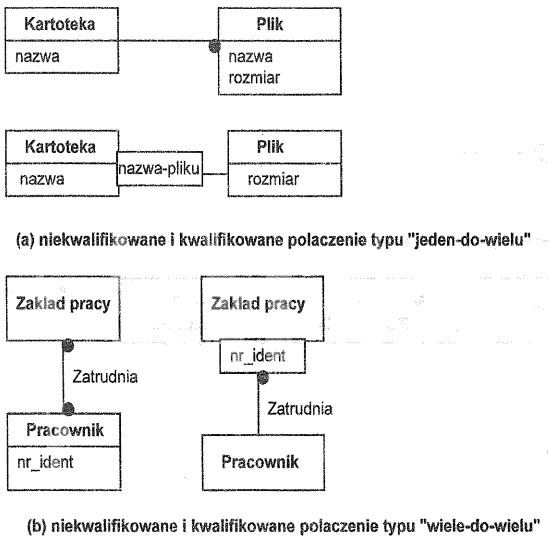
(2) Połączenia *(ang. link)*, powiązania *(ang. association)* i kompozycja *(ang. composition, aggregation)*.

Pomiędzy połączeniem i powiązaniem istnieje taka sama różnica jak między obiektem i klasą. Połączenie wiąże ze sobą dwa (lub więcej) obiekty. Na przykład, obiekty **Jan Kowalski** i **Wrocław** wiąże połączenie *Mieszka*. Jednak rozpatrując klasy **Osoba** i **Miasto**, związek *Mieszka* nazwiemy powiązaniem, a nie połączeniem.
Powiązania są z natury dwustronne. Jeżeli osoba *mieszka* w mieście, to można, z drugiej strony, stwierdzić iż miasto jest *zamieszkane* przez osoby. Warto w tym miejscu zwrócić uwagę na jeszcze jedną sprawę. Otóż, o ile jedna osoba może aktualnie mieszkać tylko w jednym mieście, o tyle miasto może być zamieszkane przez wiele osób. Wielowartościowe powiązania przedstawia rysunek 4.



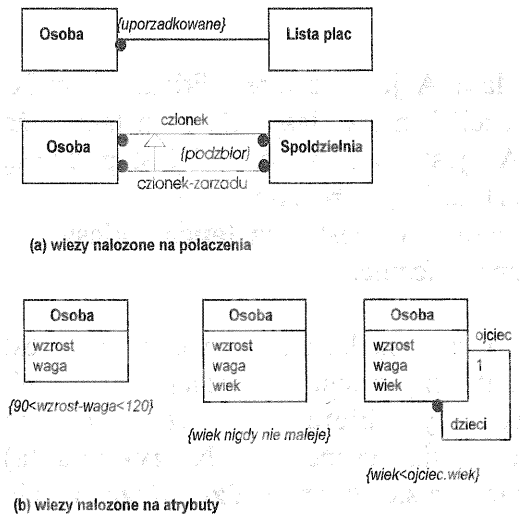
RYS. 4. Różnowartościowość powiązań

Wartościowość powiązania można zmniejszyć z "wiele" do "jeden" przez zastosowanie powiązania kwalifikowanego *(ang. qualified association)*; dotyczy to powiązań "wiele-do-wielu" oraz "jeden-do-wielu". Sposób tworzenia powiązań ilustruje rysunek 5. Zauważmy, iż w przypadku (b) zastosowanie kwalifikatora daje dodatkową korzyść. Ponieważ pracownik może pracować w więcej niż jednym zakładzie pracy, w każdym z nich może mieć przydzielony inny numer identyfikacyjny. W przypadku połączenia niekwalifikowanego, zawarcie tej informacji w klasie Pracownik byłoby z pewnością utrudnione.
Czasami jest pożądane oznaczenie obie-



RYS. 5. Połączenia kwalifikowane i niekwalifikowane

któw, które na końcu wiązania "wiele " są w jakiś sposób uporządkowane. Przykładem może być lista płac, na której umieszczono w porządku alfabetycznym osoby. Uporządkowanie jest specjalnym przypadkiem więzów, jakie można nakładać na wartości atrybutów oraz połączenia. Pojęcie to jest znane z problematyki baz danych. Przykłady więzów przedstawia rysunek 6.



RYS. 6. Więzy można nakładać na połączenia i atrybuty

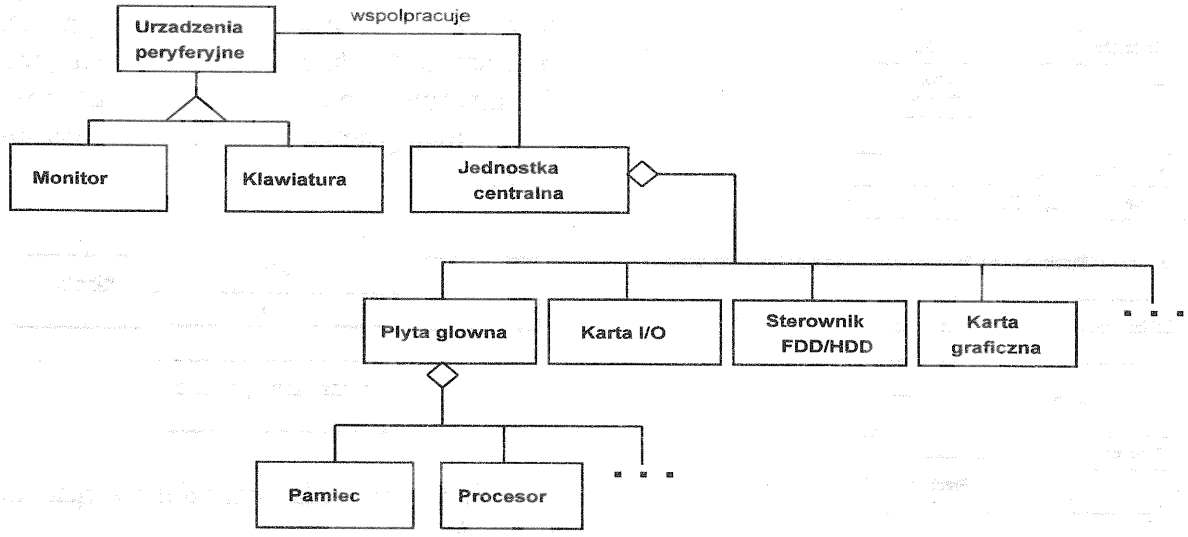
Wiązania, podobnie jak klasy, mogą również posiadać atrybuty. W przypadku powiązania *Osoba mieszka w Mieście* (rys. 7), atrybutem powiązania *mieszka* może być **rodzaj_zameldowania** (stały, czasowy).

RYS. 7. Atrybut połączenia dla powiązania wiele-do-jednego

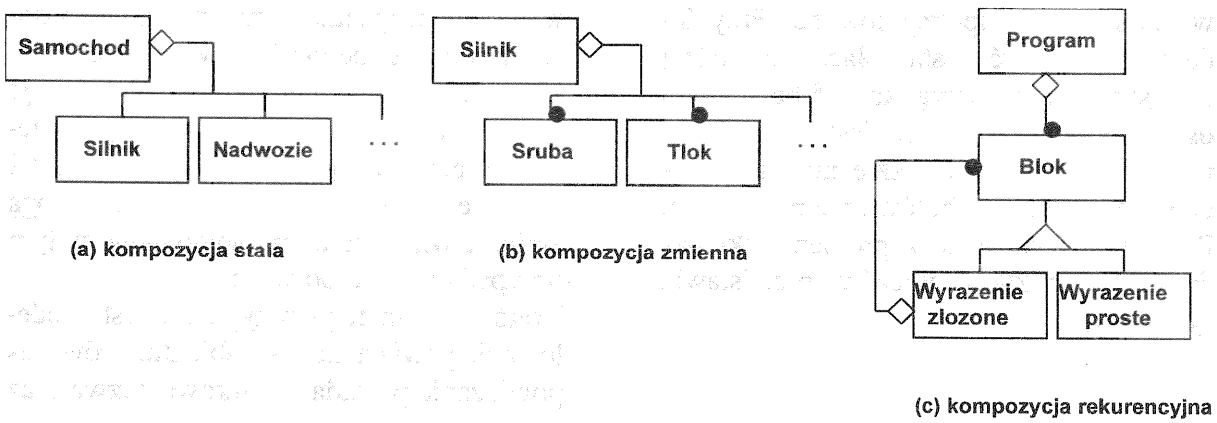
Takie rozwiązanie jest bardziej naturalne niż przyporządkowanie atrybutu rodzaj_zameldowania którejś z klas; co więcej, w przypadku powiązania wiele-do-wielu, dołączenie atrybutów połączenia do którejś z klas jest niemożliwe. Widać więc jasno, iż dołączenie atrybutów do połączenia zwiększa elastyczność systemu, gdyż przyszłe rozszerzenia systemu mogą zmienić wiązanie do niewygodnego w tym przypadku wiele-do-wielu.

Uogólnieniem tego przypadku jest modelowanie powiązania jako obiektu. Wówczas powiązanie posiada dodatkowo nazwę oraz operacje.

Specjalnym przypadkiem powiązania jest kompozycja. Można intuicyjnie przyjąć, że kompozycja jest połączeniem w rodzaju *składa-się, jest-zbudowane* itp. Obiekt, który jest *skomponowany* z innych składowych, będzie dalej nazywany agregatem *(ang. aggregate)*. Różnica między kompozycją a zwykłym połączeniem jest subtelna; polega ona na tym, że agregat jest traktowany semantycznie jako jeden obiekt. Przenosząc jednostkę centralną ze stołu na biurko nie zastanawiamy się czy osobno przenieść sterowniki, a osobno kartę graficzną. Przykład agregatu (klasa **Jednostka centralna**) przedstawia rysunek 8. Widać, iż kompozycja jest relacją przechodnią. Z tego, że jednostka centralna składa się z płyty głównej, a w skład płyty głównej wchodzą kości pamięci, wynika,



RYS. 8. Kompozycja obiektów



RYS. 9. Różne rodzaje kompozycji

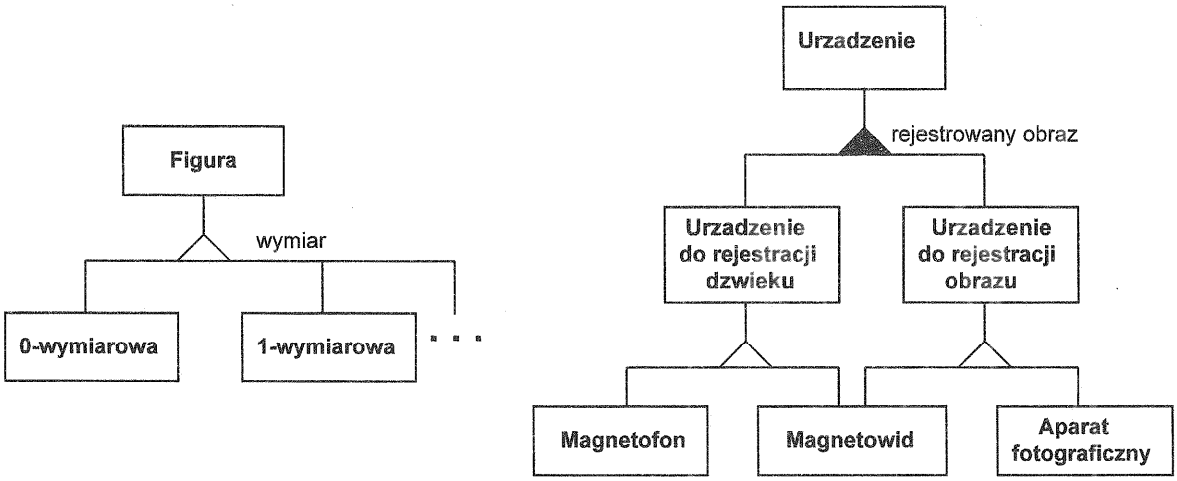
że kości pamięci wchodzą również w skład jednostki centralnej. Kompozycje można podzielić na stałe, zmienne oraz rekurencyjne. Różnice przedstawia rysunek 9.

(3) Dziedziczenie i generalizacja.

Generalizacja odzwierciedla związki pomiędzy klasami (często nazywanymi superklasami - *ang. superclass*) oraz ich podklasami *(ang. subclass)*. Dziedziczenie jest implementacją generalizacji. Ponieważ generalizacja jest związkiem przechodnim, wprowadza się pojęcia następcy *(ang. descendant)* oraz poprzednika *(ang. ancestor)*. Znaczenia tych pojęć są następujące:

* klasa A jest **poprzednikiem** klasy X, jeżeli istnieją klasy A1,...,An takie , że A1 jest podklasą A, A2 jest podklasą A1,..., X jest podklasą An,
* klasa A jest **następcą** klasy X analogicznie.

Wprowadzoną dla generalizacji notacją jest trójkąt łączący superklasę z jej podklasami. Przykłady hierarchii dziedziczenia przedstawia rysunek 10. Na rysunku (a) widać proste drzewo dziedziczenia dla wielowymiarowych figur. Klasą bazową jest **Figura**, a jej podklasami figury **0-wymiarowe**, **1-wymiarowe**, **2-wymiarowe** itd. Trzykropek oznacza, iż przedstawiona hierarchia nie jest kompletna, gdyż



RYS. 10. Wielopoziomowe hierarchie dziedziczenia

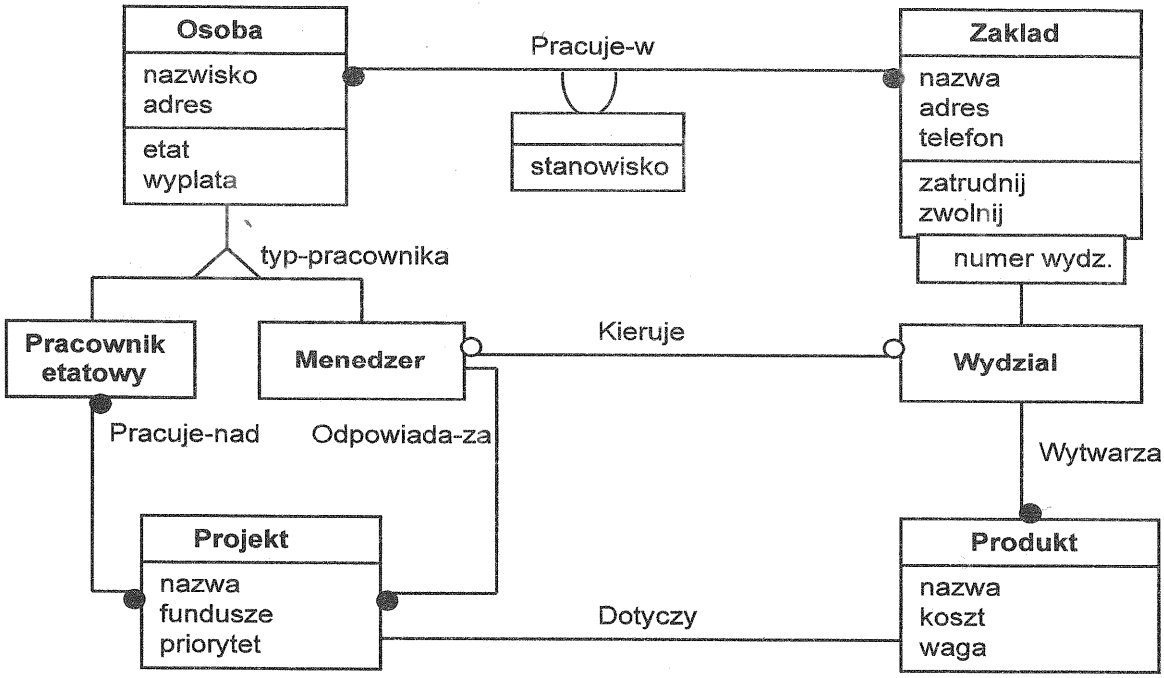
z jakichś względów nie przedstawiono jej dalszej części (być może nie jest to ważne z punktu widzenia aplikacji). Słowo "wymiar", położone przy symbolu generalizacji, jest nazywane dyskryminatorem *(ang. dyscriminator)*. Określa ono właściwość, która została wzięta pod uwagę podczas dokonywania podziału. Na rysunku (b) widać hierarchię dziedziczenia dla urządzeń audiowizualnych. Klasą bazową jest klasa **Urządzenie**. Z niej wyprowadzone są pozostałe typy. Zaczerniony trójkąt oznacza, iż dokonany podział nie jest rozłączny *(ang. disjoint)*, tzn., że istnieją klasy obiektów , będące zarówno urządzeniami do rejestracji dźwięku jak i obrazu (np. magnetowid). Każda klasa leżąca poniżej klasy bazowej jest jej następnikiem, ale tylko **Urządzenie do rejestracji dźwięku** oraz **Urządzenie do rejestracji obrazu** są jej podklasami. Klasa **Magnetowid** jest przykładem klasy łączonej - mającej więcej niż jedną superklasę - *(ang. join class)*.

Przedstawione powyżej koncepcje stanowią podstawowe narzędzia potrzebne do stworzenia prostego modelu obiektów. Proces jego tworzenia może wyglądać następująco:

1. Ze specyfikacji zadania wybierz kandydatów na obiekty.
   Najprostszą metodą jest wybranie rzeczowników ze słownego opisu zadania i wstępne przyjęcie ich jako kandydatów na obiekty. Następnie należy wyeliminować te z nich, które dotyczą implementacji oraz nie są ważne z punktu widzenia tworzonego systemu.
2. Dodaj do obiektów atrybuty i metody.
3. Utwórz połączenia między obiektami. Jeżeli jest to konieczne, dodaj do nich atrybuty lub przedstaw w postaci obiektów.
4. Uprość diagram stosując generalizację.
5. Podziel klasy na spójne pod względem świadczonych usług moduły.

Przykład: Mając dany opis słowny problemu, utwórz dla niego model obiektów.

Opis zadania.
Zakład składa się z wielu wydziałów, które są ponumerowane w sposób jednoznaczny. Większość z wydziałów ma własnego menedżera. Jednak nie każdy z zatrudnionych w zakładzie menedżerów zarządza własnym wydziałem. Każdy wydział wytwarza przynajmniej jeden produkt, zawsze od początku do końca. Produkt jest najpierw projektowany przez pracowników zakładu. Nad poprawnością każdego projektu czuwa jeden menedżer. Z każdym



RYS. Model obiektów dla postawionego problemu

wytwarzanym produktem są związane koszt, nazwa oraz waga.

Rozwiązanie.

1. Proponowane obiekty: **zakład**, **wydział**, **menedżer**, **produkt**, **pracownik**, **projekt**.
2. Proponowane połączenia:
   Osoba **pracuje** w zakładzie.
   Pracownik **pracuje nad** projektem.
   Menedżer **kieruje** wydziałem.
   Menedżer **odpowiada za** projekt.
   Wydział **wytwarza** produkty.
   Projekt **dotyczy** produktu.
3. Upraszczając diagram wprowadzono dodatkowy obiekt **Osoba**, którego podklasami są **Pracownik** oraz **Menedżer**.

Model dynamiczny.

Zadaniem modelu dynamicznego jest przedstawienie zmian, jakim podlegają obiekty w trakcie pracy systemu. Podstawowymi pojęciami, na których opiera się model są zdarzenia *(ang. events)*, stany *(ang. states)* oraz przejścia od jednego stanu do innego *(ang. transition)*.

Zdarzeniem jest coś, co wydarzyło się w jakimś momencie. Zdarzenie nie ma czasu trwania - jest jakby impulsem niosącym informację. Zdarzenia mogą być ze sobą przyczynowo powiązane (np. samolot nie może się znaleźć w powietrzu wcześniej niż wystartuje) lub nie (np. opady deszczu w Polsce i Tunezji) - takie zdarzenia nazywane są współbieżnymi *(ang. concurrent)*.

Zdarzenia przenoszą informacje od jednego obiektu do drugiego. Informacją tą może być sam impuls lub impuls z danymi, nazywanymi atrybutem zdarzenia. W zależności od potrzeb zdarzenie będzie rozumiane bądź jako klasa zdarzeń (np. opady deszczu), bądź jako pojedyncze zdarzenie (np. opady deszczu na południu Tunezji).

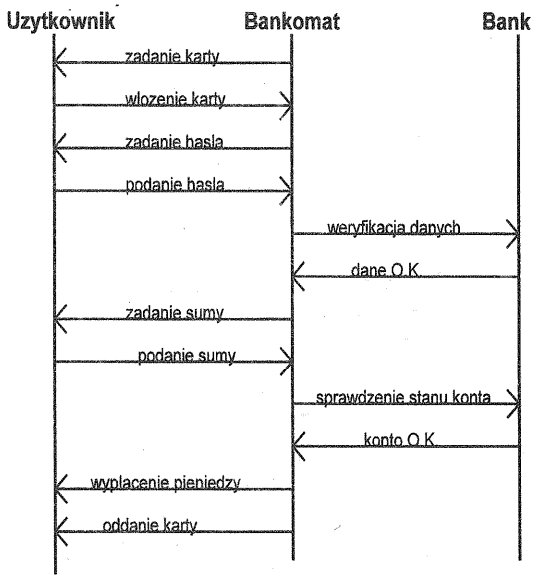
Stan obiektu można rozumieć jako zbiór wartości posiadanych przez niego atrybutów. Niech **Jan Kowalski** będzie obiektem klasy **Osoba**, posiadającym atrybuty **wiek**, **nazwisko**, **imię** i **adres**. Niech aktualnymi wartościami atrybutów będą kolejno 72, Kowalski, Jan, ul. Szewska 1/2, okre-

ślając w ten sposób jego stan. Pojawienie się zdarzenia *zmień-adres(ul. Garncarska 12/13)* spowoduje przejście obiektu do stanu 72, Kowalski, Jan, ul. Garncarska 12/13. Tak więc stan określa reakcję obiektu na pojawienie się zdarzenia. Reakcja ta może być różna dla różnych stanów. Stanom można przypisać ich czas trwania - określa je okres oddzielający pojawienie się dwóch zdarzeń.
Pierwszym etapem konstrukcji modelu dynamicznego jest napisanie scenariusza pracy. Jest to pełny cykl pracy systemu, rozumiany jako sekwencja zdarzeń pojawiających się w trakcie jednego konkretnego uruchomienia systemu. Rysunek 12 przedstawia przykładowy scenariusz dialogu użytkownik-bankomat[4].

Bankomat żąda włożenia karty.
Użytkownik wkłada kartę.
Bankomat żąda podania hasła.
Użytkownik podaje hasło.
Bankomat weryfikuje hasło.
Bankomat żąda podania sumy.
Użytkownik podaje sumę.
Bankomat sprawdza stan konta.
Bankomat wydaje pieniądze.
Użytkownik bierze pieniądze.
Bankomat oddaje kartę.
Użytkownik bierze kartę.

Rys. 12. Scenariusz dialogu użytkownik-bankomat.

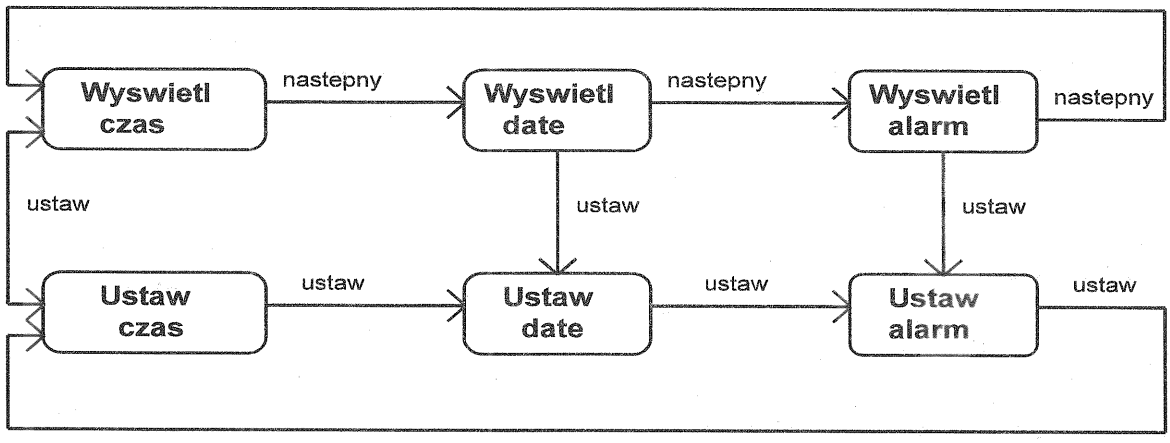
Każde zdarzenie jest transmisją informacji między obiektami. Scenariusz informuje o tym, jakie zdarzenia wystąpiły w czasie sesji, nie mówi natomiast nic o ich nadawcach i odbiorcach. Tę informację przedstawia się na diagramie zdarzeń *(ang. event trace)*. Diagram dla przedstawionego powyżej dialogu ilustruje rysunek 13. Pionowe linie na diagramie oznaczają obiekty, poziome - odpowiadają przepływowi zdarzeń. Czas płynie od góry do dołu diagramu. Diagram zdarzeń jest



RYS. 13. Diagram zdarzeń dla dialogu użytkownik-bankomat

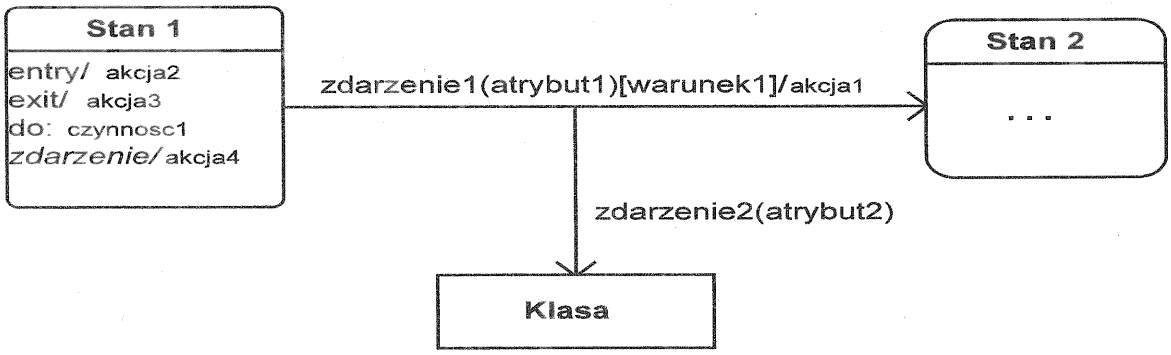
tylko przykładem jednego wykonania programu. Aby opisać cały system, należałoby utworzyć diagramy zdarzeń dla wszystkich możliwych sytuacji, jakie mogą wystąpić w trakcie pracy systemu. Takie podejście wydaje się niezbyt rozsądne.
Rozwiązaniem problemu są diagramy stanu *(ang. state diagram)* - deterministyczne automaty o skończonej liczbie stanów. Stanami są w nich stany obiektu, natomiast przejścia reprezentują zdarzenia. Stany są graficznie reprezentowane przez prostokąty o zaokrąglonych rogach; przejścia oznaczane są strzałkami. Jeżeli obiekt znajduje się w jakimś stanie i pojawi się zdarzenie, które etykietuje jedno z przejść do innego stanu, wówczas obiekt przechodzi do nowego stanu. Jeżeli żadne z przejść nie jest etykietowane nazwą zdarzenia, wówczas jest ono ignorowane. Zdarzenia są obsługiwane w porządku, w jakim nadchodzą. Prosty diagram stanu dla zegarka elektronicznego przedstawia rysunek 14.
Widać, że diagram w takiej postaci nie jest w stanie przedstawić wielu informacji. Bardzo często zdarza się, że wykonanie operacji zależy od jakiegoś warunku, np. "Kiedy dostanę wypłatę i będzie ładna pogoda, pójdę do kina". Przeanalizujmy to zdanie. Podmiot, w momencie otrzymywania wypłaty, nie będzie znajdował się

[4] Przykład zaczerpnięto z książki J. Rumbaugh: Object-Oriented Modeling and Design. Prentice-Hall, Inc., Engelwood Cliffs, New Jersey, 1991. Dla potrzeb pracy został on uogólniony. Zainteresowanych odsyłam do w/w pozycji.

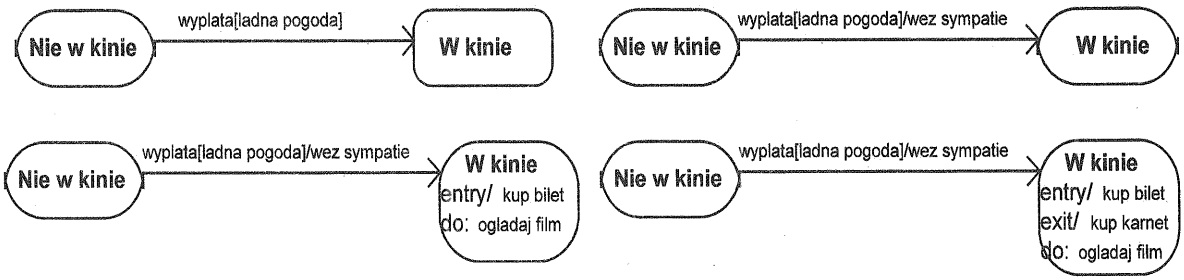


RYS. 14. Diagram stanów dla zegarka elektronicznego

w kinie (będzie to jego aktualny stan, załóżmy "nie w kinie"). Pojawi się zdarzenie "wypłata". Teraz w zależności od tego, czy warunek "ładna pogoda" będzie spełniony, podmiot albo pójdzie do kina (zmieni stan) albo pozostanie w tym samym stanie. Na diagramie warunek jest zapisywany w nawiasach kwadratowych, zaraz za nazwą zdarzenia. Jeżeli Podmiot jest odpowiednio dorosły, wówczas prawdopodobne jest, że do kina nie pójdzie sam, lecz z sympatią. W takim razie, przejściu od jednego stanu do drugiego będzie towarzyszyło wykonanie dodatkowej akcji. Rozpatrywane wcześniej zdanie będzie miało teraz postać: "Kiedy dostanę wypłatę i będzie ładna pogoda, wezmę ze sobą sympatię i pójdę do kina". Zostanie to uwidocznione na diagramie przez zapisanie akcji, poprzedzonej znakiem "/", zaraz za nazwą zdarzenia (lub warunku, jeżeli występuje). Bardziej precyzyjny Podmiot mógłby sobie postanowić: "Kiedy dostanę wypłatę i będzie ładna pogoda, wezmę ze sobą sympatię, pójdziemy do kina, kupimy bilet i poogląda my film". W takim przypadku, wejście do nowego stanu będzie poprzedzone zakupem biletu (w symbolu stanu zostanie to zapisane jako akcja, poprzedzona słowem "entry/"), co więcej, w nowym stanie będzie wykonywana czynność "oglądanie filmu" (akcja zostanie zapisana w symbolu stanu i poprzedzona słowem "do:"). Jeżeli Podmiot planuje przed wyjściem z kina kupić karnet dwuosobowy na Konfrontacje, dodatkową czynnością wykonywaną przed opuszczeniem stanu będzie: "kupno karnetu" ("/exit"+akcja). Może się też zdarzyć, iż film, który obejrzeli, tak bardzo im się spodobał, że postanowili obejrzeć go jeszcze raz. Wobec tego, bez opuszczania kina dokupili jeszcze bilety i obejrzeli film ponownie. Jest to przykład zdarzenia wewnętrznego *(ang. internal action)*, to znaczy takiego, które zostaje "obsłużone" nie powodując przy tym zmiany stanu - co więcej, nawet z niego nie wychodząc. Na diagramie akcja, która ma być wykonana w przypadku wystąpienia zdarzenia jest poprzedzona nazwą tego zdarzenia. Widać więc różnicę pomiędzy zdarzeniem, które wychodzi ze stanu, wchodząc do niego ponownie, a tym, które ze stanu w ogóle nie wychodzi. W pierwszym przypadku, przy każdym wejściu i wyjściu do i ze stanu, zostaną wykonane akcje poprzedzone słowami kluczowymi "entry/", "exit/" i oczywiście "do:". W drugim przypadku, efektem wystąpienia zdarzenia będzie wykonanie akcji poprzedzonych tylko słowem kluczowym "do:". Rysunek 15 przedstawia symboliczny zapis omówionych sytuacji; rysunek 16 jest dokładnym ich zapisem. Na rysunku 15 dodatkowo pokazano graficzny sposób przedstawiania żądań wysyłanych do innych obiektów. Strzałka oznaczająca kierunek przesyłania informacji powinna być, z jednej strony, połączona ze zdarzeniem, które rozpoczyna transmisję (w ten sposób informując, że



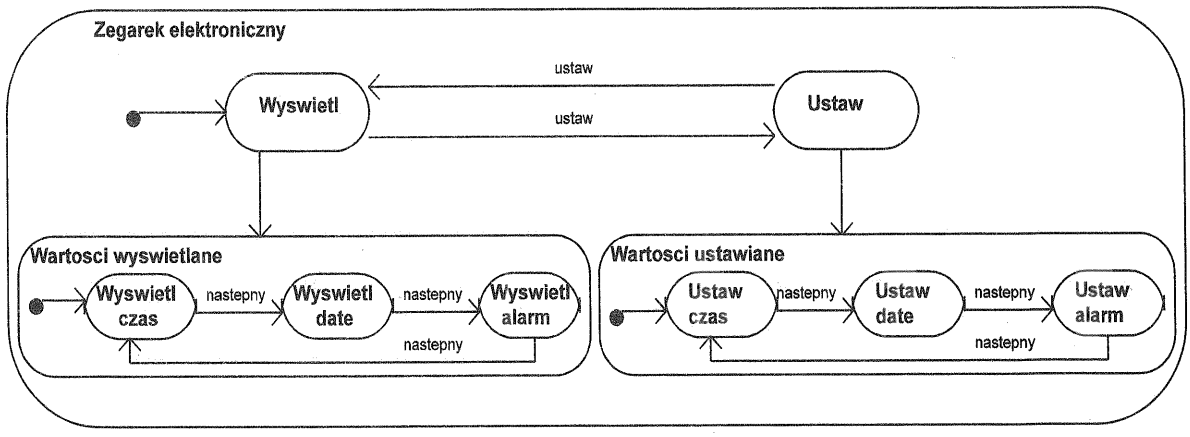
RYS. 15. Graficzna notacja diagramu stanów



RYS. 16. Przykłady diagramów stanów

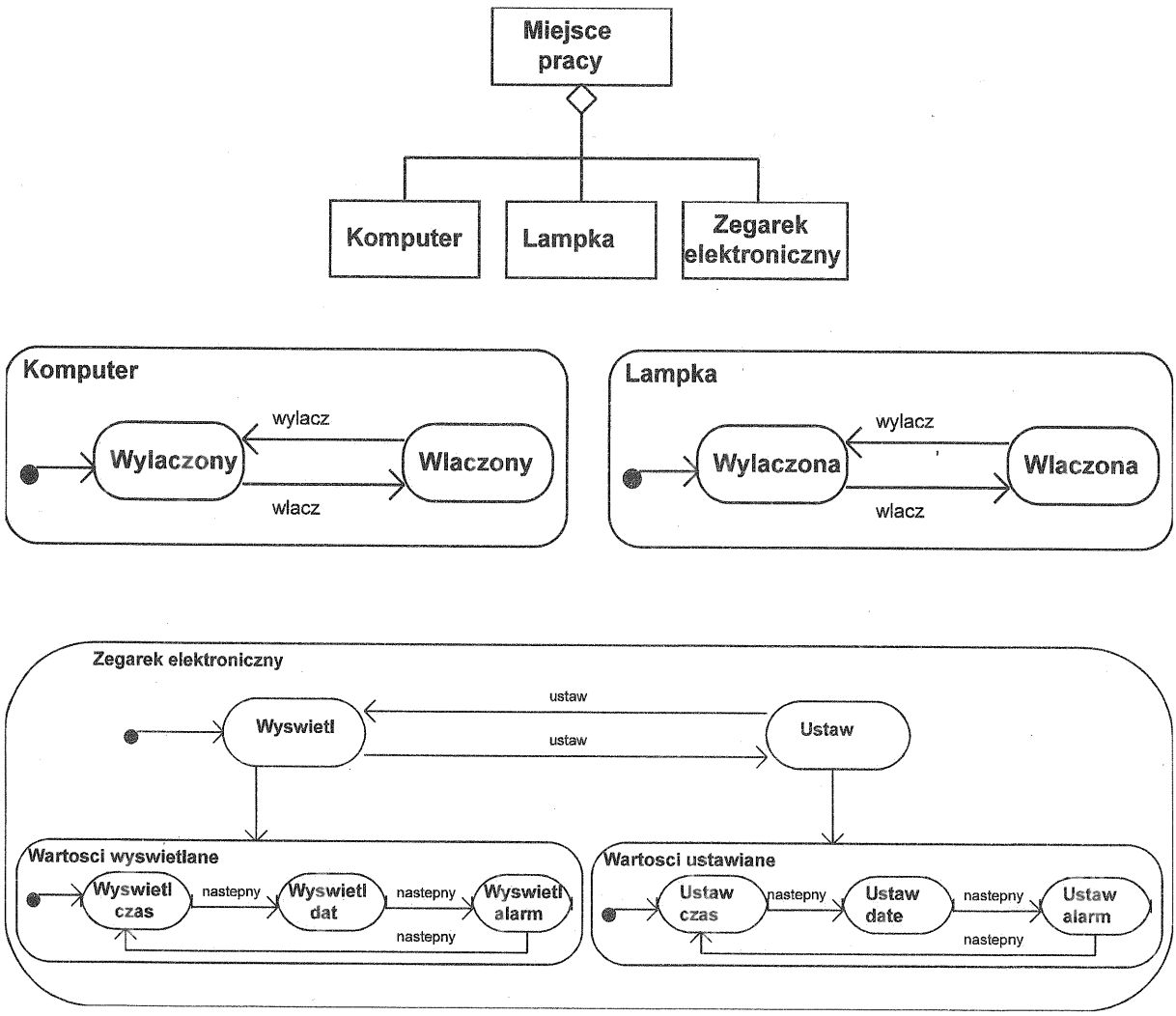
wysłanie zdarzenia2 zależy od pojawienia się zdarzenia1), a z drugiej, ze zdarzeniem w diagramie stanów docelowego obiektu (by pokazać związek między nim a przesłanym zdarzeniem).

Podobnie jak obiekty, stany można grupować w hierarchie lub komponować je z innych stanów. Odpowiednikiem generalizacji z modelu obiektów jest zagnieżdżenie diagramów *(ang. nested diagrams)*, natomiast kompozycji - stany współbieżne *(ang. concurrent state)*. Zagnieżdżanie diagramów umożliwia płynne przejście od rozważań ogólnych do bardziej szczegółowych. Stany diagramów zagnieżdżonych (podstanów) dziedziczą przejścia swoich przodków. Oznacza to, iż każde przejście i każda akcja, mające zastosowanie do jakiegoś stanu, stosują się do wszystkich jego podstanów (oczywiście pod warunkiem, że nie zostaną przedefiniowane). Przykład zagnieżdżonych diagramów przedstawia rysunek 17. Zagnieżdżonymi diagramami są **Wartości wyświetlane** oraz

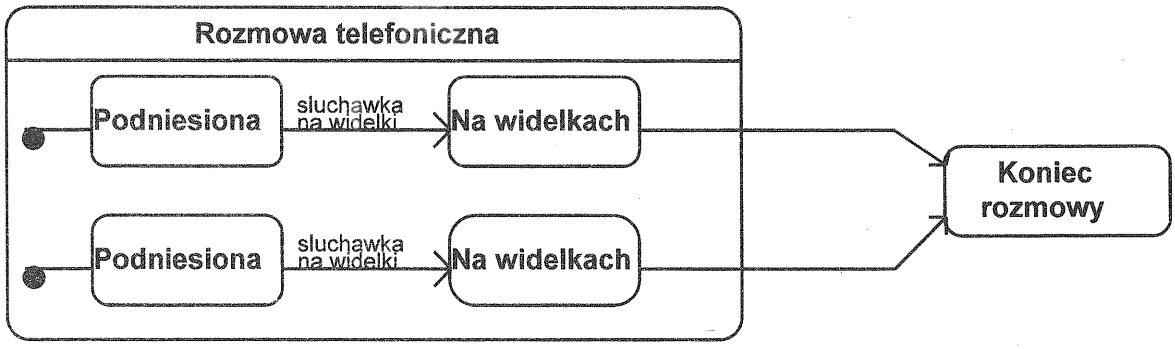


RYS. 17. Prosty diagram zagnieżdżony

Wartości ustawiane. Zaczerniony okrąg na początku przejścia oznacza stan początkowy diagramu.
Stany **Wyświetl czas**, **Wyświetl datę** oraz **Wyświetl alarm** dziedziczą po stanie **Wyświetl** przejście "ustaw". Oznacza to, że będąc w którymkolwiek z podstanów stanu **Wyświetl**, zdarzenie "ustaw" spowoduje przejście do stanu **Ustaw**. Nieetykietowane przejście (ε - przejście) od **Ustaw** do diagramu **Wartości ustawiane** powoduje, że system automatycznie znajdzie się w stanie **Ustaw czas**.
Odpowiednikiem kompozycji w modelu dynamicznym jest współbieżność. Może ona wynikać z dwóch rzeczy. Jeżeli obiekt jest kompozycją kilku innych obiektów (informacja wynika z modelu obiektów), to stan obiektu jest złożeniem stanów jego komponentów. Sytuację przedstawia rysunek 18. Na miejsce pracy składają się obiekty **Komputer**, **Lampka** oraz **Zegarek elektroniczny**. Wobec tego, stan, w jakim znajduje się w danej chwili obiekt **Miejsce pracy** jest sumą stanów poszczególnych jego komponentów.
O współbieżności można mówić również w obrębie jednego obiektu. Pojawia się ona wtedy, gdy obiekt może być podzielony na podzbiory atrybutów i powiązań, z których każdy ma swój własny poddiagram. Wówczas, na stan obiektu składają się stany wszystkich jego poddiagramów. Podział wewnątrz pojedynczego stanu złożonego (oczywiście ze współbieżnymi podstanami), oznacza się podziałem na podstany za pomocą kropkowanych linii. Rysunek 19 przedstawia stan ze współbieżnymi podstanami. W stanie **Rozmowa telefoniczna**, słuchawki osób rozmawiających



RYS. 18. Kompozycja i jej współbieżny diagram stanów

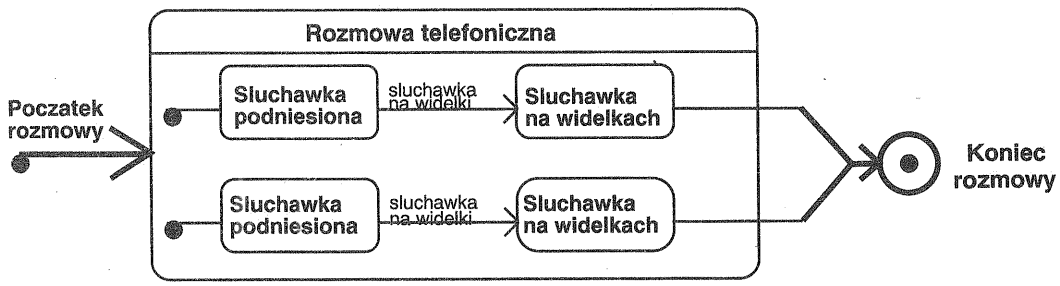


RYS. 19. Rozmowa telefoniczna ze współbieżnymi stanami

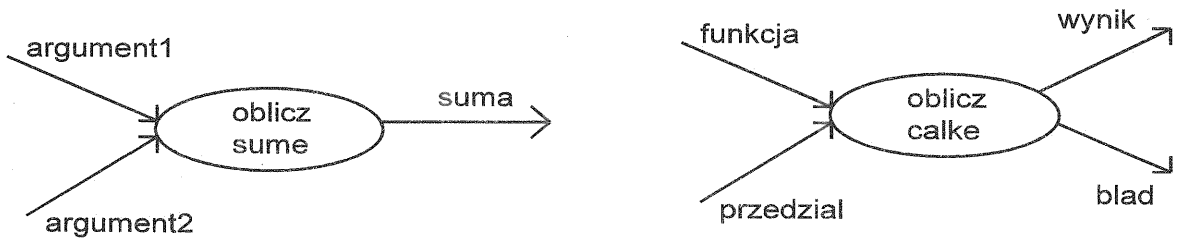
są poniesione. Jeżeli jeden z rozmówców położy słuchawkę na widełki, oznacza to przejście do stanu **Koniec rozmowy**.
Zmieniając założenia rozmowy telefonicznej tak, by jej zakończenie następowało w momencie gdy dwóch rozmówców położy słuchawki, dochodzimy do problemu synchronizacji zdarzeń. Obiekt wyjdzie więc ze stanu **Rozmowa telefoniczna** w momencie, gdy stan słuchawek obydwu rozmówców będzie **Na widełkach**. Graficzny zapis tej sytuacji przedstawia rysunek 20. Wprowadzony dodatkowo element etykietowany **Koniec rozmowy** oznacza stan końcowy diagramu.

Model funkcjonalny

Model funkcjonalny pokazuje rezultaty obliczeń, bez określania, kiedy zostają one wykonane; określa znaczenie operacji w modelu obiektów oraz akcji w modelu dynamicznym. Na model funkcjonalny składa się zbiór diagramów przepływu danych *(ang. data flow diagram)*. W ich skład, z kolei, wchodzą procesy *(ang. process)*, potoki danych *(ang. data flow)*, aktorzy *(ang. actor)* oraz kontenery *(ang. data store)*.
Zadaniem procesów jest przekształcanie danych. Proces otrzymuje dane wejściowe, przetwarza je, produkując w ten sposób dane wyjściowe. Przykładem procesu są wszelkie funkcje - są to procesy najniższego poziomu. Procesem, na wyższym poziomie, można nazwać cały diagram przepływu danych. Przykłady procesów przedstawia rysunek 21. Proces obliczający sumę dwóch argumentów jest bardzo prosty, podczas gdy dokładne numeryczne obliczenie całki może sprawić trochę kłopotów. Symbolem procesu w diagramie przepływu danych jest elipsa, zawierająca opis przekształcenia. Z każdym procesem związana jest określona liczba potoków danych wejściowych i wyjściowych.
Potoki danych łączą ze sobą procesy i obiekty. Początek potoku jest wynikiem działania procesu, koniec potoku - argumentem innego procesu (lub obiektu). Potok danych nie zmienia wartości przenoszonych danych, chociaż może je rozkładać na składowe. Przykłady potoków



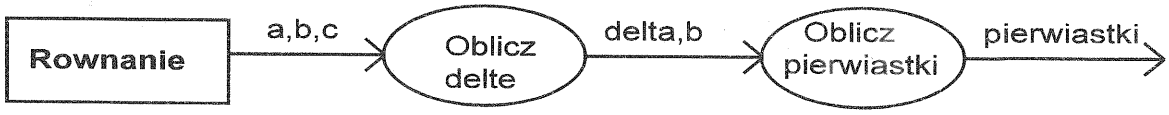
RYS. 20. Rozmowa telefoniczna z synchronizacją współbieżnych stanów



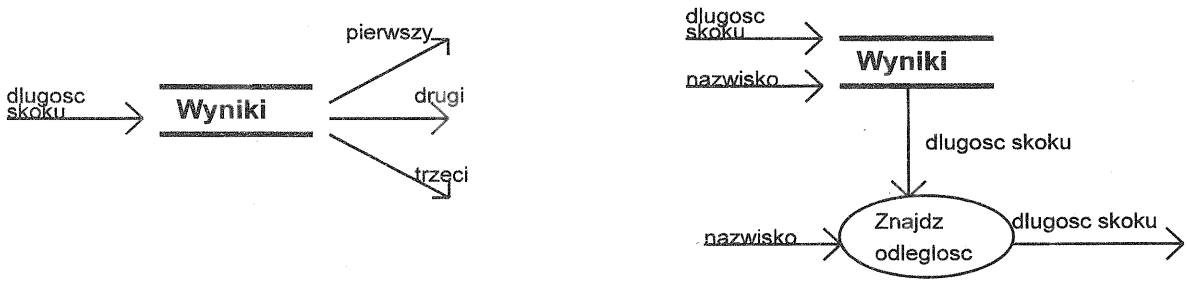
RYS. 21. Przykłady procesów

danych przedstawia rysunek 22.
Potoki danych stanowią wyniki działania określonych procesów. Jeżeli są to końcowe obliczenia, to mogą stanowić wartości znaczące z punktu widzenia aplikacji. Potoki wewnętrzne odpowiadają wartościom tymczasowym i, z reguły, nie stanowią znaczącej informacji. Potoki zewnętrzne (tzn. takie, które leżą na końcach diagramu przepływu), mogą być nie połączone z niczym (wówczas diagram stanowi poddiagram większej struktury) lub mogą być połączone z obiektem. Takie obiekty nazywane są aktorami.
Aktor nie tylko konsumuje, ale może również dostarczać dane do diagramu. Zawsze jednak leży na "skraju" diagramu. Graficzną reprezentacją aktora jest prostokąt z wpisaną wewnątrz nazwą. Na rysunku 23 aktorem jest obiekt **Równanie**.
Ostatnią składową diagramu przepływu danych są kontenery. Są one wykorzystywane do przechowywania danych. Jedynym zadaniem kontenerów jest reagowanie na żądania umieszczania w nich danych oraz ich pobierania. Kontenery pozwalają na pobieranie danych w innym porządku, niż były do nich dostarczone. Kontenery są reprezentowane w diagramie jako para równoległych odcinków, oddzielonych nazwą kontenera. Do kontenera mogą dochodzić potoki danych, wówczas są to operacje wstawiania, usuwania bądź modyfikowania przechowywanego elementu, oraz mogą z niego wychodzić oznaczając odczytanie danych. Ponieważ kontenery są obiektami, podobnie jak aktorzy, muszą być zdefiniowane w modelu obiektów. Przykłady kontenerów przedstawia rysunek 24. Pierwszy z kontenerów jest listą wyników zawodów w skoku w dal. Przez cały czas trwania zawodów wyniki są wpisywane do kontenera. Po zakończeniu zawodów wybierane są

RYS. 22. Rozdzielenie złożonej wartości na składowe oraz kopia danych w potoku



RYS. 23. Obiekt Równanie (aktor) dostarcza dane dla diagramu

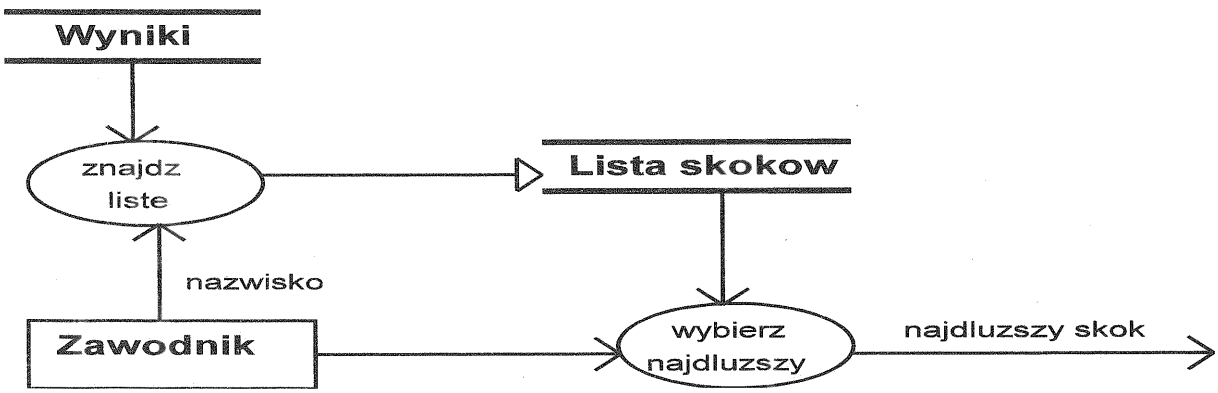


RYS. 24. Kontenery

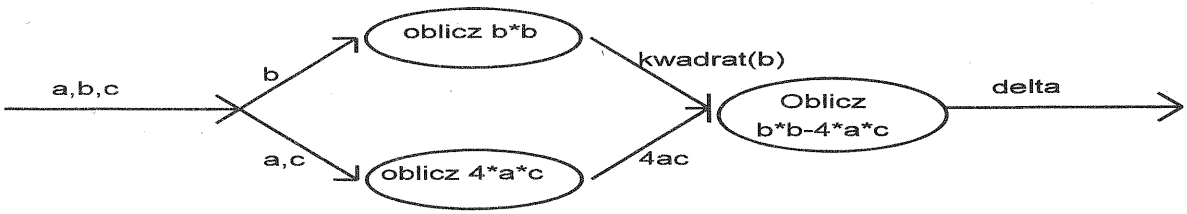
trzy najdłuższe skoki. Drugi przypadek przedstawia podobną sytuację. Do kontenera wpisywane są długości skoków oraz nazwiska sportowców. Jeżeli jest to drugi skok, do kontenera wpisywany jest dłuższy z nich. Proces **znajdź odległość**, na podstawie podanego nazwiska, wyszukuje na liście wyników odległość na jaką skoczył zawodnik.

Wynikiem odczytu z kontenera może być nie tylko pojedyncza wartość, lecz również inny obiekt. Po zmianie sposobu wpisywania wyników zawodów na listę tak, aby zapamiętywane były wszystkie skoki, żądanie wyszukania najdłuższego skoku będzie wymagało "dynamicznego" utworzenia obiektu. Obiekt taki będzie wyróżniany dodatkowo trójkątem na końcu potoku danych. Przykład ilustruje rysunek 25.

Diagramy przepływu danych mogą się zagnieżdżać w sobie. Oznacza to, że proces na wyższym poziomie może być reprezentowany przez osobny diagram na poziomie niższym. Nowy diagram może zawierać obiekty, których na poziomie wyższym nie było. Danymi dla diagramu są potoki danych wchodzące do procesu, jego rezultatem - potoki wychodzące ze stanu. Rysunek 26 pokazuje diagram przepływu danych, odpowiadający procesowi **oblicz deltę** z rysunku 23.



RYS. 25. Odczyt dający w wyniku obiekt



RYS. 26. Rozszerzenie procesu Oblicz deltę

Wzajemne związki modeli

Każdy z opisanych modeli pokazuje inny aspekt tego samego systemu. Bazę dla pozostałych modeli stanowi model obiektów. Zawiera on kluczową informację: *z czego* składa się system. Informacja w nim zawarta jest statyczna. Wiadomo wprawdzie, które obiekty ze sobą współpracują, ale nie wiadomo kiedy i jak. Pierwszą informację dostarcza model dynamiczny. Dodaje on "ducha" statycznym obiektom, pokazując *kiedy* mają rozpocząć działanie (w odpowiedzi na jakie zdarzenie) i w jakim porządku akcje mają występować. Model funkcjonalny dopełnia informację pokazując, jak wykonywać operacje.

Model obiektów definiuje strukturę aktorów, kontenerów i potoków danych (jeżeli są obiektami, a nie tylko wartościami) z modelu funkcjonalnego. Obiekty i aktorzy są w relacji 1-1. Kontenery mogą być integralnymi obiektami modelu obiektów, choć może się zdarzyć, że będą "tylko" ich atrybutami. Podobnie ma się sprawa z potokami danych. Najczęściej potok reprezentuje czystą wartość, tzn. taką, która nie ma zdefiniowanych stanów ani nie ma odpowiadającego jej modelu dynamicznego (wprawdzie niektóre języki programowania traktują wszystko jako obiekty, ale nie zmienia to sytuacji). Są od tego wyjątki, co pokazuje rysunek 25. Procesy, w zależności od poziomu abstrakcji diagramu przepływu danych, odpowiadają operacjom lub zbiorom operacji na obiektach. Procesy reprezentowane przez zwykłe funkcje można utożsamić z pojedynczymi metodami. Im bardziej abstrakcyjny staje się diagram, tym większemu zbiorowi operacji odpowiada proces.

Model dynamiczny pokazuje dokładnie kolejność wykonywania procesów w modelu funkcjonalnym. Kolejność ta zależy od napływu zdarzeń pojawiających się w systemie.

Model funkcjonalny definiuje akcje i czynności wykonywane w poszczególnych stanach modelu dynamicznego.

Nie wszystkie modele są jednakowo ważne w każdym problemie. Prawie w każdym z nich model obiektów gra pierwszoplanową rolę. Jest on odzwierciedleniem statycznej struktury modelowanego wycinka rzeczywistości. Jeżeli obraz problemu zmienia się znacząco wraz z upływającym czasem, wówczas ważny jest także model dynamiczny. Systemy mające do czynienia z dużą liczbą obliczeń będą, z kolei, miały złożony model funkcjonalny.

Analiza systemu nie jest procesem liniowym. Wymaga ona stałej współpracy z klientem oraz ekspertami z dziedziny problemu. Najczęściej proces tworzenia modeli rozpoczyna się od konstrukcji na wysokim poziomie abstrakcji, zawierających bardzo ogólne wiadomości. Nieustanne konsultacje oraz dokładniejsza analiza problemu pozwalają na przechodzenie do rozwiązań bardziej szczegółowych. Ostateczna postać modeli stanowi bazę dla etapów projektowania.

Projektowanie systemu

Projektowanie systemu jest pierwszym krokiem w stronę *jak to zrobić*. W trakcie jego trwania podejmuje się decyzje odnośnie organizacji systemu (podział na podsystemy) oraz przydziału podsystemów do oddzielnych jednostek sprzętowych i programowych. Organizacja całego systemu za pomocą podsystemów *(ang. subsystem)* jest nazywana architekturą systemu *(ang. system architecture)*. Podział problemu na mniejsze składowe pozwala na dalszą niezależną pracę nad każdym z nich. W trakcie projektowania systemu powinno się wziąć pod uwagę następujące czynniki:

- podział systemu na podsystemy,
- identyfikację współbieżności wewnątrz systemu,
- przydział podsystemom osobnych procesorów i procesów,
- ustalenie sposobu dostępu do wspólnych zasobów,
- ustalenie metody implementacji sterowania w systemie,

- obsługę warunków brzegowych.

W zależności od specyfiki projektowanego systemu, niektóre czynniki będą wymagały bardzo uważnej analizy, podczas gdy część z nich będzie można pominąć w rozważaniach.

Podział systemu na podsystemy

Pierwszym krokiem przy projektowaniu systemu powinien być jego podział na podsystemy. Podział powinien być dokonany w ten sposób, by każda nowo utworzona składowa tworzyła spójną pod jakimś względem całość. Za kryterium podziału przyjęło się brać usługi, jakie podsystem ma wykonywać. Jeżeli projektowany jest system obsługi klienta w banku, podsystemy mogłyby zawierać interfejs graficzny, obsługę kont, obsługę bankomatów itd. Każdy podsystem powinien mieć precyzyjnie zdefiniowany interfejs z resztą systemu. Dzięki takiemu rozwiązaniu możliwe będzie dokonywanie ewentualnych zmian wewnątrz podsystemu bez ingerencji w pozostałą część systemu. Wobec tego, każdy podsystem będzie mógł być tworzony niezależnie od innych.
Podział systemu może być dokonywany na dwa sposoby. Pierwszy polega na podziale na "poziome" warstwy. Każda warstwa jest spojrzeniem na problem "z innej wysokości". Warstwę górną buduje się albo na podstawie tylko warstwy leżącej bezpośrednio pod nią - wówczas mamy do czynienia z architekturą zamkniętą *(ang. closed architecture)*, lub wszystkich warstw leżących pod nią - taką architekturę nazywa się otwartą *(ang. open architecture)*. W przypadku architektury zamkniętej, odwołania w warstwie górnej odnoszą się tylko do jednej warstwy. Oznacza to, że ewentualne zmiany w którejś z warstw spowodują konieczność uaktualnienia jedynie warstwy leżącej bezpośrednio pod nią. Dzięki temu łatwo można przenosić systemy na różne platformy sprzętowe, zmieniając tylko najniższą warstwę. Rozwiązanie ma jednak tę wadę, iż występuje duża redundancja informacji. Na każdym poziomie należy przedefiniowywać wszystkie obiekty poprzedniej warstwy. Architektura otwarta zmniejsza redundancje informacji. Z drugiej strony, zmiany w którejś z warstw mogą mieć wpływ na każdą inną warstwę leżącą powyżej, co może sprawić sporo kłopotów w przypadku awarii oraz przy ewentualnej przyszłej rozbudowie systemu.
Alternatywą dla podziału na poziome warstwy jest podział na pionowe partycje. Każda partycja świadczy określone usługi. Jednocześnie partycje są bardzo słabo ze sobą powiązane, zmniejszając w ten sposób liczbę wysyłanych do siebie komunikatów. Jak zwykle bywa w życiu, praktyka pokazała, iż najlepszym rozwiązaniem jest połączenie obydwu metod. Tworzy się więc systemy "poziomo-pionowe".
Związki między podsystemami mogą być dwojakiego rodzaju: klient-dostawca *(ang. client-supplier)* lub równoprawny *(ang. peer-to-peer)*. W pierwszym przypadku klient żąda od dostawcy wykonania określonej usługi. Dostawca wykonuje ją dając określoną odpowiedź. Dostawca musi wcześniej znać interfejs dostawcy, podczas gdy znajomość interfejsu klienta przez użytkownika jest zbędna. Dzięki temu, takie połączenie jest proste do wykonania i pielęgnacji. Przy związku równoprawnym obydwa podsystemy mogą żądać od siebie wykonania usług. Każdy z nich musi znać sposób komunikacji ze sobą.
Na zakończenie etapu należy wybrać odpowiednią topologię systemu. Sposób komunikacji podsystemów może być różny. Zawsze należy wybierać najprostszą topologię, tzn. taką by jak najbardziej zmniejszyć liczbę wzajemnych odwołań między podsystemami.

Rozpoznawanie współbieżności

Do tej pory wszystkie obiekty występujące w systemie były traktowane jako niezależne. Działo się tak, gdyż modelowane obiekty miały odpowiadać obiektom rzeczywistym. Niestety wciśnięcie Rzeczywistości w sztywne ramy zer i jedynek nie może się obejść bez skutków ubocznych.

W praktyce obiekty muszą dzielić się ograniczonymi zasobami wspólnych dóbr. Dlatego też należy wydobyć te obiekty, które *muszą* pracować równolegle oraz te, których działanie wzajemnie się wyklucza, czyli *nie mogą* równocześnie funkcjonować.
Współbieżność obiektów łatwo jest odczytać z diagramu stanów. Dwa obiekty są współbieżne, jeżeli mogą niezależnie od siebie odbierać zdarzenia i stosować się do nich. Niezależne systemy są o tyle wygodne, że można im przydzielić osobne jednostki sprzętowe, nie obciążając się dodatkowymi kosztami wzajemnej komunikacji. Nie oznacza to, że niezależne systemy nie mogą pracować z tym samym procesorem. Dobrą platformą są dla nich systemy, umożliwiające pracę wielozadaniową *(ang. multitasking)*.
W praktyce niektóre obiekty, pomimo tego, iż konceptualne są niezależne, zależą w jakimś stopniu od siebie. Obiekty takie można umieścić na jednym wątku *(ang. thread of control)*. Wątkiem nazywa się ścieżkę prowadzącą przez zbiór diagramów stanów, na której tylko jeden obiekt jest aktywny. Jeżeli dotychczas aktywny obiekt przekazuje kontrolę innemu obiektowi, pozostając jednocześnie w stanie w oczekiwaniu na inne zdarzenie, wątek zostaje przekazany temu drugiemu obiektowi i pozostaje w jego diagramie tak długo, aż kontrola nie zostanie z powrotem przekazana obiektowi poprzedniemu. Jeżeli obiekt wysyła zdarzenie do drugiego obiektu, a sam nie przestaje działać, wątek zostaje rozdzielony. Wątki są implementowane w systemie jako procesy *(ang. task)*.

Przydział procesora i procesu podsystemom

Każdy współbieżny podsystem musi być przydzielony do jednostki czy to sprzętowej, czy programowej, umożliwiającej realizację współbieżności. Platformą sprzętową może być jeden z procesorów, dedykowany wyłącznie do obsługi podsystemu. Może nią również być wyspecjalizowany układ scalony, realizujący określone zadanie. Wybór zależy od wymagań stawianych systemowi. Jeżeli system ma do czynienia z dużą ilością danych, które muszą być przetworzone w rozsądnym czasie, nie wystarczy symulacja współbieżności na jednym procesorze. W takich wypadkach należy uruchomić system na maszynie wieloprocesorowej. Pożądane efekty może dać także zastosowanie wyspecjalizowanych układów scalonych, świadczących pewne określone usługi. Rozwiązanie ma tę wadę, w stosunku do metod programistycznych, że nie pozwala na łatwą zmianę stosowanych algorytmów. Jak zwykle w takich przypadkach, warto pomyśleć o kompromisie.

Zarządzanie danymi

Zarządzanie danymi jest kluczowym problemem przy konstrukcji praktycznie każdego systemu. Systemy meteorologiczne przechowują dane o pogodzie z ostatnich kilku miesięcy, systemy militarne zawierają informacje, które nie powinny wyjść poza mury jednostki. Najprostszym rozwiązaniem jest umieszczanie danych w plikach. Jest to prosty i skuteczny sposób. Ma on jednak kilka znaczących wad. Po pierwsze, implementacja plików różni się praktycznie w każdym rodzaju sprzętu - co w dużym stopniu ogranicza przenośność systemu. Poza tym, dostęp do plików jest dokonywany na dość " niskim" poziomie, co powoduje, iż system powinien być dodatkowo wyposażony w wygodny mechanizm obsługi.
Innym rozwiązaniem problemu jest zastosowanie istniejących systemów baz danych. Systemy baz danych są wyposażone w wygodne metody dostępu do danych (szczególną zaletą jest ogólnie przyjęty, standardowy sposób manipulowania danymi w postaci języka SQL) oraz efektywne algorytmy uaktualniania i przeszukiwania swoich zasobów. Wadą ich stosowania jest nienajlepsza współpraca z proceduralnymi językami programowania oraz trudność w zapamiętywaniu złożonych struktur danych.

Zarządzanie wspólnymi zasobami

W systemie, w którym niezależne procesy muszą korzystać ze wspólnych zasobów, istnieje problem synchronizacji dostępu do nich. Zasoby systemu można podzielić na kilka kategorii:

- fizyczne - procesor, drukarka,dyski, pamięć, itd.,
- logiczne - identyfikatory obiektów (ponieważ każdy obiekt jest jednostką niezależną i rozróżnialną, musi, w związku z tym, posiadać niepowtarzalny identyfikator), nazwy klas i plików.

Dostęp do każdego globalnego zasobu powinien być kontrolowany. Typowym rozwiązaniem tego problemu jest stosowanie monitorów *(ang. monitor, guardian object)*. Monitory są pośrednikiem pomiędzy procesem wysyłającym żądanie dostępu do zasobu, a samym zasobem. Monitor przechwytuje żądanie dostępu i sprawdza, czy dostęp jest możliwy. Jeżeli jest - wówczas zasób jest przydzielany procesowi. Za pomocą monitorów możemy zarządzać pamięcią czy czasem procesora (jeżeli współbieżność jest realizowana w jednostce z jednym procesorem). Czasami konieczne jest, by dostęp do zasobu odbywał się bez udziału monitora. Wówczas w zasobie umieszcza się zamek *(ang. lock)*, przez który procesy komunikują się bezpośrednio z zasobem. Takie rozwiązanie jest niebezpieczne, gdyż niekontrolowany dostęp do zasobu może spowodować wiele zamieszania. Innym problemem systemów ze współdzielonymi zasobami, są blokady *(ang. deadlock)*. Blokada występuje w momencie gdy dwa, lub więcej, procesów stara się o dostęp do zasobu i żaden z nich nie ma szans otrzymania go. Przykładem może być dostęp do przewijaków taśm. Jeżeli dwa procesy starają się o dostęp do przewijaków i każdy z nich zajął już jeden (a każdemu potrzebne są dwa), wówczas następuje blokada. Rozwiązaniem problemu jest zastosowanie algorytmów wywłaszczających *(ang. preemptive algorithm)*. Istnieją także metody radzenia sobie z dostępem do wspólnych zasobów logicznych. Zabezpieczenie przed nadawaniem przez różne procesy tych samych numerów identyfikacyjnych różnym obiektom uzyskuje się przydzielając każdemu procesowi przedział, do jakiego mogą należeć identyfikatory tworzonych w nim obiektów.

Ustalenie metody implementacji sterowania w systemie

Można wyróżnić dwa rodzaje przepływu sterowania w systemie: przepływ zewnętrzny i przepływ wewnętrzny. Przepływ zewnętrzny odpowiada przepływowi widzialnych na zewnątrz aplikacji zdarzeń pomiędzy obiektami, podczas gdy przepływ wewnętrzny odpowiada przepływowi sterowania wewnątrz procesu. Wyróżniamy trzy kategorie zewnętrznych przepływów sterowania: sterowane proceduralne *(ang. procedure-driven)* - w którym sterowanie znajduje się wewnątrz kodu programu, sterowanie zdarzeniami *(ang. event-driven)* - w którym sterowanie jest w rękach dyspozytora *(ang. dispatcher)* oraz współbieżne - w którym sterowanie znajduje się jednocześnie w niezależnych obiektach.

Zarządzanie warunkami brzegowymi

Część pracy przy projektowaniu systemu należy poświęcić warunkom brzegowym systemu, na które składają się: inicjalizacja, zakończenie oraz awarie. W momencie inicjalizacji należy pamiętać o nadaniu wartości początkowych odpowiednim składowym systemu. Należą do nich, między innymi: stałe, zmienne globalne, zadania, monitory itp. Zakończenie pracy systemu powinno się wiązać przede wszystkim z usunięciem z pamięci wszystkich niepotrzebnych obiektów oraz ze zwolnieniem zajętych zasobów. Błędy powinny być tak obsługiwane, by drobna usterka nie powodowała zawieszenia całego systemu.

Etap projektowania systemu jest pierwszym z etapów rozpatrywania systemu w dziedzinie implementacji. Następnym krokiem ży-

cia systemu jest projektowanie obiektów.

Projektowanie obiektów

Jest to ostatni etap poprzedzający implementację. Jego zadaniem jest takie zoptymalizowanie systemu, by w jak największym stopniu spełniał stawiane mu wymagania. Optymalizacja może dotyczyć pamięci operacyjnej, szybkości działania, przyszłej rozbudowy czy pielęgnacji. Na tym etapie należy wybrać algorytmy implementujące operacje, struktury danych reprezentujące powiązania i atrybuty. Projektowanie obiektów jest więc kolejnym etapem " ulepszania" systemu, przez dodanie do niego nowych szczegółów. Aby ten proces ułatwić i nieco zautomatyzować, warto się kierować poniżej przedstawionymi wskazówkami:

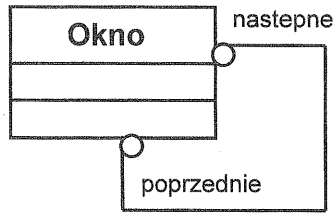
- połącz trzy modele analizy w jeden,
- zaprojektuj algorytmy implementujące operacje,
- zoptymalizuj ścieżki dostępu do danych,
- zaprojektuj powiązania,
- wyznacz reprezentację obiektów,
- utwórz z klas logiczne moduły.

Wynikiem analizy systemu są trzy modele, odpowiadające trzem ortogonalnym sposobom widzenia systemu. Model obiektów jest statycznym "szkieletem" (klasy, atrybuty, powiązania), z którym kojarzy się odpowiednie zachowanie (operacje). Zadaniem projektanta jest przekształcenie akcji, zawartych w modelu dynamicznym, oraz procesów z modelu funkcjonalnego, na operacje na obiektach.
Każdy diagram stanu pokazuje cykl życia pewnego obiektu *(ang. object life cycle)*. Przejścia między stanami odpowiadają wykonaniu pewnej operacji na obiekcie. Wynika stąd, iż każdemu zdarzeniu odpowiada konkretna akcja, dokonana na obiekcie. Zdarzenie wysyłane przez obiekt może reprezentować operacje wykonywane na innym obiekcie. Ponieważ zdarzenia zwykle pojawiają się w parach, odpowiadających wykonaniu operacji i zwróceniu wartości (a tym samym - powrocie sterowania do pierwszego obiektu), wynika stąd, że należą one do jednego wątku, przekazywanego między obiektami, i mogą być reprezentowane jako jedno zadanie. Akcjom w diagramie stanów mogą odpowiadać całe diagramy przepływu danych w modelu funkcjonalnym. Zadaniem projektanta jest przekształcenie tego grafu w liniową sekwencję kroków algorytmu. W zależności od poziomu, na którym rozpatrywane są procesy, algorytm może być na przykład prostym przejściem przez strukturę modelu obiektów , w celu odnalezienia wartości, albo skomplikowanym procesem, składającym się z wywołań wielu operacji. Od wielu innych kryteriów może zależeć wybór algorytmu. W przypadku problemów numerycznych lub inżynierskich ważną rzeczą jest złożoność obliczeniowa. Wybrane zostaną z pewnością te metody, których zapotrzebowanie na czas lub pamięć jest najmniejsze. Jeżeli tworzony system stanowić ma bazę dla innych aplikacji, pod uwagę należy wziąć łatwość dokonywania zmian oraz rozszerzeń. Algorytmy silnie związane ze strukturami danych, bądź korzystające ze specyficznej własności obiektu, mogą sprawić wiele kłopotu, nawet w przypadku małych zmian. Jeżeli jest to możliwe, warto pisać algorytmy "ogólniejsze". Podobna uwaga dotyczy typów argumentów obiektów. Argumenty "ściśle dopasowane" do funkcji obiektu w systemie mogą okazać się niewystarczające w przypadku modyfikacji lub rozbudowy systemu.
Na etapie projektowania obiektów często zachodzi potrzeba uzupełnienia listy operacji obiektów o te, których istnienie związane jest z implementacją. Z tej samej przyczyny do systemu mogą być dodane również całe obiekty. Przykładem mogą być omówione wcześniej monitory, synchronizujące dostęp do wspólnych zasobów, lub niskopoziomowe klasy, odpowiedzialne za graficzny sprzęg z użytkownikiem. Bardzo często w takich przypadkach pojawiają się problemy z umieszczeniem niektórych operacji w klasach (problem ten praktycznie nie występuje w przypadku

obiektów z dziedziny problemu). Szczególnie jest to widoczne wtedy, gdy z operacją jest związanych kilka obiektów. Ogólną regułą w takich przypadkach jest umieszczanie operacji w tych obiektach, na których jest ona wykonywana, a nie w tych, które operacje wywołują. Czasami spojrzenie na sieć powiązań ułatwia decyzję - jeżeli obiekty i powiązania między nimi tworzą gwiazdę wokół jednego z obiektów, wówczas jest on obiektem docelowym.

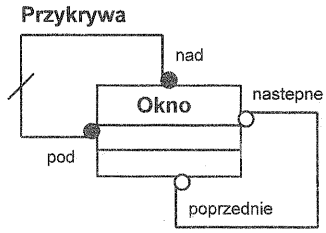
Jak wspomniano wcześniej, istnieje odwrotna zależność pomiędzy systemem optymalnym (ogólnie pojętym) a rozszerzalnym. Najczęściej struktura połączeń obiektów otrzymana w procesie analizy jest mało optymalna. Poprawiając ją warto zwrócić uwagę na trzy aspekty:

- czy nie dodać powiązań, które wprawdzie nie stanowią dodatkowej informacji, ale mogą przyspieszyć działanie lub ułatwić dostęp? Szczególnie przydatne w takich przypadkach są powiązania kwalifikowane, implementowane jako indeksy - jeżeli stosunek elementów spełniających zadany warunek do liczności całej populacji jest mały, a wyszukiwanie jest wykonywane często, wówczas indeks znacznie je przyspieszy. Możliwe są także usprawnienia wewnątrz obiektu posiadającego listę cech. Kosztem większego wysiłku można ją zaimplementować jako tablicę haszowaną lub przynajmniej zastosować jakiś porządek.
- czy można dodać atrybuty lub połączenia, które wprawdzie można wyprowadzić, ale w gotowej postaci mogłyby przyspieszyć działanie? Jeżeli obliczenia tych wartości są częste, wówczas umieszczenie ich *explicite* wewnątrz obiektu dałoby z pewnością pożądany efekt. "Sztandarowym" przykładem są nakładające się na ekranie okna. Pierwszym nasuwającym się rozwiązaniem jest diagram postaci:



RYS. 27. Okno "znające" jedynie swego następcę i poprzednika

W momencie, gdy usunąć należy okno znajdujące się w środku "stosu", należy sprawdzić, które okna poniżej były przez nie przykryte, i je " odsłonić". Jest to rzecz prosta, ale czasochłonna. Jeżeli operacja będzie przeprowadzana często, wówczas strata czasu będzie duża. Przyspieszenie można uzyskać dodając nowe połączenie, symbolizujące przykrywane przez obiekt okna. Pokazuje to rysunek 28.



*RYS. 28. Wyprowadzone powiązanie **Przykrywa***

Przekreślony symbol powiązania oznacza, iż jest ono wyprowadzone *(ang. derived)* z istniejących już powiązań. Wartość wyprowadzona musi być uaktualniana w momencie, gdy zmienia się jeden z jej atrybutów bazowych (takich, od których jest ona zależna). Aktualizację można wykonywać na wiele sposobów. Najbardziej naturalne jest dokonywanie aktualizacji w momencie, gdy zmienia się dowolny atrybut bazowy. Jeżeli zmiany te będą częste, wtedy, niestety, proces będzie zajmował dużo czasu. Można więc wyznaczyć okres, po upłynięciu którego atrybuty wyprowadzane będą aktualizowane. Jest to " hurtowe" podejście do rzeczy. Wadą jest to, iż w przypadku dużej liczby wyprowa-

dzonych atrybutów aktualizowane są wszystkie bez względu na to, ile bazowych atrybutów zmieniło swoją wartość.
Uważnego potraktowania przez projektanta wymaga sterowanie przebiegiem programu. Przebieg sterowania został już określony w modelu dynamicznym. Teraz zadanie polega na jego realizacji. Pierwszą metodą jest umieszczenie sterowania systemu w samym programie *(ang. procedure-driven system)*. Jest to najbardziej rozpowszechniona do tej pory metoda. Stanem programu jest miejsce, w którym sterowanie w danej chwili się znajduje. Zdarzeniami są dane wejściowe. W zależności od otrzymanej danej, program rozgałęzia się (decydują o tym umieszczone w programie instrukcje warunkowe). Sposób przechodzenia od diagramu stanu do programu może wyglądać tak:

(a) najpierw należy ustalić główną ścieżkę diagramu stanu. Będzie ona odpowiadała "normalnemu" wykonaniu programu,
(b) przejściom w diagramie, odgałęziającym się od głównej ścieżki, należy przyporządkować odpowiednie wyrażenia warunkowe,
(c) cykle występujące w diagramie należy zaimplementować jako pętle. Pozostałe ścieżki są po prostu wyjątkami - mogą to być na przykład błędy.

Ponieważ generalizacja zwiększa powtórną używalność klas, warto jej również poświęcić trochę uwagi. Pierwszych ulepszeń można dokonać wyszukując wspólne metody w hierarchii klas. Zdarza się czasami, że operacje zdefiniowane w różny sposób, posiadające różne argumenty i różne nazwy, w gruncie rzeczy świadczą te same, lub bardzo zbliżone usługi. W takim razie można je "doprowadzić do jednej postaci" i umieścić we wspólnym przodku.
Na tej samej zasadzie można zmienić strukturę obiektów, wyszukując klasy o podobnym zachowaniu. Jeżeli znajdą się takie klasy, wówczas warto utworzyć wspólnego przodka (najczęściej jest to abstrakcyjna klasa), który będzie zawierał wspólne dla grupy klas cechy. Tworzenie klas abstrakcyjnych bardzo zwiększa możliwości dalszej rozbudowy systemu. Dzieje się tak z racji tego, iż zawierają one jedynie ogólne aspekty modelowanego zjawiska. Wszelka specjalizacja dokonuje się na niższych poziomach, przez co dodawanie nowych klas polega jedynie na uzupełnianiu brakujących cech.
Częstym błędem jest nieodpowiednie używanie mechanizmów dziedziczenia. Dziedziczenie powinno być stosowane tylko wtedy, gdy obiekt dziedziczący jest istotnie rozszerzeniem poprzednika. Często używa się dziedziczenia w sytuacji, w której *akuratnie* można to zrobić, traktując dziedziczenie jako mechanizm implementacji. W takich sytuacjach warto się zastanowić, czy kompozycja obiektów nie jest lepszym rozwiązaniem.

Podsumowanie

Technologie obiektowe są, w dalszym ciągu, swego rodzaju nowością. Nie wytworzyły się jeszcze żadne standardy, zarówno w przypadku narzędzi jak i metodologii. Najdalej posunęły się chyba obiektowo-zorientowane języki programowania. Do tej pory raczkują obiektowe bazy danych (tworzy się różne "mutacje" polegające na przykład na przeróbkach systemów relacyjnych). Obiektowe narzędzia CASE czy I-CASE też nie są jeszcze powszechne. Wszystko jednak wskazuje na to, iż najbliższe lata będą zdominowane przez technologię obiektową.
Przedstawiona w pracy technika tworzenia systemów obiektowych jest jedną z wielu, które powstały w ostatnim czasie. Jak wszystkie techniki obiektowe, ma tę zaletę, że na każdym etapie cyklu życia obiektu posługuje się tą samą terminologią i notacją. Jej autorzy twierdzą, iż nie jest to tylko nowa metodologia, jest to również nowy sposób myślenia.
Celem pracy nie było nauczenie stosowania techniki OMT, lecz zasygnalizowanie problemu. Skrótowe potraktowanie, a nawet opuszczenie, pewnych partii materiału wynika, w głównej mierze, z obszerności materiału. Zainteresowanych szczegółami techniki odsyłam do literatury.

Bibliografia

[1] Martin J., Odell J.: *Object-Oriented Analysis and Design*, Prentice-Hall, Engelwood Cliffs, New Jersey, 1992.

[2] Rumbaugh J.,...: *Object-Oriented Modeling and Design*, Prentice-Hall Inc., Engelwood Cliffs, New Jersey, 1991.

[3] Winblad A. L.,...: *Object-Oriented Software*, Addison-Wesley, 1990.

[4] Rumbaugh J.: *An object or not an object?*, Journal of Object-Oriented Programming, 5(3), str. 20-25.

[5] Odell M.: *What is object state?*, Journal of Object-Oriented Programming, 5(2), str. 19-22.

[6] Harel D.: *On visual formalism*, Communications of the ACM, 31(5), str. 514-529.

[7] Davis A. M.: *A comparison of techniques for the specification of external system behavior*, Communications of the ACM, 31(9), str. 1098-1115.

Zbigniew Benedykt
Zakładowy Ośrodek Informatyki, BEFAMA, Bielsko-Biała

Co daje obiektowe podejście do analizy i projektowania systemów informatycznych

W poszukiwaniu technik i narzędzi ułatwiających tworzenie oprogramowania oraz podnoszących jego niezawodność dobrnęliśmy wreszcie do technik obiektowo zorientowanych. Tak, tak, minęło już ćwierć wieku od pojawienia się SIMULI 67 - niekwestionowanego prekursora tego podejścia. Język stworzony dla potrzeb modelowania, z definicji musi wywierać wpływ na sposób percepcji i opisu rzeczywistości. Można stąd wnioskować, że techniki obiektowe odziedziczyły tę cechę po swoim "wielkim przodku", i tak jest istotnie.

Zawsze istniała jakaś współzależność pomiędzy technikami stosowanymi przez programistów i projektantów oraz analityków systemów. W tym przypadku zależność ta jest znacznie bardziej wyrazista i nic w tym dziwnego, że oprócz programowania obiektowego istnieje obiektowe podejście do modelowania i projektowania systemów. Metody - techniki - rozwijane przez tak długi okres nie powodują rewolucyjnych zmian w metodach dotychczas stosowanych. To oczywiste, przecież to co było cenne i dało się zaadoptować wcześniej, weszło już do praktyki. Istnieje jednak pewien próg, po przekroczeniu którego zaczynamy nazwom metod dodawać kolejny przymiotnik. Obiektowo zorientowana metoda (technika), to taka, w której rozpatrujemy świat i organizujemy system informatyczny jako kolekcję dyskretnych obiektów łączących ze sobą zarówno dane (stany) jak i zachowanie się bytów istniejących w postrzeganym świecie. Większość dotychczas stosowanych metod rozdziela opis struktur danych od opisu procesów, aby w dalszych etapach analizy skojarzyć te dwa opisy dokonując weryfikacji ich zupełności i niesprzeczności. Dla każdego z etapów stosowane są różne techniki notacyjne (diagramy związków encji, przepływów danych, historii życia encji). Prowadzi to do swoistego dualizmu "danowo-akcyjnego", a przecież rzeczywiste byty są jednorodne, postrzegać je można tylko wtedy, gdy następuje interakcja pomiędzy podmiotem i przedmiotem. Zatem - obiekt jest konstrukcją bardziej adekwatną do opisu świata niż encja i proces! Biorąc pod uwagę to co zostało wyrażone wyżej, można oczekiwać, że metody obiektowe wyeliminują oddzielne diagramy dla poszczególnych etapów, wprowadzając jeden typ diagramu odpowiedni dla całego procesu tworzenia systemu. Ułatwi to kontrolę poszczególnych fragmentów systemu oraz relacji istniejących pomiędzy nimi i zwiększy czytelność projektu. Dotychczas nie zetknąłem się z ogłoszeniem takiej metody, lecz to co zostało już dokonane w tym zakresie daje podstawy do doprecyzowania takiego rozwiązania.

Innym aspektem stosowania obiektowo zorientowanych metod budowania systemów jest dalece zaawansowany stan rozwiązań programowych. Języki obiektowe (lub z mechanizmami programowania obiektowego) oraz nadchodząca "epoka" baz obiektowych dają podstawy aby oczekiwać, że w niedługim czasie można

będzie izomorficznie przetworzyć specyfikację systemu w oprogramowanie. To z kolei stwarza nadzieje na automatyczne implementowanie struktur obiektowych wprost ze specyfikacji systemu.

Istnieje wiele "klasycznych" metod budowania systemów, w większości z nich można dopatrzeć się elementów podejścia obiektowego i można je ulepszyć poprzez implantację "chwytów" obiektowych. Można więc wyobrazić sobie wiele metod projektowania obiektowo zorientowanego. Nie jest moim zamiarem prezentowanie tutaj jakiejkolwiek z nich, chodzi mi raczej o pokazanie potencjalnych możliwości tkwiących w tym podejściu. Wygodnie jednak jest mieć gotową notację oraz przykładową metodę. W tym celu posłużyłem się notacją i elementami metody OMT (Object Modeling Technique) przedstawionej w [1]. Spróbujmy więc na jej bazie prześledzić, co twórcy systemu daje podparcie się pojęciami: obiekt, enkapsulacja, dziedziczenie i polimorfizm.

Szkic metody OMT

Istotą metody OMT jest stworzenie modelu rzeczywistości, który posłuży do zbudowania systemu informatycznego. Model ten powinien wiernie oddawać syntaktyczne i semantyczne własności tego fragmentu rzeczywistości, który pozwoli nam na rozwiązanie problemów stawianych do rozwiązania przez system. Powinien on (ten model) być łatwo implementowalny, tzn. powinno się go dać łatwo, bez zbędnego wysiłku przetransponować na system informatyczny. Oczywiście model taki nie powstanie w jednym akcie twórczym. Proces jego tworzenia rozciąga się w czasie i wymaga zaangażowania różnej wiedzy specjalistycznej. Metoda zakłada podział całego procesu na fazy: analizy, projektowania systemu, projektowania obiektów oraz implementacji.

Analiza
Zaczynając od sformułowania problemu, analityk buduje model sytuacji w świecie rzeczywistym, pokazując w nim istotne jego właściwości. Analityk musi pracować z zamawiającym system, aby zrozumieć jego potrzeby oraz określić zasięg "terytorialny" systemu. W swojej pracy musi precyzyjnie określić, co system ma robić, nie zajmując się tym jak to będzie realizowane.

Projekt systemu
Generalne decyzje dotyczące architektury systemu, podział docelowego systemu na podsystemy. Decyzje dotyczące charakterystyk eksploatacyjnych, wyboru strategii rozwiązania problemu oraz próbny podział zasobów.

Projekt obiektów
Uzupełnienie modelu o szczegóły implementacyjne. Uwaga projektanta skupiona na strukturach danych oraz algorytmach niezbędnych do implementacji klas. Dla abstrakcyjnych struktur danych dobierane są odpowiedniki z repertuaru dostępnego w pakietach oprogramowania. Dobierane są algorytmy optymalizujące wykonanie ważnych funkcji systemu.

Implementacja
Klasy obiektów i powiązania zaprojektowane w poprzednich fazach są przekładane na konkretny język programowania i bazę danych. Kodowanie powinno być niewielką i mechaniczną częścią cyklu budowania systemu, ponieważ wszystkie znaczące decyzje powinny zostać podjęte w fazach wcześniejszych.

Wyznaczone w fazie analizy obiekty i ich klasy zachowują swą ważność przez cały cykl życia systemu, jedynie poziom szczegółowości opisu się zwiększa. W każdej fazie mogą pojawiać się nowe obiekty nie występujące w fazach poprzednich oraz nowe metody. Poprzez wszystkie fazy przechodzi się stosując te same konwencje notacyjne. OMT posługuje się trzema rodzajami diagramów do opisu systemu. Są to: model obiektów, model dynamiki obiektów oraz model funkcjonalny.

Model obiektów opisuje statyczną strukturę obiektów w systemie i ich powiązania. Model prezentowany jest w postaci - przypominających diagramy Chena - grafów, których węzły reprezentują klasy obiektów a łuki powiązania pomiędzy nimi.

Model dynamiki prezentuje schematy zmiany stanów obiektów systemu. Przedstawiany jest w postaci diagramów zmian stanów, są to grafy, których węzłami są stany, a łukami ich zmiany powodowane przez zdarzenia.

Model funkcjonalny prezentuje przepływy danych w systemie. Przedstawiany jest w postaci diagramów przepływu danych.

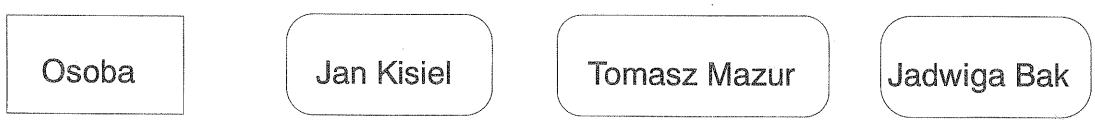
Te trzy modele stanowią ortogonalne części opisu systemu. Wprawdzie model obiektów jest fundamentalnym, jednak nie przedstawia on dynamiki obiektów, stąd te pozostałe dwa elementy. Twierdzę, że można zmodyfikować metodę i diagramy tak, aby wszystkie informacje o systemie zamknąć w jednym tylko modelu, ale to już całkiem inny referat. Ponieważ najwięcej korzyści z podejścia obiektowego widać w modelu obiektów, tylko on będzie przedmiotem dalszych rozważań.

Model obiektów

Zacznijmy od pojęć bazowych. Są nimi pojęcie obiektu i klasy. Najbardziej lapidarnie można stwierdzić, że obiektem jest coś, co ma dla nas jakiś sens w kontekście analizowanego systemu. Obiektami są na przykład: Jan Kisiel, faktura nr 13/93, wydział narzędziowni itd. Każdy obiekt jest odróżnialny od innych, może znajdować się w określonym stanie, wykonywać określone akcje i/lub podlegać określonym presjom. Połączenie stanu z akcjami jest tym co istotnie różni pojęcie obiektu od pojęcia encji wg Chena i jego kontynuatorów. Grupę obiektów posiadających jednakowe typy atrybutów, wspólne schematy zachowań, identyczne związki z innymi obiektami oraz jednakową semantykę nazywamy klasą obiektów. Obiekt kojarzymy z inkarnacją bytu zaś klasę z jego typem, ale uwaga: klasa jest również obiektem wyznaczonym przez wspólne (dla wszystkich obiektów klasy) stany i zachowania. Na przykład: liczba jest klasą obiektów takich jak liczba całkowita, wymierna itd. ale liczba sama jest obiektem; jeśli interesują nas stany magazynowe, to materiał o danym identyfikatorze jest obiektem ale z drugiej strony jest to klasa obrotów magazynowych. Na diagramach klasę przedstawiamy w postaci prostokąta, obiekt w postaci pudełka o zaokrąglonych rogach (rys. 1).

Zazwyczaj obiekty posiadają atrybuty, których wartości określają stan obiektu. Przedstawiając na diagramie obiekt, wpisujemy przykładowe wartości atrybutów (stan obiektu). Przedstawiając klasę podajemy nazwę oraz typ atrybutu (rys. 2). Często ze względów implementacyjnych posługujemy się specjalnym atrybutem służącym do identyfikacji obiektu. Na diagramach obiektów nie umieszcza się takich atrybutów ponieważ nie należą one do świata rzeczywistego.

Oprócz stanów obiekty miewają metody (akcje), które mogą wykonywać lub "znosić" np. pracownik może zmieniać



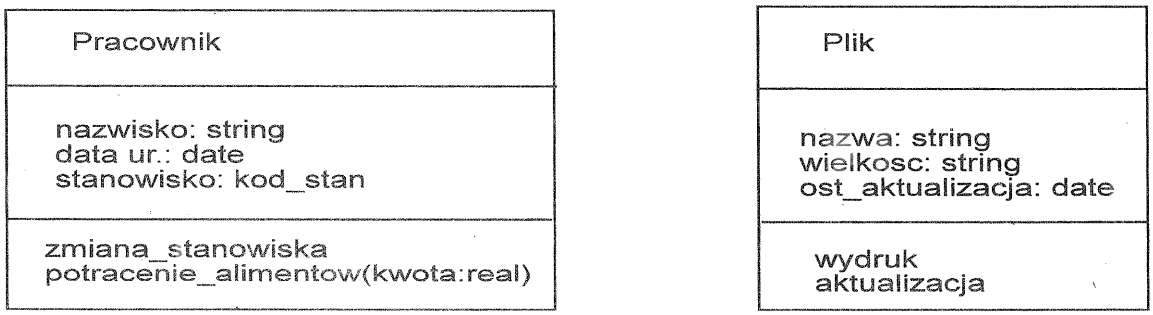
Klasa **Obiekty (klasy Osoba)**

RYS. 1.

RYS. 2

stanowisko, być zwalniany, zmuszony do płacenia alimentów itd. Metody specyfikuje się tylko na diagramach klas. Tak naprawdę są to procedury lub funkcje, zatem ich specyfikacja odpowiada tym kategoriom. Przykłady przedstawia rys. 3.

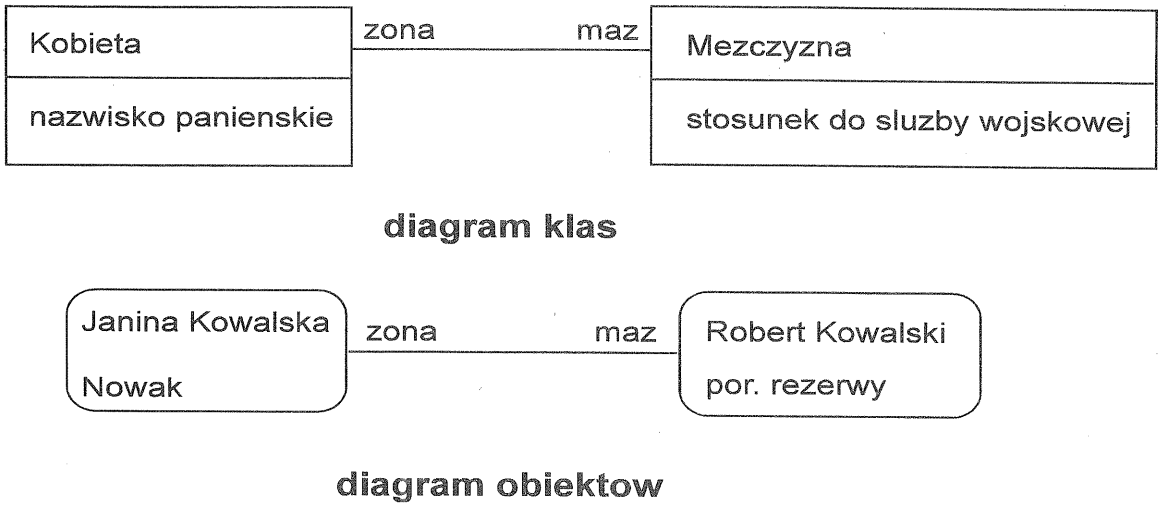
w tym zakresie oferuje OMT? Zacznijmy od rozróżnienia powiązania obiektów od związków pomiędzy klasami. Jeśli weźmiemy dwa obiekty, to mogą one być ze sobą powiązane np. Maria Kowalska może być żoną Jana Nowaka. Powiązanie to wyraża



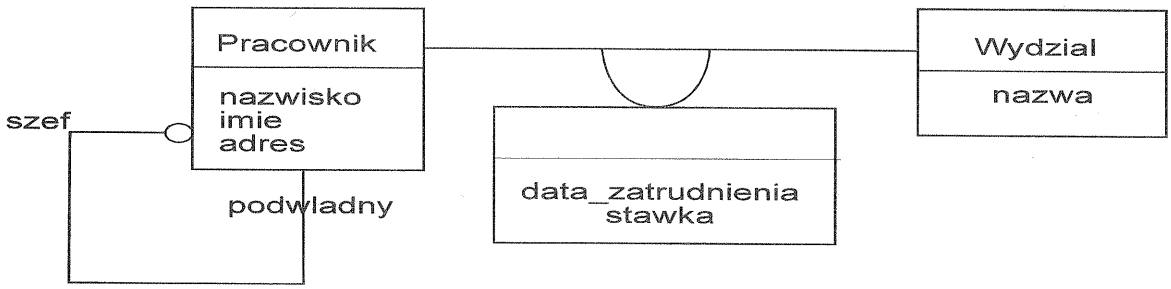
RYS. 3

Świat byłby mało interesujący gdyby obiekty nie wchodziły w związki pomiędzy sobą. Praca analityka staje się frapująca dopiero wtedy gdy może odkrywać te związki. Prowadzi to często również do odkrycia nie uświadamianych wcześniej obiektów. W "klasycznych" metodach brakuje mechanizmów pozwalających wystarczająco dokładnie zarejestrować semantykę związków encji. Powód wydaje się oczywisty. Znaczenie związków ujawnia się często poprzez działania (metody) obiektów, te zaś są rozpatrywane poza diagramami związków encji. Siłą podejścia obiektowego jest większa "wykrywalność" rzeczy, których mógł nam nie przekazać użytkownik systemu. Mamy również mechanizmy pozwalające zanotować te cechy i przekazać je do dalszej obróbki przez projektantów i programistów. Co istniejący stan. Jeśli zaś spojrzymy na związek pomiędzy klasami, to zobaczymy, że jest to tylko szablon możliwej sytuacji. Na diagramach zarówno powiązanie obiektów jak i związki pomiędzy klasami wyraża się przy pomocy linii łączących odpowiednie pudełka rys. 4.

Takie własności związków jak: krotność, opcjonalność, arności weszły już - tak mi się wydaje - do folkloru informatycznego, możemy więc tutaj o nich nie mówić, po prostu cechy te przenoszą się bez większych zmian na teren obiektów. Powszechnie znanym jest fakt, że związki wiele do wielu "generują" nową encję. W podejściu obiektowym, "generują" nowy obiekt, pozostaje tylko kwestia notacji, przedstawia ją rysunek 5.



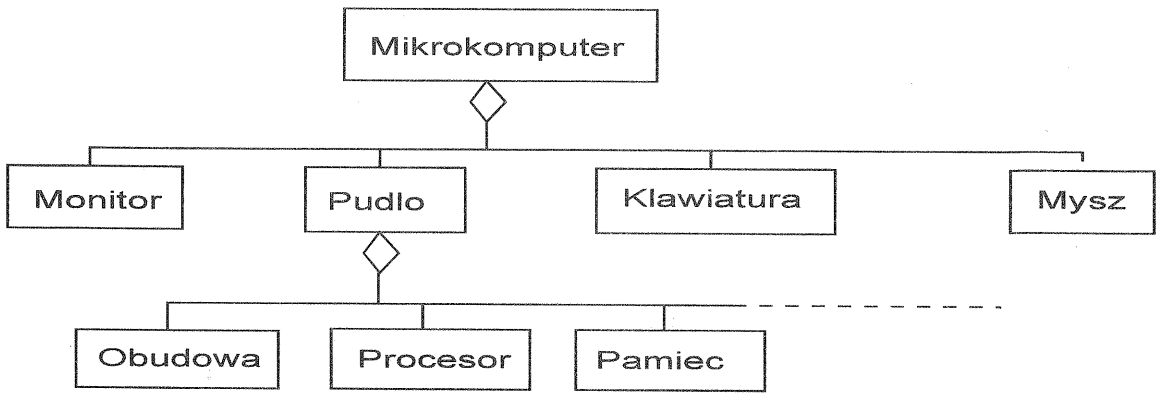
RYS. 4



RYS. 5

Z rysunku możemy odczytać, że wydział musi zatrudniać co najmniej jednego pracownika, pracownik może pracować w wielu wydziałach. Atrybutami tego powiązania (a zarazem nowego obiektu) są: data_zatrudnienia i stawka. Istnieje związek pomiędzy pracownikami. Pracownik może mieć co najwyżej jednego szefa, szefem jest się jeśli ma się przynajmniej jednego podwładnego.

Specyficzne cechy związków ujawniają się wtedy, gdy jeden obiekt jest klasą do której należy drugi. Rozróżnia się tu dwie kategorie związków: agregację i generalizację. Agregacja jest związkiem typu "część całości", wyraża fakt, że klasę tworzy zbiór obiektów (części). Na diagramach agregację przedstawiamy jak inne związki, ale po stronie złożenia rysujemy mały romb (rys. 6).

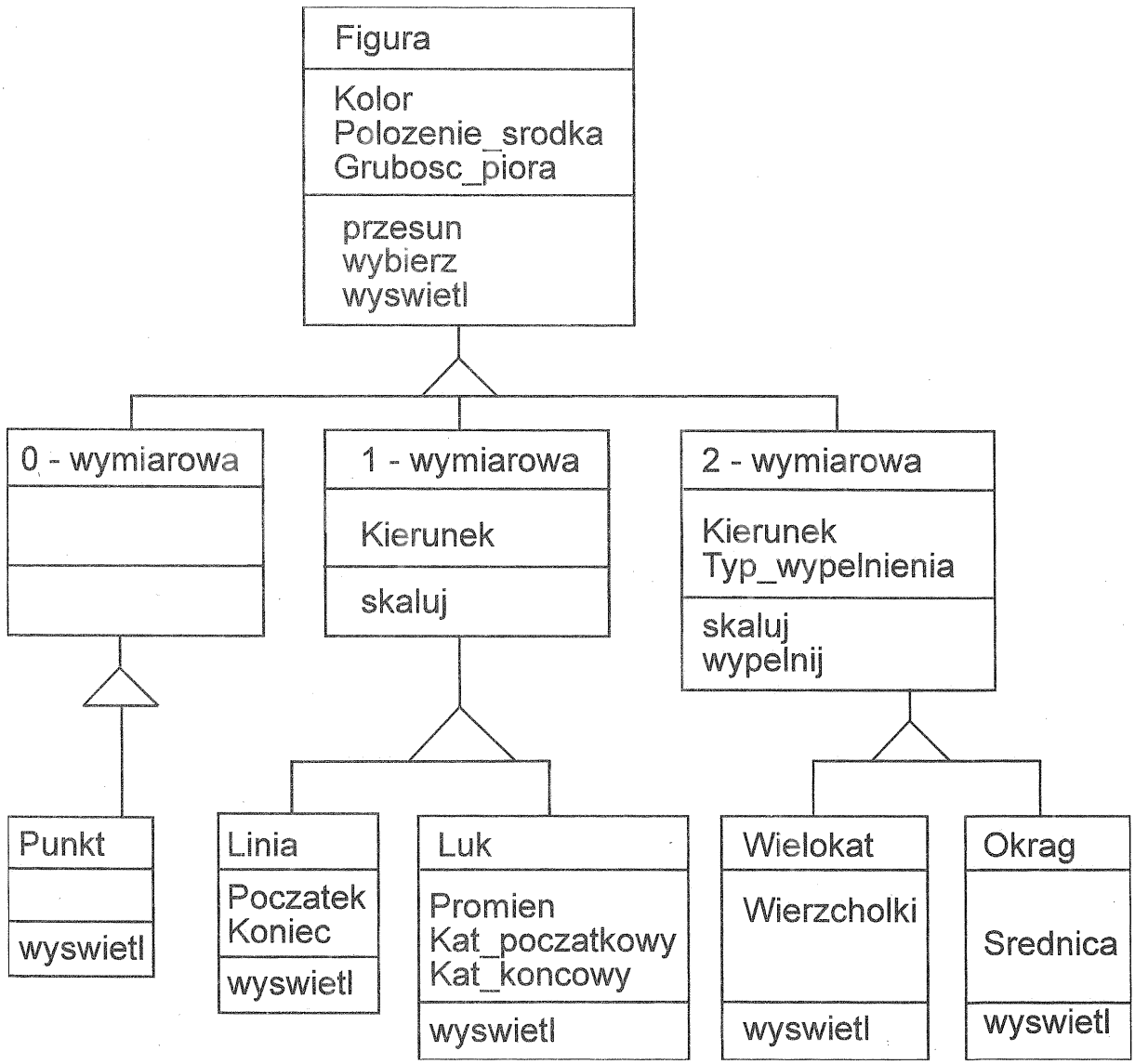


RYS. 6

Agregacja może być wielopoziomowa. Na rysunku 6 przedstawiony jest przykład agregacji o ustalonej liczbie składowych (linia przerywana oznacza dalsze pojedyncze elementy - pełni tutaj rolę wielokropka). Można jednak modelować sytuacje gdy ilość obiektów na niższym poziomie jest zmienna np. książka składa się z wielu rozdziałów, rozdział z wielu paragrafów i liczba paragrafów w rozdziale jest zmienna. Można również modelować agregację rekurencyjną np. struktura konstrukcyjna wyrobów.

Generalizacja jest też związkiem pomiędzy klasą a jej obiektami (lub jak kto woli - pomiędzy klasą i jej uszczegółowieniem) lecz w tym przypadku klasę wyznaczają cechy podobieństwa obiektów a nie ich kolekcja.

Rysunek 7 przedstawia typową sytuację gdzie generalizacja odgrywa pierwszorzędną rolę, pozwalając istotnie uprościć model. Jeśli się przyjrzeć uważnie, to można zauważyć, że pomiędzy generalizacją a dziedziczeniem istnieje pełna zgodność. To właśnie powoduje, że generalizacja jest bardzo użyteczną konstrukcją nie tylko na poziomie konceptualnym ale i implementacyjnym. Diagram obiektów łączy klasy i związki pomiędzy nimi w jedną logiczną całość. Mogą na nim współwystępować związki pomiędzy klasami nie należącymi do wspólnej hierarchii, jak i wszelkie związki



RYS. 7

hierarchiczne. Niektórym można nadać jeszcze inne znaczenia np. definicja przez przypadki jest formą generalizacji.

Podsumowując, model obiektów przedstawia wszystkie obiekty systemu (w kolejnych etapach budowy systemu ich przybywa) oraz związki pomiędzy klasami obiektów. Wszystko to, co można zapisać (przedstawić) na diagramach związków encji jest rejestrowane na modelu obiektów. Dodatkowo mając do dyspozycji opakowane wraz z danymi procedury, możemy dokładniej wyrazić semantykę tych związków. Brakuje tu jednak możliwości wyrażenia scenariuszy zdarzeń uruchamiających metody i powodujących zmiany stanów obiektów. Jest to robione na diagramach dynamiki i funkcjonalnym.

Środki implementacji

Modele budujemy nie po to, by się cieszyć ich urodą, lecz by je przekształcić w "chodzące" systemy służące do rozwiązywania konkretnych problemów. Jeśli krok jaki musimy wykonać aby przekształcić model w system jest niewielki i dobrze zdefiniowany, to mamy szansę na szybkie i w pełni kontrolowane uzyskanie celu. Nowoczesne systemy zarządzania bazami danych pozwalają na stosunkowo łatwą implementację struktur danych, odciążają projektanta od zajmowania się wieloma uciążliwymi problemami z zakresu ich ochrony oraz organizacji wielodostępu. Teoria struktur danych daje nam narzędzia do odwzorowywania jednych struktur w inne. Potrzebna jest teraz, możliwość uczynienia tego samego z oprogramowaniem, lecz można to zrobić tylko wtedy, gdy dobrze zdefiniujemy akcje wykonywane przez obiekty i ich porządek, już na poziomie modelu konceptualnego. Wiele z tych akcji wynika bezpośrednio ze związków pomiędzy klasami, stąd tak ważnym jest aby związki te były bardzo precyzyjnie zdefiniowane. Podejście obiektowe pozwala nam lokować je we właściwych miejscach. Wykorzystanie mechanizmu dziedziczenia pozwala na ograniczenie się do jednokrotnej tylko definicji procesu, a polimorfizm umożliwia określenie klasy procesów do wykonania takiej samej metody dla różnych klas obiektów. To bardzo pięknie porządkuje nam obraz świata, który modelujemy i daje możliwość łatwego ustalenia homomorfizmów pomiędzy światem rzeczywistym i modelem oraz pomiędzy modelem a światem komputera. Zapewne przyjdzie nam jeszcze trochę poczekać na wdrożenie do praktyki baz danych, które będą mogły pamiętać i obsługiwać obiekty, wykorzystywać mechanizmy dziedziczenia oraz dopasowywać do obiektów metody polimorficzne. Ale tak jak można odwzorować encję w rekord, tak też można przekształcić standardowe klasy metod w gotowe szablony procedur i udostępnić je do użytku w środowisku baz danych. Takie rozwiązanie dostępne jest np. w bazie MAGIC (firmy MSE z Izraela). Wbudowano tam operacje połączeń typu jeden do jednego oraz jeden do wielu z możliwością aktualizacji powiązanych krotek. (Interesujące byłyby takie mechanizmy pozwalające obsługiwać agregację.)

Bibliografia

[1] Rumbaugh J., Blaha M., Premerlani W., Eddy F., Lorensen W.: *Object-Oriented Modeling and Design*, Prentice-Hall Inc., 1991.

[2] Materiały konferencji: *CASE'91*, Warszawa.

[3] Materiały konferencji: *INFORMATION SYSTEM DEVELOPERS WORKBENCH*, Gdańsk 1990, 1992.

[4] Yourdon E.: *Modern Structured Analysis*, Prentice-Hall, 1989.

[4] Pinson L. J., Wiener R. S.: *An Introduction to Object-Oriented Programming and Smalltalk*, ADDISON-WESLEY PUBLISHING COMPANY, 1988.

[5] Dokumentacja pakietu MAGIC, MSE.

Stanisław Kędzierski
Katedra Informatyki Akademii Ekonomicznej w Katowicach

Modelowanie obiektów dynamicznych

1. Wstęp

Pojęcie obiektu pojawiło się w latach siedemdziesiątych w wyniku prac nad nowymi metodami programowania. Booch [1] wprowadza klasyfikację języków programowania według kryterium abstrakcji, którymi się one posługują:

- generacja pierwsza (1954 - 1958) - abstrakcja matematyczna,
- generacja druga (1959 - 1961) - abstrakcja algorytmiczna,
- generacja trzecia (1962 - 1970) - abstrakcja danych,
- luka generacyjna (1970 - 1980),
- N-ta generacja (1980 - dotychczas) abstrakcja zorientowana na obiekty.

Programy są jednym z wielu elementów systemu informatycznego (SI). Stanowią one rezultat procesu projektowania. Projektowanie zaś jest niczym innym jak modelowaniem istniejącej rzeczywistości. Z punktu widzenia projektanta interesująca jest część otaczającego świata nazywana *dziedziną przedmiotową*. Dziedzina przedmiotowa *(ang. UoD - Universe of Discourse)* to zbiór wszystkich możliwych jednostek, które aktualnie leżą w zasięgu zainteresowania odpowiedzialnych osób (projektanci, analitycy). *Model konceptualny* dziedziny przedmiotowej to pewna abstrakcyjna jednostka, w której zawarte jest wspólne pojmowanie dziedziny przedmiotowej przez projektantów. Model konceptualny powstaje z dziedziny przedmiotowej jako rezultat procesu *modelowania konceptualnego* (występującego też pod nazwą analizy informacyjnej). Projektant powinien być zainteresowany tym aby doprowadzić ten proces do takiej postaci (przy użyciu metod formalnych), by mógł on stanowić podstawę do automatycznego generowania rozwiązania.

2. Modele konceptualne obiektowo-zorientowane

W modelowaniu konceptualnym należy odciąć się od modeli danych, które są zorientowane na komputer (opisując rozmieszczenie danych w mc.). Należy rozpatrywać dwa rodzaje jednostek: obiekty i masy. Obiekt to cokolwiek, co może być pomieszczone w zbiorze (w sensie matematycznym). Zawsze mówi się o zbiorze (aktualnym lub możliwym) ludzi, przedsiębiorstw itp., czyli że obiekty mogą być policzone. Zbiór obiektów nie musi być przeliczony ale ma swą moc. Z drugiej strony masa nie może być policzona czyli nie może być elementem zbioru. Za przykłady mas mogą służyć woda, zysk, waga itp. O masy można się pytać: jak dużo? ale nie konkretnie: ile? Zwrócić należy uwagę, że masy nigdy nie są reprezentowane w bazach danych czy modelach konceptualnych. Dla mas, o których powinny być przechowywane jakieś informacje, definiowana jest funkcja miary z masy do obiektu dyskretnego: np. zysk w jednostkach pieniężnych, waga w kilogramach.

Obiekty charakteryzują się posiadaniem atrybutów. Wiele obiektów może mieć te same własności, przy czym nie są to te same obiekty np.: książki w bibliotece, samochody opuszczające fabrykę. Są one takie same (autor, tytuł, treść; marka,

kolor, wyposażenie) a różni je tylko numer (sygnatury lub fabryczny). Każdy obiekt musi posiadać unikalny globalny identyfikator - a to celem umożliwienia rozróżnienia pomiędzy dwoma dowolnymi z nich. Rozpatrując identyfikatory obiektów należałoby bardziej wnikliwie skoncentrować się na ich cechach:

- obiekt ma swój identyfikator podczas całego czasu istnienia niezależnie od stanu w jakim znajduje się dziedzina przedmiotowa,
- identyfikator obiektu jest pewnego określonego typu. Licząc obiekty liczymy identyfikatory np.: możemy przyjąć, że zbiór pasażerów autobusu był trzyelementowy, mimo że ta sama osoba trzy razy korzystała z przejazdu. Zawsze mówimy, że było nie: "50" lecz: "50 pasażerów" czy też: "50 osób" (pasażer lub osoba jest tutaj typem).
- identyfikatory mogą być strukturalnie zależne od innych identyfikatorów np.: dodanie nowej osoby do zbioru może zmienić wartości pól średni wiek czy też największy wiek.

Zorientowany obiektowo model konceptualny wytwarzany jest w procesie modelowania. Można powiedzieć, że każdy model konceptualny jest pewną abstrakcją dziedziny przedmiotowej i można patrzeć na model konceptualny jako na schemat konceptualny co przedstawia rys. 1.

Schemat modelu konceptualnego (SMK) powinien mieć formalną postać (czyli, żeby można było formalnie opisać relacje pomiędzy SMK i MK). Przykładowo chcemy aby SMK był teorią w formalnym języku z MK jako modelem.

Metodyki projektowania proponują różne języki specyfikacji schematu modelu konceptualnego oraz pewien zbiór struktur, który może być reprezentowany w MK, jak i też grupę porad praktycznych w jaki sposób z wiedzy analityka o dziedzinie przedmiotowej wytworzyć specyfikację MK. Porady praktyczne to głównie zbiór kolejnych zadań do wykonania co w klasycznych metodykach przyjmowało formę liniowego następstwa. Zadania sprecyzowane były aż do poziomu żądanych detali (projekty zbiorów, wydawnictw, algorytmów programów).

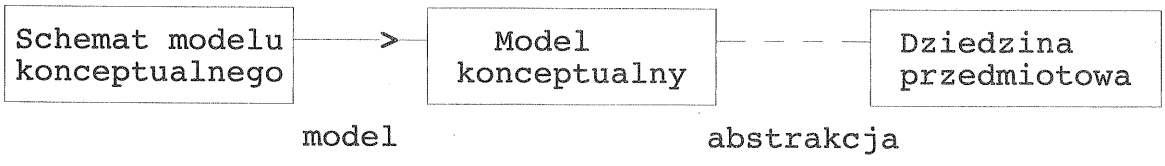
Metodyka może być obiektowo zorientowana, czyli że modeluje ona rzeczywistość w sposób uwypuklający obiekty. Istnieją trzy powody wyboru takiego podejścia:

1) orientacja na dziedzinę przedmiotową umożliwia zdobycie dostatecznej wiedzy o niej i uniknięcie błędów w dalszych etapach,
2) umożliwienie identyfikacji obiektów, ich klasyfikacja i dynamika (Booch uważa, że koncentracja na obiektach z DP koresponduje z zasadą modularyzacji software'u),
3) specyfikacja modelu przed specyfikowaniem funkcjonalnym, czyli powodowanie większej stabilności systemu informatycznego.

3. Klasyfikacja obiektów

Obiekty charakteryzują się szeregiem istotnych i ciekawych cech. Wymienienie ich wydaje się bardzo pouczające:

1) obiekty są dyskretne,
2) operacje łączenia lub rozdzielania mas daje masę tego samego typu,
3) każdy obiekt musi mieć globalny unikalny identyfikator.



RYS. 1. Modelowanie konceptualne

Obiekty z dziedziny przedmiotowej można pogrupować w klasy. Aby wskazać obiekt wskazuje się konkretne wystąpienie w jednej lub większej liczbie *klas* obiektów, jak np.: człowiek, zatrudniony, samochód itp.

Klasy (uniwersa) z filozoficznego punktu widzenia charakteryzują się dwoma cechami:

1) typ (klasa) orzeka o rzeczy, np. **czerwony** jest typem ponieważ mówi, że coś jest czerwone. Typ może też być egzemplarzem innego typu np. **czerwony** jest wystąpieniem typu **kolor**. Typy posiadają wystąpienia poszczególne zaś obiekty reprezentują tylko siebie, np.: Warszawa nie jest typem ponieważ orzeka tylko o sobie.
2) typy orzekają o **wielu** rzeczach, czyli o tym co jest wspólne dla pewnej ilości elementów.

Wszystkie obiekty należące do danej klasy mają tę samą strukturę. Może być wiele wystąpień o wielu atrybutach w danej klasie. Umysł analityka ogranicza ilość atrybutów. Można powiedzieć, że atrybuty określają w pewnym sensie klasę. Klasę można scharakteryzować poprzez podanie jej właściwości.

Pierwszą własnością jest to, że klasa powinna być naturalna czyli, że w prosty sposób można ją łączyć lub dzielić i określić przynależność obiektu do klasy. Używając pojęć logiki można powiedzieć, że klasą naturalną jest największy zbiór wystąpień, dla którego prawdziwe są warunki przynależności do klasy. Przykładowo zbiór osób w pewnej dziedzinie przedmiotowej może być największym zbiorem wystąpień określonych warunkami: posiadanie nazwiska oraz wiek < 120 lat.

Drugą własność klasy naturalnej stanowi fakt, iż istnieje ona w swojej 'naturze'. Analityk może znaleźć informacje o niej w DP.

Trzecią cechą klasy naturalnej jest jej podstawowość lub warunkowość. Podstawowość oznacza niezmienianie cech podczas przechodzenia DP przez różne stany np.: człowiek nie może istnieć jako wystąpienie nie będąc elementem klasy osoba. Warunkowość zaś oznacza możliwość przestania bycia elementem danej klasy jeśli zmienił się stan DP np.: można przestać być elementem klasy student czy też pracownik. Często mówi się, że obiekt z klasy warunkowej gra *rolę*. Aby móc grać rolę obiekt musi należeć do klasy naturalnej podstawowej np.: osoba może grać rolę pracownika, studenta, czytelnika itp. Wynikają stąd wnioski mówiące o tym, że:

- rola musi być podklasą klasy naturalnej,
- żadna klasa naturalna nie może być podklasą roli.

Obiekt posiada strukturę, która może być w zależności od momentu podstawowa lub warunkowa. Wszystkie podstawowe struktury obiektu wyrażone są poprzez identyfikator. Struktura warunkowa może zmieniać się w czasie np.: zbiór pracowników wydziału, wiek osób, kolejność występowania są reprezentowane przez wartości atrybutów co można sformalizować następująco:

```
zatr      :WYDZ      —> OSOBA
wiek      :OSOBA     —> NAT
max-wiek  :OSOBY     —> NAT
```

gdzie:
WYDZ, OSOBA, OSOBY są typami odpowiednio identyfikatorów wydziałów, osób i skończonych zbiorów osób.

Interesujące z punktu widzenia możliwości automatyzacji dalszych prac nad projektem systemu informatycznego jest formalne zapisanie specyfikacji tegoż systemu. Jedną z propozycji jest CMSL (Conceptual Modelling Specification Language [3]). Jako przykład zastosowania tego języka może posłużyć definiowanie obiektów dotyczących czytelników:

```
   import
     OsobaIde, Nazwisko, Adresy
   identifier sorts
     OSOBA
   attributes
     nazwisko :OSOBA  -> NAZWISKO
     adres    :OSOBA  -> ADRES
end spec OsobaAtrybuty

object spec CzytelnikAtrybuty
   import
     Osoba, Boolean,
                ZbiórSubskrybentów
   roles
     CZYTELNIK < OSOBA
   attributes
     subskrypcja :CZYTELNIK  ->
                          SUBSKRYPCJA
end spec CzytelnikAtrybuty
```

Powyższe dwa opisy mają swą graficzną reprezentację przy wykorzystaniu grafów skierowanych (rys. 2).

Niezaetykietowana strzałka na rys. 2 oznacza funkcję identyczności, czyli obrazuje to, że CZYTELNIK jest podklasą OSOBY.
Górna część rysunku przedstawia prostą hierarchię mówiącą o tym, że osoba reprezentuje połączenie nazwiska z adresem.
Przerywaną linią oznaczona jest rola jaką może pełnić osoba.

W opisie specyfikacji OsobaAtrybuty importowane były trzy specyfikacje, przy czym milcząco założone było, że są one już opisane. I tak OsobaIde określa identyfikatory typu OSOBA:

$$p_0,\ p_1,\ p_2,\ p_3,\ \ldots$$

Na tym typie określona jest funkcja sprawdzająca równość dwu identyfikatorów:

```
eq: OSOBA * OSOBA --> BOOL
```

OsobaIde oraz Nazwisko określają *abstrakcyjne typy danych* identyfikatorów osób i ciągów znaków. Adresy specyfikują *abstrakcyjne typy obiektów* tak jak czynią to OsobaAtrybuty. Tak więc każdą osobę można traktować jako trójkę o przykładowej postaci:

```
<p1,<nazwisko: 'Kowalski',
                    adres: a15>>.
```

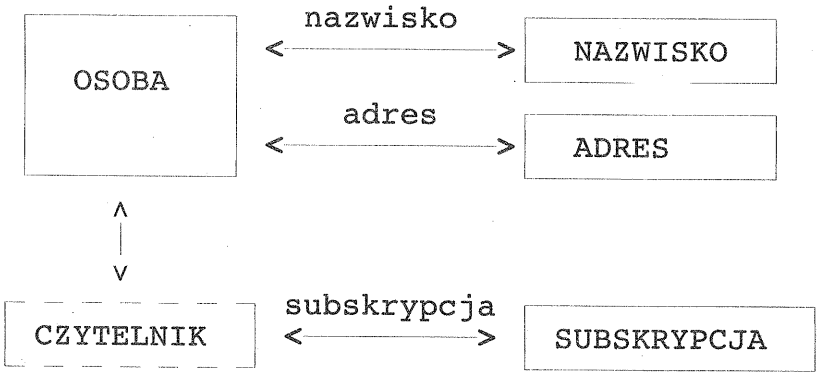
CzytelnikAtrybuty deklaruje rolę CZYTELNIKA dla osób oraz zbiór subskrypcji czytelnika. Notacja:

```
CZYTELNIK < OSOBA
```

oznacza, że klasa czytelników zawiera się w klasie osób, przy czym możliwe jest aby klasa CZYTELNIK była pusta. Formalny zapis bycia czytelnikiem przybiera postać:

```
<p1,<nazwisko: 'Kowalski',
        adres: a15,
  subskrypcje: {s1,...,sn}>>
```

gdzie $s_1,\ldots,s_n$ są identyfikatorami subskrypcji.



RYS. 2. Struktura semantyczna osób i czytelników

4. Specyfikacja zdarzeń

Obiekty mogą pełnić pewne role, ale aby do tego doszło muszą zajść jakieś zdarzenia. Zdarzenia potrzebne są do modelowania zmian zachodzących w DP. Ogólnie rzecz biorąc można wyróżnić dwa rodzaje zdarzeń:

- zmiana roli,
- zmiana wartości atrybutu.

Czyli mówiąc o zdarzeniu ma się na myśli to, że obiekt wykonuje je lub że podlega jego działaniu. Ponieważ obiekty mają swe identyfikatory więc można powiązać zdarzenia z konkretnymi obiektami. Pomiędzy obiektami o_1 a o_2 zachodzi *interakcja* jeśli obserwują one swe działania. Rozumieć to należy następująco: istnieje co najmniej jeden obiekt (dla innych może być to niedostępne), który obserwuje pełnienie ról lub zmianę atrybutów innego obiektu. Każde zdarzenie może zmienić stan obiektu (oczywiście za wyjątkiem jego identyfikatora). Zmiana stanu obiektu może być niedeterministyczna, czyli że ze stanu `st1` obiekt przechodzi w jeden ze stanów z pewnego zbioru. Niedeterminizm można ograniczyć żądając aby zdarzenie było funkcją wartości atrybutów obiektu. Zdarzenie `e` jest wykonywane przez obiekt `o` z atrybutem `a` tak, że po realizacji `a` będzie miało określoną wartość.

Jak widać istnieje związek pomiędzy zmianą stanu a zdarzeniem. Jeśli obiekt zmienia stan - wykonuje on zdarzenie. W drugą stronę to stwierdzenie niekoniecznie musi być prawdziwe np.: po wykonaniu zdarzenia obiekt może pozostać w tym samym stanie. Obserwator tego zdarzenia mógł zmienić swój stan, czyli że można tylko powiedzieć, iż *coś* się zdarzyło. Ogólnie wystąpienie zdarzenia zmienia stan *gdzieś* w DP lub w obiekcie wykonującym zdarzenie lub u obserwatora. Kiedy już wiadomo, że nie było zmian w DP to można powiedzieć, że nie wystąpiło żadne zdarzenie.

Zdarzenia zmieniające stan pojedynczego obiektu są zdarzeniami lokalnymi. Zdarzenia takie nie zmieniają już nic więcej w DP.
Jeśli istnieje powiązanie wykonania zdarzenia z obiektem to można to rozszerzyć na klasę obiektów. Przykładowo jeśli zdarzenie `zwiększ-wiek(p)` jest wykonywane przez osobę to `zwiększ-wiek` jest zadeklarowane dla wszystkich możliwych osób (z klasy `OSOBA`).

Inną ważną cechą zdarzeń jest ich atomowość, czyli spełnienie dwu warunków:

- zdarzenie nie trwa dłużej niż pewien okres czasu,
- zdarzenie nie składa się z podzdarzeń.

Natychmiastowym wnioskiem jest to, że wykonywanie zdarzeń nie ma stanów pośrednich.

Obiekty powinny móc komunikować się między sobą co można modelować jako wspólną akcję dwu obiektów, które dzielą się zdarzeniem. Formalny zapis *enkapsulacji* i *funkcji komunikacji* jest następujący:

$$e_1 | e_2 = e_3$$

gdzie: | oznacza konieczność komunikowania się zdarzenia e_1 ze zdarzeniem e_2 i że zdarzenie e_3 składa się z synchronicznych wykonań e_1 i e_2. Zdarzenie e_3 zachowuje atomowość ponieważ nie ma stanów pośrednich. Przykładem tej definicji jest zdarzenie:

```
zwiększ-stanowisko|
        zwiększ-pensję=awans.
```

Zdarzenia zenkapsulowane nazywane są *komunikatami (messages)*. Enkapsulacja i komunikacja odbywa się na dwu poziomach: lokalnym i globalnym. Komunikat lokalny występuje tylko wtedy gdy wykonywane są synchronicznie co najmniej dwa zdarzenia na tym samym obiekcie, globalny - gdy wykonywane są synchronicznie zdarzenia na różnych

obiektach. Analogicznie określana jest komunikacja lokalna i globalna.

W komunikowaniu się obiektów obowiązują następujące zasady:

I. zdarzenia wykonywane przez pojedynczy obiekt wykluczają się wzajemnie,
II. dowolny zbiór zdarzeń bez *message* wykonywany przez różne obiekty może być synchroniczny,
III. efekt komunikacji wykonywanej przez różne obiekty na obiektach wykonujących to równy jest efektowi uczestniczących zdarzeń na tych obiektach.

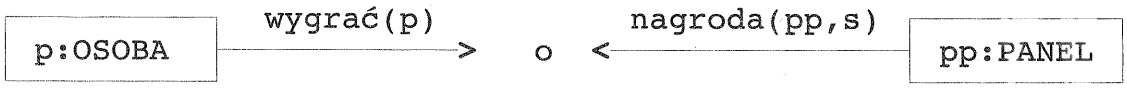
Jako przykład specyfikacji zdarzeń opisane będą: zmiana adresu, subskrypcja i udział w konkursie:

```
object spec OsobaZdarzenie
  import
    OsobaAtrybuty
  events
    zmiana-adresu:
        OSOBA×ADRES ---> OSOBA
  variables
    p: OSOBA
    a: ADRES
  local event constrains
    adres(
        zmiana-adresu(p,a))=a
end spec OsobaZdarzenie
```

OsobaZdarzenie opisuje jedno zdarzenie dla osoby a mianowicie zmianę adresu. CzytelnikZdarzenie podaje możliwość prenumeraty i stwarza osobie możliwość grania roli czytelnika i powiększania zbioru prenumerat. Każdy czytelnik może wziąć udział w konkursach

```
object spec CzytelnikZdarzenie
  import
    CzytelnikAtrybuty
  events
    subskrypcja:OSOBA*subskrypcja ---> CZYTELNIK global message
    dołączanie :OSOBA*subskrypcja ---> CZYTELNIK global message
    wejście    :OSOBA*subskrypcja ---> CZYTELNIK global message
    wygrać     :CZYTELNIK ---> CZYTELNIK global message
  variables
    p:OSOBA
    s:subskrypcja
  local event constrains
    subskrybowanie(subskrypcja(p,s))={s}
    subskrybowanie(subskrypcja(p,s))=subskrybowanie(p)+{s}
end spec CzytelnikZdarzenie
```

```
conceptual model spec Tytuł
  import
    CzytelnikZdarzenie, Panel, Skrzynka
  variables
    pp:PANEL
    p :OSOBA
    s :subskrypcja
    sk:SKRYTKA
  global communications
    załóż-SUBSKRYPCJA(db)|subskrypcja(p,s)
    załóż-SUBSKRYPCJA(db)|dołączanie(p,s)
    nagroda(pp,s)|wygrać(p)
    wejście(p,s)|otrzymać(m,s)
end spec Tytuł
```



RYS. 3. Diagram komunikacyjny

z szansą na wygraną. W modelu konceptualnym Tytuł importowane są specyfikacje odpowiednich obiektów oraz dodawane informacje globalne a w szczególności specyfikacja komunikacji globalnej.

Komunikacja w modelu konceptualnym przedstawiona jest za pomocą *diagramu komunikacyjnego* (rys. 3).

5. Zakończenie

W referacie poruszone zostały podstawowe problemy obiektowo zorientowanego modelowania konceptualnego systemów informatycznych.
Ten sposób pojmowania i modelowania rzeczywistości (DP) wnosi nowe wartości do pracy analityka i projektanta. Nie będzie on wolny od błędów ale z pewnością zdecydowanie zmniejszy ich ilość.

Należy spodziewać się, że dalszy rozwój metod informatycznych będzie wiązał się z ich formalizacją, co przyczyni się do automatyzacji prac. Omówiony w niniejszej pracy język CMSL jest jednym z przykładów takiego sformalizowanego podejścia.

Literatura

[1] Booch G.: *Object-oriented development.* IEEE Transactions on Software Engineering, 1986, ss. 211-221.
[2] Cameron J. R.: *An overview of JSD*, IEEE Transactions on Software Engineering, 1986, ss. 222-240.
[3] Wieringa R.: *Steps towards a method for the formal modeling of dynamic objects*, Data & Knowledge Engineering, 1991, ss. 509-540.

Marek Wierzbicki
Student V roku Politechniki Łódzkiej
Wydział Elektryczny-Automatyka

Tworzenie obiektów poprzez dziedziczenie mono- i polimorficzne

Programowanie obiektowe jest najnowszym sposobem programowania. Idea ta powstała jako dalsze udoskonalenie i rozszerzenie programowania strukturalnego. Umożliwia ukrycie wszystkich szczegółów (które dotąd zaciemniały obraz sytuacji) tylko po to, aby całość rozwiązania była jak najprostsza i najbardziej przejrzysta. Zbliża również modele świata opisywane w komputerze do rzeczywistości, przez co programowanie staje się bardziej naturalne.

Dla osób, które nigdy dotąd nie miały styczności z programowaniem obiektowym najważniejsze będą trzy innowacje:

1. Nowa klasa zmiennych: programowanie obiektowe realizuje się z wykorzystaniem zmiennych nowego typu. Typy te są projektowane przez użytkownika podobnie jak rekord. Oprócz danych mogą jednak zawierać także procedury i funkcje. Cała taka struktura, składająca się z danych i procedur nazywana jest właśnie obiektem.

2. Dziedziczność: jeśli do definicji nowego obiektu używamy opisu innego, wcześniej skonstruowanego, wtedy ten nowy obiekt będzie dziedziczył wszystkie cechy (zmienne i procedury) po obiekcie, którego użyliśmy do jego opisu (będzie je posiadał w swojej strukturze).

3. Polimorfizm: najatrakcyjniejsza cecha programowania obiektowego, umożliwiająca zmniejszenie pracy przy pisaniu programu korzystającego z obiektów do minimum. Idea polimorfizmu polega na tym, że każdy obiekt wykorzystuje tylko swoje własne sposoby działania, nawet jeśli korzystamy z tych sposobów pośrednio (to znaczy używając procedur nie opisanych konkretnie w tym obiekcie, a tylko dziedziczonych).

Te trzy innowacje dają już pełne możliwości programowania obiektowego. Dokładnie takie same możliwości dostępne są w językach C++, Turbo Pascal, Simula, SmallTalk czy Eiffel. Warto jednak zwrócić uwagę na różne nazewnictwo i trochę inne sposoby konstrukcji obiektów w różnych językach - wynikają one najczęściej ze struktury samego języka, który posłużył za bazę realizacji idei obiektowej. W poniższym artykule będę korzystał ze składni i nazewnictwa stosowanego w Turbo Pascalu. Na zakończenie przedstawię główne różnice między Turbo Pascalem a C++ - drugim najpopularniejszym językiem programowania udostępniającym możliwość programowania obiektowego.

Zajmiemy się teraz podstawami programowania obiektowego i przyjrzymy się dokładnie wszystkim innowacjom wprowadzonym do języka strukturalnego. Dotychczas w Turbo Pascalu dostępna była tylko jedna klasa zmiennych, która mogła być projektowana przez użytkownika. Był to rekord - struktura, która umożliwia połączenie (zgromadzenie) kilku zmiennych prostych opisujących np. jeden przedmiot, pod jedną nazwą. Takie połą-

czenie ułatwia operacje na wszystkich danych zawartych w rekordzie. Jeśli chcielibyśmy opisać za pomocą zmiennych kilka cech człowieka, należałoby skonstruować rekord, który zawierałby takie pola (poszczególne zmienne) jak wzrost, waga, wiek, płeć, kolor skóry itd. Wykorzystanie takich rekordów np. w bazach danych bardzo ułatwia pracę. Zmienne Człowiek_1 i Człowiek_2, które są tego samego typu, mają dokładnie taką samą strukturę. Dość łatwo można więc zaprojektować procedury, które zewnętrznie oddziaływują na te zmienne. Jednak do potrzeb symulacji komputerowych takie rozwiązanie było niewystarczające. Wiadomo, że każdy człowiek, mimo ogólnych podobieństw, zachowuje się jednak trochę inaczej. Najbardziej rzucającym się w oczy rozwiązaniem było więc przypisanie do każdej zmiennej złożonej całego zestawu procedur, opisujących działania tylko i wyłącznie tej zmiennej. Od takiego rozwiązania był już tylko mały krok do stworzenia obiektu - połączenia w jednej strukturze danych i procedur opisujących działania na tych danych. Cała taka struktura oglądana z zewnątrz dawała już dużo bliższą symulację życia codziennego. Wystarczyło do każdej struktury danych dodać procedury opisujące działania na niej (podobne, lecz nie jednakowe) aby otrzymać modele niezbędne w symulacji.

Z języków symulacyjnych, obiekty szybko przeniesiono do zwykłych języków programowania. Co więcej, każdy z nas nieświadomie używa kilku konstrukcji, które są typowo obiektowymi rozwiązaniami. Potrzebujecie przykładu? Proszę bardzo. Wyobraźmy sobie, że mamy dwie pary zmiennych: aw i bw - liczby typu Word oraz ar i br - liczby typu Real. Jeśli w programie wykonujemy mnożenie liczb aw i bw oraz ar i br, używamy do tego (w obu przypadkach) operatora *. Jednak te dwa mnożenia wykonane będą na dwa różne sposoby (inny zestaw instrukcji zastąpi mnożenie liczb typu Word, a inny mnożenie liczb typu Real). Nie jest to nic innego, jak skojarzenie pewnych procedur (metod działania) z pewnymi typami zmiennych. Oczywiście w Turbo Pascalu nie jest to rozwiązane w sposób obiektowy (razem z każdą zmienną nie trzymamy w pamięci kopii wszystkich procedur i funkcji działających na tej zmiennej). Teoretycznie można by wykonać to właśnie tak (nie zważając na ograniczenia pamięci i niemożność przeciążania operatorów w Turbo Pascalu). Oczywiście tworzenie tak prostych obiektów służy tylko zaprezentowaniu możliwości i wprawkom programistycznym. Naprawdę atrakcyjne efekty osiąga się dopiero przy tworzeniu dużych struktur i programów.

Wyobraźmy sobie, że przygotowujemy się do stworzenia dużego programu komputerowego. W poniższym artykule zajmiemy się sprawą komunikacji tego programu z użytkownikiem (nie będziemy prowadzili rozważań o jego merytorycznym działaniu). Podstawowym obiektem naszej biblioteki będzie obiekt Prostokat, którego definicja pokazana jest poniżej:

```
type
  Prostokat=object
    AX, AY: Integer;
    BX, BY: Integer;
  end;
```

Poszczególne pola oznaczają położenie lewego górnego rogu (AX, AY) i prawego dolnego rogu (BX, BY) we współrzędnych ekranowych. Cały obiekt określa tylko współrzędne prostokąta i jako taki nie jest jeszcze zbyt użyteczny. Będzie jednak mógł być używany do dalszego dziedziczenia. Przykładem stworzenia obiektu, który nadal nie posiada żadnych metod jest definicja pokazana poniżej:

```
type
  Okno_Dane=object(Prostokat)
    Ramka: String[8];
    Wnetrze: Pointer;
    Tlo: Pointer;
    CienX, CienY: Byte;
    CienAttr: Byte;
  end;
```

Obiekt Okno_Dane zawiera najważniejsze informacje o oknie. Oprócz mechanizmu dziedziczenia w definicji tego obiektu nie ujawniają się żadne inne cechy programowania obiektowego. Pokazana powyżej definicja określa obiekt, który logicznie ma postać:

```
type
  Okno_Dane=object
    AX, AY: Integer;
    BX, BY: Integer;
    Ramka: String[8];
    Wnetrze: Pointer;
    Tlo: Pointer;
    CienX, CienY: Byte;
    CienAttr: Byte;
  end;
```

Jak widać, mimo że pola określające położenie prostokąta nie zostały jawnie zadeklarowane, istnieją w tym obiekcie. Jak dotąd obiekt niewiele różni się od standardowego rekordu. Nową możliwością jest tylko dziedziczenie. Dodajmy teraz do obiektu procedury, które będą działaniami operującymi na obiekcie (czyli na własnych danych). Poniżej przedstawiam definicję obiektu OknoBazoweM, które oprócz pól (dziedziczonych z obiektu Okno_Dane) posiada procedury działające na nim (rysujące to okno, przesuwające je itd.):

```
type
  OknoBazoweM=object(Okno_Dane)
    procedure ZapamietajTlo;
    procedure OdtworzTlo;
    procedure Rysuj;
    procedure RysujRamke;
    procedure RysujCien;
    procedure RysujWnetrze;
    procedure Przesun;
  end;
```

Interesujące nas w tych rozważaniach metody wymienione w definicji mają następującą implementację:

```
procedure OknoBazoweM.Przesun
                  (DX, DY: Integer);
begin
  OdtworzTlo;
  AX:=AX+DX;
  BX:=BX+DX;
  AY:=AY+DY;
  BY:=BY+DY;
  Rysuj;
end;

procedure OknoBazoweM.Rysuj;
begin
  ZapamietajTlo;
  RysujCien;
  RysujRamke;
  RysujWnetrze;
end;

procedure OknoBazoweM.RysujWnetrze;
begin
end;
```

Jak widać, implementacja metod jest bardzo zbliżona do implementacji zwykłych procedur. Mimo tego, że metoda RysujWnetrze nie wykonuje żadnej akcji, wywoływana jest ona z metody Rysuj w celu zachowania konwencji. Obiekt OknoBazoweM może być teraz rozszerzane na wiele różnych sposobów. Możemy z jego użyciem stworzyć okno dialogowe, okno edycyjne itp. Dla przykładu pokażę fragment definicji i deklaracji obiektu OknoDialogoweM:

```
type
  OknoDialogoweM=
                object(OknoBazoweM)
    procedure Przesun(DX, DY:
                      Integer);
    procedure Rysuj;
    procedure RysujWnetrze;
  end;

procedure
          OknoDialogoweM.Przesun
              (DX, DY: Integer);
begin
  OdtworzTlo;
  AX:=AX+DX;
  BX:=BX+DX;
  AY:=AY+DY;
  BY:=BY+DY;
  Rysuj;
end;

procedure OknoDialogoweM.Rysuj;
begin
  ZapamietajTlo;
  RysujCien;
  RysujRamke;
  RysujWnetrze;
end;

procedure
      OknoDialogoweM.RysujWnetrze;
begin
  {rysowanie wnętrza w sposób}
  {charakterystyczny dla okna}
  {dialogowego na podstawie danych}
  {opisanych strukturą}
  {znajdującą się pod adresem}
  {określonym przez Wnetrze}
end;
```

W implementacji metod obiektu OknoDialogoweM powinni Państwo zwrócić uwagę na metody Rysuj i Przesun. Jak widać, są one identyczne jak metody o tych samych nazwach obiektu po którym OknoDialogoweM dziedziczy własności. Dlaczego ponowna implementacja (a właściwie skopiowanie) tych metod jest konieczne? Dla procedury Przesun problemem jest działanie metody Rysuj, a dla Rysuj działanie metody RysujWnetrze. RysujWnetrze, które znajduje się na końcu kolejki wywołań, zostało zmienione w stosunku do metody RysujWnetrze obiektu OknoBazoweM. Naturalne wydaje się więc, że wszystkie metody, które pośrednio lub bezpośrednio odwołują się do RysujWnetrze muszą zostać zmienione.
Takie podejście wydaje się naturalne dla osób, które nie miały dotąd styczności z programowaniem obiektowym i z obiektami polimorficznymi. Poniżej przedstawię fragmenty definicji dwóch obiektów polimorficznych, które funkcjonalnie będą takie same, jak przedstawione wcześniej. OknoBazoweP jest obiektem polimorficznym, który może służyć jako baza do tworzenia następnych obiektów (okna dialogowego, edycyjnego itp.). Definicja i fragment implementacji tego obiektu przedstawione są poniżej:

```
type
  OknoBazoweP=object(Okno_Dane)
    constructor Init(NAX, NAY, NBX,
                     NBY: Integer);
    destructor Done;
    procedure ZapamietajTlo;
    procedure OdtworzTlo;
    procedure Rysuj;
    procedure RysujRamke;
    procedure RysujCien;
    procedure RysujWnetrze;
                          virtual;
    procedure Przesun(DX, DY:
                      Integer);
  end;

procedure OknoBazoweP.Przesun
                (DX, DY: Integer);
begin
  OdtworzTlo;
  AX:=AX+DX;
  BX:=BX+DX;
  AY:=AY+DY;
  BY:=BY+DY;
  Rysuj;
end;
```

```
procedure OknoBazoweP.Rysuj;
begin
  ZapamietajTlo;
  RysujCien;
  RysujRamke;
  RysujWnetrze;
end;

procedure OknoBazoweP.RysujWnetrze;
begin
end;
```

W stosunku do obiektu OknoBazoweM obiekt ten ma dwie modyfikacje:
1. Metoda Rysuj jest metodą wirtualną (polimorficzną);
2. Dołożono dwie specjalizowane metody:
   - konstruktor Init,
   - destruktor Done.

Możliwości i zalety metody RysujWnetrze, która ma teraz nową kwalifikację, omówione zostaną później. Jak państwo widzą, metoda ta, podobnie jak w obiekcie OknoBazoweM jest pusta (nie wykonuje żadnej akcji). Specjalizowane metody - konstruktor i destruktor służą do prawidłowej inicjacji i dezaktywacji obiektu. Implementacja konstruktora i destruktora prawie nie różni się od standardowych metod. Najważniejsze są operacje, które dopisywane są automatycznie przez kompilator zarówno do konstruktora jak i destruktora. Operacje te są ważne, ze względu na techniczny sposób przechowywania informacji o obiektach i metodach polimorficznych (inny, niż w przypadku obiektów monoformicznych). Zalety polimorfizmu ujawniają się dopiero w przypadku dziedziczenia i tworzenia obiektu o pochodnych własnościach. Poniżej przedstawiona jest definicja i implementacja obiektu OknoDialogoweP:

```
type
  OknoDialogoweP=
                  object(OknoBazoweP)
    procedure RysujWnetrze;
                              virtual;
  end;
```

```
procedure
      OknoDialogoweP.RysujWnetrze;
begin
  {rysowanie wnętrza w sposób}
  {charakterystyczny dla okna}
  {dialogowego na podstawie danych}
  {opisanych strukturą}
  {znajdującą się pod adresem}
  {określonym przez Wnetrze}
end;
```

Jak widać, jedyną metodą, która odróżnia obiekt OknoDialogoweP od obiektu OknoBazoweP jest RysujWnetrze (jest to metoda polimorficzna - wirtualna). Mimo tego obiekty OknoDialogoweP i OknoDialogoweM działają tak samo. Jak to możliwe ? Otóż jest to właśnie zaleta polimorfizmu. Wszystkie metody wirtualne wywoływane są w inny sposób niż metody standardowe (statyczne). Można to wyjaśnić przedstawiając logiczną implementację metody Rysuj:

```
procedure OknoBazoweP.Rysuj
                        (var Self);
begin
  OknoBazoweP.ZapamietajTlo;
  OknoBazoweP.RysujCien;
  OknoBazoweP.RysujRamke;
  Self.RysujWnetrze;
end;
```

Jak widać wszystkie odwoływania do metod statycznych dotyczą metod, które są zaimplementowane w tym obiekcie (lub są dziedziczone przez ten obiekt). Inaczej jest z wirtualną metodą RysujWnetrze. Odwołanie do niej realizowane jest poprzez deskryptor Self. Określa on wskazanie do aktualnie używanego obiektu (niekoniecznie do OknoBazoweP). Jeśli więc metoda Rysuj zostanie wywołana jako metoda obiektu OknoDialogoweP wtedy jako Self podstawiony zostanie adres tego obiektu i metoda ta będzie miała postać:

```
procedure OknoDialogoweP.Rysuj;
begin
  OknoBazoweP.ZapamietajTlo;
  OknoBazoweP.RysujCien;
  OknoBazoweP.RysujRamke;
  OknoDialogoweP.RysujWnetrze;
end;
```

Jak widać, mimo tego, że w chwili projektowania obiektu OknoBazoweP nic nie było jeszcze wiadomo o obiekcie OknoDialogoweP (nie istniał on jeszcze), metoda Rysuj wywoła metodę RysujWnetrze obiektu OknoDialogoweP.

Czy to wszystko? Tak! Naprawdę idea polimorfizmu jest bardzo prosta i polega na dynamicznym sposobie wywoływania własnych metod danego obiektu (odwołania do tych metod są konstruowane w czasie pracy programu a nie jego konsolidacji). Metody te będą prawidłowo wywoływane nawet jeśli w chwili tworzenia procedur wywołujących nie były znane obiekty końcowe. Jakie są zalety stosowania obiektów polimorficznych w stosunku do standardowych (monomorficznych)? Przede wszystkim szybkość i łatwość tworzenia nowych obiektów. Porównajcie Państwo definicję i implementację obiektów OknoDialogoweP i OknoDialogoweM. Obiekt polimorficzny jest dużo prostszy w definicji i dużo szybszy w implementacji. Mimo tego, że ta sama metoda zrealizowana jako polimorficzna zajmuje więcej miejsca (dochodzą tu niewidoczne dla programisty nagłówki), program wykorzystujący obiekty polimorficzne jest mniejszy (nie trzeba przechowywać zestawów jednakowych procedur w pamięci). Dodatkowo, co częściowo było założeniem twórców programowania zorientowanego obiektowo, łatwiej jest ukrywać niektóre szczegóły implementacyjne poprzez tworzenie tzw. obiektów abstrakcyjnych (obiekt abstrakcyjny to jakby szkielet, w którym w procesie dziedziczenia implementowane są tylko metody polimorficzne, a szkielet pozostaje stały). Dobrze zaprojektowany bazowy obiekt abstrakcyjny przyspiesza pracę przy tworzeniu programu ponad dwukrotnie.
Różnicę między Turbo Pascalem a C++ w programowaniu obiektowym nie są zbyt wielkie. Jako, że używałem w swoim artykule składni i określeń charakterystycznych dla Turbo Pascala, czuję się w obowiązku zaznaczyć różnice związane z problemami omawianymi powyżej, które będą łatwe do wychwycenia dla znawców języka C:

1. W C++ do definicji obiektów (klas) używa się tych samych słów kluczowych, które używane były do definicji rekordów (class, struct, union).

2. Konstruktory i destruktory mają w C++ nazwy obligatoryjne, pochodzące od nazw klas. W Turbo Pascalu nazwy są dowolne, choć preferowane są odpowiednio Init i Done.

3. Konstruktory i destruktory klas są w C++ uruchamiane automatycznie w chwili deklaracji użycia zmiennej obiektowej (przydzielanie pamięci) i przy zakończeniu jej życia (zwalnianie pamięci). W Turbo Pascalu konstruktory i destruktory trzeba wywoływać jawnie.

4. W Turbo Pascalu może istnieć więcej niż jeden destruktor (w C++ jest to niemożliwe). Jeśli nie definiujemy żadnego destruktora to go po prostu nie ma (w C++ tworzony jest domyślny - pusty destruktor).

Mirosław Wieczorek

Podstawowe zagadnienia programowania zorientowanego obiektowo - polimorfizm i dziedziczenie klas w języku C++.

Wstęp

Język C++, opracowany na początku lat osiemdziesiątych przez Bjarne Stroustrupa, jest nowoczesnym językiem programowania ogólnego przeznaczenia. Z paroma małymi wyjątkami C++ jest nadzbiorem języka C, co oznacza, ze prawie wszystko co jest dozwolone w C jest dozwolone również w C++. Oczywiście w C++ wprowadzono wiele rozszerzeń usprawniających pracę programisty oraz mechanizmy umożliwiające programowanie zorientowane obiektowo. I właśnie ta możliwość programowania obiektowego w połączeniu z efektywnością programowania odziedziczoną po języku C sprawiła, że C++ stało się nowym językiem programowania i coraz bardziej zyskuje na popularności.

W tradycyjnym, proceduralnym podejściu do programowania, program opisuje serię czynności, które mają zostać wykonane żeby osiągnąć pożądane wyniki, czyli po prostu algorytm. Natomiast w podejściu obiektowym program opisuje system czy też zbiór wzajemnie oddziaływujących na siebie obiektów, w pewien sposób hierarchicznie zorganizowanych i wymieniających między sobą informacje.

Programowanie obiektowe opiera się zasadniczo na kilku kluczowych koncepcjach. Są to:

1. Abstrakcja danych i procedur czyli oderwanie się od szczegółów implementacji struktur danych i procedur a skoncentrowanie się na istocie rzeczy.
2. Koncepcja klasy czyli definiowanego przez programistę nowego typu określonego przez pewien zbiór wartości i operacji, które na tym typie mogą być wykonywane. Z koncepcją klasy wiąże się ściśle pojęcie hierarchii klas.
3. Hermetyzacja czyli ukrywanie szczegółów implementacyjnych i rozróżnienie między częścią implementacyjną a definicyjną.
4. Dziedziczenie kodu czyli możliwość wykorzystania już istniejących definicji klas czy też całej hierarchicznej ich struktury.

Oczywiście nic nie stoi na przeszkodzie pisaniu programów opartych na tych założeniach przy wykorzystaniu dowolnego języka programowania, jednak tylko w przypadku języków zorientowanych obiektowo koncepcje te są elementem języka i są ściśle związane z jego opisem.
Język C++ stał się językiem obiektowym dzięki wprowadzeniu do jego definicji koncepcji klasy zapewniającej wszystkie wymienione powyżej istotne cechy wyróżniające programowanie obiektowe. W dalszej części zostanie dokładnie przedstawiony sposób definicji **klasy** w języku C++, przykłady konstruowania

hierarchii klas wraz z przykładami dziedziczenia kodu.

Oprócz tego zostanie zaprezentowany sposób definicji i użycia funkcji wirtualnych dających możliwość tzw. dynamicznego łączenia (dynamic binding) oraz stosowania funkcji polimorficznych.

Pojęcie klasy

W języku C++ klasa jest typem danych definiowanym przez użytkownika. Deklaracja klasy określa reprezantację i zbiór wartości obiektu danej klasy oraz zbiór operacji, które można na obiektach danej klasy przeprowadzić. Klasy mogą służyć, jak wspomniano, do wprowadzania nowych typów np. klasa **Drzewo_binarne** czy **Stos** oraz do definiowania abstrakcji, które w naturalny sposób nie odpowiadają predefiniowanym typom z języka C++, na przykład klasa **Okno**, klasa **Student** czy też klasa **Dom**. Ilość rodzajów klas jest ograniczona tylko inwencją programisty.
W języku C++ z klasą są związane cztery atrybuty:

1. Zbiór pól składowych określających reprezentację klasy, przy czym ilość pól może wynosić zero.
2. Zbiór funkcji składowych określających zbiór operacji, które można wykonywać na obiekcie danej klasy. W klasie nie jest konieczne deklarowanie funkcji składowych.
3. Widoczność elementów klasy, pozwalająca na hermetyzację i ukrywanie reprezentacji klasy.
4. Nazwa klasy, służąca jako identyfikator typu i mogąca występować wszędzie tam gdzie mogą wystąpić identyfikatory typów predefiniowanych.

Definicja klasy

Definicja klasy składa się z dwóch części: nagłówka klasy złożonego ze słowa kluczowego **class** oraz nazwy klasy oraz ciała klasy objętego nawiasami klamrowymi. Po zamykającym nawiasie klamrowym musi wystąpić średnik lub lista deklaracji. Na przykład:

```
class okno
{
   int xl, yl, rozmiar_x,
                        rozmiar_y ;
   char *bufor ;
   int ramka ;
} ;
```

Zdefiniowaliśmy klasę o nazwie **okno** zawierającą sześć pól: cztery opisujące położenie okna i jego rozmiar, jedno zawierające adres obszaru przechowującego zawartość okna oraz jedno określające rodzaj ramki. W ten sposób opisaliśmy reprezentację klasy.
Tak zdefiniowanej klasy i jej nazwy możemy używać wszędzie tam, gdzie używamy typów predefiniowanych. Na przykład:

```
okno o1, o2 ;
        // dwa obiekty klasy okno
okno tablica_okien[ 20 ] ;
        // tablica 20 elementów
        // klasy okno
int s = sizeof( okno ) ;
        // przypisanie zmiennej s
        // rozmiaru pamięci
        // zajmowanej przez obiekt
```

Zdefiniujemy teraz klasę zawierającą oprócz pól składowych również zestaw operacji:

```
class lista
{
   int wartosc ;
   lista *nast ;
   void dodaj( int war ) ;
   void drukuj() ;
} ;
```

Klasa **lista** oprócz pól składowych zawierających przechowywaną wartość oraz wskaźnik do następnego elementu w liście zawiera również zbiór, w tym wypadku dwóch, operacji, które można wykonywać na obiekcie klasy **lista**, realizujących dodawanie nowego elementu i drukowanie zawartości listy. W tej chwili nie będziemy się zajmować implementacją tych operacji.

Do tej pory mieliśmy do czynienia z trzema atrybutami klasy, nic nie mówiliśmy o widoczności i dostępności składowych klasy, a jest to niestety bardzo istotne. Tak istotne, że poprzednio zdefiniowane klasy są w zasadzie bezużyteczne.

Widoczność każdej ze składowych pewnej klasy można określić na trzy sposoby: składowa klasy może być publiczna, prywatna lub zabezpieczona. Co to znaczy ?

Otóż dostęp do składowych publicznych nie jest z żaden sposób ograniczany, czyli mogą być używane (są widoczne) w dowolnym miejscu w programie, w którym jest widoczna deklaracja klasy, natomiast w przypadku składowych prywatnych, są one widoczne tylko w obrębie funkcji składowych danej klasy (dla ścisłości również w funkcjach zaprzyjaźnionych i w funkcjach składowych klas zaprzyjaźnionych). Pole, czy też składowa zabezpieczona ma podobne zasady widoczności jak składowa prywatna z tym, że składowa zabezpieczona jest również widoczna w funkcjach składowych klas pochodnych. Co to dokładnie znaczy, zostanie omówione w dalszej części tej pracy.

Dlaczego więc poprzednie definicje są w zasadzie bezużyteczne? Mając zdefiniowaną klasę **lista** zadeklarujmy obiekt o nazwie **l1**:

```
lista l1 ;
```

Dostęp do składowych klasy (pól i funkcji) odbywa się identycznie jak w przypadku struktur i unii znanych z języka C, czyli podaje się identyfikator zmiennej (w naszym przypadku będzie to reprezentant naszej klasy czyli obiekt) a następnie po kropce identyfikator składowej (pola albo funkcji). Na przykład:

```
l1.wart = 12 ;
```

albo

```
l1.drukuj() ;
```

W naszym przypadku odwołania te będą jednak nieprawidłowe ponieważ w klasie lista **wszystkie** składowe są **prywatne**, ponieważ przyjmuje się, że bez wyszczególnionego specyfikatora dostępu wszystkie składowe klasy są prywatne. Zmieńmy teraz definicję naszej klasy lista w ten sposób aby była bardziej przydatna.

```
class lista
{
  public:
     int wart ;    // pole publiczne
     void dodaj( int war ) ;
                   // funkcja publiczna
     void drukuj() ;
                   // funkcja publiczna
  private:
     lista *nast ; // pole prywatne
} ;
```

Specyfikator widoczności składa się z jednego z trzech słów kluczowych: **public:**, **private:** i **protected:**, i powoduje zmianę domyślnej widoczności składowych klasy. Zmiana ta rozciąga się od miejsca wystąpienia specyfikatora aż do nawiasu klamrowego kończącego definicję klasy lub do wystąpienia kolejnego specyfikatora. W ramach jednej definicji klasy może wystąpić dowolna ilość specyfikatorów i w dowolnej kolejności. Rozważmy następujące odwołania do pól składowych:

```
lista l1, l2 ;

l1.wart = 123 ;      // o'k
l1.dodaj( 1 ) ;      // o'k
l1.drukuj() ;        // o'k
l1.nast = &l2 ; // blad, pole nast
          //jest polem prywatnym
```

Właściwy dobór specyfikatorów widoczności umożliwia właśnie hermetyzację i ukrywanie szczegółów implementacyjnych. To czego nie widać, czyli składowe prywatne, można w dowolny sposób zmieniać bez potrzeby zmian w później stworzonym kodzie. Oczywiście o ile te zmiany będą sensowne. Zalecane jest również, aby dostęp do pól składowych

był realizowany przez zbiór odpowiednich funkcji składowych a same pola były prywatne. Uniemożliwia to przypisanie polom składowym niepożądanych wartości.

Mówiąc o definicji klasy nie od rzeczy będzie wspomnieć o specjalnych funkcjach składowych jakimi są konstruktory i destruktory.
Konstruktor jest funkcją składową o nazwie identycznej z nazwą klasy i jest niejawnie wywoływany w momencie tworzenia obiektu danej klasy. Konstruktor w najprostszym przypadku ma za zadanie inicjalizację pól składowych klasy określonymi wartościami. Dla bardziej skomplikowanych klas może to być jeszcze otwieranie plików (np. klasa opisująca plik zorganizowany w strukturę B-drzewa) realizowanie przydziału pamięci (np. klasa opisująca tablicę dynamiczną). U nas to będzie po prostu nadanie polu **nast** wartości 0 co oznacza pusty wskaźnik. Rozszerzmy naszą klasę **lista** o taki właśnie konstruktor:

```
class lista
{
  public:
      lista() { nast = NULL ; }
    int wart ;
      void dodaj( int war ) ;
      void drukuj() ;
  private:
      lista *nast ;
                  // pole prywatne
} ;
```

Konstruktor nie może zwracać żadnej wartości i **musi** być niestatyczną funkcją składową danej klasy. Warto zauważyć, że w ciele definicji klasy pojawiła się definicja funkcji. Jest to całkowicie poprawne. Definicja funkcji składowej może znajdować się albo w ciele definicji klasy jak w ostatnim przykładzie, albo poza nią. Jednak w tym drugim przypadku w definicji klasy powinna się pojawić jej deklaracja prototypowa (tak jak we wszystkich poprzednich przykładach).

Destruktor jest funkcją, która jest niejawnie wywoływana w momencie niszczenia obiektu danej klasy. Dla obiektów statycznych będzie to miało miejsce po zakończeniu działania programu. Dla obiektów automatycznych odbędzie się to w momencie opuszczenia bloku w którym obiekt został utworzony. Nazwa destruktora odpowiada nazwie klasy poprzedzonej znakiem tyldy (~). Do naszej definicji klasy dopiszemy teraz destruktor:

```
class lista
{
                  // poprzednie składowe
      ~lista() ;
   // ciało funkcji w innym miejscu
  private:
      lista *nast ;
};
```

To co destruktor ma wykonywać zależy tylko i wyłącznie od programisty, może to być na przykład zamknięcie pliku czy też zwolnienie przydzielonej obiektowi pamięci. W naszym przypadku destruktor powinien zwolnić pamięć przydzielaną kolejnym elementom w funkcji **dodaj()**.

Dziedziczenie klas

Dziedziczenie jest, jak już wspomniano, jedną z kluczowych koncepcji programowania zorientowanego obiektowo. Umożliwia wykorzystanie już istniejącego kodu oraz, co wydaje się ważniejsze, wykorzystanie funkcji polimorficznych. Rozpatrzmy następującą definicje klasy:

```
class osoba
{
  public:
    char nazwisko[ 30 ] ;
} ;
```

Dla uproszczenia pominiemy funkcje realizujące dostęp do pól klasy, a same pola uczynimy publicznymi. Mamy klasę o nazwie **osoba**, która to klasa opisuje po prostu jakąś osobę. Powiedzmy , że chcemy teraz uczynić z tej osoby pracownika pewnego zakładu. Zamiast implementować drugą klasę, która będzie zawierać informacje o nazwisku i, powiedzmy, pensji możemy wykorzystać

już istniejącą klasę **osoba** i utworzyć nową klasę poprzez dziedziczenie.

```
class pracownik : public osoba
{
  public:
    long pensja ;
} ;
```

Przy takiej definicji klasy **pracownik** mamy do czynienia z klasą pochodną (**pracownik**), która dziedziczy po klasie **osoba**, która jest klasą bazową. W czasie dziedziczenia, wszystkie składowe klasy bazowej wchodzą do klasy pochodnej. Można nawet powiedzieć, że klasa pochodna **jest** klasą bazową ewentualnie rozszerzoną o nowe możliwości. W naszym przykładzie **pracownik** jest jednocześnie **osobą**. Natomiast **osoba** nie jest **pracownikiem**. I nie należy tego rozumieć, że klasa **pracownik** zawiera pole typu **osoba**. Określenie tego czy jakaś klasa jest pochodną pewnej klasy bazowej jest zapisane w nagłówku definicji klasy.

Po słowie kluczowym **class** występuje nazwa klasy a po niej dwukropek. Po dwukropku natomiast występuje **lista** klas bazowych (w naszym przypadku jest to jedna klasa). Nazwy klas bazowych są rozdzielane przecinkami. Wynika z tego, że klasa może dziedziczyć właściwości niekoniecznie po jednej klasie bazowej. Mamy wtedy do czynienia z dziedziczeniem wielobazowym, którym jednak nie będziemy się na razie zajmować. Przed nazwą klasy, tak jak w naszym przykładzie może wystąpić specyfikator dostępu. Specyfikator **public** oznacza, że klasa bazowa jest upubliczniona. Brak takiego specyfikatora powoduje, że domyślnie klasa bazowa jest uprywatniona. Co to wszystko oznacza? W przypadku uprywatnienia klasy bazowej, wszystkie pola klasy bazowej stają się, z punktu widzenia klasy pochodnej, polami prywatnymi, natomiast w przypadku upublicznienia pola klasy bazowej zachowują swój tryb widoczności.

Wspomniana już zależność między klasą bazową a klasą pochodną powoduje to, że obiektowi klasy bazowej można bez żadnych konwersji przypisać obiekt klasy pochodnej. Nigdy odwrotnie i tylko wtedy gdy klasa bazowa jest upubliczniona:

```
osoba os1, os2 ;
pracownik p1, p2 ;

os1 = os2 ;          // o'k
p1  = p2  ;          // o'k
os1 = p1  ;          // o'k
p2  = os2 ;          // błąd !
```

Drugą interesującą rzeczą jest możliwość przypisywania wskaźnikowi na obiekt klasy bazowej adresu (lub wskaźnika) obiektu klasy pochodnej i, tak jak w poprzednim przypadku, klasa pochodna musi mieć upublicznioną klasę bazową oraz przypisanie odbywa się bez specjalnych operatorów konwersji. Na przykład:

```
osoba os, *pos ;
pracownik pr, *ppr ;

pos = &os ;          // o'k
ppr = &pr ;          // o'k
pos = &pr ;          // o'k
pos = ppr ;          // o'k
ppr = &os ;          // błąd
```

Wprowadzając następną w hierarchii klasę możemy napisać:

```
class kierownik : public pracownik
{
  public:
    char wydział ;
    long dodatek ;
} ;
kierownik k ;
osoba os ;
os = k ;                    // o'k
```

Powyższy zapis jest całkowicie poprawny ponieważ kierownik **jest** pracownikiem a co za tym idzie jest również osobą.

Pewne wątpliwości może budzić mechanizm takiego przypisania, który w rzeczywistości jest bardzo prosty. Polom obiektu **os** klasy **osoba** zostaną nadane wartości odpowiednich pól z obiektu **k** typu **kierownik**.

```
#include <iostream.h>
#include <string.h>
class osoba
{
  public:
     char nazwisko[ 30 ] ;
     virtual zarobek()
     {
        cout << nazwisko << " nie jest pracownikiem\n" ;
     }
} ;
class pracownik : public osoba
{
  public:
     long pensja ;
     zarobek()
     {
        cout << nazwisko << " zarabia " << pensja << endl ;
     }
} ;
class kierownik : public pracownik
{
  public:
     char wydział ;
     long dodatek ;
     zarobek()
     {
        cout << nazwisko<< " jest szefem i zarabia "
             << pensja+dodatek << endl;
     }
} ;
main()
{
  osoba *pracownicy[ 20 ] ;
  int liczba_osób  = 0 ;

  osoba os ;
  strcpy( os.nazwisko, "Kowalski" ) ;

  pracownik pr ;
  strcpy( pr.nazwisko, "Nowak" ) ; pr.pensja = 10000 ;

  kierownik kier ;
  strcpy( kier.nazwisko, "Berda" ) ;
  kier.pensja = 100000 ; kier.dodatek = 20000 ;

  pracownicy[ liczba_osób++ ] = &pr ;
  pracownicy[ liczba_osób++ ] = &os ;
  pracownicy[ liczba_osób++ ] = &kier ;

  for( int i = 0 ; i < liczba_osób ; i++ )
     pracownicy[ i ]->zarobek() ;
}
```

WYDRUK 1.

Funkcje polimorficzne

Na początek rozpatrzmy prosty przykład:

```
#include <iostream.h>
class A
{
  public:
    virtual void pisz()
    {
      cout << "Klasa A\n" ;
    }
} ;
class B : public A
{
  public:
    void pisz()
    {
      cout << "Klasa B\n" ;
    }
} ;
A a, *pa ;
B b, *pb ;
main()
{
  pb.pisz()
        // drukuje napis "Klasa B"
  pa = &b
  pa->pisz() ;   // również drukuje
              // napis "Klasa B" !!
}
```

Dlaczego tak się dzieje? Przecież wskaźnik **pa** jest typu **A *pa** więc zapis **pa->pisz()** powinien sugerować wywołanie funkcji **pisz()** należącej do klasy **A**. Dlaczego więc nie wywołuje ? W tym przypadku mamy bowiem do czynienia z bardzo istotnym udogodnieniem C++ - funkcjami wirtualnymi czy też polimorficznymi. Polimorficznymi dlatego, że funkcja o nazwie **pisz()** zachowuje się różnie dla obiektów o różnych typach, nawet jeśli typ wskaźnika mówi zupełnie coś innego.

W C++ deklaracja funkcji wirtualnej zaczyna się od słowa kluczowego **virtual**, po którym następuje normalna deklaracja funkcji, to znaczy typ wyniku, identyfikator funkcji oraz lista parametrów. Funkcja taka musi być niestatyczną funkcją składową klasy. Taka deklaracja tworzy w zasadzie rodzinę funkcji polimorficznych bowiem każda funkcja zadeklarowana w klasie pochodnej i mająca taką samą nazwę, ten sam typ zwracanego rezultatu oraz taką samą listę parametrów jest funkcją wirtualną. Niezależnie od tego czy została zadeklarowana ze słowem kluczowym **virtual** czy też bez niego. Można również powiedzieć, że w ciągu klas funkcja staje się wirtualna od momentu zadeklarowania jako wirtualna, wcześniej jest to zwykła funkcja składowa. Gdyby w poprzednim przykładzie jako wirtualną zadeklarować funkcję w klasie **B** a nie w **A**, na ekranie pojawiłyby się napisy:

```
Klasa A
Klasa B
```

Rozpatrzmy teraz bardzo prosty przykład użycia funkcji wirtualnych:
W przykładzie tym tworzymy tablicę zawierającą informacje o trzech osobach a następnie drukujemy informacje o ich wynagrodzeniu. Wynikiem działania naszego programu jest następujący wydruk:

```
Nowak zarabia 10000
Kowalski nie jest pracownikiem
Berda jest szefem i zarabia 120000
```

Najbardziej interesujące w tym programie jest to, że w pętli drukującej zarobek w żaden sposób nie sprawdzamy, którą funkcję drukującą wywołać. To wszystko robi za nas kompilator. Pisząc ten program w C lub w innym języku, umożliwiającym stosowania funkcji wirtualnych musielibyśmy wprowadzić w definicjach klas (czy też raczej używalibyśmy struktur lub rekordów) dodatkowe pola identyfikujące typ obiektu, a następnie musielibyśmy badać wartość tego pola w pętli drukującej informacje, na przykład za pomocą rozbudowanej instrukcji **switch**.
Byłoby to szczególnie kłopotliwe jeśli operowalibyśmy na dużej liczbie typów obiektów.

Bibliografia

[1] Stroustrup B.: *The C++ Programming Language*, Addison-Wesley 1987.
[2] Stroustrup B., Ellis M.: *The Annotated C++ Reference Manual*, Addison-Wesley 1990.
[3] Lippman S. B.: *C++ Primer*, 2-nd Edition, Addison-Wesley 1991.

Marek Średniawa
Instytut Telekomunikacji Politechniki Warszawskiej

Podejście obiektowe do specyfikacji na przykładzie języka CCITT SDL'92

Streszczenie

Przedmiotem referatu są zmiany i rozszerzenia wprowadzone przez CCITT do najnowszej wersji standardu języka specyfikacji SDL, które umożliwiają stosowanie podejścia obiektowego do specyfikacji systemów telekomunikacyjnych, protokołów komunikacyjnych oraz szerokiej klasy systemów czasu rzeczywistego. Dla przypomnienia, w zwięzły sposób, zostanie przedstawiona podstawowa wersja języka SDL. Główna uwaga zostanie skupiona na rozszerzeniach obiektowych tzn. mechanizmach abstrakcji, parametryzacji, hermetyzacji, specjalizacji, dziedziczenia, polimorfizmu. Prezentacja uwzględnia również inne modyfikacje, istotne z punktu widzenia specyfikacji rozproszonych systemów czasu rzeczywistego. W podsumowaniu rozważone zostaną aspekty praktyczne stosowania języka SDL.

1. Wprowadzenie

Język SDL (Specification and Description Language) ma dość długą tradycję użytkowania, nie tylko w zastosowaniach telekomunikacyjnych, i od 1976 roku kiedy po raz pierwszy został ujęty w normę międzynarodową, podlegał ewolucji w cyklu czteroletnim (zgodnie z harmonogramem pracy CCITT). Początkowe wersje języka wywodziły się z nieformalnych lub półformalnych metod opartych na graficznych diagramach stanów przejść. Ewolucja przebiegała w kierunku wzbogacenia języka o konstrukcje pozwalające specyfikować szeroką klasę systemów oraz zmierzała ku zdefiniowaniu semantyki języka w sposób formalny. Wersja języka zdefiniowana w roku 1988 była już stabilnym standardem powszechnie akceptowanym w środowisku telekomunikacyjnym. Dość długi okres zbierania doświadczeń oraz prowadzone studia doprowadziły do ustabilizowania się i upowszechnienia stosowania SDLa przez duże firmy telekomunikacyjne i krajowe administracje łączności. SDL jest także wykorzystywany przez CCITT i ETSI (European Telecommunication Standards Institute) do formułowania zaleceń i standardów (por. np. seria zaleceń Q opisująca system sygnalizacji nr 7, seria Q.1200 dotycząca modelu i usług sieci IN, zalecenia serii I dotyczące usług dodatkowych w ISDN).

Wśród innych znanych formalnych metod specyfikacji i opisu, określanych terminem FDT (Formal Description Techniques), takich jak np. Estelle czy LOTOS, język SDL wyróżnia się następującymi cechami:

- istnieniem notacji graficznej SDL/GR i równoważnej jej semantycznie notacji tekstowej; możliwe jest mieszanie obu postaci składni w tej samej specyfikacji;
- możliwością umieszczania w specyfikacji elementów nieformalnych, co pozwala na stopniowe uszczegółowianie i uściślanie specyfikacji w kolejnych etapach projektowania;
- dostosowaniem do obiektowego stylu specyfikacji i projektowania;
- intuicyjnością modelu.

Język SDL jest używany głównie do specyfikacji protokołów komunikacyjnych, oprogramowania sterującego dla central telefonicznych i węzłów sieci komputerowych, usług dodatkowych (np. dla ISDN czy IN) oraz innych systemów należących do kategorii rozproszonych systemów czasu rzeczywistego. Zwraca się uwagę, że z natury rzeczy, systemy telekomunikacyjne są systemami otwartymi, w związku z czym użycie języka SDL wiąże się także ze stosowaniem modelu odniesienia systemów otwartych OSI (Open System Interconnection Reference Model) będącego ugruntowanym standardem ISO.

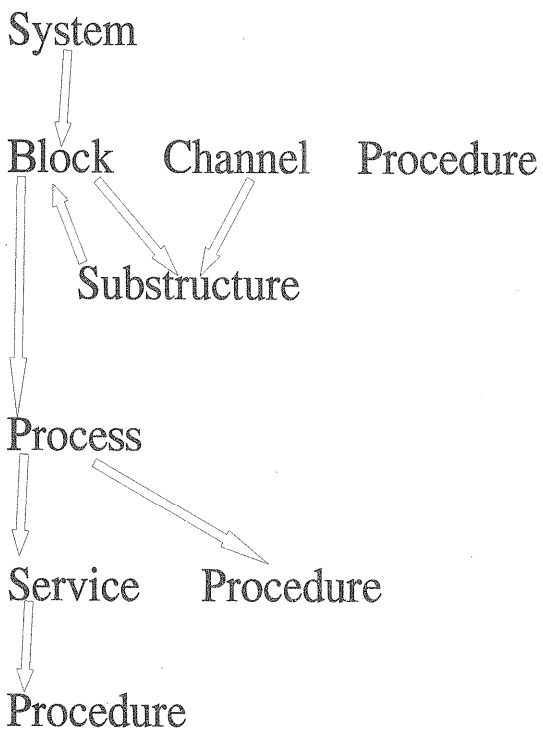
Fakt, że SDL jest oparty na ogólnej, uniwersalnej i intuicyjnie zrozumiałej koncepcji komunikujących się asynchronicznie za pomocą sygnałów automatów skończonych sprawia, że język i metodologia jego stosowania zyskały akceptację w środowisku telekomunikacyjnym i może być przedmiotem zainteresowania dla wszystkich osób zajmujących się projektowaniem rozproszonych i współbieżnych systemów czasu rzeczywistego. Rozszerzenia obiektowe, wprowadzone w najnowszej wersji języka SDL'92, czynią go wygodnym środkiem do projektowania dużych systemów.

2. Podstawowy SDL - omówienie

W niniejszym rozdziale przedstawiona jest bardzo zwięzła charakterystyka języka SDL [2], [5] (SDL był już prezentowany podczas jednej z poprzednich Szkół PTI, w kontekście zastosowań do specyfikacji systemów czasu rzeczywistego [1]), której celem jest wyjaśnienie modelu semantycznego, oraz zasad budowy specyfikacji zarówno z punktu widzenia statycznego jak i dynamicznego.

Specyfikacja w języku SDL opisuje model systemu otwartego. Oznacza to, że system zdefiniowany przez specyfikację może współdziałać z otoczeniem, którego model zachowania daje się również wyrazić zgodnie z regułami języka SDL. Interakcja systemu z otoczeniem odbywa się za pomocą sygnałów, które są przenoszone kanałami łączącymi bloki systemu z otoczeniem. Zakłada się, że otoczenie zachowuje się tak jak inny system wyspecyfikowany w języku SDL, tzn. że rozważa się je jako złożone z bloków zawierających procesy wysyłające i odbierające sygnały do i z systemu poprzez kanały.

Z punktu widzenia statycznej struktury specyfikacji **system** jest zbiorem **bloków** połączonych **kanałami** FIFO między sobą i z **otoczeniem** systemu. Bloki i kanały mogą być dalej, w sposób rekurencyjny, dekomponowane na odpowiednio: substruktury bloków i kanałów. Bloki są dalej dekomponowane na procesy sekwencyjne. Procesy mogą definiować procedury i korzystać z nich. Dodatkowy mechanizm dekompozycji procesu stanowią tzw. usługi, które odzwierciedlają rozłączne, wzajemnie wykluczające się funkcje - wątki procesu. Statyczny aspekt budowy specyfikacji ilustruje rys. 1.



RYS. 1. Struktura statyczna specyfikacji

Z punktu widzenia struktury dynamicznej specyfikacja stanowi zbiór procesów sekwencyjnych komunikujących się za pomocą sygnałów, przy czym procesy mogą być dynamicznie kreowane i unicestwiane.

Zachowanie się bloku jest określone przez zachowanie procesów do niego należących. Definicja procesu opisuje wzorzec zachowania wystąpień procesów kreowanych dynamicznie na podstawie tej definicji.

Procesy współdziałają z innymi procesami i z otoczeniem za pomocą sygnałów. Komunikacja za pomocą sygnałów jest asynchroniczna: proces nadawca po wysłaniu sygnału do procesu adresata nie jest przez niego wstrzymywany. W definicji procesu można określić liczbę jego wystąpień powoływanych do życia w chwili rozpoczęcia interpretacji systemu oraz maksymalną liczbę jednocześnie aktywnych wystąpień tego procesu.

W celu identyfikacji procesów wprowadzono predefiniowany typ danych **PId** reprezentujący tożsamość wystąpień procesów. Z każdym wystąpieniem procesu związane są cztery atrybuty, o wartościach typu **PId** określające odpowiednio tożsamość:

- wystąpienia procesu "ojca" (**PARENT**);
- własną wystąpienia procesu (**SELF**);
- wystąpienia ostatnio wykreowanego potomka (**OFFSPRING**);
- wystąpienia procesu, od którego został ostatnio odebrany sygnał (**SENDER**).

Z sygnałem mogą być związane dane i wówczas ich wartości są przekazywane z procesu do procesu. Sygnał nadany przez proces w akcji wyjścia jest przekazywany do pojedynczego procesu adresata, jawnie wskazanego lub arbitralnie wybranego spośród wystąpień procesu pewnego typu. Istnieje również możliwość jednoczesnego wysłania sygnału do wielu procesów *(multicast)*.

Kanały stanowią kolejki FIFO reprezentujące drogi przenoszenia sygnałów między blokami. Kanały z definicji wnoszą pewne niedeterministyczne opóźnienie przy przekazywaniu sygnałów, z wyjątkiem przypadku, w którym definicja kanału została opatrzona atrybutem NODELAY (bez opóźnienia).

Drogi sygnałowe łączą procesy należące do bloku między sobą oraz z kanałami dochodzącymi i wychodzącymi z bloku; Drogi sygnałowe przenoszą sygnały między procesami należącymi do tego samego bloku i między procesami a kanałami, nie wprowadzając, w odróżnieniu od kanału, opóźnienia. Drogi sygnałowe dochodzące do procesu połączone są z **portem procesu**.

Całość drogi łączącej parę procesów i przenoszącej sygnały z procesu do procesu stanowi **marszrutę sygnałów**. Gdy sygnał przybywa marszrutą do procesu docelowego tzn. znajdzie się na końcu kolejki sygnałów związanej z marszrutą, to jest on odbierany i przechowywany w porcie procesu aż do chwili skonsumowania tego sygnału. Sygnały są konsumowane w porządku odbierania. Modelem portu jest kolejka FIFO o nieograniczonej pojemności.

Proces w trakcie swego cyklu życiowego bądź znajduje się w **stanie**, oczekując na sygnał, bądź w **przejściu**, wykonując sekwencję **akcji**. Procesy powoływane do życia w chwili kreacji systemu mają wyróżnione miejsce - symbol startu, od którego rozpoczyna się przejście inicjalizujące proces. W przejściu po symbolu startu może wystąpić dowolna akcja właściwa dla przejścia, w szczególności akcja kreacji wystąpienia innego procesu.

Gdy proces znajduje się w stanie, to może pobrać z portu oczekujący sygnał (z początku w kolejki portu). Jeśli jest to sygnał, który ma być **zachowany** to pozostaje on w porcie i pobierany jest następny sygnał. Zachowanie sygnału jest wyrażane przez

wyspecyfikowanie dla niego czynności zachowania (**save**) zamiast wejścia, w danym stanie. Jeśli jest to sygnał nie odpowiadający żadnemu wyspecyfikowanemu w tym stanie wejściu czy symbolowi zachowania, to zostanie on skonsumowany przez "**domyślne wejście**" związane z każdym stanem a proces pozostanie w tym samym stanie.

Jeśli jest to sygnał odpowiadający wyspecyfikowanemu **wejściu**, to zostanie on skonsumowany, dane w nim zawarte stają się dostępne dla procesu i rozpoczyna się wykonanie przejścia związanego z tym wejściem.

Repertuar akcji w przejściach obejmuje:
- zadania - odwołujące się do danych i czasu oraz modyfikujące dane;
- ustawianie i kasowanie limitów czasów oczekiwania *(time-out)*;
- wyjście - wysłanie sygnału;
- kreację nowego wystąpienia procesu;
- unicestwienie wystąpienia procesu;
- decyzję lub wybór wielościeżkowy odwołujące się do danych i czasu, które wprowadzają w specyfikacji rozgałęzienia;
- wskazanie następnego stanu lub miejsca kontynuacji oznaczonego etykietą;
- wywołanie procedury (w tym mechanizm zdalnego wywoływania procedur - RPC);
- powrót z procedury;
- wywołanie makrodefinicji;
- wybór opcji.

W trakcie wykonywania przejścia dane procesu mogą być modyfikowane (w zadaniach), proces może wysłać sygnał, który może przenosić dane do innego procesu, może kreować inne procesy, wywoływać procedury. Przejście może się rozgałęziać w decyzjach uwarunkowanych wartościami obiektów. Istnieje również możliwość odwoływania się do wartości zmiennych należących do innych procesów pod warunkiem użycia odpowiedniego trybu deklaracji danych. Przejście kończy się osiągnięciem przez proces następnego stanu lub wykonaniem akcji **STOP** powodującej unicestwienie procesu.

W specyfikacji istnieje możliwość odwoływania się do czasu absolutnego reprezentowanego przez **NOW** oraz do czasu względnego, czyli wyrażania różnicy czasu i czasów trwania lub oczekiwania. Wprowadzony jest specjalny typ obiektu - **TIMER** służący do deklarowania stoperów odmierzających czas oczekiwania.

Podobnie jak w innych językach również w SDLu istnieją mechanizmy definiowania, deklarowania i odwoływania się do danych. Różnica w stosunku do języków programowania polega na przyjęciu matematycznego modelu danych, w którym nie występują problemy związane z zakresem, skończonością i dokładnością reprezentacji.

W specyfikacji można odwoływać się do predefiniowanych typów danych lub samemu konstruować aksjomatyczne definicje nowych typów danych. Lista predefiniowanych typów danych obejmuje następujące typy:

- całkowity - **Integer**;
- rzeczywisty - **Real**;
- naturalny - **Natural**;
- logiczny - **Boolean**;
- znakowy - **Character**;
- identyfikacyjny - **PId**;
- czasowy: czas trwania - **Duration**;
- czasowy: moment czasu - **Time**;
- napisowy - **Charstring**.

Deklaracje zmiennych mogą pojawić się tylko w definicjach procesów i procedur. Oznacza to, że zmienne są lokalne w procesach i procedurach czyli nie ma zmiennych "globalnych", które mogłyby być modyfikowane przez wiele procesów. Natomiast definicje typów danych mogą występować na poziomie systemu, bloku, substruktury bloku oraz w procesach i procedurach. Są one widoczne na zasadzie dziedziczenia nazw w jednostkach zagnieżdżonych. Ewentualny konflikt nazw jest

rozstrzygany przez kwalifikację nazwy nazwą jednostki, w której została ona wprowadzona.

Podstawowym sposobem przekazywania wartości danych pomiędzy procesami jest komunikacja za pomocą sygnałów, lecz oprócz tego wprowadzono dodatkowy mechanizm udostępniania wartości lokalnej zmiennej procesu innemu procesowi.

Nowe typy danych są definiowane aksjomatycznie. Definicja nowego typu wymaga:
- podania nazwy typu;
- określenia zbioru wartości z uwzględnieniem literałów;
- zdefiniowania operatorów;
- podania zbioru aksjomatów.

Nowe typy danych można definiować w zwięzły sposób, w oparciu o już zdefiniowane typy danych, korzystając z mechanizmu dziedziczenia. Polega on na wskazaniu, które elementy (operatory, literały, aksjomaty) są dziedziczone i na rozszerzeniu tego zbioru o dodatkowe wymagane cechy.

Warunkiem przydatności języka do specyfikacji systemów czasu rzeczywistego jest możliwość wyrażania uwarunkowań czasowych. W języku SDL przewidziano następujące mechanizmy:

- Predefiniowane typy danych **TIME** i **DURATION**. Pierwszy wyraża czas bezwzględny - wartość momentu czasu. Drugi reprezentuje czas trwania. Operator bezargumentowy **NOW** zwraca bieżący czas. Dla typu **TIME** zdefiniowane są operatory sumy, różnicy i przypisania. Dla typu **DURATION** dodatkowo zdefiniowane są operatory mnożenia i dzielenia. Ponadto dla obu typów zdefiniowane są operatory relacyjne.
- Deklaracja stoperów (**TIMER**) w połączeniu z akcjami **SET** i **RESET** wprowadzają do specyfikacji mechanizm odmierzania limitów czasów oczekiwania.
- Sygnał ciągły z uwarunkowaniem czasowym.
- Wejście uwarunkowane dozorem odwołującym się do czasu.

3. Podejście obiektowe do specyfikacji w języku SDL

3.1 Wprowadzenie

Jednym z głównych celów podejścia obiektowego do programowania jest umożliwienie wykorzystania napisanego już kodu w innych "pokrewnych" aplikacjach. Ma to znaczenie przy profesjonalnej implementacji dużych systemów oprogramowania, w których problemem jest panowanie nad złożonością oprogramowania w sensie rozmiaru, wersji i wariantów.

Innym uzasadnieniem stosowania podejścia obiektowego jest jego własność naturalnego porządkowania pojęć i zjawisk w odniesieniu do szerokiej klasy aplikacji. Właśnie z uwagi na tę cechę podejście obiektowe może być przydatne nie tylko w fazie implementacji lecz także we wczesnych fazach projektowania na etapie analizy i formułowania wymagań oraz specyfikacji systemów.

W paradygmacie obiektowym zjawiska charakterystyczne dla danej aplikacji są wyróżniane i reprezentowane jako *obiekty*. Zjawiska są usystematyzowane za pomocą pojęć reprezentowanych przez klasy obiektów. Organizacja klas w hierarchie pojęć pomaga zidentyfikować podobieństwa i różnice między zjawiskami. Nacisk na wczesną analizę różnic i analogii ułatwia konstrukcję uogólnionych klas obiektów, charakterystycznych dla pewnej kategorii zastosowań, które nadają się do wykorzystania również w innych aplikacjach.

Wspomniane zalety podejścia obiektowego dobrze korespondują z głównymi proble-

mami napotykanymi przy projektowaniu i implementacji systemów telekomunikacyjnych tzn. ze złożonością, rozmiarem, długim okresem eksploatacji oraz wielowersyjnością. Aparat pojęciowy i mechanizmy paradygmatu obiektowego ułatwiają rozwiązanie tych problemów. Rozważmy dla przykładu oprogramowanie dla rodziny central, od poziomu centrali końcowej do poziomu centrali międzynarodowej, które zawiera wspólne moduły oprogramowania i elementy sprzętowe występujące we wszystkich konfiguracjach. Można tu określić zarówno dla oprogramowania jak i dla konfiguracji sprzętowej relacje "rodzinne", które dają się dobrze wyrażać za pomocą mechanizmów dziedziczenia, specjalizacji i uogólniania oraz parametryzacji naturalnych w podejściu obiektowym.

W tradycyjnym podejściu aspekty funkcjonalne (dekompozycja funkcjonalna) oraz reprezentacji i modelowania danych są traktowane oddzielnie - w paradygmacie obiektowym elementy te można rozpatrywać łącznie.

Wprowadzenie mechanizmów obiektowych do języka SDL opiera się na następujących przesłankach:

- rzeczywiste systemy i ich modele są postrzegane w kategoriach zjawisk i pojęć;
- na ogół zjawiska są reprezentowane przez obiekty, natomiast pojęcia - za pomocą klas obiektów;
- mechanizm definiowania podklas jest właściwy do wyrażania specjalizacji pojęć, przy czym nacisk kładzie się na zagwarantowaniu zachowania własności klasy nadrzędnej przez obiekty podklasy;
- mechanizmem dziedziczenia powinny być objęte nie tylko dane i procedury ale również sekwencje akcji związane z obiektami.

Klasy obiektów są reprezentowane poprzez definicje typów, natomiast same obiekty są reprezentowane przez wystąpienia tych typów. Ponieważ w języku występuje szereg rodzajów obiektów to odpowiadają im różne definicje typów. W terminologii obiektowej specjalizacja jest wyrażana przez definiowanie podklas - w SDLu, w celu zachowania spójności z poprzednią wersją języka, do wyrażania specjalizacji służy definiowanie podtypów.

W poprzedniej wersji języka (1988) definicje sygnałów, procedur, stoperów, typów danych były definicjami typów, natomiast mechanizm definicji typów nie obejmował elementów struktury specyfikacji takich, jak system, blok, proces (definicje wprowadzały do specyfikacji od razu wystąpienia elementów struktury). Nowa wersja SDL'92 [1], [3] umożliwia definiowanie systemu, bloku i procesu jako typu.

Ponieważ wystąpienia bloków są połączone kanałami, a procesy używają dróg sygnałowych, istnieje potrzeba odpowiedniego parametrycznego sposobu wyrażenia kontekstu dla definicji typów systemowego, blokowego i procesowego. W tym celu wprowadzono pojęcie bramy *(gate)*, które reprezentuje potencjalny punkt połączenia dla kanału lub drogi sygnałowej.

Typ może być zdefiniowany jako specjalizacja innego typu. Typ macierzysty jest nazywany typem nadrzędnym, natomiast typ specjalizowany jest nazywany podtypem. Podtyp dziedziczy wszystkie własności wprowadzone w definicji typu nadrzędnego oraz może uzupełniać definicję o dodatkowe własności.

Obiektowa wersja SDL umożliwia redefinicję typów lokalnych (bloków, procesów i procedur) oraz przejść zdefiniowanych w typach nadrzędnych i powiązanie jej ze specjalizacją.

Typy i przejścia, które mogą być modyfikowane w definicjach podtypów są nazywane odpowiednio typami i przejściami wirtualnymi. Zredefiniowana definicja lub przejście w dalszym ciągu zachowują włas-

ność wirtualności, czyli możliwości dalszej redefinicji w kolejnych podtypach. Nadanie atrybutu *finalized*, czyli sfinalizowanie definicji, usuwa wirtualność definicji blokując możliwość dalszej modyfikacji.

Każdy typ może być sparametryzowany za pomocą tzw. parametrów generycznych, co pozwala na uniezależnienie definicji od konkretnego kontekstu. Takie sparametryzowane definicje mogą być umieszczane w modułach bibliotecznych do wykorzystania w specyfikacjach innych systemów. Typowymi parametrami generycznymi są sygnały. Parametr generyczny jest nazwą formalną elementu lub obiektu występującą w zasięgu otaczającym daną definicję typu.

W celu umożliwienia przeprowadzania kontroli zgodności typów zarówno bramy, parametry generyczne i obiekty wirtualne mogą być opatrzone ograniczeniami. Parametry generyczne, które są typami muszą być opatrywane ograniczeniami wyrażonymi w kategoriach typu ograniczającego. Parametry konkretne zastępujące generyczne parametry formalne muszą być podtypami typu ograniczającego. Podobnie redefinicje typów wirtualnych z ograniczeniami muszą określać podtypy typu ograniczającego.

Mechanizm modułów bibliotecznych umożliwia wielokrotne wykorzystywanie tych samych definicji typów w różnych specyfikacjach. Definicje typów umieszczone w modułach bibliotecznych *(package)* mogą być w całości lub części wykorzystywane ("importowane") w różnych specyfikacjach oraz w innych modułach bibliotecznych.

W celu ułatwienia procesu tworzenia specyfikacji a przede wszystkim umożliwienia wielokrotnego wykorzystania opracowanych wcześniej definicji lub ich elementów, do języka SDL wprowadzono mechanizm definicji typów dla elementów struktury specyfikacji (system, blok, proces, usługa). Mechanizmowi temu towarzyszą środki językowe służące do wyrażania parametryzacji, uogólniania, specjalizacji, hermetyzacji i polimorfizmu, związane z pojęciem dziedziczenia własności charakterystycznego dla obiektowego podejścia do projektowania.

Wspomniane mechanizmy językowe umożliwiają:
- wprowadzanie definicji typów dla konstrukcji strukturalnych (definicje te mogą być umieszczane zarówno w definicji systemu jak i w modułach, które mogą być później wykorzystywane w całości lub częściowo w innych specyfikacjach);
- definiowanie wystąpień konstrukcji strukturalnych w oparciu o wcześniej zdefiniowane typy;
- parametryzację definicji typów przez wykorzystanie parametrów kontekstowych;
- specjalizację typów poprzez definiowanie podtypów dziedziczących własności typu nadrzędnego i rozszerzających go o dodatkowe własności;
- wyrażanie polimorfizmu typów za pomocą mechanizmu redefinicji typów i przejść wirtualnych.

3.2 Typy, wystąpienia i bramy

W SDLu odróżnia się specyfikację wystąpień od specyfikacji typów. W specyfikacji typu strukturalnego, w odróżnieniu od specyfikacji wystąpienia, nie występują punkty połączeń z konkretnymi kanałami czy drogami sygnałowymi. Ze względu na ogólny charakter definicji typu strukturalnego są one reprezentowane przez tzw. bramy. Definicja typu stanowi podstawę do definiowania wystąpień danego typu. Mechanizmem definicji typów objęte są:
- system,
- blok,
- proces,
- procedura.

3.2.1 Definicje odwołujące się do typów

Definicje wystąpień odwołujące się do definicji typów strukturalnych (systemo-

wego, blokowego, procesowego, procedury) definiują konkretne wystąpienia obiektów danego typu, o własnościach wynikających z definicji tego typu. Ogólny mechanizm definiowania wystąpień odwołujących się do definicji typów polega na wykorzystaniu tzw. wyrażeń określających typ <type expression>. Wyrażenia określające typ umożliwiają definiowanie wystąpień odwołujących się do definicji typów przez podanie typu bazowego oraz tzw. kontekstowych parametrów konkretnych, które zastępują kontekstowe parametry formalne.

Wyrażenie określające typ <type expression> oznacza: albo typ określony przez identyfikator typu bazowego, albo typ nie wyróżniony nazwą, wynikający z zastąpienia w typie sparametryzowanym, określonym identyfikatorem typu bazowego, kontekstowych parametrów formalnych przez odpowiadające im parametry konkretne.

W przypadku gdy dla pewnego typu podano parametry konkretne tylko dla części kontekstowych parametrów formalnych to wówczas typ ten pozostaje nadal typem sparametryzowanym, przy czym jego lista kontekstowych parametrów formalnych obejmuje te parametry, które nie zostały zastąpione parametrami konkretnymi (kolejność parametrów formalnych w nowej liście zachowuje porządek wprowadzony na pierwotnej liście).

Typ systemowy

Definicja typu systemowego służy jako podstawa do definiowania wystąpień systemów danego typu. Wszystkie wystąpienia systemu mają własności określone w definicji tego typu systemowego. Definicja systemu jako typu umożliwia jego parametryzację. Do tego celu służą kontekstowe parametry formalne. W definicji typu systemowego kontekstowymi parametrami formalnymi mogą być: procedury, sygnały, typy danych i synonimy. Semantyka wystąpień systemów odwołujących się do typu systemowego jest zgodna z opisem definicji systemu.

Procedury

Procedura zachowuje swoje normalne podstawowe znaczenie. Przedstawione w niniejszym rozdziale elementy stanowią rozszerzenie definicji procedury z punktu widzenia podejścia obiektowego do specyfikacji.

Użycie w preambule procedury <procedure preamble> słowa kluczowego EXPORTED oznacza, że procedura może być wywoływana zgodnie z zasadami wywoływania procedur oddalonych (RPC - remote procedure call).

Użycie kontekstowych parametrów formalnych <formal context parameters> umożliwia uogólnioną parametryzację definicji procedury według zasad dla typów generycznych.

Zakres redefinicji procedury może być zawężony przez podanie ograniczenia <virtuality constraint>.

Użycie specjalizacji <specialization> charakteryzuje sposób dziedziczenia własności procedury przez definicje podtypów, dla których stanowi ona typ nadrzędny.

3.3 Brama

Wystąpienia bloków i procesów są łączone ze swoim otoczeniem za pomocą konkretnych ścieżek komunikacyjnych składających się z kanałów i dróg sygnałowych. Interpretacja ścieżek komunikacyjnych dla typów blokowego i procesowego jest odmienna. W związku z tym, że typy te stanowią wzorzec używany w różnych kontekstach to w definicjach typów blokowego i procesowego zamiast ścieżek komunikacyjnych występują **definicje bram** <gate definition>. Brama jest wyróżnionym przez nazwę punktem połączenia, który może zostać

odmiennie skonfigurowany w różnych kontekstach. Brama przenosi pewne sygnały, określone za pomocą listy sygnałowej, do lub od definicji typu blokowego/procesowego, przy czym możliwe jest wprowadzenie ograniczenia **[ATLEAST] <identifier>** zawężającego możliwość dołączenia bramy do ścieżek komunikacyjnych prowadzących tylko do wystąpień określonego typu.

3.4 Generyczne definicje typów

Generyczna definicja typu jest niekompletna, w tym sensie, że może zawierać i odwoływać się do nazw obiektów, które nie są w pełni zdefiniowane. Nazwy te są nazywane **kontekstowymi parametrami formalnymi** <formal context parameters> tego typu. Wewnątrz generycznej definicji typu zdefiniowane są tylko te własności kontekstowych parametrów formalnych (tzw. ograniczenia formalne), które są niezbędne do ich użycia.

Generyczną formę definicji typu można użyć w odniesieniu do typów: systemowego, blokowego i procesowego oraz dla procedur, sygnałów i typów. Jako kontekstowe parametry formalne mogą wystąpić: procesy, procedury, procedury oddalone, sygnały, zmienne, stopery, synonimy i typy.

Kontekstowe parametry formalne są umieszczane bezpośrednio po nazwie typu w nawiasach kątowych (< i >).

Użycie generycznej definicji typu polega na zastąpieniu kontekstowych parametrów formalnych przez kontekstowe parametry konkretne. Parametry konkretne muszą spełniać formalne ograniczenia związane z odpowiadającymi im parametrami formalnymi. Przez podanie różnych parametrów konkretnych pojedyncza generyczna definicja typu może być wykorzystana do zdefiniowania rodziny typów o podobnych własnościach.

Formalne ograniczenia kontekstowego parametru formalnego mogą być określone za pomocą **ograniczenia sygnatury** <signature constraint>, które polega na określeniu pewnych wymaganych własności kontekstowego parametru konkretnego lub za pomocą ograniczenia ATLEAST, które oznacza, że kontekstowy parametr aktualny jest tego samego typu co podane ograniczenie lub jest jego podtypem.

Kontekstowe parametry formalne służą do parametryzacji definicji typów co umożliwia wykorzystywanie tych definicji w różnych kontekstach. W definicjach wystąpień i podtypów kontekstowe parametry formalne są zastępowane przez kontekstowe parametry konkretne.

Kontekstowe parametry formalne mogą zostać użyte w następujących definicjach:

- typu systemowego,
- typu blokowego,
- typu procesowego,
- procedury,
- sygnału,
- typu danych.

Kontekstowe parametry formalne typu są wiązane z konkretnymi parametrami kontekstowymi za pomocą wyrażeń określających typ <type expression> lub reguł dziedziczenia <inheritance rule>. Mechanizm wiązania polega na zastępowaniu poszczególnych parametrów formalnych, wprowadzonych w sparametryzowanej definicji typu, przez odpowiadające im parametry konkretne.

Kontekstowe parametry formalne, które pozostały nie związane (nie zostały podane odpowiadające im parametry konkretne) pozostają kontekstowymi parametrami formalnymi nowego typu (są dziedziczone).

3.5 Specjalizacja

Specjalizacja umożliwia definiowanie typów przez wykorzystanie własności innych typów. Specjalizacja może mieć charakter prosty tzn. polegać na wzbogace-

niu własności obiektu, lub złożony tzn. pozwalać na redefinicję tzw. własności wirtualnych. Specjalizacja umożliwia rozszerzenie własności obiektu o definicje nowych elementów (np. typów, sygnałów, zmiennych) oraz wprowadzenie nowych przejść w typie procesowym lub procedurze. Własnościami, które mogą być poddane redefinicji są definicje typów dla konstrukcji strukturalnych charakteryzujących się zachowaniem wewnętrznym tzn. typ blokowy, typ procesowy i procedura oraz przejścia w typie procesowym i procedurze.

Specjalizacja wiąże się z hierarchią definicji typów w specyfikacji, w tym sensie, że typy specjalizowane stanowią podtypy swoich typów nadrzędnych. Podtypy dziedziczą własności typu nadrzędnego i mogą go wzbogacać o nowe własności i elementy oraz mogą redefiniować wirtualne typy lokalne i przejścia. Hierarchia i co za tym idzie dziedziczenie mogą być wielopoziomowe.

Specjalizacja w definicji podtypu typu nadrzędnego jest wyrażana za pomocą klauzuli **INHERITS** <type expression>, w której wyrażenie określające typ <type expression> oznacza ogólny typ nadrzędny. Zgodnie z zasadą wielopoziomowości dziedziczenia specjalizacja podtypu stanowi podtyp ogólnego typu nadrzędnego.

Z typami wirtualnymi mogą być związane ograniczenia tzn. własności, które muszą być zachowane we wszystkich redefinicjach tego typu wirtualnego. Służą one zagwarantowaniu określonych własności w dowolnej redefinicji.

Wynikowy diagram procesu/procedury w postaci GR dla specjalizowanej definicji procesu/procedury stanowi sumę diagramów procesów/procedur typu nadrzędnego i specjalizowanego podtypu.

W specjalizowanej definicji sygnału można wprowadzić nowe typy, uzupełniające listę typów podaną w definicji sygnału w typie nadrzędnym.

W specjalizowanej definicji podtypu można wprowadzać do definicji typów danych nowe operatory, literały i aksjomaty.

3.6 Polimorfizm i typy wirtualne

W przypadkach gdy definicja podtypu stanowi niewielką modyfikację ("mutację") typu nadrzędnego wówczas właściwym mechanizmem jest użycie typów wirtualnych. Mechanizm definicji typów wirtualnych umożliwia lokalną redefinicję typów przy specjalizacji typu nadrzędnego. Mechanizm ten umożliwia redefinicję przejść w typie procesowym i procedurach. W definicji typu nadrzędnego wskazuje się, które jego elementy mogą zostać zredefiniowane w specjalizowanej definicji podtypu. Elementy te (typy i przejścia) są nazywane wirtualnymi.

Typ lub przejście wirtualne są w definicji typu nadrzędnego wyróżniane za pomocą słowa kluczowego **VIRTUAL**. W przypadku gdy w definicji podtypu następuje redefinicja typu lub przejścia wirtualnego i podtyp ten dopuszcza dalszą redefinicję jest to zaznaczane przez użycie słowa kluczowego **REDEFINED** (w tym przypadku rozważany podtyp w dalszym ciągu pozostaje typem wirtualnym), w przeciwnym razie gdy następuje zakończenie - finalizacja definicji, jest to wskazywane za pomocą słowa kluczowego **FINALIZED**.

Typ wirtualny, który nie został poddany redefinicji w definicji podtypu, zachowuje swoje własności (definicję) z typu nadrzędnego.

Typ wirtualny może być ograniczony przez podanie w części **ATLEAST** <identifier> identyfikatora typu ograniczającego. Pominięcie ograniczenia w definicji typu wirtualnego oznacza, że ograniczeniem jest ten sam typ wirtualny.

Typ wirtualny z podanym jawnie ograniczeniem, lecz bez specyfikacji dziedzicze-

nia, domyślnie dziedziczy własności typu ograniczającego.

Typ zredefiniowany lub sfinalizowany bez podanego jawnie ograniczenia, dziedziczy własności w sposób niejawny.

3.6.1 Przejścia wirtualne

Wirtualizacja przejść dotyczy następujących elementów definicji procesów i procedur:

- przejścia startowego procesu/procedury,
- wejścia,
- sygnału ciągłego,
- przejścia spontanicznego,
- operacji zachowania,
- wejścia i operacji zachowania związanego z mechanizmem wywoływania procedur oddalonych;

Znaczenie

Redefinicja przejść wirtualnych odbywa się na zasadach analogicznych do redefinicji typów.

Wirtualne przejście startowe w redefinicji procesu/procedury może być zastąpione innym przejściem.

Wirtualne wejście może być w redefinicji zastąpione innym wejściem lub operacją zachowania.

Wirtualna operacja zachowania może być w redefinicji zastąpiona przez wejście.

Podobnie jak w przypadku definicji typów wirtualnych użycie słów kluczowych **REDEFINED** i **FINALIZED** oznacza odpowiednio redefinicję umożliwiającą dalszą modyfikację i zakończenie - sfinalizowanie redefinicji przejścia lub operacji zachowania.

3.7 Moduły

Jedną z podstawowych cech podejścia obiektowego do projektowania systemów jest mechanizm wielokrotnego wykorzystywania już istniejących elementów i struktur do budowy specyfikacji innych nowych systemów.

Sam mechanizm definicji typów, opisany wcześniej w niniejszym rozdziale, umożliwia wykorzystanie raz zdefiniowanego typu w wielu miejscach danej pojedynczej specyfikacji do definiowania wystąpień oraz innych typów lub podtypów za pomocą parametryzacji, dziedziczenia i specjalizacji. W celu umożliwienia wykorzystania istniejących definicji w wielu innych specyfikacjach wprowadzono do SDL pojęcie modułu <package>.

Obiekty zdefiniowane w modułach mogą być wykorzystywane, w całości lub w części, w wielu specyfikacjach. Skorzystanie z definicji umieszczonej w module wymaga użycia w module jej używającym klauzuli odwołania do tego modułu <package reference clause>. Każda specyfikacja może korzystać z wielu definicji umieszczonych w odrębnych modułach.

Jeżeli specyfikacja styku (<interface>) została pominięta, moduł udostępnia nazwy wszystkich zdefiniowanych w nim obiektów. W przypadku podania specyfikacji styku moduł udostępnia tylko nazwy obiektów, które jawnie umieszczono na liście wyboru definicji <definition selection list> .

W przypadku pominięcia listy wyboru <definition selection list> klauzula odwołania do modułu <package reference clause> "importuje" wszystkie nazwy zdefiniowane w powoływanym module, natomiast w przypadku użycia listy wyboru importowane są tylko te nazwy, które zostały umieszczone na liście.

Jawny import nazwy z modułu może pociągać za sobą domyślny niejawny import dodatkowych nazw pochodnych.

Z każdą definicją systemu i modułu jest domyślnie związana klauzula odwołania do

wyróżnionego modułu **Predefined**:

USE Predefined;

który zawiera predefiniowane typy danych języka SDL.

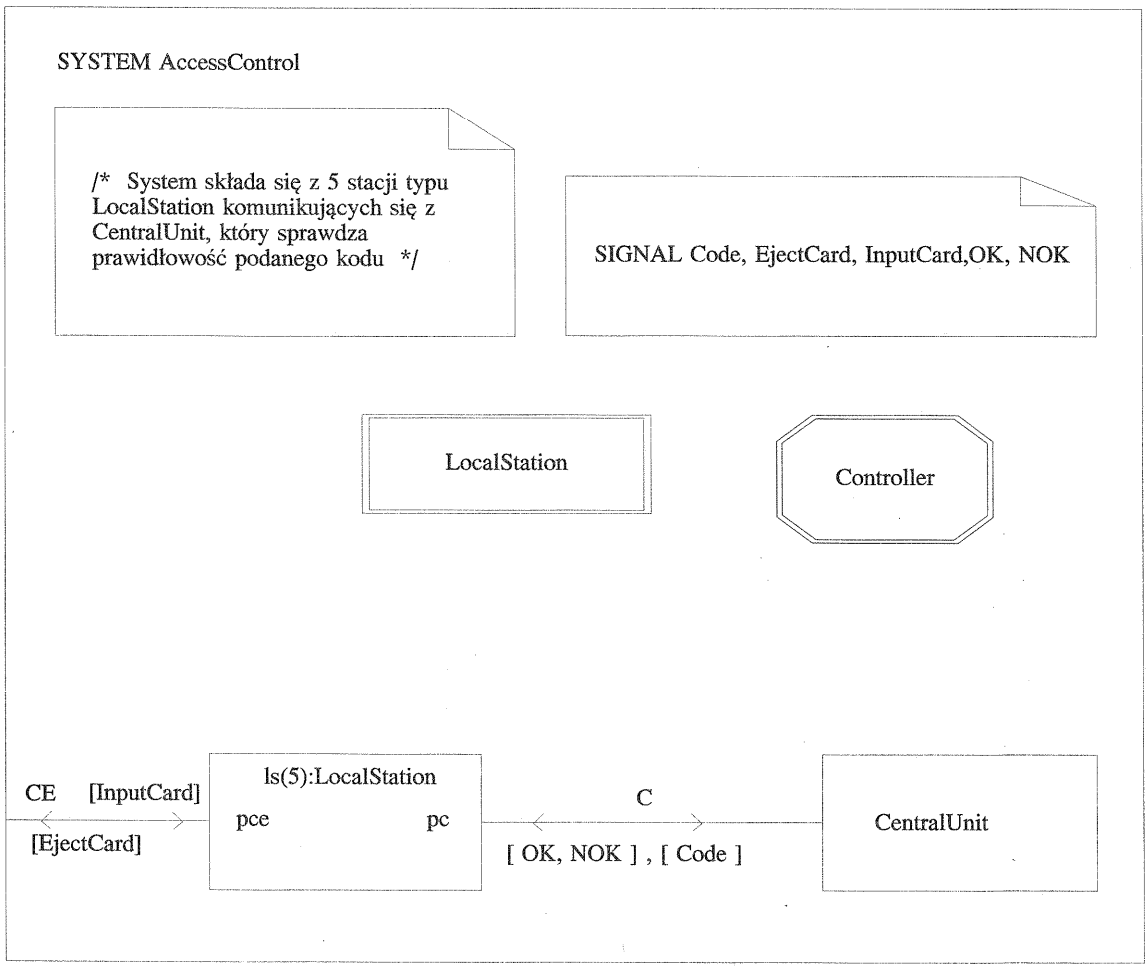
4. Przykład

W celu ilustracji użycia obiektowej wersji języka SDL rozważymy przykład systemu zabezpieczenia dostępu i jego przystosowanie do konkretnego zastosowania bankowego (wykorzystano przykład użyty w pracy [4] oraz w Dodatku do zalecenia CCITT Z.100 poświęconym wskazówkom metodologicznym dla użytkowników).

Przeznaczeniem systemu jest sterowanie dostępem do pewnych usług. System jest z natury rozproszony, ponieważ usługi są świadczone za pomocą lokalnych stacji zainstalowanych w różnych miejscach. Centralny moduł sterujący systemu przechowuje informację o użytkownikach i ich uprawnieniach, kartach umożliwiających dostęp oraz poufnych numerach identyfikacyjnych. W celu uproszczenia przykładu pominiemy reprezentację za pomocą sygnałów takich zdarzeń i czynności jak wybieranie kodów na klawiaturze, otwieranie i zamykanie drzwi.

Architektura systemu

Pierwsza wersja systemu *AccessControl* składa się z pięciu stacji lokalnych *(LocalStation)* oraz jednej stacji centralnej *(CentralUnit)*. Stacje lokalne są wprowadzone do specyfikacji systemu za pomocą definicji typu *LocalStation* a następnie definicji pięciu wystąpień bloków *ls* według definicji typu *LocalStation* (por. rys. 2).



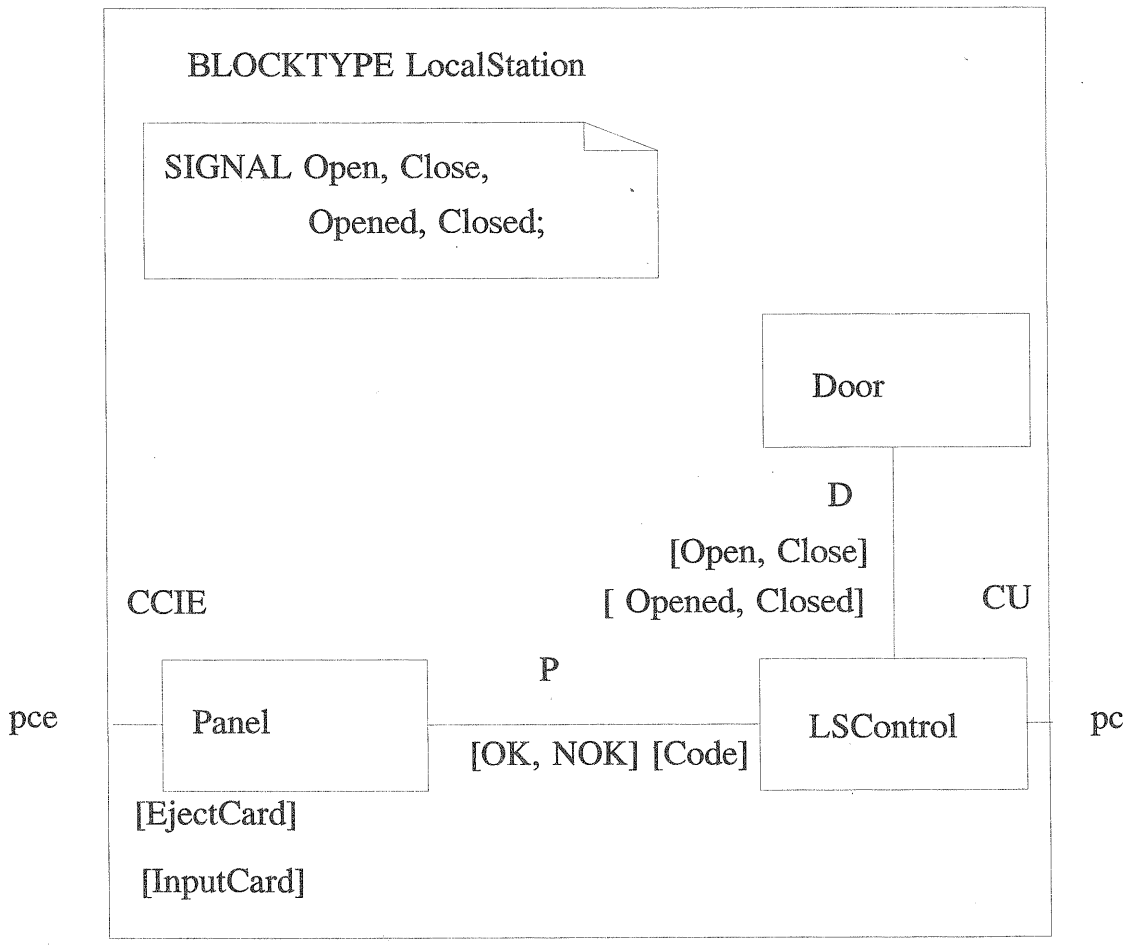
RYS. 2. Struktura systemu AccessControl

Definicja typu procesowego *Controller* została także wprowadzona na poziomie definicji systemu, aby umożliwić jej wykorzystanie w definicjach typów blokowych. W obiektowej wersji SDLa definicje typów mogą być umieszczane na wyższych poziomach, niż definiowane w oparciu o nie obiekty, co pozwala odwoływać się do typów wszędzie tam, gdzie jest to możliwe (w sensie reguł widoczności nazw).
W systemie występuje 5 kanałów łączących poszczególne wystąpienia bloków typu *LocalStation* z węzłem centralnym *CentralUnit*. Każdy z kanałów przenosi wymienione w specyfikacji sygnały. Definicja typu blokowego *LocalStation* (rys. 3) zawiera bramy *pc* i *pce* reprezentujące symbolicznie punkty połączenia z kanałami. Zdefiniowanie wystąpień bloków *ls* typu *LocalStation* powoduje wprowadzenie kanałów związanych z konkretnymi wystąpieniami bloku. Kanały muszą spełniać ograniczenia podane w definicji typu blokowego, tzn. lista sygnałów i kierunek ich przekazywania muszą być zgodne z definicją typu. W tym przypadku kanały dołączone do punktu *pce* muszą przenosić sygnały *EjectCard* i *InputCard*, a kanały dołączone do punktu *pc* powinny przekazywać sygnały *Code, OK i NOK.*

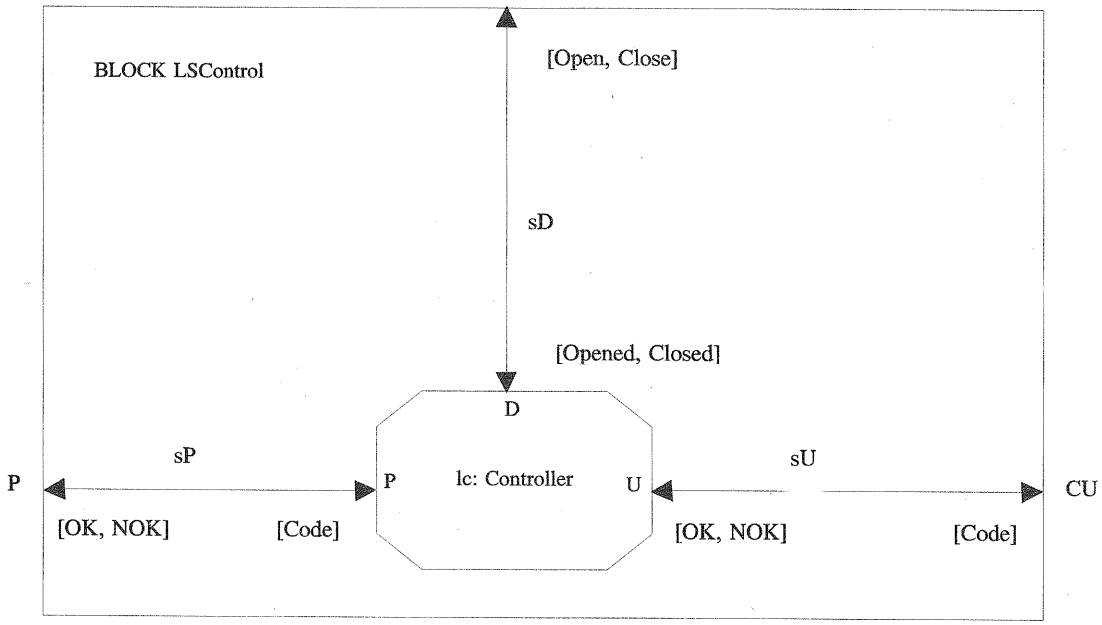
Blok *LsControl* (rys. 4) opisuje sterowanie stacją lokalną. Blok zawiera definicję procesu *lc* odwołującą się do definicji typu procesowego *Controller* (rys. 5). W procesie wyróżniono trzy bramy: *P, D* i *U*.

Dziedziczenie proste

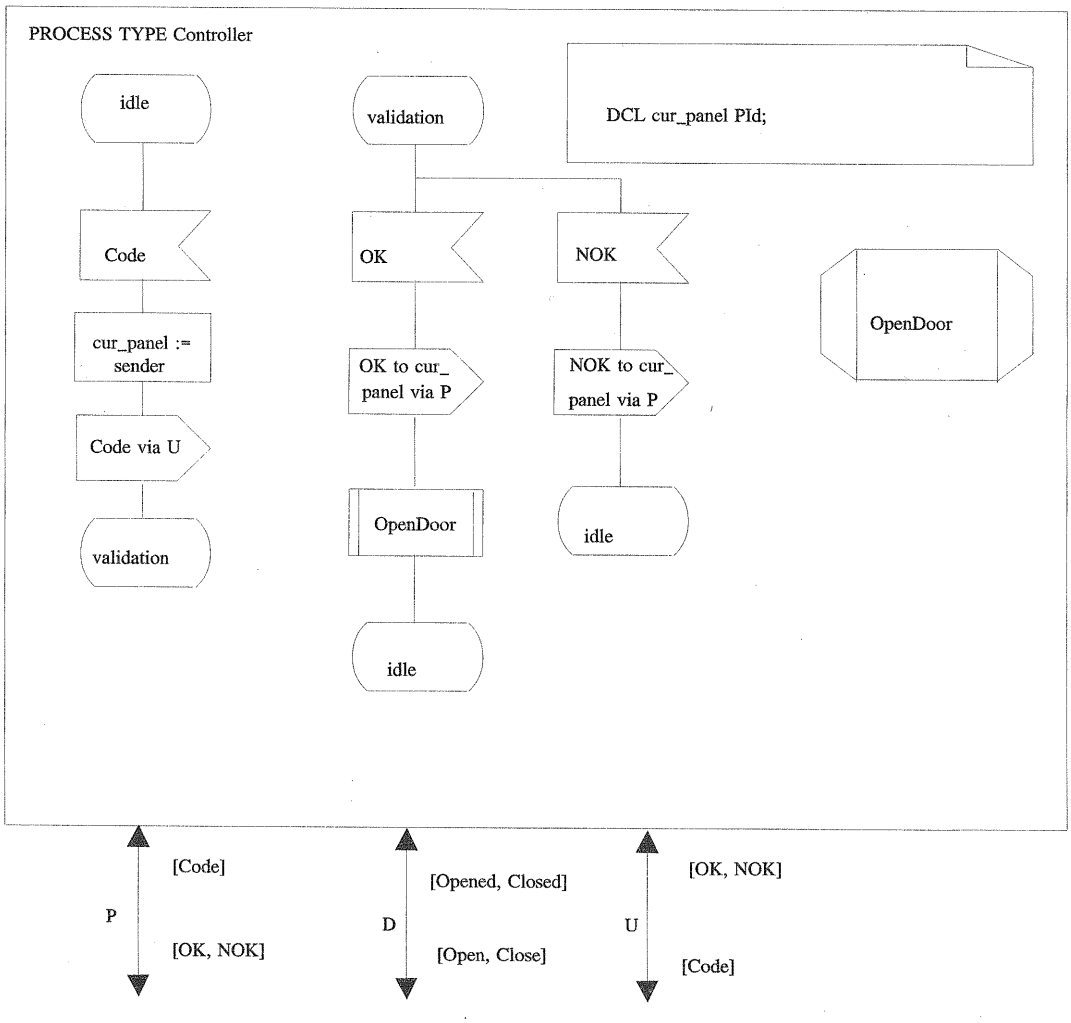
W celu ilustracji mechanizmu dziedziczenia prostego przyjmiemy w przykładzie, że niektóre stacje lokalne typu *LocalStation* są wyróżnione w sensie funkcjonalnym i można je zablokować na żądanie wysłane



RYS. 3. Typ blokowy LocalStation



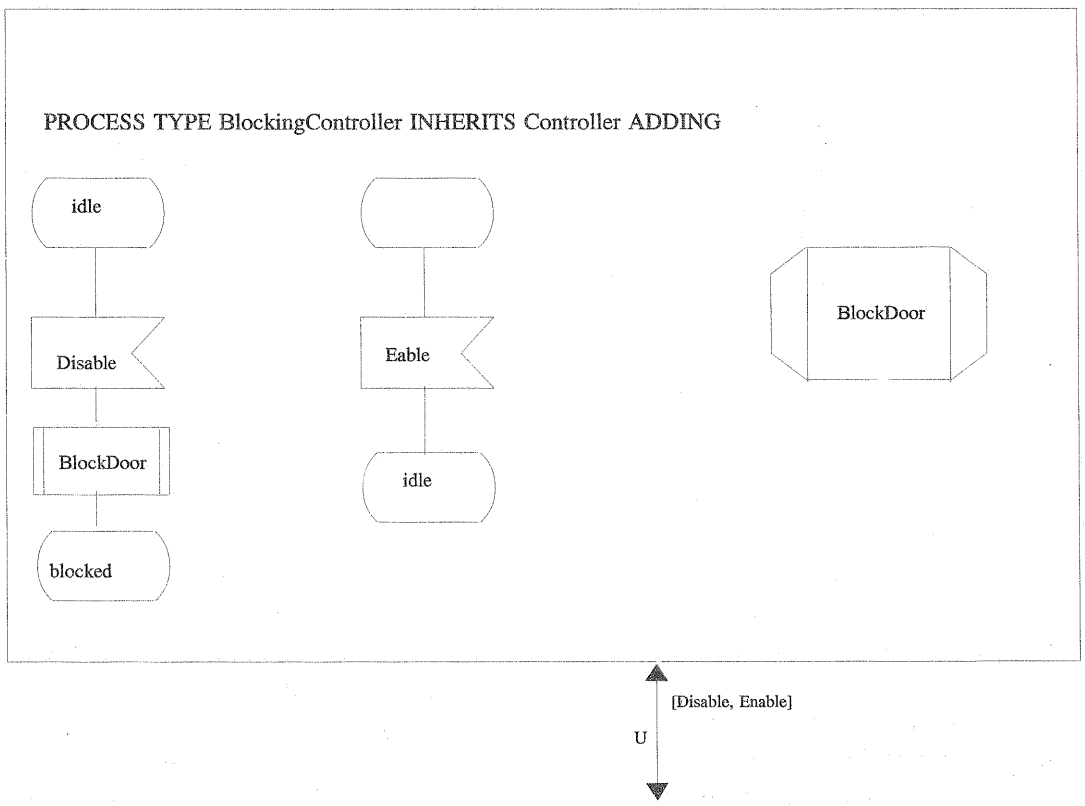
RYS. 4. Struktura bloku LSControl



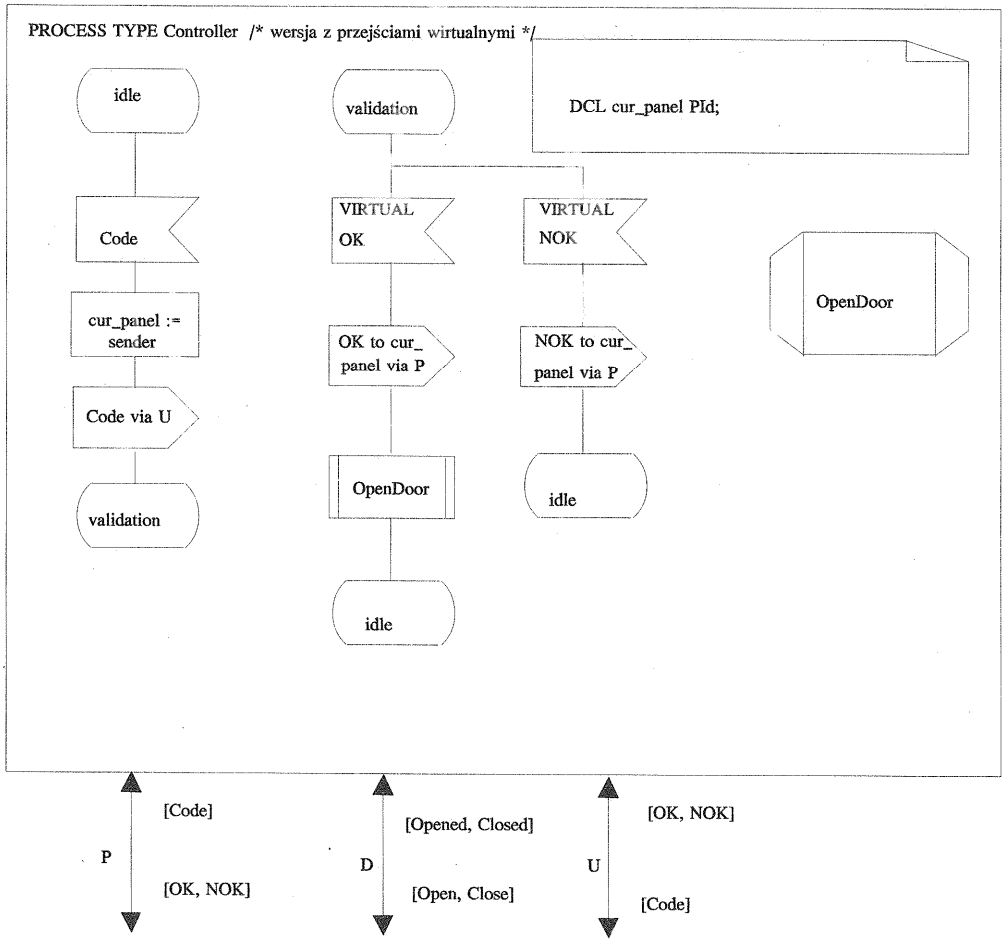
RYS. 5. Definicja typu procesowego Controller

z węzła centralnego *CentralUnit*. Cel ten można osiągnąć przez wprowadzenie dodatkowego sygnału *Disable* wysyłanego przez *CentralUnit*. Sygnał ten powinien być odbierany w obu stanach procesu *Controller* oraz w dodatkowym stanie *Blocked*, w którym odbierany jest tylko sygnał *Enable* aktywujący ponownie normalne zachowanie stacji lokalnej. Nowy diagram procesu, w którym dodano odbiór nowych sygnałów, obsłuży część nowych wymagań związanych z przetwarzaniem. Efekt o który chodzi uzyskuje się przez zdefiniowanie nowego typu procesowego *BlockingController* jako specjalizacji rozszerzającej własności typu procesowego *Controller* o pożądane cechy. Ponieważ bramy typu procesowego *Controller* są dziedziczone, nie są już one specyfikowane. Ograniczenia są również takie same - dziedziczone. Uzupełniono natomiast specyfikację bramy *U* o nowe sygnały *Disable* i *Enable*. Analogicznie jak w przypadku definicji typu blokowego *LocalStation*, w którym jest odwołanie do typu procesowego *Controller*, tak nowy typ blokowy *BlockingStation* odwołuje się do typu procesowego *BlockingController*. Zmiana ta będzie również widoczna na poziomie definicji systemu. Diagram tego typu procesowego składa się ze wszystkich elementów zdefiniowanych w typie procesowym *Controller* uzupełnionych o nowe przejścia uwidocznione w definicji typu procesowego *BlockingController* (por. rys. 6). Zwraca się uwagę, że nowa definicja powstaje nie przez skopiowanie i modyfikację pewnej już istniejącej innej definicji, lecz przez określenie relacji pomiędzy typem nadrzędnym i jego podtypem. Z tego powodu ewentualna modyfikacja typu nadrzędnego (w tym przypadku typu *Controller*) spowoduje, że wszystkie wprowadzone zmiany zostaną przeniesione - odziedziczone przez wszystkie podtypy (w tym przypadku typ *BlockingController*).

Obiektowa wersja języka SDL oprócz mechanizmów typowych dla języków ukierunkowanych obiektowo, takich jak



RYS. 6. Definicja typu procesowego BlockingController, który stanowi specjalizację typu Controller



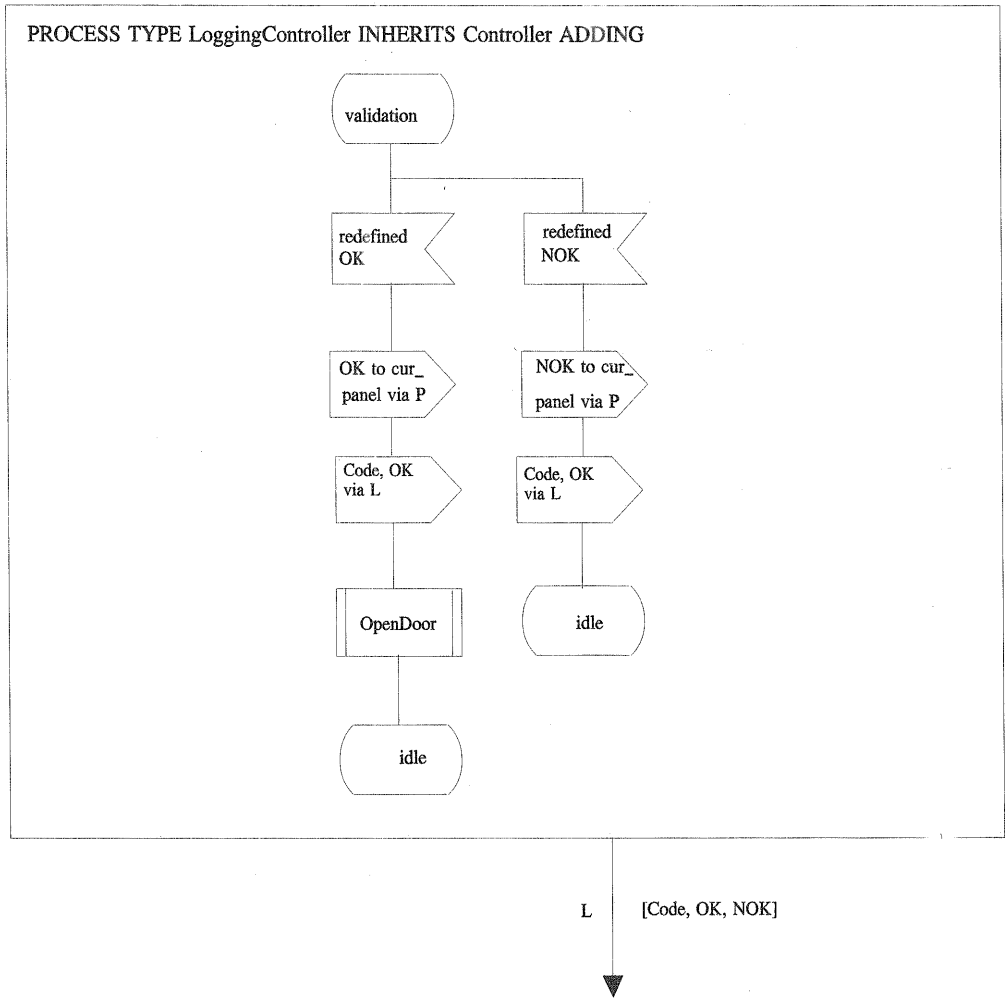
RYS. 7. Definicja typu procesowego Controller zawierająca przejścia wirtualne

dziedziczenie i specjalizacja w odniesieniu do zmiennych i procedur umożliwia także dziedziczenie i specjalizację sekwencji akcji (zilustrowane diagramami procesów w wersji SDL/GR).

Wirtualne własności typów

Niektóre własności podtypów nie mogą być wyrażone wyłącznie w oparciu o mechanizm prostego dziedziczenia i specjalizacji. W rozważanym systemie zabezpieczenia dostępu *AccessControl* pragniemy wyróżnić pewne stacje lokalne, które oprócz swoich standardowych funkcji dokonują rejestracji zdarzeń - prowadzą dziennik systemowy. W tym celu wprowadzimy nowy typ blokowy *LoggingStation* uzupełniający definicję typu *LocalStation* o dodatkowy blok *LogDevice*, którego zadaniem jest rejestracja sygnałów wysyłanych z bloku *LSControl*. Natomiast w bloku *LSControl* zostanie zdefiniowany proces typu *LoggingController*. Proces ten nie obsługuje odbioru nowych sygnałów ani nie wprowadza nowych stanów, natomiast oprócz funkcji właściwych dla typu *Controller* rejestruje sygnały *OK*, *NOK* i *Code*, tzn. wysyła je do bloku *LogDevice*.

Co prawda proces *LoggingController* (rys. 8) operuje tymi samymi sygnałami, co proces *Controller*, lecz wykonywane przez oba procesy akcje, w związku z odebranymi sygnałami, różnią się. Np. w przejściu inicjowanym odebraniem sygnału *OK* z *CentralUnit* oprócz wykonania procedury *OpenDoor* sygnały *Code* i *OK* są wysyłane do bloku *LogDevice*, podobnie w przejściu



RYS. 8. Typ procesowy LoggingController, który stanowi redefinicję i specjalizację typu Controller

aktywowanym przez odbiór sygnału *NOK* dodatkowo wysyłane są sygnały *Code* i *NOK* do bloku *LogDevice*.

W opisanym przypadku celem było uogólnienie i uelastycznienie definicji typu procesowego *Controller* tak aby w definicjach podtypów pochodnych można było modyfikować przejścia. Oczywiście problem ten można było rozwiązać przez wprowadzenie odrębnej nowej definicji typu procesowego *LoggingController*. Ponieważ poza opisanymi różnicami omawiane typy wykazują duże podobieństwo, to znacznie prostszym i naturalnym rozwiązaniem jest wykorzystanie definicji wirtualnych w powiązaniu z mechanizmem dziedziczenia.

Przejścia, dla których dopuszcza się możliwość redefinicji w specyfikacji podtypu nazywają się przejściami wirtualnymi. Przejścia wirtualne specyfikuje się przez opatrzenie elementów, które mogą ulec modyfikacji (wejść, wyjść, operacji zachowania) atrybutem virtual.

W ten sposób nowa uogólniona definicja typu procesowego *Controller* może wyglądać jak na rysunku 7. Wówczas definicja typu procesowego *LoggingController* (rys. 8) będzie dziedziczyła własności typu *Controller* uzupełniając go o zredefiniowane przejścia i bramę *L* do komunikacji z blokiem *LogDevice*. W redefinicji wirtualne elementy przejść, które są poddane ostatecznej modyfikacji są opatrzone

atrybutem **redefined**. W przypadku, gdy redefinicja ma charakter ostateczny, używa się atrybutu **finalized** zamiast **redefined**. Podtyp zawierający zredefiniowane przejścia (elementy z atrybutem **redefined** lub **virtual**) zachowuje własność wirtualności tzn. możliwość dalszej redefinicji w innych podtypach pochodnych. W przypadku, gdy pragnie się przerwać możliwość redefinicji przejść w podtypach, odpowiednie elementy przejść opatruje się atrybutem **finalized**.

W niektórych przypadkach pożądany sposób dziedziczenia i ewentualnej modyfikacji przejść może mieć bardziej systematyczny charakter, tzn. pewne elementy przejścia są dziedziczone bez zmian, natomiast inne mogą być modyfikowane. Typowym przykładem jest sytuacja, w której potencjalnie zmienne elementy przejścia są wyrażone za pomocą wywołania procedur. Wówczas przejście w definicji podtypu jest w całości dziedziczone z typu nadrzędnego z wyjątkiem wywołania procedury. Dla tego rodzaju przypadków wprowadzono pojęcie procedury wirtualnej, tzn. procedury, która może być poddana specjalizacji w definicjach podtypów. Specyfikacja tego rodzaju procedur jest opatrzona atrybutem virtual. W definicji procedury wirtualnej, podobnie jak w przypadku bramy, można użyć ograniczenia, które wyraża fakt, że redefinicja procedury musi stanowić specjalizację podanej procedury ograniczającej, co oznacza, że wszystkie specjalizacje danej procedury będą miały co najmniej własności procedury ograniczającej.

Na rys. 9 przedstawiono odpowiednio uogólniony diagram typu procesowego *Controller* zawierający definicję wirtualnych procedur *WhenOK* i *WhenNOK* oraz typu procesowego *LoggingController* (rys. 10), w którym specyfikacje wymienionych procedur są redefiniowane.

W podobny sposób można definiować i poddawać modyfikacji typy blokowe. Generalna zasada pozwala wszystkie definicje lokalne opatrywać atrybutem wirtualności.

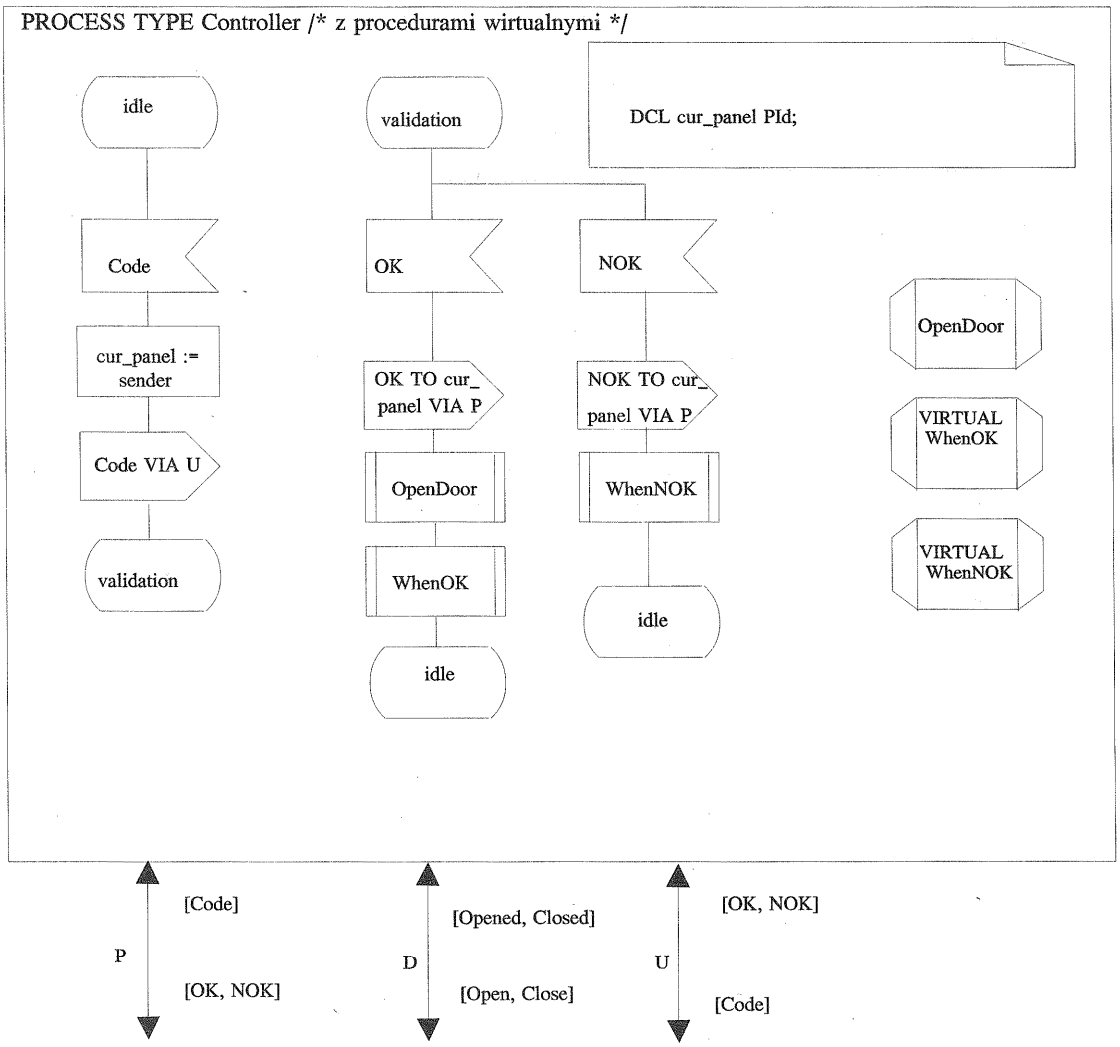
Moduły biblioteczne i definicje generyczne

Typową cechą każdego większego projektu jest istnienie pewnego podstawowego zestawu wzorców definicji obiektów, struktur danych, procedur, stałych tworzących jedną lub więcej bibliotek projektu, do których odwołują się specyfikacje różnych wersji, wariantów i konfiguracji systemu. Reprezentatywnym przykładem tego rodzaju systemów są rodziny central telefonicznych czy węzłów sieci komputerowych. Wzorce zakwalifikowane do bibliotek powinny być na tyle ogólne i sparametryzowane, aby mogły znaleźć uniwersalne zastosowanie w wielu różnych wersjach systemu, realizując postulat wielokrotnego wykorzystania istniejących elementów specyfikacji, charakterystyczny dla podejścia obiektowego.

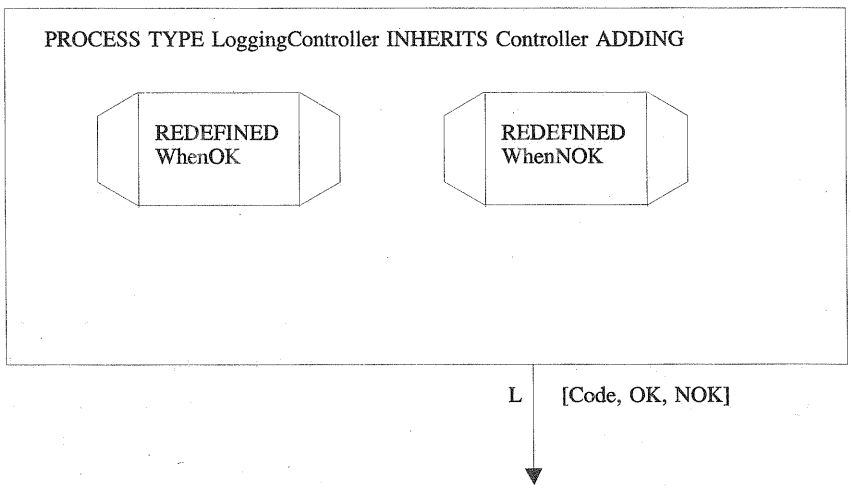
Sam mechanizm definicji typów umożliwia wykorzystanie raz zdefiniowanego typu w wielu miejscach danej pojedynczej specyfikacji do definiowania wystąpień oraz innych typów lub podtypów za pomocą parametryzacji, dziedziczenia i specjalizacji. W celu umożliwienia wykorzystania istniejących definicji w wielu innych specyfikacjach wprowadzono do SDLa pojęcie modułu <package>.

Obiekty zdefiniowane w modułach mogą być wykorzystywane, w całości lub w części, w wielu specyfikacjach. Skorzystanie z definicji umieszczonej w module wymaga użycia w module jej używającym klauzuli odwołania do tego modułu <package reference clause>. Każda specyfikacja może korzystać z wielu definicji umieszczonych w odrębnych modułach.

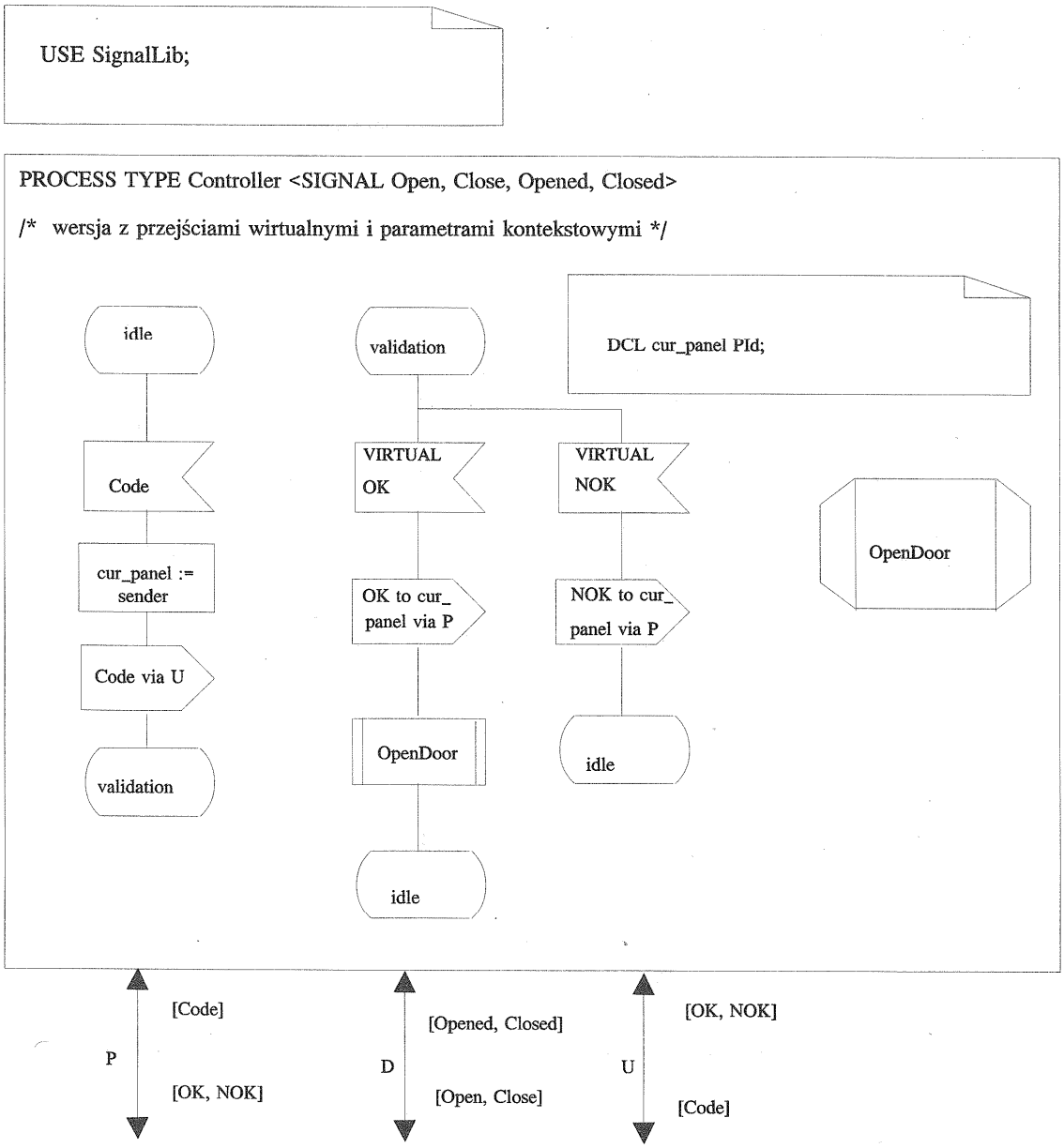
Na rys. 11 przedstawiono przykładową uogólnioną generyczną definicję dla typu procesowego Controller, zawierającą



RYS. 9. Definicja typu procesowego Controller w wersji z procedurami wirtualnymi



RYS. 10. Redefinicja procedur wirtualnych w typie procesowym LoggingController



RYS. 11. *Sparametryzowana generyczna definicja typu procesowego Controller odwołująca się do biblioteki zewnętrznej SignalLib*

kontekstowe parametry formalne i odwołującą się do modułu bibliotecznego SignalLib, w którym znajdują się definicje sygnałów używanych w specyfikacji.

5. Wnioski

Najbardziej interesującym dla użytkowników problemem jest: jak skorzystać z mechanizmów obiektowych języka SDL. Generalnie obiektowe rozszerzenia języka SDL mogą być używane w dwojaki sposób:

- przez wykorzystywanie mechanizmu specjalizacji do tradycyjnych specyfikacji opartych na funkcjonalnej dekompozycji systemów;
- przez konsekwentne stosowanie mechanizmów obiektowych, tzn. specjalizacji, dziedziczenia i polimorfizmu, od samego początku; główne zadanie polega na identyfikacji klas typów procesów, które stanowią podstawowe obiekty specyfikacji systemów w języku SDL.

Niewątpliwie wprowadzenie do języka SDL mechanizmów obiektowych oraz innych rozszerzeń, takich jak np. jednoczesne wysyłanie sygnałów do wielu adresatów *(multicast)*, zdalne wywoływanie procedur, wybór arbitralny, moduły biblioteczne, ułatwią stosowanie języka do dużych projektów charakteryzujących się ewolucyjną zmiennością. Odbywa się to jednak kosztem pewnego skomplikowania składni języka, który mimo tego zachowuje swoją sugestywność i intuicyjność.

Z uwagi na uniwersalność modelu, sugestywną notację graficzną, możliwość wyrażania uwarunkowań czasowych oraz dostępność oprogramowania wspomagającego (narzędzia uwzględniające zmiany wprowadzone w 1992 roku są w trakcie opracowywania), SDL może znaleźć wiele zastosowań pozatelekomunikacyjnych, w szczególności w dziedzinie specyfikacji i projektowania dużych rozproszonych systemów czasu rzeczywistego.

LITERATURA

[1] Średniawa M.: *Zastosowanie języka SDL w projektowaniu systemów czasu rzeczywistego*, mat. konf. II Wyższa Międzynarodowa Górska Szkoła PTI, Szczyrk 1990.

[2] *CCITT: Recommendation Z.100 - CCITT Specification and Description Language (SDL)*, COM X-R 17-E, Geneva, 1992.

[3] Faergemand O., Olsen A.: *Tutorial on new features in SDL'92*, mat. konf. FORTE-92, Perros-Guirec, 1992.

[4] Moller-Pedersen B.: *Object Oriented SDL*, TELE nr 1, 1991, str. 23-32.

[5] Ostrowski P., Średniawa M.: *Język specyfikacji SDL*, Informatyka nr 9/10, 1991.

Roman Siminski
Uniwersytet Śląski

Projekt i realizacja obiektowo zorientowanego podsystemu komunikacji z użytkownikiem

Współczesny rynek oprogramowania stawia wysokie wymagania przed oprogramowaniem, które miałoby na nim znacząco zaistnieć. Jednym z czynników mających istotny wpływ na atrakcyjność oprogramowania jest łatwość jego obsługi. Wśród systemów o zbliżonych możliwościach, przyszły użytkownik z pewnością faworyzować będzie te, które wyposażone są w wygodny, ergonomiczny i przejrzysty podsystem komunikacji z użytkownikiem *(ang. user interface - UI)*. Z racji postępującego w tej dziedzinie procesu standaryzacji (np. SAA/CUA) oraz potencjalnej powielarności elementów podsystemu komunikacji, celowe wydaje się opracowanie i implementacja biblioteki programowej realizującej funkcje komunikacji z użytkownikiem. Tego typu rozwiązania istnieją już dla wielu platform sprzętowych, środowisk systemowych czy implementacji języków programowania. Obecnie zaobserwować można trzy kierunki rozwoju tego typu produktów:

1. Biblioteki dołączane do konkretnych implementacji języków programowania (dla DOS występują np. jako uzupełnienia produktów firm Borland, MicroSoft, Zortech-Symantec).
2. Rozbudowane systemy tworzenia podsystemów komunikacji z użytkownikiem dedykowane dla całej grupy implementacji języków programowania, dostarczane często z pełnym kodem źródłowym (Vermont Views, Vitamin C, Code Screen itp).
3. Pakiety dostarczane przez producentów oprogramowania systemowego (lub innych producentów, lecz bazujące na nich) umożliwiające konstruowanie podsystemów komunikacji z użytkownikiem zgodnie z regułami obowiązującymi w danym środowisku systemowym (SDK dla Windows, OS/2; OSF/Motif, OpenLook).

Dzięki bogactwu propozycji w tej dziedzinie można wybrać rozwiązanie najkorzystniejsze dla określonych potrzeb. Praktyczne doświadczenia dowodzą jednak, że dla dość sporej grupy zastosowań kompleksowe rozwiązania dostarczone przez produkty omówione w pkt. 1, 2 są nieoptymalne.

Istnieją systemy nie wymagające bardzo rozbudowanych UI, stąd zastosowanie narzędzi takich jak Turbo Vision czy Vitamin C powoduje nadmierny wzrost rozmiaru programu, zwiększenie wymagań odnośnie wolnej pamięci itp. Dodatkowo przy tworzeniu UI konieczne jest dostosowanie się do standardów proponowanych przez te pakiety. Mimo reklamowanej łatwości modyfikacji i możliwości dostosowania do indywidualnych potrzeb, proces ten jest trudny i wymaga często dogłębnej znajomości zasad funkcjonowania pakietu. Zbytnie przywiązanie się do takiej biblioteki może też znacznie ograniczyć możliwości przenoszenia kodu źródłowego. Może okazać się również, że postawieni przed koniecznością zmiany środowiska systemowego lub nawet wersji językowej, zostaniemy pozostawieni w nowych warunkach sam na sam z problemem realizacji podsystemu komunikacji

z użytkownikiem (zgodnie z zasłyszanym gdzieś stwierdzeniem - "Turbo Vision i problem z głowy, lecz w głowie nic nie przybyło"). Celowe zatem wydaje się zaprojektowanie i implementacja własnej biblioteki służącej do tworzenia i obsługi UI. Będzie ona uwzględniać nasze własne upodobania, będzie realizować wybrany podzbiór funkcji UI, nie będzie przy tym zbyt rozbudowana, a co za tym idzie będzie umożliwiała sprawną i optymalną implementację.

Nie będzie truizmem stwierdzenie, że użytkownik postrzega aplikację przez pryzmat jej podsystemu komunikacji. Zwykle określone funkcje powiązane są z wybranymi elementami podsystemu. Elementy te nie mają więc charakteru pasywnego. Po uaktywnieniu danego elementu podsystemu komunikacji może on wykonywać określone akcje, dzięki czemu użytkownik podświadomie utożsamia te akcje z danym elementem. Dla użytkownika podsystem komunikacji ma więc charakter obiektowy - dany element UI posiada nie tylko swą reprezentację graficzną na ekranie ale także potrafi wykonać pewne czynności. To intuicyjnie obiektowe postrzeganie podsystemu komunikacji może znaleźć odbicie w jego obiektowo zorientowanym projekcie. Wybór takiej metodyki jest podyktowany również innymi względami. Jednym z nich jest postulowana elastyczność podsystemu, łatwość jego rozbudowy oraz duża modyfikowalność. Ta ostatnia cecha ma szczególnie istotne znaczenie. Projekt UI nie powinien bowiem odwoływać się do danego zastosowania, jednak powinien umożliwiać późniejszą jego lokalizację w ramach konkretnego zastosowania. Większość tych postulatów można pogodzić stosując konsekwentnie zasady projektowania i programowania obiektowego. Kluczem do elastycznej i efektywnej implementacji będą następujące mechanizmy:

- Abstrakcja i enkapsulacja danych osiągnięta dzięki wykorzystaniu klas;
- Dynamiczne wiązanie akcji związanych z komunikatami przesyłanymi do elementów UI *(dynamic binding, late chaining)* zrealizowane dzięki dziedziczeniu polimorficznemu;
- Rozszerzalność cech funkcjonalnych elementów UI dzięki dziedziczeniu monomorficznemu.

Zastosowanie podejścia obiektowego umożliwia wykorzystanie mechanizmu programowania sterowanego zdarzeniami *(event driven programming)*, który stanowi jedną z najpopularniejszych metod sterowania podsystemami komunikacji z użytkownikiem.

Zadania podsystemu komunikacji z użytkownikiem

Jednym z problemów występujących na etapie projektowania podsystemu komunikacji z użytkownikiem jest wybór koncepcji zarządzania elementami tego podsystemu. Dana aplikacja działająca pod kontrolą określonego systemu operacyjnego poddawana jest bowiem różnego rodzaju wymuszeniom, które powodują zwykle zmianę stanu aplikacji. Wymuszenia pochodzące od użytkownika (docierające do systemu za pośrednictwem wybranych urządzeń wejściowych) powodują najczęściej określone zmiany w podsystemie komunikacji, następnie podsystem komunikacji powoduje wykonanie określonych czynności. Podsystem komunikacji spełnia więc następujące funkcje:

1. Odpowiedzialny jest za przepytywanie określonych źródeł wymuszeń. Załóżmy, że zaistnienie pewnego wymuszenia nazwiemy zdarzeniem. Podsystem komunikacji winien każde zdarzenie zidentyfikować i zapamiętać.
2. Zapamiętane zdarzenie powinno być przefiltrowane przez podsystem. Zdarzenia nieistotne, błędne lub docierające ze źródeł przez aplikację nieobsługiwanych powinny zostać w pewien sposób przez podsystem obsłużone. Jednym z możliwych podejść

jest ignorowanie takich zdarzeń lub realizacja pewnych specyficznych działań mających np. zapobiec potencjalnym sytuacjom błędnym lub awaryjnym. Podsystem może również dokonać konwersji zdarzenia jednego typu na zdarzenie innego, znanego aplikacji typu.

3. Po przeprowadzeniu działań z pkt. 2 podsystem powinien przystąpić do przekazania określonego zdarzenia do jądra aplikacji, odpowiedzialnego za właściwą obsługę danego zdarzenia. Zwykle bywa to poprzedzone ewentualną zmianą stanu elementów podsystemu (znajdującą swe odbicie na ekranie).
4. Podsystem powinien uczestniczyć w obsłudze zdarzenia bądź to na zasadzie pasywnego monitoringu lub udostępniać użytkownikowi możliwość ingerencji w trwający aktualnie proces.

Omówione powyżej funkcje podsystemu komunikacji wymagają:

1. Precyzyjnego zdefiniowania klas zdarzeń, które mogą zaistnieć. W ramach danej klasy sprecyzowane winny być możliwe zdarzenia. Rozpoznane zdarzenie powinno być zapamiętane w wybranej strukturze danych, znanej wszystkim komponentom podsystemu komunikacji.
2. Scentralizowania podsystemu komunikacji, tzn. wyróżnienia takiego elementu, który będzie realizował przepytywanie możliwych źródeł wymuszeń, będzie je identyfikował, klasyfikował zaliczając do określonej klasy zdarzeń i zapamiętywał w wybranej strukturze danych.
3. Wyróżniony element podsystemu zwany dalej zarządcą podsystemu komunikacji (ZPK) musi również realizować proces lokalizacji tego elementu podsystemu, do którego dane zdarzenie jest adresowane, przekazywania zdarzenia do tego elementu oraz rozpoznawania faktu obsłużenia (lub nie obsłużenia) zdarzenia.
4. Niezależnie od postulowanego scentralizowania podsystemu, poszczególne jego elementy muszą posiadać możliwość komunikowania się między sobą bez pośrednictwa ZPK. Jest to konieczne np. w przypadku, gdy otrzymanie zdarzenia przez pewien element powinno pociągnąć za sobą zmianę stanu innych elementów podsystemu w sposób automatyczny, bez udziału ZPK. Wymaga to sprecyzowania mechanizmu komunikacji (wymiany informacji) pomiędzy komponentami podsystemu.

Kolejka zdarzeń

Zdarzenia mogą docierać do aplikacji w dowolnej chwili, nawet wtedy gdy aplikacja nie może nadchodzącego zdarzenia obsłużyć. Aby żadne zdarzenie nie zostało zagubione, korzystne wydaje się wprowadzenie kolejkowania zdarzeń. ZPK po zidentyfikowaniu zdarzenia będzie wstawiać je do globalnej (dla aplikacji) kolejki zdarzeń. Proces przepytywania źródeł zdarzeń będzie polegał na pobieraniu oczekujących w kolejce zdarzeń. Taka globalna kolejka umożliwia również komunikację między elementami podsystemu. Załóżymy, że pewna klasa elementów podsystemu może generować zdarzenia i wstawiać je na początek kolejki. Takie zdarzenie zostanie następnie pobrane do obsługi w pierwszej kolejności. Mechanizm ten można wykorzystać do organizacji obsługi tzw. menu, przycisków *(buttons)*, pasków narzędziowych *(tool bar, speed bar)*.

Ogólna koncepcja organizacji podsystemu komunikacji z użytkownikiem

Aby pogodzić przedstawione powyżej wymagania i przedstawić jeden z możliwych sposobów organizacji podsystemu, wykorzystać można postulowane wcześniej obiektowe podejście do omawianego problemu. Przyjrzyjmy się więc elementom wchodzącym zwykle w skład podsystemu

komunikacji z użytkownikiem, dokonując klasyfikacji wspólnych cech tych elementów. Najczęściej elementy te są postrzegane przez użytkownika jako pewne wycinki ekranu. Zwykle najlepszym przybliżeniem takiego wycinka bywa prostokąt (choć w podsystemach graficznych nie jest to zasadą). Taki prostokąt opisują jednoznacznie współrzędne elementu, które można przedstawić w postaci punktów opisujących np. przeciwległe narożniki.

Załóżmy, że dla roboczego opisu przedstawionych koncepcji używać będziemy konwencji zapisu klas z języka C++. Dla prostoty i przejrzystości stosowane będą znaczne uproszczenia, których w docelowej implementacji być nie powinno. Dotyczą one głównie ograniczenia funkcjonalności definiowanych klas, co ma na celu wyeksponowanie cech najistotniejszych na danym etapie rozważań, jak również pewnych nieścisłości syntaktycznych.

Punkt ekranu może być reprezentowany przez klasę o definicji przedstawionej na wydruku 1.
Wybrany prostokąt ekranu opisywać może klasa przedstawiona na wydruku 2.

Załóżmy, że dowolny element podsystemu będzie posiadał wszystkie cechy prostokąta. Jednak to zbyt mało. Każdy element powinien być również wyposażony w pewien zestaw standardowych operacji, które może wykonywać. Zastanówmy się, jakie operacje powinien wykonywać dany element podsystemu. Na pewno powinien posiadać umiejętność wyświetlania na ekranie swej graficznej reprezentacji. Niech ta operacja nazywa się show. Komplementarną operacją będzie wymazanie własnego obrazu z ekranu - tę operację nazwiemy hide. Ponieważ w danym momencie na ekranie współistnieć może wiele elementów, musi istnieć mechanizm uaktywniania jednego wybranego elementu. Towarzyszyć temu będzie określone wyróżnienie graficznej reprezentacji danego elementu. Tę funkcję realizować będzie operacja select, oraz operacja przeciwna unselect. Każdy element podsystemu komunikacji z użytkownikiem powinien potrafić odpowiadać na przesyłane do niego komunikaty, którymi będą (upraszczając sprawę) najczęściej zdarzenia. Zdarzenie przesłane do elementu może zostać obsłużone lub nie a fakt ten powinien zostać zasygnalizowany. Niech operacja obsługi nadesłanego zdarzenia nazywa się process_event. Ten zestaw operacji stanowi teoretycznie zadawalające minimum. Wynikiem powyższej analizy może być abstrakcyjna klasa stanowiąca wzorzec, szablon dla dowolnego elementu

```
class POINT
{
  public:

    // Konstruktor
    POINT( int x, int y )
    {
      POINT::x = x;
      POINT::y = y;
    }

    // Pobranie współrzędnych punktu
    int   get_x( void ) { return x; }
    int   get_y( void ) { return y; }

    // Ustawienie współrzędnych istniejącego punktu
    void set_point( int x, int y ) { POINT::x = x; POINT::y = y; }
    void set_x( int x ) { POINT::x = x ; }
    void set_y( int y ) { POINT::y = y ; }

  private:
    int x,y;
};
```

WYDRUK 1: Klasa reprezentująca punkt ekranu

```
class RECTANGLE
{
  public:

    // Konstruktory
    RECTANGLE( int left, int top, int right, int bottom )
    : lt( left, top ), rb( right, bottom )
    {
    }

    // Pobranie parametrów prostokąta
    int get_width ( void ) { return ( rb.get_x()  - lt.get_x() + 1 ); }
    int get_height( void ) { return ( rb.get_y()  - lt.get_y() + 1 ); }

    int get_left  ( void ) { return lt.get_x(); }
    int get_right ( void ) { return rb.get_x(); }
    int get_top   ( void ) { return lt.get_y(); }
    int get_bottom( void ) { return rb.get_y(); }

    // Ustawianie współrzędnych istniejącego prostokąta
    void set_rect( int left, int top, int right, int bottom )
    {
      lt.set_point( left, top );
      rb.set_point( right, bottom );
    }

  protected:
    // Narożniki - lewy górny, prawy dolny
    POINT lt, rb;
};
```

WYDRUK 2: Klasa opisująca prostokąt ekranu

podsystemu. Z tej klasy wywieść można na zasadzie dziedziczenia dowolny element, dodając nowe metody oraz przesłaniając istniejące już metody wirtualne. Przykładowa deklaracja tej abstrakcyjnej klasy może wyglądać następująco (wydruk 3):

```
class ABSTRACT_FORM : public RECTANGLE
// Abstrakcyjna forma ekranowa
{
  public:

  ABSTRACT_FORM( const RECTANGLE & scr_coords ) : RECTANGLE( scr_coords )
  {
  }

  ABSTRACT_FORM( int left, int top, int right, int bottom )

                : RECTANGLE( left, top, right, bottom )

  {
  }

  virtual ~ABSTRACT_FORM( void )
  {
  }

  // Metody wirtualne wymagające przesłonięcia w klasach pochodnych
  virtual int show( void ) = 0;
  virtual int select  ( void ) = 0;
  virtual int unselect( void ) = 0;
  virtual int process_event( EVENT & ) = 0;
  virtual int hide( void ) = 0;
};
```

WYDRUK 3: Deklracja abstrakcyjnej klasy wzorcowej

Powyższa klasa zawiera deklaracje metod wirtualnych, które muszą być przesłonięte wirtualnie w klasach pochodnych.

Załóżmy, że każdy element podsystemu komunikacji to obiekt klasy pochodnej od ABSTRACT_FORM. Zgodnie z zasadami programowania obiektowego, zmiennej wskaźnikowej związanej z typem wskazywanym w postaci klasy bazowej można przypisać wskazanie dowolnego obiektu klasy pochodnej. Załóżmy następujące deklaracje:

```
class SCR_FORM : public
                        ABSTRACT_FORM
// Pewna pochodna klasa zawierająca
// deklaracje własnych metod
// wirtualnych
{
   virtual int show();
      .
      .
      .
   virtual int hide();
};

ABSTRACT_FORM * form_ptr;

// Obiekt klasy SCR_FORM
SCR_FORM  screen_form;
```

Można zatem:

```
form_ptr = & screen_form;
```

oraz:

```
form_ptr->show();
form_ptr->unselect();
.
.
.
form_ptr->select();
.
.
.
form_ptr->hide();
.
.
```

Uaktywnianie metod wirtualnych odbywa się na podstawie typu wskazywanego, a nie typu wskaźnika. Zatem każde z powyższych wywołań spowoduje wykonanie akcji określonych metodami klasy SCR_FORM. Jeżeli metoda show realizuje wyświetlenie okna dialogowego, to ta operacja zostanie wykonana. Jeżeli metoda show powoduje wyświetlenie okna menu, pojawi się ono na ekranie. Uogólniając, wywołanie tej metody spowoduje wyświetlenie takiej graficznej reprezentacji elementu, która została określona w ciele metody show tej klasy, którą reprezentuje aktualnie wskazywany obiekt. To samo dotyczy innych metod wirtualnych. Zarządca podsystemu komunikacji może więc manipulować obiektami dowolnej klasy pochodzącej od klasy ABSTRACT_FORM. Typowa operacja wyświetlenia na ekranie określonego elementu podsystemu będzie mogła być realizowana zgodnie z umieszczonym powyżej schematem. Efekt tej operacji zależy od natury danego elementu, która nie jest zarządcy znana. Aby takie podejście mogło zostać zrealizowane, każdy element podsystemu musi być obiektem klasy pochodzącej od ABSTRACT_FORM, musi on przesłaniać odpowiednie metody wirtualne, które realizować będą odpowiednie dla danego elementu akcje. Zarządca podsystemu komunikacji będzie operował na pewnej strukturze danych, której elementami będą obiekty reprezentujące elementy składowe podsystemu. Wszystkie operacje wykonywane będą za pośrednictwem zmiennych wskaźnikowych, co zapewni uaktywnienie właściwych metod wirtualnych. Przed bliższym przedstawieniem zasady działania zarządcy podsystemu komunikacji, słów kilka należy powiedzieć o reprezentacji zdarzeń.

Reprezentacja zdarzeń

Metoda process_event otrzymuje jako parametr daną typu EVENT - zawierającą opis zaistniałego zdarzenia. Jak może wyglądać deklaracja takiej struktury danych? Załóżmy, że źródłami zdarzeń mogą być klawiatura i myszka. Musimy zakodować te dwa rodzaje zdarzeń, służyć do tego będzie pole kind. Zdarzenie pochodzące z klawiatury jest jednoznacznie identyfikowane przez kod klawisza, zgodny z pewną ustaloną konwencją. Zdarzenie pochodzące od manipulatora jakim jest mysz wymaga nieco więcej danych. Powinno być opisane przez pole typu

POINT zawierające ekranowe współrzędne kursora myszy w chwili zaistnienia zdarzenia. Niech to pole nazywa się where. Istotny będzie również odpowiednio zakodowany stan przycisków myszy. Przykładowa uproszczona deklaracja klasy EVENT może wyglądać następująco:

```
enum EVENT_KINDS
{
   NO_EVENTS     = 0x0000u,
   FROM_MOUSE    = 0x0001u,
   FROM_KBD      = 0x0002u,
};// EVENT_KINDS

enum MOUSE_EVENTS
{
   MOUSE_LEFT_DOWN     = 0x0001u,
   MOUSE_LEFT_UP       = 0x0002u,
   MOUSE_RIGHT_DOWN    = 0x0004u,
   MOUSE_RIGHT_UP      = 0x0008u,
   MOUSE_MOVED         = 0x0010u,
   MOUSE_REPEAT_DOWN   = 0x0020u
};//MOUSE_EVENTS//

enum KEY_CODES
{
   .
   .
   .
};

class EVENT
{
  public:

   EVENT_KINDS   kind;
   union
   {
     KEY_CODES     key;
     // Kod klawisza
     MOUSE_EVENTS  mouse;
     // Kod zdarzenia MOUSE_EVENTS
   };

  POINT    where;
  int  clear()
  {
    return ( kind = NO_EVENTS );
  }
  int  pool_mouse();
  int  pool_kbd();
};
```

Metoda clear wywołana dla obiektu klasy EVENT powoduje, że pole kind otrzymuje wartość NO_EVENTS. Metodę tę uaktywniać będą odpowiednie metody process_event dla zaznaczenia faktu, iż dane zdarzenie zostało obsłużone. Metody pool_kbd oraz pool_mouse realizują proces przepytywania tych dwóch źródeł zdarzeń oraz zapis wykrytego zdarzenia do kolejki.

Przedstawione wyżej deklaracje klas umożliwiają omówienie zasady działania zarządcy podsystemu. Realizuje on cykliczne przepytywania źródeł zdarzeń. Niech realizuje to funkcja

look_for_events(EVENT & e)

pobierająca zdarzenia z kolejki zdarzeń.

Ogólny algorytm działania zarządcy podsystemu może wyglądać następująco:

```
powtarzaj
{
  pobierz_zdarzenie();
  obsłuż_zdarzenie();
}
dopóki( nie trzeba zakończyć
                    działania )
```

co można zapisać zgodnie ze stosowaną konwencją:

```
EVENT event;
event.kind = NO_EVENTS;

exit_flag = FALSE;
do
{
   look_for_event( event );
   process_event( event );
}
while( ! exit_flag )
```

Wobec powyższej organizacji obsługi zdarzeń, zarządca podsystemu posiada także własną metodę process_event, jest więc on szczególnym przypadkiem abstrakcyjnej klasy opisującej element podsystemu komunikacji. Metoda process_event realizuje proces obsługi zdarzeń. Stanowi centralne ogniwo podsystemu. Tutaj właśnie następuje zidentyfikowanie elementu, do którego adresowane jest dane zdarzenie oraz inicjowana jest obsługa zdarzenia. Aby zarządca zdarzeń mógł odnaleźć odpowiedni element, muszą one być połączone w pewien łańcuch. Skład tego łańcucha może się zmieniać, istotne jest to, by w danym momencie zarządca podsystemu mógł w nim odnaleźć właściwy element. Gdy taki istnieje, uaktywniana jest metoda process_event tego elementu, odpowiedzialna za właściwą obsługę zdarzenia. Metoda process_event zarządcy podsystemu dokonywać może wstępnego filtrowa-

nia zdarzeń, np. zdarzenia związane z uaktywnianiem kolejnych elementów podsystemu będą obsługiwane przez zarządcę, który uaktywniać będzie metody select i unselect odpowiednich obiektów. Dzięki temu, że wszystkie elementy podsystemu posiadają tego samego przodka w postaci omówionej klasy abstrakcyjnej, zarządca zdarzeń potrafi nimi manipulować, nie wiedząc nic o ich właściwym wcieleniu. Elementami łańcucha mogą być okna dialogowe, edycyjne, komunikacyjne, elementy menu, przyciski itp. Jeżeli wszystkie są wywiedzione z jednej klasy bazowej oraz konsekwentnie przesłonięte zostały wszystkie metody wirtualne abstrakcyjnej klasy bazowej, działanie zarządcy zdarzeń jest procesem nietrudnym do realizacji. Załóżmy, że łańcuch elementów to dwukierunkowa lista obiektów pewnej klasy wywiedzionej z abstrakcyjnej klasy bazowej (patrz wydruk 4):

Pola next, prev stanowią łączniki w liście dwukierunkowej obiektów klasy SCR_FORM. Pole on_screen określa czy dany element jest wyświetlony na ekranie, pole selected mówi czy dany element jest aktualnie wyróżniony (uaktywniony).

Zarządca podsystemu komunikacji powinien być właścicielem listy obiektów klasy SCR_FORM. Powinien on umożliwiać dodanie (usunięcie) elementu do (z) listy. Zwykle jeden element podsystemu jest wyróżniony. Zarządca powinien wiedzieć, który to element. Powinien być wyposażony w możliwości odnajdywania tego elementu, do którego adresowane jest dane zdarzenie. W rzeczywistych podsystemach proces ten bywa bardzo skomplikowany, związane to jest min. z problemem obsługi tzw. klawiszy szybkiego reagowania *(hot keys)*, pasków narzędziowych itp. Przykładowa definicja klasy reprezentującej zarządcę podsystemu operu-

```
class SCR_FORM : public ABSTRACT_FORM
{
  public:

    SCR_FORM( int left, int top, int right, int bottom )
    : ABSTRACT_FORM( left, top, right, bottom ),

    {
      on_screen = selected = FALSE;
      next = prev = NULL;
    }

    virtual ~SCR_FORM( void )
    {
      on_screen = selected = FALSE;
      next = prev = NULL;
    }


    virtual int show( void ) { return ( on_screen = TRUE  ); }
    virtual int hide( void ) { return ( on_screen = FALSE ); }

    virtual int select  ( void ) {  return ( selected = TRUE  ); }
    virtual int unselect( void ) {  return ( selected = FALSE ); }

    virtual int process_event( EVENT & ) { return FALSE; }

  protected:

    int selected;
    int on_screen;

    SCR_FORM * next;
    SCR_FORM * prev;

};
```

WYDRUK 4: Deklaracja klasy obiektów będących elementami listy zdarzeń

```
class CHAIN : public SCR_FORM
{
  public:

    CHAIN( int left, int top, int right, int bottom );

    virtual ~CHAIN();

    virtual int show( void );
    virtual int exec( void );
    virtual int run( void );
    virtual int hide( void );
    virtual int select( void );
    virtual int unselect( void );
    virtual int process_event( EVENT & );
    virtual int look_for_event( EVENT & event );

    void operator += ( SCR_FORM & f );
    void operator -= ( SCR_FORM & f );

    add( SCR_FORM * );
    sub( SCR_FORM * );

    sel( SCR_FORM * );
    sel_next( void );
    sel_prev( void );

  protected:
    int       exit_flag;

    // Pierwszy element łańcucha
    SCR_FORM * first;

    // Ostatni element łańcucha
    SCR_FORM * last;

    // Aktualnie wyróżniony element
    SCR_FORM * curr;

    SCR_FORM * find_who_has_mouse( POINT & p );
};
```

WYDRUK 5: Definicja klasy reprezentującej zarządcę podsystemu

jącego na łańcuchu obiektów przedstawiona jest na wydruku 5.

Do listy można dodawać (i kasować) elementy wykorzystując przeciążone operatory += , -=, oraz metody add i sub. Metody sel, sel_next, sel_prev służą zarządcy do uaktywniania aktualnego elementu podsystemu, czemu towarzyszy wywołanie odpowiednich metod select i unselect dla obiektów stanowiących listę. Metoda find_who_has_mouse umożliwia odnalezienie w łańcuchu elementu wskazanego przez kursor myszy. Metoda exec realizuje opisany wyżej iteracyjny proces obsługi zdarzeń. Pole exit_flag decyduje o zakończeniu działania zarządcy podsystemu. Przykładowa sekwencja programowa obrazująca tworzenie łańcucha elementów podsystemu komunikacji może wyglądać następująco:

```
class DIALOG_WIN : public SCR_FORM
{
  ...
};

class BUTTON : public SCR_FORM
{
  ...
};

class MENU_WIN : public DIALOG_WIN
{
  ...
};
```

```
class EDIT_WIN : public DIALOG_WIN
{
  ...
};
// Deklaracja elementów składowych
// podsystemu
CHAIN   manager;
DIALOG_WIN d;
BUTTON     b;
EDIT_WIN   e;
MENU_WIN   m;
// Budowa struktury podsystemu
manager += d;
manager += b;
manager += e;
manager += m;
// Uruchomienie zarządcy podsystemu
// - uaktywnienie podsystemu
manager.exec();
```

Okazuje się, że klasa CHAIN służyć może nie tylko do globalnego koordynowania działania podsystemu, lecz również może mieć zastosowanie lokalne. Obiekt klasy CHAIN może być bowiem elementem łańcucha! Wykorzystując tę cechę można konstruować rozbudowane podsystemy komunikacji, zawierające wiele bardzo wyrafinowanych elementów. Wyobraźmy sobie, że jednym z elementów podsystemu jest pasek narzędziowy złożony z przycisków. Jest więc on łańcuchem elementów. Deklaracja takiego paska dla edytora może wyglądać następująco:

```
CHAIN  tool_bar;
BUTTON    center,    left_just,
right_just, bold, underline;
tool_bar += center;
tool_bar += left_just;
tool_bar += right_just;
tool_bar += bold;
tool_bar += underline;
manager += tool_bar;
```

Obiekt tool_bar jest reprezentowany poprzez jego elementy składowe. Przekazanie zdarzenia do metody process_event obiektu tool_bar powoduje wykonanie czynności właściwych dla zarządcy podsystemu reprezentowanego przez łańcuch przycisków. Wywołanie metody exec spowoduje ograniczenie działania całego podsystemu, aktualnym odbiorcą zdarzeń będzie właśnie tool_bar i ten stan będzie występował tak długo aż nie zaistnieją warunki przerywające działanie metody exec.

Podsumowując, zarządzanie elementami łańcucha jest stosunkowo mało skomplikowane dzięki zastosowaniu techniki projektowania obiektowego. Wykorzystujemy maksymalnie istniejące już komponenty, mamy możliwość rozszerzania ich własności funkcjonalnych (dziedziczenie, dodawanie metod), jak również ich zmianę bez ingerencji w kod źródłowy (przesłanianie metod wirtualnych). Autor zrealizował bibliotekę programową realizującą funkcję podsystemu komunikacji opartą na zasygnalizowanej koncepcji. Współpracuje ona z kompilatorami języka C++ dla systemu DOS. Niestety dokładniejszy opis problemów związanych z tą realizacją oraz możliwych ich rozwiązań przekracza ramy tej publikacji. Autor zobowiązuje się udzielić wszelkich dodatkowych informacji osobom, którym powyższy opis wydał się zbyt ubogi, mało konkretny lub mało komunikatywny.

Borys Stokalski, Andrzej Maciej Wierzba
InfoVIDE Sp. z o.o.

LBMS Systems Engineer Zintegrowane środowisko do tworzenia systemów informatycznych

Wstęp

Bez wdawania się w akademickie definicje możemy przyjąć ogólnie uznany podział narzędzi typu CASE na pakiety wysokiego poziomu (Upper CASE) i niskiego poziomu (Lower CASE). Pakiety wysokiego poziomu są narzędziami wspomagającymi koncepcyjne etapy pracy nad systemem informatycznym - przede wszystkim etapy Strategii, Analizy i Projektowania. Pakiety niskiego poziomu to przede wszystkim generatory kodu, a więc narzędzia pozwalające zamienić logiczną specyfikację systemu na działający produkt. Pakiety oferujące możliwości obu powyższych poziomów określa się mianem I-CASE (Integrated CASE), czyli zintegrowanych pakietów CASE. Oczywiście terminologia jest rzeczą wtórną, istotna jest natomiast oferowana przez zintegrowane pakiety CASE możliwość przejścia od analizy wymagań użytkownika do działającego programu w środowisku gwarantującym zachowanie spójności projektu.

Znaczenie zintegrowanych pakietów CASE w procesie produkcji oprogramowania

Mówiąc o pakietach CASE, w pierwszym rzędzie należy sobie zadać pytanie o sens wykorzystania tego typu kosztownych narzędzi. Stosunek wielu producentów oprogramowania do technologii CASE jest zwykle mieszaniną fascynacji, nadziei i zdrowego sceptycyzmu. Fascynujemy się nowością rozwiązań, mamy nadzieję, że oto jest wreszcie panaceum na dzisiejsze bolączki informatyki, a z drugiej strony sceptycznie patrzymy na koszty wdrożenia nowych rozwiązań. Nie zawsze łatwo ekonomicznie uzasadnić koszty związane z wprowadzaniem technologii CASE w kraju, gdzie większość systemów przetwarzania danych tworzy się prostymi narzędziami typu dBASE, użytkownik nie wierzy, że system powinien zostać zaprojektowany, zespoły informatyczne są niewielkie, a płace programistów umiarkowane. Przyglądając się sytuacji w krajach, gdzie informatyka stoi na najwyższym poziomie można wysnuć wniosek, że stosowanie technologii CASE wymuszane jest najczęściej następującymi problemami:

- Skalą projektów - projekty o wielkości kilkuset osobo-lat to niezmiernie złożone przedsięwzięcia organizacyjne, praktycznie nie do ogarnięcia bez metodycznego podejścia.
- Żądaniem poprawy jakości systemów związanym z potrzebą zapewnienia funkcjonalności systemów zgodnej z oczekiwaniami użytkowników.
- Koniecznością podniesienia jakości i efektywności pracy wymuszaną przez konkurencję.

Wszystkie te powody wiążą się z przejściem od mniej lub bardziej chaotycznego procesu wytwarzania oprogramowania do sposobów pracy opartych o metodykę. Metodyki wprowadzają do projektów dyscyplinę, kontrolę jakości, standardy,

metody zarządzania. Wymagają jednak tworzenia sporej ilości dokumentacji. Zarządzanie taką dokumentacją jest zwykle nie do pomyślenia bez narzędzi typu CASE.

Jeśli nawet dotychczas nie było w Polsce miejsca dla systemów CASE - problemy we wdrożeniach dużych, ważnych dla kraju systemów informatycznych zdają się ogłaszać zmiany - oto nadchodzą duże projekty, duże zespoły, i naprawdę duże kłopoty.

Tym co prawdziwie decyduje o przydatności systemów CASE są funkcje "wysokiego poziomu" - dotyczące etapu analizy i projektowania. Szczegółowa specyfikacja techniczna powstaje po wnikliwej analizie z udziałem użytkowników, na podstawie projektu wykorzystującego efekty analizy. Jest ona dobrym materiałem zarówno dla programisty jak i dla programu generującego kod. Przy dobrze zaprojektowanym planie testów zarówno generator jak i programista mogą być niemal równie wydajnymi "narzędziami" tworzącymi fizyczny system. Tym co decyduje o jakości systemu jest bowiem w pierwszym rzędzie jakość pracy koncepcyjnej, a zwłaszcza wykrycie i właściwe odwzorowanie wymagań użytkownika.

LBMS Systems Engineer

Systems Engineer, produkt firmy LBMS (Learmonth & Burchett Management Systems) jest propozycją zintegrowanego pakietu CASE. Ten zestaw programów działających w środowisku MS Windows zapewnia wspomaganie prac projektowych na wielu poziomach - od zarządzania, poprzez proces analizy i projektowania, po generację aplikacji i produkcję wysokiej jakości raportów i analiz. Przy tworzeniu tego zestawu narzędzi wykorzystano wieloletnie doświadczenie firmy, która jest jednym z czołowych twórców metodyk strukturalnych na świecie. LBMS działa jako firma konsultingowa i szkoleniowa od ponad piętnastu lat, a od dziesięciu obecna jest na rynku CASE. Jej produktem jest między innymi jedna z najbardziej popularnych metodyk strukturalnych - SSADM, standard zalecany przez rząd brytyjski i (od niedawna) węgierski. Podstawowe cele jakie postawiono sobie tworząc pakiet Systems Engineer to:

- Elastyczne dopasowanie metodyki i narzędzi,
- Wspomaganie pełnego cyklu rozwojowego systemu,
- Kierowanie projektem i zarządzanie cyklem projektowym,
- Zachowanie spójności projektu w środowisku wielu użytkowników,
- Integracja w środowisku "biurowym".

Elastyczne dopasowanie metodyki i narzędzi

Mówiąc o systemach CASE należy zdawać sobie sprawę z tego że są one tylko narzędziem pomagającym zrealizować pewną metodę pracy nad systemem informatycznym. CASE nie zapewni automatycznie zwiększonej wydajności i lepszej jakości. Te tak pożądane efekty może zapewnić jedynie wprowadzenie dyscypliny w pracy zespołu, stosowanie standardów dokumentacji, wybór właściwych technik realizacji projektu. Inwestycja w CASE wiąże się nierozerwalnie z inwestycją w ludzi i w organizację - niepowodzenia przy wprowadzaniu tej technologii biorą się zwykle z niedoceniania wagi tego faktu. Udane wdrożenia technologii CASE wiążą się z intensywnymi programami szkoleń dotyczącymi metod analizy, projektowania i zarządzania projektami. Istotnym czynnikiem pomagającym "uruchomić" nowy sposób pracy jest też wykorzystanie doświadczonych konsultantów w pierwszych, pilotowych projektach. Jest to zwłaszcza istotne tam, gdzie technologia CASE ma służyć stworzeniu dużych systemów o strategicznym znaczeniu.

W związku z tym, pożądaną cechą pakietu CASE jest możliwość zastosowania go do wybranej metodyki. Możliwość ta zależy

w pierwszym rzędzie od możliwości realizacji wszelkich istotnych dla danej metodyki technik (takich jak diagramy modelujące statyczną i dynamiczną stronę systemu, macierze incydencji, prototypy dialogu z użytkownikiem). Z drugiej strony użytkownik powinien mieć dużą dowolność wyboru kolejności realizacji poszczególnych fragmentów projektu. Jednym z założeń pakietu System Engineer była możliwość zastosowania go do wspomagania różnych metodyk strukturalnych. Najpełniej wspomaga oczywiście metodyki firmy LBMS - System Engineering i SSADM - jest jednak z powodzeniem wykorzystywany przez organizacje stosujące metodyki takie jak Information Engineering, czy Summit-D. Warto podkreślić, że dostosowanie do metodyk nie polega na prostej zmianie konwencji notacyjnych dla diagramów. Chodzi bowiem przede wszystkim o możliwość definiowania różnorodnych logicznych zależności pomiędzy elementami projektu. Pozwala to na kontrolę wewnętrznej spójności projektu na wszystkich jego etapach.

Wspomaganie pełnego cyklu rozwojowego systemu

Chyba każdy użytkownik systemów CASE spodziewa się, że narzędzie przeprowadzi go od początkowych faz projektu aż do działającej aplikacji. Postulat ten dotyczy jednak w pierwszym rzędzie metodyki - to ona musi określać zadania i produkty wszystkich etapów projektu. Nie każda metodyka to robi. Na przykład SSADM dotyczy tylko etapów analizy i projektowania, nic nie mówiąc o planowaniu, strategii, czy implementacji. Systems Engineering jest metodyką pokrywającą w pełni cykl życia systemu informatycznego, i zawiera rozwiniętą część dotyczącą projektu fizycznego, konstrukcji, testowania i wdrożenia systemu. Jest w tym wydatnie wspierana narzędziem, które umożliwia zarówno bezpośrednią generację systemów na różne platformy, generację standardowego (ANSI) kodu dla SQL i Cobolu, wymianę danych z pakietami CASE niskiego poziomu, jak i prostą implementację własnych, specyficznych mechanizmów generacji kodu na podstawie bazy danych projektu. Docelowe architektury to:

- Client-Server - pełna generacja aplikacji - części serwerowej dla różnych systemów baz danych i części "klient" działającej w środowisku MS Windows),
- Host-Terminal dla dużych systemów IBM z monitorem transakcji CICS i terminalami znakowymi.

Kierowanie projektem i zarządzanie cyklem projektowym

Wymienić można następujące aspekty pracy kierownictwa projektu informatycznego, które są bezpośrednio wspomagane przez pakiet Systems Engineer:

- Planowanie i szacowanie projektu,
- Profilowanie metodyki,
- Kontrola dostępu,
- Tworzenie wersji, etapów i kontrola jakości.

Powyższe zadania realizowane są przez dwa składniki pakietu - moduł Project Engineer i moduł Information Manager.

Project Engineer jest narzędziem pozwalającym na zaplanowanie przebiegu prac nad projektem, wyznaczenie zadań, wymaganych umiejętności i zasobów. Project Engineer zawiera biblioteki typowych dla metodyki SSADM lub Systems Engineering struktur zadań oraz biblioteki metryk pozwalających szacować pracochłonność poszczególnych etapów prac. Plan można przenieść do typowego pakietu typu "Project Management" - np. MS Project gdzie odbywa się bieżące śledzenie i modyfikacja przebiegu projektu. Project Engineer pozwala następnie na weryfikację początkowego planu i jego uściślenie w miarę jak w bazie danych

projektu pojawiają się kolejne elementy opisu systemu. Przeszacowania kolejnych etapów można dokonywać opierając się na wielkości modelu danych, ilości transakcji w projekcie, ilości węzłów struktury dialogu itp.
Dane "zwrotne", to znaczy faktyczne rezultaty osiągane w czasie realizacji projektu umożliwiają następnie modyfikację bibliotek programu Project Engineer, a więc de facto na profilowanie metodyki i "przykrajanie" jej do specyfiki projektów i realiów danej organizacji.

Information Manager jest modułem pozwalającym na kontrolę dostępu użytkowników do elementów projektu, oraz na tworzenie i zarządzanie wersjami dokumentów projektowych. Umożliwia to na przykład sterowanie procesem kontroli jakości. Prawa dostępu do dokumentów ukończonych przez projektantów czy analityków mogą być np. przekazane grupie kontroli jakości. Pozwala to na weryfikację "zamrożonego" zbioru dokumentów, a tym samym umożliwia rzetelne wykrycie wszelkich błędów i niedociągnięć.

Zarządzanie wersjami dotyczy w pakiecie Systems Engineer dokumentów projektu, a nawet całych zbiorów takich dokumentów. Zbiory te tworzą poszczególne etapy projektu - np. specyfikację wymagań użytkownika, projekt koncepcyjny, analizę strategiczną. Umożliwia to kontrolę zmian oraz ocenę wpływu zmian jednych elementów projektu na pozostałe. Taka ocena może być dobrą podstawą np. do szacowania kosztów zmian wymagań przez użytkownika.

Spójność dokumentów projektu

Jak wspomnieliśmy wcześniej podstawową korzyścią oferowaną przez zintegrowane pakiety CASE jest zachowanie spójności projektu na wszystkich etapach jego realizacji. Jest to możliwe jedynie wtedy, gdy wszystkie składniki pakietu korzystają z jednego słownika danych projektu. Dokumenty projektu są wówczas tworzone na podstawie tych samych danych (np. danych elementarnych, definicji procesów, przepływów, modeli encji). Ważną i unikalną cechą słownika danych pakietu Systems Engineer jest prawdziwie wielodostępny charakter słownika danych. Eliminuje to konieczne w większości systemów CASE uciążliwe procedury konsolidacji efektów pracy poszczególnych członków zespołu ze słownikiem projektu. W pakiecie Systems Engineer najbardziej aktualna wersja każdego z obiektów projektu jest natychmiast dostępna dla wszystkich członków zespołu projektowego!

Kolejnym istotnym mechanizmem integrującym efekty pracy zespołu jest możliwość określania różnego rodzaju związków pomiędzy dokumentami projektu. Związki te wynikają zwykle z przyjętej metodyki pracy nad systemem. W wielu metodykach strukturalnych (SSADM, Systems Engineering) żąda się np. zapewnienia możliwości śledzenia realizacji wymagań użytkownika w dokumentach projektu *(requirement traceablity)*. Wiązanie pozycji katalogu wymagań z produktami etapów analizy, projektu, a nawet implementacji pozwala właśnie na prześledzenie ścieżki realizacji postulatów użytkownika w każdym momencie pracy nad projektem. Związki są jednocześnie świetnym mechanizmem pozwalającym na kontrolę kompletności i wzajemnej odpowiedniości różnych "perspektyw" widzenia systemu. Przykładem może być choćby kompletność zestawu diagramów przepływu danych (DFD), zbioru transakcji i modelu danych (diagramów ERD).

Integracja w środowisku "biurowym"

Ostatnim, ale bynajmniej nie poślednim aspektem integracji środowiska Systems Engineer są jego możliwości współpracy z elementami typowego środowiska biurowego, a więc z arkuszami kalkulacyjnymi, procesorami tekstu, czy programami do

tworzenia publikacji. Systems Engineer działa w środowisku MS Windows i wykorzystuje wszelkie mechanizmy wymiany danych dostępne w tym środowisku - od edycji w stylu Cut, Copy, Paste, aż do dynamicznej wymiany danych (DDE). Składnik zestawu Systems Engineer o nazwie Workplace pozwala na tworzenie dowolnych zestawień informacji z bazy danych projektu, zawierających teksty, liczby, diagramy w postaci kwerend dynamicznej wymiany danych. Umieszczenie takiej kwerendy w dokumencie procesora tekstów (np. MS Word for Windows lub Ami Pro) zamienia ten dokument w "żywy" raport, korzystający zawsze z najbardziej aktualnych danych o projekcie. Może to być raport analizujący stan zaawansowania prac projektowych, lub pełna specyfikacja wymagań użytkownika. Procesor tekstu, narzędzie powszechne stosowane przez kadrę kierowniczą czy personel biurowy stają się w ten sposób mechanizmem pozwalającym na śledzenie przebiegu prac osobom nie znającym się wcale na zawiłościach pakietu CASE.

Podsumowanie

Powyższy tekst stanowił próbę opisania tych cech środowiska CASE które wykraczają poza tradycyjny obraz systemu CASE jako "edytora diagramów". Autorzy artykułu są głęboko przekonani że są to jednocześnie te cechy, które decydują o sensowności stosowania tej technologii. Inwestycja w CASE może być dziś postrzegana jako ryzykowna, ale tym co otrzymujemy w zamian nie jest tylko skomplikowany i drogi program. Sądzimy że podstawowym pozytywnym efektem takiej inwestycji jest poprawa jakości i wydajności procesu tworzenia oprogramowania.

Piotr Woźniak

SuperMemo as a new tool increasing the productivity of a programmer. A case study: programming in Object Windows

Abstract

The paper provides general guidelines for increasing the productivity of a programmer by means of SuperMemo - a new speed-learning technique. The author has been working on a software development project involving programming in Object Windows (an application framework designed by Borland for Turbo Pascal for Windows). Simultaneously, a SuperMemo database has been being developed for training programmers working with Object Windows. Conclusions coming from the author's assignment are described and illustrated with examples.

Numerous studies on productivity in software engineering have shown that programming, which still combines elements of art and science, can be numbered among those few domains of engineering, where individual differences in productivity, often measured in the number of statements per day, vary greatly beyond ordinary expectation. The often quoted relationship is like 1 to 20 among programmers employed in the same project [1]. This paper presents a way to increasing the programmer's productivity by means of simple training techniques that can be introduced at a very little cost by software developers or even by individual programmers on their own.

There are two opposite trends affecting the complexity of problems encountered by an engineer using state-of-the-art technology in development projects:

- introduction of new concepts and techniques at the receiving-end of new technologies increases the complexity of tasks facing the developer,
- introduction of new tools and techniques at the level of the developer's workshop makes his attention shift from methodological aspects to conceptual aspects of the considered problem.

The subject of this paper is a new training technique, SuperMemo, which can be employed in offsetting the decrease of a programmer's productivity observed in association with increasing complexity of the development environment.

The attention is focused on programming in Object Windows - an object-oriented application framework designed by Borland for Windows programmers privy to Pascal. Introduction of Microsoft Windows version 3.0 in 1990 resulted in an increasing interest in graphic interface applications for DOS based personal computers. As the user of the application was provided with new standardized application interface tools such as pull-down menus, icons, dialog boxes, scroll bars, command buttons, etc., the programmer had to face new challenges posed by mastering close to 1000 functions of the Application Programming Interface, as well as not lower a number of constants, messages, structures, etc. Windows' Graphic Device Interface, and a number of other improvements at the level of the operating system, increased the productivity at the end-user level, and at the same time, rapidly increased the complexity of the development environment offered to the programmer. This situation lead to a popular saying that Windows programs are as good as they are because only the best programmers can complete them.

In consequence of a greater challenge

facing the programmer, a number of application frameworks have been developed in order to counterbalance the growing complexity of programming in Windows. Increasingly, object-oriented approach has been used in similar situations over the past decade. One of the most popular application frameworks for Pascal programmers are Object Windows developed as a standard library for Turbo Pascal and Borland Pascal for Windows. Optimally, development of new programming tools should gradually reduce the complexity of routine programming tasks in software engineering, and clearly separate the conceptual effort of the developer from details of the operating system, programming language and the programming techniques. The optimum situation, however, seems to be ever more elusive as new requirements are imposed on software application with availability of new technologies appearing periodically in all fields of computer and communications technology.
Professional programming was, and continues to be the domain of the few, and success in dealing with complexity has its fundamental bearing on the overall effectiveness in software development projects.

There are four basic sources of information for a programmer working in software development:
1. printed documentation and publications,
2. on-line help facilities,
3. peer-level and technical support consultation,
4. programmer's knowledge acquired in the course of programming studies and programming practice.

There is a natural tendency to comprise the printed sources of information, as well as human-to-human exchange of information in increasingly intelligent computer help facilities. In other words, a clear split emerges between the two forms of storing knowledge: in a computer and in the human brain; phenomenon appearing more and more visibly in all kinds of scientific and engineering tasks.
The three related problems are:

1. how to divide knowledge between computer and natural memory storage,
2. how to represent knowledge in a computer knowledge base, and how to implement effective querying procedures,
3. how to represent knowledge for human training, and how to effectively transfer it to the subject's brain.

In the presented paper, the first and the third problems are considered.
The obvious advantages of knowledge stored in the human brain are its high level of associativity and relatively short access and query time. Artificial intelligence system do not seem to closely approach humans in a number of highly complex intellectual task, therefore the main criterion of what to store and what not to store in human memory remains the time necessary for training. In other words, time lost for training, later called time loss (TL) must offset the time lost in result of using an external reference. Time saved by not having to refer to external sources of knowledge will later be called time gain (TG).
Time loss and time gain, whose relationship determines which elements of knowledge should be mastered in the course of the training, can be computed individually on a one-by-one basis for each of the pieces of information. Those pieces considered in the analysis will later be called items, while the criterion TG > TL will be called the memorization criterion.
Let us consider the time gain side of the memorization criterion. There are three major components here:

1. reference time, i.e. time needed to look up the item in the external source of information, e.g. in a help facility,
2. deadline penalty, i.e. extra time penalty coming from not having the item available before a certain deadline,
3. non-association penalty, i.e. extra time penalty coming from not being able to produce associations, which form the basis of creative thinking.

Only the first component of the time gain can easily be estimated and practically

employed in determining knowledge that should be transferred into the programmer's brain in the course of the training. The difficulty with the two other components is that the deadline penalty is a rare phenomenon in the practice of programming, while losses coming from non-association are very hard to estimate. Inability to form associations, a result of not having simultaneous access to unrelated components of the relevant knowledge, may involve a penalty that greatly surpasses the penalty coming from the need for extra knowledge references (first component of time gain).

The final form of the time gain equation in the practice of programming could be as follows:

```
TG = RefNo * RefT + NAP    (1)
```

where:

- TG - time gain,
- RefNo - number of references needed for a given item,
- RefT - average reference time,
- NAP - non-association penalty.

The time loss side of the memorization criterion can be computed with a high degree of accuracy, on condition that a systematic approach to the learning process is assumed.

SuperMemo is a learning technique, in which the spacing of repetitions of knowledge is optimized by means of an algorithm proprietary to SuperMemo World. As it can be demonstrated, SuperMemo produces a repetition spacing that is close to the optimum spacing for a selected level of knowledge retention.

From the optimum spacing of repetition, and from practical measurements of mean repetition times, the time loss per item can be computed for a selected period of time and a selected level of knowledge retention.

The accurate formula for time loss could like this:

```
TL = RepNo * RepT          (2)
```

where:

- TL - time loss in the training period,
- RepNo - number of repetitions in the training period,
- RepT - average repetition time.

Practice shows that knowledge retention of about 95% provides the best balance in the retention vs acquisition rate trade-off in learning material useful from the programmer's standpoint (this level of knowledge retention is also recommended in majority of practical applications of SuperMemo [4]).

It can also be shown that the knowledge of the highest degree of applicability in the practice of programming has extraordinarily short lifetime. Unlike in sciences or medicine, the knowledge of a programmer must constantly be turned over, with a large proportion of items losing their applicability is a period of time as short as 1-3 years.

As an illustration of item attrition in databases pertaining to the programming knowledge, let us consider an example of a SuperMemo database which was developed by the author with the view to learning Pascal programming , starting with Turbo Pascal 3.0 and ending with Borland Pascal 7.0 for Windows (Table I).

Year	Language	BP7	TPW	TP6	TP4	TP3	Total
1987	TP3	-	-	-	-	800	800
1989	TP4	-	-	-	1000	600	1000
1991	TP6	-	-	1500	1000	600	1500
1992	TPW	-	3500	1100	700	400	3500
1993	BP7	4000	3100	1100	700	400	4000

Table I.

Where:

- Year - the year in which database analysis was done,
- Language - the Pascal superset in question (Turbo Pascal 3.0, Turbo Pascal 4.0, Turbo Pascal 6.0, Turbo Pascal for Windows 1.0, and Borland Pascal 7.0 for Windows respectively),
- BP7, TPW, TP6, TP4, TP3 - number of items coming from the particular superset in the entire SuperMemo database.

The above figures illustrate (1) a substantial growth in the number of items coming from the increase in the programming system complexity resulting from the introduction of Turbo Pascal 1.0 for Windows, as well as (2) a high attrition rate of items in the database in accompaniment of employing new language elements, tools and techniques to Turbo Pascal.

The average number of repetitions per item is about 5-6 in a well-structured SuperMemo database. As the average repetition time is from 5 to 7 seconds, the time loss per item is about 30-40 seconds in the considered period of time.

From equations (1) and (2) we can easily obtain a simple formula for estimating the memorization criterion:

```
RefNo * RefT + NAP > RepNo * RepT
```

Considering that in the period of 2 years RepNo=6 and RepT=6 we can reduce the inequality to the following form:

```
RefNo * RefT + NAP > 36 [seconds]
```
(3)

We will use the above formula for the memorization criterion in the analysis of exemplary items to verify their suitability for memorization as opposed to being left for further reference.

In the course of developing a SuperMemo database on programming in Object Windows, the author has identified the following six dominant aspects that should be subject of training by means of SuperMemo:

1. Foundations of computing sciences: algorithms, data structures, computational complexity, probability calculus, operation research, numerical methods, software engineering, statistics, etc.
2. Elements of the operating system and the assembly language (in the considered case: DOS, Windows and the 80x86).
3. Elementary terminology and concepts of the development environment (for example, in Windows: client area, control menu, system key, modal vs modeless dialog, child vs popup window, logical vs device co-ordinates, interface object vs interface element, virtual method table, polymorphism, etc.).
4. Programming tools (debugger, profiler, resource workshop, help compiler, etc.).
5. Elements of the language; syntax and use: procedures, functions, variables, constants, objects and their hierarchy, fields, methods, messages, run-time errors (but not compiler errors), etc.
6. Practical tips, observations, and experimental findings not included in the documentation (these should be collected by the programmer in the course of the programming practice). For example, compiler bugs, freaks of the operating system, frequently encountered traps (e.g. using more than five device contexts in the GDI), etc.

Let us consider the memorization criterion for exemplary items used to learn programming in Object Windows, and belonging to the fifth knowledge domain, that of elements of the language. All the examples are taken from the database developed by the author. Letter Q and A are used to denote questions and answers.

Example 1:

Q: What is the name of the pointer to the main window (defined by TApplication)?
A: MainWindow.

In Object Windows, the application instance is defined as an object whose type it TApplication. One of the fields in TAp-

plication is a pointer to the main application window, MainWindow.
The name of the pointer can easily be found in the Borland Pascal on-line help. The reference time (RefT) can be estimated to be no less than 20 seconds assuming the knowledge of the name TApplication, or at least 40 seconds otherwise (486 class computers). The number of references (RefNo) depends greatly on the programmer's needs and his or her programming style. MainWindow is used in the method InitMainWindow, which must be specified for every Object Windows program. If the programmer works on a single project, he is more likely to reference MainWindow as the parent of other windows or dialog boxes. RefNo can be estimated as varying from 0 to 100 for the period of two years. The item in question asks for an identifier, and as such involves a very low non-association penalty (NAP). A conservative substitution to the memorization criterion could therefore look as follows: RefNo * 20 > 36, which justifies introducing the analysed item into a SuperMemo database as soon as the expected number of references is greater than one! It is important to notice, that a large number of references (above 20) may result in memorization of the item in question without the help of SuperMemo. However, memorization criterion does not lose its validity as long as the first reference does not result in a memory engram that can be sustained until the need for the next reference arises.

Example 2:

Q: What are the two stages of MakeWindow?
A: (1) Checking for memory conditions (ValidWindow),
(2) Creating the window (Create).

In Object Windows, MakeWindow is a method of TApplication, which takes a pointer to a window as its parameter, and is responsible for creating the relevant interface element. The knowledge of the stages of MakeWindow is not needed by the programmer in the coding process therefore RefNo can safely be put to zero. However, because of its educational aspect, and because of possibility of using the method Create, which is not recommended, the non-association penalty (NAP) can safely be set as greater than 36 seconds needed to satisfy the memorization criterion.

Example 3:

Q: What is HWindow?
A: handle to the interface element of a given object.

HWindow is used by all objects TWindowsObject and its descendants to store the handed to the corresponding interface element, or to store zero if the interface element has been destroyed or not created at all. Interestingly, this item does not satisfy the memorization criterion. Passive recognition of the identifier HWindow is needed in programming Object Windows; however, because of its frequent use and intuitiveness, the recognition can easily be achieved without extraneous training (i.e. any training different than active use of the identifier in the programming practice). The reason for placing this item in the database is different, and comes from the principles of SuperMemo, which demand that multifaceted approach to learning be taken. What the programmer has to remember and what satisfies the memorization criterion is the active recall of the name HWindow. However, adding a symmetrical item which, in a sense, duplicates the memorized piece of knowledge and asks for its passive recognition, has been proved to reduce the overall workload, i.e. the time loss side of the memorization criterion (TL). This happens by reducing the difficulty of the items in question; the fact reflected by a lower number of repetitions per item, e.g. 6 repetitions in two years instead of 10 repetitions, and by a shorter repetition time, e.g. 3 seconds instead of 5 seconds (the overall difference

here would be as between 2 items * 6 repetitions * 3 seconds = 36 seconds, and 1 item * 10 repetitions * 5 seconds = 50 seconds).

Example 4:

Q: What object types define HWindow?
A: TWindowsObject (and all the descendants).

Though again, the estimated reference number (`RefNo`) is zero, this item has been introduced into the database in result of a combination between the non-association penalty and the multifaceted approach stipulated by SuperMemo.

Example 5:

Q: What are the parameters of GetText in edit controls?
A: (1) retrieved text and (2) its maximum size.

Edit controls are standard Windows interface objects used in editing null-terminated strings. The method GetText defined by the object type TEdit, a descendant of TWindowsObject in Object Windows, retrieves the text from an edit control, and requires the maximum size of the retrieved text as the second parameter. This item is a typical example of a language item that fully satisfies the memorization criterion. Here, the reference time should rather be called a recompilation penalty time, because the main hitch in using GetText comes from the fact that it returns the size of the retrieved text, not the text itself. The mistaken reference of GetText will easily be discovered at compile time as a type mismatch. The penalty involved could span from 10 to 45 seconds depending on the size of the compiled code. The reference number (RefNo) can range from 10-100, and a conservative substitution to the memorization criterion could look as follows: `RefNo * 10 > 36` [seconds], which implies that four recompilations would justify introducing the item into the database.

Example 6:

Q: What are the descendant types of TStatic?
A: TEdit.

Static controls, encapsulated in Object Windows in the object type TStatic, are standard Windows interface objects which define static texts such as edit control labels in a dialog box.
The above item is an example of benefits coming from reasonable knowledge structuring in learning highly associative subject domains. From the assertion implied here, and from the understanding of inheritance in object-oriented programming, the programmer can infer a number of properties of objects typed as TEdit. For example, by specifying items needed for understanding the SetText method of TStatic, the programmer does not have to duplicate the same questions for TEdit.
Obviously, the multifaceted approach principle could be raised to justify such redundancy; however, some extra components of the memorization criterion would have to be added in order to balance the trade-off between time gain coming from multifacetedness and time loss coming from the extra workload.

Let us have a short look at the overall workload involved in the training based on SuperMemo. In the course of the project which started 401 days before writing this paper, the analysed SuperMemo database has grown to comprise 3605 items. After 401 days, the average number of repetitions per day was 47, the average number of repetitions per item was 4.5, the average repetition time was 7.4 seconds per item, and the average daily workload was 5 minutes and 44 seconds (excluding time needed to extract items from the documentation and to put them into the database). On average, 1 page of documentation produced 3-4 items, which indicates that the database makes up an equivalent of

900 pages of printed reference (Borland Pascal documentation, DOS and Windows manuals, and the Windows programming reference total over 7000 pages altogether).

Basing on the experience gained in the course of compiling the discussed database, the following exemplary 8-hour working schedule could be proposed for a software development project, in which new tools or a new environment are to be employed:

- Working with the documentation and studying how-to literature with the accompanying extraction of relevant items that should be memorized (about 2 hours). Here, only basic concepts and terminology should sparingly be marked for memorization with SuperMemo.
- Programming training, perhaps including the development of the code for the ultimate project, with extraction of items useful in programming tasks and in overcoming most typical programming hurdles (about 5 hours). Here, only the most frequently used details of the language and the development environment should be considered. Particular emphasis should be put on specifying items that solve bottleneck problems that have been encountered in the course of the practice.
- Development of a SuperMemo database and repetitions of the outstanding items (about 1 hour).

Obviously, the time needed for studying the documentation and the time needed for SuperMemo repetitions can decline rapidly with the advancement of the project, especially in programmers with a good background on the considered language and environment. Similarly, the availability of customized, ready-made SuperMemo databases could cut the third component of the training by up to 70% time.

The figures presented earlier in reference to the effectiveness of using SuperMemo in software development training may sound encouraging; however, it is not easy to convince an effective programmer with a firmly established modus operandi to adopt new solution, especially if they touch the neglected and under-appreciated field of training.

The major obstacle in popularization of SuperMemo among programmers seems to be the fact that development of well-structured databases that could result in a sharp increase in productivity is not as easy a task as it might seem at first. Therefore, an inexperienced user of SuperMemo software may quickly become disillusioned with rapid accumulation of repetitions resulting from wrong selection of the material, as well as from formulating items that do not comply with the principles of effective learning as stated in the definition of the SuperMemo method. However, the author is absolutely positive that it is only the question of time before SuperMemo becomes a standard tool used in routine training of a modern programmer.

References

[1] Bell D., Morrey I., Pugh J.: *Software engineering*, Prentice Hall, 1987.

[2] Borland Int.: *Object Windows. Programming Guide*, Borland International, 1992.

[3] Petzold Ch.: *Programming Windows 3.1.*, Microsoft Press, 1992.

[4] Woźniak P. A.: *Optimization of learning. A new approach and computer application*, 1990.

[5] Woźniak P. A., Biedalak K., *Metoda SuperMemo. Optymalizacja procesu uczenia się*, Informatyka, 1992, Vol. 10, p. 1.

[6] Woźniak P. A., Gorzelańczyk E. J.: *Optimal scheduling of repetitions in paired-associate learning*, Acta Neurobiologiae Experimentalis, 1992, Vol. 52, p. 189.

[7] Woźniak P. A.: *Speed-learning with SuperMemo for Windows*, SuperMemo World, 1993.

[8] Woźniak P. A.: *A new method of fast learning and its software implementation*, Polish Technical Review, 1993, Vol. 1, p. 8.

Joanna Kurzok-Derda
Instytut Systemów Sterowania, Katowice

System ekspertowy EKSSTER dla wspomagania projektowania i dokumentowania układów automatyki przemysłowej na bazie sterownika STERISS-2

Wstęp

System ekspertowy EKSSTER dla wspomagania projektowania i dokumentowania układów automatyki przemysłowej, a w szczególności schematów rozplanowania szaf sterowniczych i okablowania urządzeń automatyki, przeznaczony jest do przygotowywania projektów układów automatyki realizowanych na bazie programowalnych sterowników STERISS-2 produkowanych w ISS. System ekspertowy pozwala na przyspieszenie i optymalizację rozwiązań w procesie projektowania i sporządzania:

- schematów szaf sterowniczych przeznaczonych dla sterowników STERISS-2;
- schematów listew zaciskowych, a konkretnie tabeli połączeń sygnałów obiektowych;
- kosztorysu szafy sterowniczej wraz z kasetami zawierającymi pakiety sterownika STERISS-2, a także w ramach tegoż kosztorysu - wykazów i zestawień materiałowych.

Ponadto opracowany system pozwala na:

- tworzenie i zapamiętywanie schematów rysunkowych szaf sterowniczych;
- automatyzację przygotowywania dokumentacji.

1. Ogólna struktura systemu

1.1 Programowalny sterownik przemysłowy STERISS-2

System ekspertowy EKSSTER dla wspomagania projektowania i dokumentowania układów automatyki przemysłowej zrealizowany został na bazie sterownika programowalnego STERISS-2 produkowanego w Instytucie Systemów Sterowania w Katowicach. STERISS-2 jest sterownikiem programowalnym przeznaczonym do automatyzowania obiektów przemysłowych, a w szczególności do sterowania, regulacji, monitorowania i rejestracji. Sterownik ten charakteryzuje się się łatwością dopasowania wejść i wyjść do obiektu dzięki modularnej konstrukcji, łatwością programowania dzięki strukturalności programów oraz standardowym modułom programowym. STERISS-2 stwarza także możliwość ręcznego wykonywania operacji sterowniczych z klawiatury alfanumerycznej i funkcyjnej. Ponadto charakteryzuje się łatwością czynności serwisowych dzięki wbudowanym testom diagnostycznym.

Sterownik STERISS-2 łączy cechy typowego programowalnego sterownika logicznego oraz przemysłowego modularnego mikrokomputera. Jako programowalny sterownik logiczny realizuje funkcje sterowania sekwencyjnego. Jako przemy-

słowy mikrokomputer z wielozadaniowym systemem operacyjnym czasu rzeczywistego przeznaczony jest do realizacji złożonych algorytmów sterowania i komunikacji w systemach sterowania rozproszonych przestrzennie.

1.2 Środowisko programowe

Realizacja dowolnego systemu ekspertowego wymaga podjęcia ważnej decyzji o rodzaju sprzętu, za pomocą którego będzie on używany oraz podjęcia decyzji o tym, w jaki sposób zostanie przygotowany program dla tego sprzętu.
Biorąc pod uwagę tendencje na świecie oraz aktualny stan sprzętu komputerowego w Polsce można było przyjąć, że odpowiednim rozwiązaniem dla realizacji systemu ekspertowego dla wspomagania projektowania i dokumentowania układów automatyki przemysłowej na bazie sterownika STERISS-2 jest wybór mikrokomputera IBM PC.

Program określający działanie dowolnego systemu ekspertowego może być przygotowany w sposób dowolny. Celowe jest użycie specjalnych narzędzi programowych, takich jak:
- odpowiednie języki programowania,
- szkieletowe systemy ekspertowe (shell'e).

Kryteria stawiane tym narzędziom programowym są zależne od dziedziny, dla której opracowywany jest system ekspertowy. Do najczęściej rozpatrywanych wymagań zaliczyć należy:
- możliwość modyfikowania i uzupełniania bazy wiedzy na podstawie doświadczeń uzyskanych podczas realizacji wcześniejszych sesji systemu ekspertowego,
- możliwość badania bazy wiedzy i wykrywanie w niej różnych nieprawidłowości,
- możliwość ustalania sposobu komunikacji systemu ekspertowego z użytkownikiem w postaci dialogu prowadzonego w języku quasi-naturalnym.

Złożone systemy ekspertowe wymagają bardzo dużego nakładu pracy programistów oraz inżynierów wiedzy w przypadku, gdy systemy te opracowywane są bez pomocy takich narzędzi jak systemy szkieletowe lub metasystemy ekspertowe. Brak było odpowiedniego (wyposażonego w odpowiedni układ wnioskujący, układ objaśniający, układ sterujący, w odpowiednią pustą bazę wiedzy oraz w puste bazy danych) systemu ekspertowego szkieletowego, który mógłby być szybkim narzędziem programowym dla skonstruowania systemu ekspertowego dla wspomagania projektowania i dokumentowania układów automatyki przemysłowej. Zatem system ten konstruowany był od podstaw.

Do języków programowania szczególnie dogodnych dla omawianych potrzeb zaliczyć należało LISP i PROLOG.
LISP umożliwia proste zapisywanie działań na listach.
PROLOG natomiast pozwala na łatwą realizację rachunku predykatów.

Opracowując program w LISP-ie należy określić jakie dane i w jaki sposób przetwarzane będą zgodnie z programem. Opracowując natomiast program w PROLOG-u należy określić, jakie dane będą przetwarzane według programu, przy czym szczegóły sposobu przetwarzania danych będą w znacznej mierze ustalane automatycznie bez bezpośredniego udziału programisty (zgodnie z zasadą rezolucji). W języku C są również opracowywane złożone systemy ekspertowe. Jest to jednak związane z bardzo dużymi nakładami pracy ze strony programistów.

Z uwagi na powyższe spostrzeżenia i krótki termin realizacji, system ekspertowy EKSSTER dla wspomagania projektowania i dokumentowania układów automatyki przemysłowej na bazie sterownika STERISS-2 zrealizowany został w języku TURBO PROLOG (wersja 2.0).

Podsystem graficzny ElectroCAD konstruowania układów automatyki przemysłowej

napisany został w języku C++ firmy Borland. Uruchomiony został za pomocą zestawu kompilacyjnego Borland C++ działającego pod systemem operacyjnym MS DOS.
W zakresie grafiki komputerowej podsystem konstruowania układów geometrycznych nie opiera się na żadnym międzynarodowym standardzie graficznym. Wykorzystuje jedynie odpowiednio skuteczne przetwarzanie następujących prymitywów graficznych:
- linia,
- łuk,
- przewód,
- tekst o zmiennym kącie lub rozmiarze.

System ekspertowy EKSSTER został natomiast uruchomiony za pomocą zestawu kompilacyjnego Borland TURBO PROLOG wersja 2.0 działającego pod systemem operacyjnym MS DOS.
Z kolei interfejs programowy pozwalający na przesyłanie rozwiązań systemu ekspertowego EKSSTER do podsystemu ElectroCAD konstruowania układów automatyki przemysłowej napisany został w języku C++. Ów interfejs programowy tworzy na podstawie baz danych (w jęz. TURBO PROLOG) systemu ekspertowego EKSSTER odpowiednie bazy danych (w jęz. C++) wykorzystywane przez podsystem ElectroCAD (na życzenie użytkownika) w celu wyświetlenia na ekranie monitora bądź wyprowadzenia na drukarkę następujących schematów graficznych:
- rozkładu pakietów w poszczególnych kasetach szafy sterowniczej,
- tabeli stanowiącej kosztorys wyprodukowania zaprojektowanej i wypełnionej pakietami szafy sterowniczej,
- rysunku wszystkich listew zaciskowych związanych z zaprojektowaną szafą sterowniczą,
- tabeli połączeń sygnałów obiektowych związanej z zaprojektowaną szafą sterowniczą.

Oprócz tego napisany został w jęz. TURBO PROLOG wersja 2.0 moduł nadrzędny stwarzający możliwość wywołania:
- systemu ekspertowego EKSSTER,
- edytora graficznego ElectroCAD,

a także umożliwiający powrót do systemu DOS.
Po uruchomieniu tegoż modułu nadrzędnego, na ekranie komputera pojawi się menu postaci:

1. EKSSTER
2. ElectroCAD
3. KONIEC

1.3 Środowisko sprzętowe

System ekspertowy EKSSTER dla wspomagania projektowania i dokumentowania układów automatyki przemysłowej przeznaczony jest na środowisko mikrokomputerowe IBM PC. Może być użyty mikrokomputer klasy co najmniej AT z pamięcią operacyjną nie mniejszą niż 640 kB oraz pamięcią dyskową typu Winchester. System ekspertowy EKSSTER wykorzystuje:
- kartę graficzną EGA,
- dysk twardy co najmniej 20 MB,
- stacje dysków elastycznych 5.25" (1.2 MB),
- drukarkę,
- mysz.

1.4 Struktura systemu

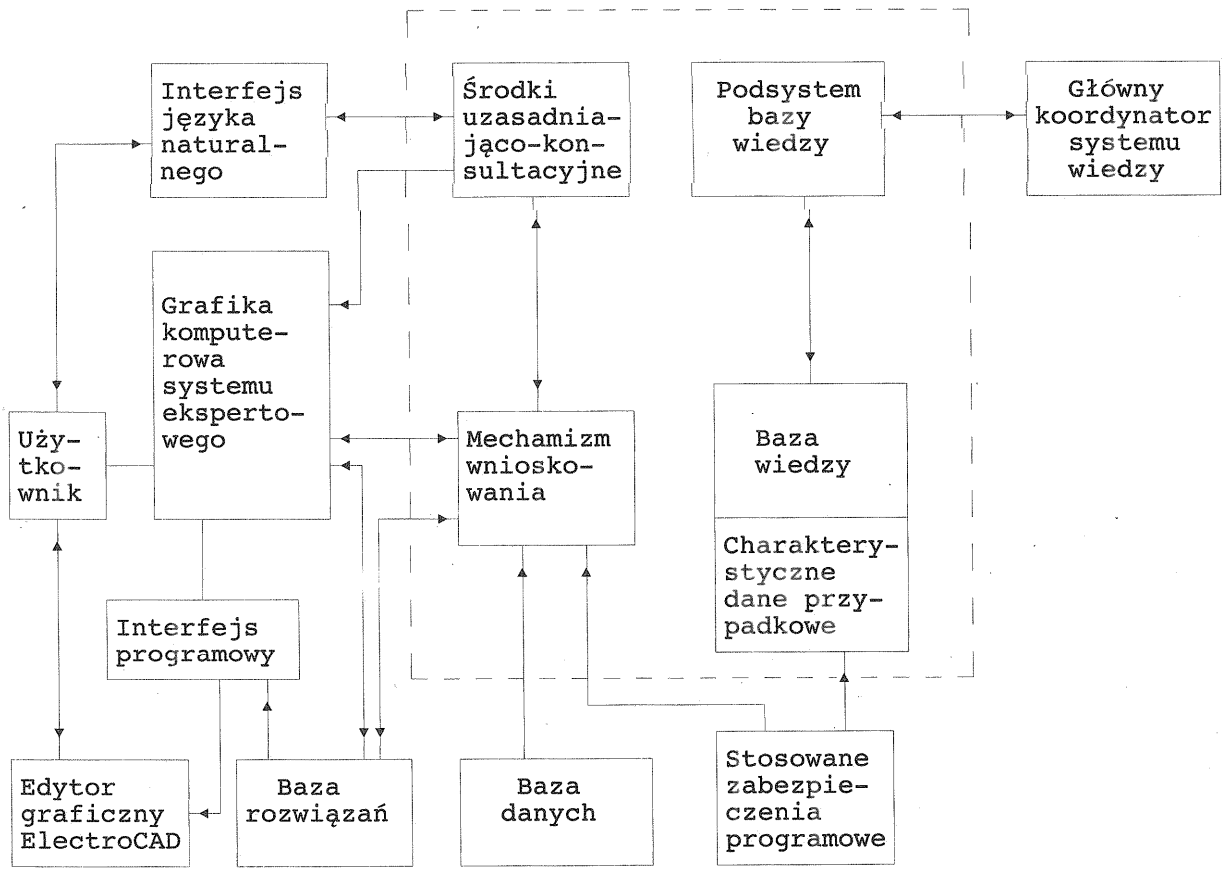
Do realizacji systemu ekspertowego EKSSTER wspomagania projektowania i dokumentowania układów automatyki przemysłowej konieczne jest wyposażenie systemu w następujące grupy funkcjonalne:
- prostą grafikę komputerową realizowaną w języku TURBO PROLOG;
- obsługę urządzeń, takich jak: ekran, drukarka, ploter, mysz;
- mechanizm wnioskowania;
- zabezpieczenie programowe;
- interfejs języka naturalnego;
- interfejs programowy przekształcający struktury danych zrozumiałe dla systemu ekspertowego, w struktury danych zrozumiałe dla edytora graficznego ElectroCAD.

Bardzo uproszczona grafika komputerowa realizowana jest w ramach projektowanego systemu ekspertowego za pomocą mechanizmów TURBO PROLOG-u.

Obsługa urządzeń, takich jak drukarka, ploter, ekran, mysz realizowana jest w ramach edytora graficznego ElectroCAD. Natomiast mechanizm wnioskowania, środki uzasadniająco-konsultacyjne, zabezpieczenia programowe, interfejs języka naturalnego, a także częściowo obsługa urządzeń, takich jak ekran, drukarka działają w obrębie części ekspertowo-doradczej projektowanego systemu. Część ta realizowana jest w języku TURBO PROLOG firmy Borland (wersja 2.0). Model funkcjonalny systemu EKSSTER przedstawiony został na rys. 1.1.

System ekspertowy EKSSTER złożony jest z czterech niezależnych modułów funkcjonalnych umożliwiających:

(1) tworzenie i aktualizację bazy danych systemu zawierającej wykaz wszystkich niezbędnych elementów konstrukcyjnych (dla dokonania rozmieszczenia pakietów w szafie sterowniczej, dla ustalenia tabeli połączeń sygnałów obiektowych, dla sporządzenia kosztorysu i równocześnie dla sporządzenia w ramach kosztorysu wykazu podzespołów szafy sterowniczej do skompletowania) wraz z aktualnymi parametrami technicznymi i ceną;
(2) tworzenie przez użytkownika projektu rozmieszczenia pakietów w poszczególnych kasetach szafy sterowniczej za pomocą prostego edytora graficznego, oraz na żądanie użytkownika - weryfikację tak utworzonego projektu, a także wyświetlanie uwag i sugestii modułu dokonującego ekspertyzy rozmieszczenia pakietów w szafie sterowniczej;
(3) tworzenie schematu zawierającego li-



RYS. 1.1. Model funkcjonalny systemu ekspertowego EKSSTER

stwy zaciskowe dla zaakceptowanego przez użytkownika rozkładu pakietów w szafie sterowniczej oraz tworzenie tabeli połączeń sygnałów obiektowych;

(4) sporządzenie kosztorysu dla utworzonego przez użytkownika rozmieszczenia pakietów w szafie sterowniczej oraz sporządzenie w ramach kosztorysu wykazów i zestawień materiałowych.

Bazy danych utworzone za pomocą modułu (1) przez tzw. projektanta systemu ekspertowego lub inaczej przez głównego koordynatora systemu wiedzy, są zbiorami wejściowymi dla modułów (2), (3) oraz (4). W trakcie realizacji modułu (2) tworzona jest dodatkowa baza danych zawierająca informacje o sposobie rozmieszczenia pakietów w poszczególnych kasetach szafy sterowniczej. Ponadto na podstawie tej dodatkowej bazy danych tworzony jest za pomocą specjalnego interfejsu programowego (napisanego w jęz. C++) - plik w formacie edytora graficznego ElectroCAD przedstawiający rozmieszczenie pakietów w szafie sterowniczej.

W trakcie realizacji modułu (2) projektant rozmieszczenia pakietów w szafie ma możliwość wywoływania programu dokonującego ekspertyzy i weryfikacji projektu.

Owa dodatkowa baza danych zawierająca projekt rozmieszczenia pakietów w szafie sterowniczej jest bazą danych niezbędną dla modułu (3), a także niekiedy dla modułu (4).

Na podstawie baz danych utworzonych w trakcie realizacji modułów (3) i (4) generowane są za pomocą specjalnego interfejsu programowego (napisanego w jęz. C++) - pliki w formacie edytora ElectroCAD zawierające odpowiednio: kosztorys dla utworzonego rozmieszczenia pakietów w szafie sterowniczej, schemat listew zaciskowych związanych z zaprojektowaną szafą sterowniczą oraz schemat tabeli połączeń sygnałów obiektowych związanych z zaprojektowaną szafą sterowniczą.

Pliki ze schematami utworzonymi w trakcie realizacji modułów (2), (3) i (4) stanowią podstawę dokumentacji technicznej projektu, i mogą podlegać dalszej obróbce przez edytor graficzny ElectroCAD.

Edytor graficzny ElectroCAD jest programem umożliwiającym w sposób łatwy i wygodny rysowanie schematów elektrycznych. Praca z edytorem odbywa się w przyjaznym dla użytkownika systemie rozwijanych menu zawierających polecenia edytora.

ElectroCAD jest programem przygotowanym w konsultacji z projektantami z branży elektrotechniki i energetyki. Jest on edytorem przeznaczonym dla wspomagania prac w tychże środowiskach projektowo-badawczych. ElectroCAD powstał na bazie nowoczesnych środków programowych jakie oferuje oprogramowanie Borland C++ w zakresie oprogramowania obiektowego.

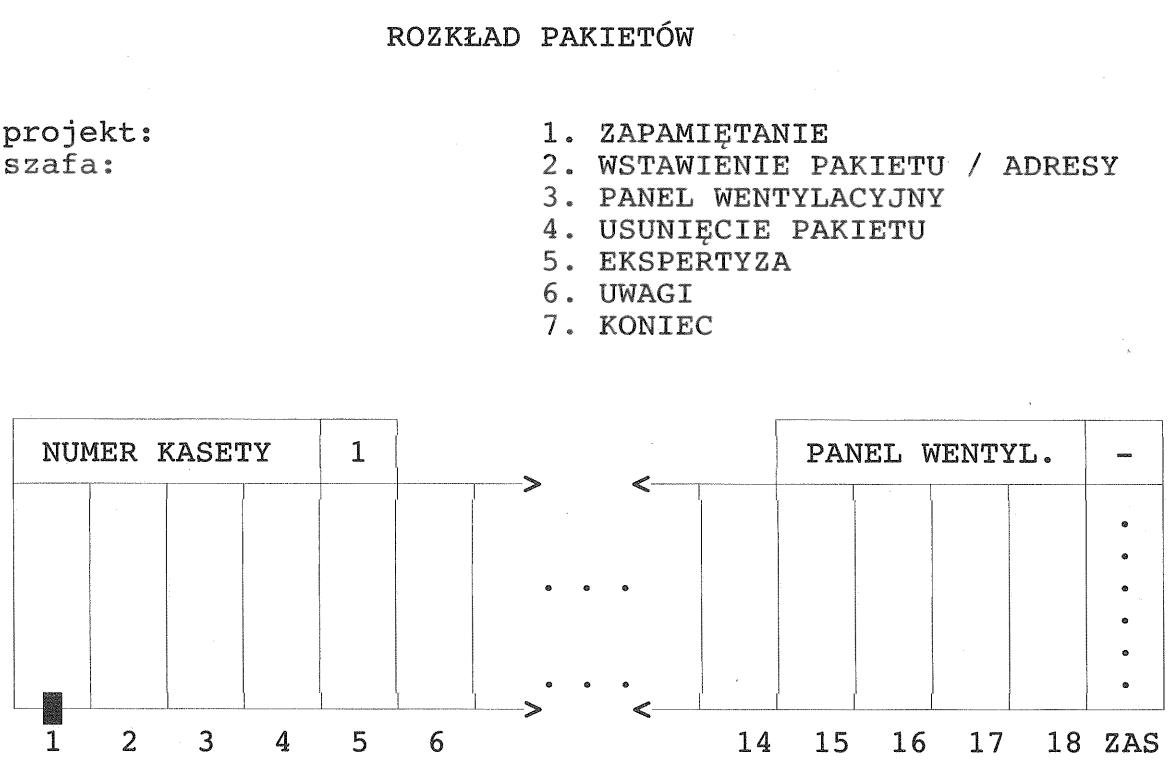
2. Opis działania systemu EKSSTER.

Po uruchomieniu systemu ekspertowego EKSSTER, na ekranie komputera pojawi się menu główne w postaci:

1. BAZY DANYCH
2. ROZKŁAD PAKIETÓW
3. LISTWY ZACISKOWE
4. KOSZTORYS
5. KONIEC

W celu wybrania dowolnej pozycji z wyświetlonego menu należy nacisnąć klawisz z odpowiednią cyfrą (od 1 do 5). Bezpośrednio po naciśnięciu klawisza nastąpi wywołanie odpowiedniej funkcji realizującej wybraną pozycję menu głównego. Pozostałe klawisze (różne od 1 do 5) będą przez system ignorowane.

Funkcja BAZY DANYCH przeznaczona jest do tworzenia i aktualizacji baz danych (SZAFA, KASETA, ZASILACZ, PAKIET, WERSJE, SYGNAŁY) elementów mecha-



RYS. 2.1. Menu podstawowe funkcji ROZKŁAD PAKIETÓW

nicznych i elektrycznych koniecznych do realizacji funkcji wywoływanych przez wybranie pozycji 2, 3 i 4 menu głównego systemu. Bazy te muszą zostać utworzone przed wyborem którejkolwiek z tych pozycji.

Funkcja ROZKŁAD PAKIETÓW umożliwia rozmieszczenie poszczególnych pakietów w kasetach wybranej szafy sterowniczej, oraz weryfikację tego rozkładu za pomocą programu ekspertowego. Po wprowadzeniu przez użytkownika nazwy projektu oraz numerów szafy sterowniczej i zasilacza, na ekranie wyświetlona zostanie aktualna nazwa projektu, odpowiednie menu oraz rysunek kasety, tak jak przedstawiono na rys. 2.1. W slotach kasety przedstawionej na rys. 2.1. automatycznie wyświetlone zostaną nazwy pakietów (o ile rozmieszczenie pakietów w szafie było wcześniej zaprojektowane).

Na początku wyświetlona zostanie kaseta nr 1. Zmiana kasety możliwa jest poprzez naciskanie klawiszy edycyjnych: ⟨ ↓ ⟩ lub ⟨ ↑ ⟩. Przemieszczanie się między sąsiednimi slotami kasety odbywa się za pomocą klawiszy: ⟨ ← ⟩ lub ⟨ → ⟩. Wybrany slot zaznaczany jest kursorem graficznym.

Funkcja LISTWY ZACISKOWE powoduje tworzenie schematu zawierającego listwy zaciskowe dla projektu rozmieszczenia pakietów w poszczególnych kasetach szafy sterowniczej, zaakceptowanego przez użytkownika, zaś funkcja KOSZTORYS powoduje sporządzenie kosztorysu dla tego projektu. Funkcja LISTWY ZACISKOWE pozwala również utworzyć tabelę połączeń sygnałów obiektowych. Rozszywanie sygnałów obiektowych odbywać się może wg jednego z trzech algorytmów wybieranego na życzenie użytkownika.

Danuta Kajrunajtys
Katedra Informatyki, Akademia Ekonomiczna w Krakowie

Komputerowe systemy wspomagania decyzji a modele sytuacji decyzyjnych

1. Wstęp

Dostrzeżenie, iż informacja jest towarem stało się w świecie podstawową przyczyną rozwoju zastosowań informatyki w zarządzaniu i tworzenia aplikacji opartych na coraz bardziej zaawansowanych modelach logicznych.

Istotą procesu zarządzania są procesy informacyjno - decyzyjne, mające na celu permanentne dopasowywanie aktualnego stanu obiektu (opisanego gromadzonymi charakterystykami liczbowymi i jakościowymi) do wyznaczonego wzorca [1], [2]. Na procesy tc składają się:

- pozyskiwanie danych,
- gromadzenie i aktualizowanie,
- porządkowanie,
- przetwarzanie,
- wyszukiwanie,
- prezentowanie,
- formułowanie problemów,
- generowanie wariantów rozwiązania,
- opis i analiza struktury wariantów,
- ocenianie wariantów,
- wybór postępowania,
- wdrażanie decyzji i aproksymowanie skutków.

Zdecydowana większość obecnie eksploatowanych aplikacji została zaprojektowana i zbudowana jako odpowiedź na potrzebę wspomagania gromadzenia, aktualizowania, porządkowania, przetwarzania, wyszukiwania czy wreszcie prezentowania danych. Takim celom (kilku lub wszystkim z tej listy) przyjętym bez modyfikacji od użytkowników podporządkowane zostały zarówno struktury danych jak i układy funkcjonalne eksploatowanych rozwiązań. System wspomagania decyzji jest natomiast produktem, w którym z uwagi na przeznaczenie i spełniane funkcje zaimplementowane zostać powinny:

- doświadczenia psychologii behawioralnej dostarczającej narzędzi opisu zachowania się osób dokonujących wyboru przy uwzględnieniu możliwości percepcyjnych człowieka oraz metod opisu klas sytuacji decyzyjnych,
- propozycje analizy decyzyjnej korzystającej z modeli normatywnych określających warunki racjonalności decyzji,
- mechanizmy zaproponowane przez teorię zbiorów rozmytych jako doskonale nadające się do opisu relacji w systemach ekonomicznych, [9]
- doświadczenia sztucznej inteligencji pozwalające na zbudowanie modułu odpowiedzialnego za akwizycję i udostępnianie wiedzy zawartej w systemie.

Celem niniejszego opracowania jest prezentacja jednego z podejść do istoty pozostałych etapów procesów informacyjno - decyzyjnych a na tej podstawie, koncepcji systemu wspomagania decyzji.

2. Modele sytuacji decyzyjnych

Rozważania rozpoczniemy od przeglądu modeli sytuacji decyzyjnych. Są one w literaturze klasyfikowane w zależności od stopnia formalizacji na:

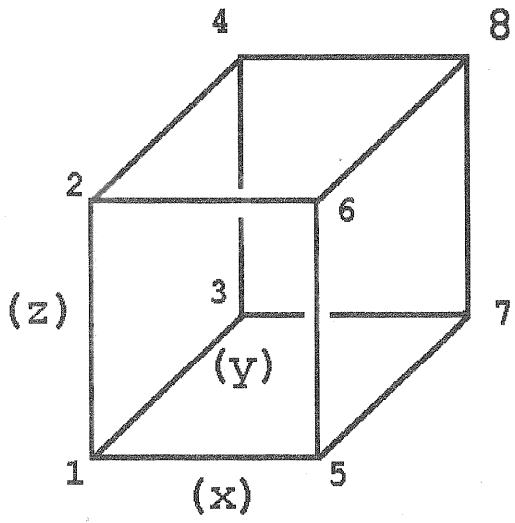
- normatywne (matematyczne),
- opisowe, powstałe jako modyfikacje modeli normatywnych poprzez zastąpienie niektórych zmiennych obiektywnych w modelach normatywnych wielkościami subiektywnymi lub poprzez wprowadzenie do modeli normatywnych nowych parametrów wykrytych w analizie kierunków subiektywnych odchyleń od modelu,
- modele odzwierciedlające system preferencji decydenta i stosowane przez niego psychofizyczne reguły wyboru.

Ostatnia grupa modeli, rozwijana przez psychologię behawioralną, nie tylko respektuje ale wręcz wymaga od decydenta, aby wyboru dokonał zgodnie ze swoją skalą wartości. Sytuacja decyzyjna rozumiana jest tutaj jako:

czynność C podmiotu P w sytuacji S

Decydent rozpoznaje sytuację i podejmuje działania, które jego zdaniem powinny przyczynić się do rozwiązania problemu. Sytuacja, w której zachodzi konieczność podjęcia działania jest częścią środowiska, w jakim człowiek ten działa. Dlatego też sytuację (analogicznie jak i środowisko) można przedstawić w trójwymiarowej przestrzeni XYZ w następujący sposób [5]:

wymiar (x) -
to stopień **niepewności** określający, na ile można przewidzieć wystąpienie (lub niewystąpienie) pewnych zdarzeń,

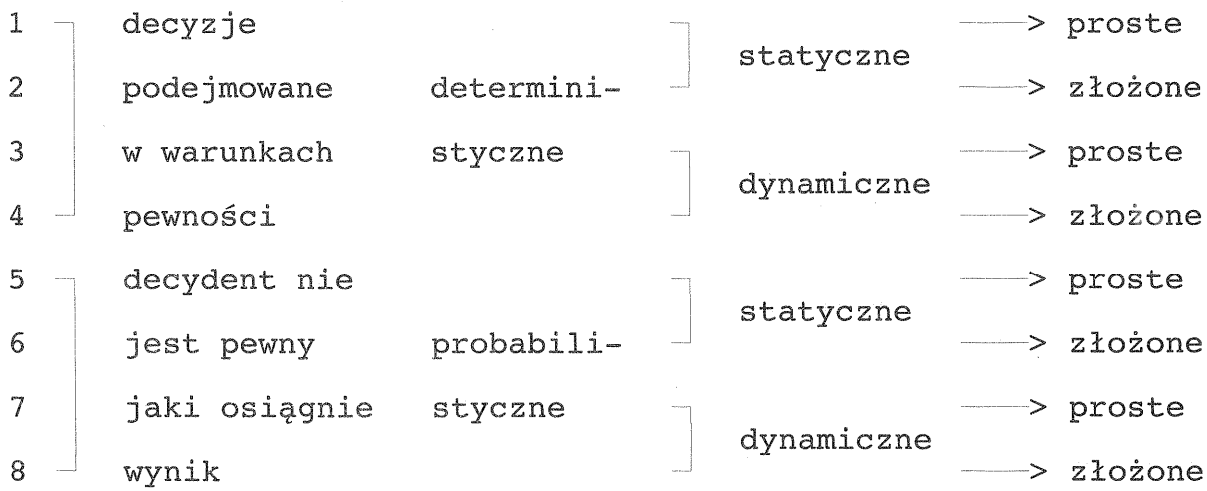


RYS. 1. Trójwymiarowy model środowiska

wymiar (y) -
to stopień **dynamiki** określający zmienność z jaką sytuacja ulega modyfikacji i przeobrażeniom wraz z upływem czasu,

wymiar (z) -
to stopień **złożoności** wyznaczony przez liczbę zmiennych lub czynników, które zostaną uwzględnione przy podejmowaniu decyzji, sytuacja jest tym bardziej skomplikowana, im więcej zmiennych ją opisuje.

W zależności od lokalizacji analizowanej sytuacji w powyższej przestrzeni (środowisku) można wyróżnić następujące klasy sytuacji decyzyjnych (rys. 2):



RYS. 2. Klasy sytuacji decyzyjnych

Modele sytuacji decyzyjnych bez względu na typ powinny zawierać: podstawową definicję modelowanej sytuacji (zwykle jednoargumentowej), aksjomaty tej definicji, prezentację sytuacji wieloaspektowej, interpretację preferencyjnej niezależności atrybutów w rozpatrywanym modelu, zestawienie procedur oceny jednowymiarowych funkcji wartości.

Statyczna sytuacja decyzyjna (SSD) (za [5], [8]) reprezentowana na rys. 1 przez punkty 1, 2, 5, 6 jest opisywana przez:

SSD ===== <D, H, u>

gdzie:

$D = \{d_1, \ldots d_n\}$ - skończony zbiór możliwych do podjęcia działań,

$H = \{h_1, \ldots, h_m\}$ - skończony zbiór możliwych stanów rzeczy (hipotez o stanie rzeczy) determinujących wynik poszczególnych działań. Hipoteza h_i to sąd o przewidzianych zdarzeniach [5].

$u = \{u_1, \ldots, u_m\}$ - funkcja rzeczywista wyników (konsekwencji), zwana użytecznością [3], określona na iloczynie kartezjańskim D x H w ten sposób, że:

$u(d_i, h_j)$ - jest wynikiem, który decydent otrzymuje, gdy wybierze działanie d_i i hipoteza h_j okaże się, prawdziwa.

Dynamiczna sytuacja decyzyjna (DSD) - decydent może modyfikować rozkład prawdopodobieństwa na zbiorze stanów rzeczy (na rys. 1 punkty 3, 4, 7, 8):

DSD ===== <D, H, u, E>

gdzie:

$E = \{e_1, \ldots, e_r\}$ - czynności pozwalające na uzyskanie nowej informacji o użyteczności działań lub prawdopodobieństwa stanów rzeczy.

Szczególnym przypadkiem dynamicznej sytuacji decyzyjnej są te, których konsekwencje są rozłożone w czasie. Jakkolwiek są to interesujące problemy, to z uwagi na to, iż prace teoretyczne nad nimi są jeszcze słabo zaawansowane (w zakresie znanym autorce), zostaną wyłączone z dalszych rozważań.

Model statycznej sytuacji deterministycznej - celem analizy decyzyjnej w odniesieniu do decyzji podejmowanych w warunkach pewności (na rys. 1 punkty 1 i 2) jest określenie na zbiorze możliwych do podjęcia działań porządkowej funkcji użyteczności v odzwierciedlającej preferencje użytkownika takiej, że dla wszystkich konsekwencji x, y[1] [7]:

$$v(x) \geq v(y) \Leftrightarrow x \succeq y \qquad (1)$$ [2]

Z powyższego wynika, że funkcja użyteczności v może być określona, gdy spełnione są następujące dwa postulaty racjonalności:

- *spójności*, będący interpretacją formuły (1):
  oznacza, iż z każdej pary alternatyw można wskazać tę preferowaną,
- *przechodniości:*
  ze słabej preferencji x nad y i y nad z powinna wynikać słaba preferencja x nad z.

Jednak w przypadku wieloaspektowej alternatywy zintegrowanie opisujących ją atrybutów w całościową ocenę wymagałoby od decydenta nieprzeciętnych zdolności poznawczych (percepcji, przechowywania i przetwarzania informacji opisujących te alternatywy). Często więc postulaty te zostają naruszone. Niezbędny jest formalizm, który pozwoli wyznaczyć wartość złożonych alternatyw poprzez wyznaczenie wartości poszczególnych atrybutów. Funkcja pozwalająca integrować częściowe oceny (wartości poszczególnych atrybutów) w globalną wartość działania przyjmuje ogólną postać [7]:

$$v(x_1, x_2, \ldots, x_n) = f(v_1(x_1), v_2(x_2), \ldots, v_n(x_n)) \qquad (2)$$

[1] x, y - działania podejmowane przez decydenta,

[2] symbol $\succeq$ oznacza słabą preferencję decydenta

Subiektywne oceny wartości atrybutów będą mogły zostać zintegrowane w globalną ocenę alternatywy przy pomocy funkcji, która powinna:
- uwzględniać postulaty racjonalności wymienione powyżej,
- być formułą względnie prostą (pracochłonność analizy i racjonalny czas oczekiwania na wyniki),
- odzwierciedlać preferencyjną niezależność wszystkich wymiarów jak i łączną niezależność wszystkich wymiarów.

Warunki te spełnia następująca funkcja **addytywna** [7]:

$$v(x_1,x_2,...,x_n) = \sum_{i=1}^{n} v(x_i) \qquad (4)$$

Jej użycie musi poprzedzić sprawdzenie, czy preferencje dla wartości poszczególnych wymiarów są niezależne od stałych wartości pozostałych wymiarów oraz czy n-1 par wymiarów jest preferencyjnie niezależna celem wykrycia ewentualnej łącznej zależności wymiarów.

Model statycznej sytuacji probabilistycznej - celem analizy decyzji podejmowanych w warunkach ryzyka jest określenie na zbiorze niepewnych zdarzeń funkcji użyteczności (rozumianej jako wybór parametrów loterii [3]) poprzez przypisanie prawdopodobieństwa tym zdarzeniom zgodnie z subiektywną oceną decydenta oraz wybranie najkorzystniejszej alternatywy. Do wybrania najkorzystniejszej alternatywy wykorzystuje się zasadę maksymalizacji wartości oczekiwanej. **Wartość oczekiwana działania niepewnego** prowadzącego do różnych możliwych wypłat z odpowiednimi prawdopodobieństwami jest sumą iloczynów wypłaty i prawdopodobieństwa jej otrzymania, przy czym prawdopodobieństwo to jest subiektywnie przypisaną przez decydenta wielkością. Do pomiaru subiektywnie przypisywanych prawdopodobieństw wykorzystuje się zasadę maksymalizacji oczekiwanej użyteczności. **Oczekiwana użyteczność (pożądalność)** loterii, o wynikach otrzymanych z odpowiednimi prawdopodobieństwami jest określona przez sumę pożądalności jej wyników ważonych prawdopodobieństwem uzyskania każdego z wyników.

Aksjomaty teorii użyteczności sytuacji probabilistycznej:
- *spójności*: konsekwencje można uporządkować wg ich atrakcyjności,
- *przechodniości*: uporządkowanie preferencji jest przechodnie,
- *redukcji loterii złożonych*: atrakcyjność zakładu nie powinna zależeć od jego struktury,
- *ciągłości*: nie ma konsekwencji nieskończenie pożądalnych bądź nieskończenie niepożądalnych,
- *podstawialności*: gdy wyniki loterii rozważane oddzielnie są indyferentne, to takimi są również, gdy wchodzą w skład dowolnej loterii,
- *dominacji*: loteria, która może pogorszyć wynik, nie może być przedkładana nad pewny wynik lepszy i odwrotnie, loteria, która może tylko poprawić wynik, powinna być przekładana nad pewny wynik gorszy.

Postulat spójności w probabilistycznej sytuacji wieloatrybutowej przyjmuje jedną z dwóch postaci:
1) jest **multiplikatywną** funkcją jednowymiarowych ocen poszczególnych atrybutów,
2) jest **addytywną** funkcją jednowymiarowych ocen poszczególnych atrybutów.

Wobec funkcji użyteczności ryzykownych alternatyw, tak jak wobec alternatyw deterministycznych zachodzi konieczność sprawdzania warunków niezależności

[3] *loterię* opisują dwa parametry: *prawdopodobieństwo* wystąpienia wszystkich możliwych zdarzeń niepewnych oraz *wypłata* - oszacowane konsekwencje poszczególnych zdarzeń niepewnych.

poszczególnych wymiarów jak i łącznej niezależności par atrybutów.

3. Techniki ustalania funkcji użyteczności

Funkcja użyteczności jest funkcją rzeczywistą wyników (konsekwencji) działań podjętych przez decydenta.
Procedura określania wielowymiarowej funkcji użyteczności składa się z dwóch etapów [3][4]:
- oceny jednowymiarowych funkcji wartości,
- integrowanie ich w poszukiwaną funkcję wieloargumentową.

Procedura ta odnosi się tak do modelu deterministycznego jak i probabilistycznego.

3.1 Model deterministyczny

W literaturze [5] proponuje się zastosowanie m.in. następujących technik do szacowania funkcji użyteczności:
- **technika sekwencji standardowych:**
  Decydent oznacza na skali pomiarowej punkt zerowy (tj. punkt, w którym najgorsza wartość alternatyw ze względu na wymiary je różnicujące wynosi 0). Następnie decydent podaje kolejno wyższe wartości poszczególnych atrybutów dotąd, dopóki zachodzi indyferencja pomiędzy wszystkimi wartościami rozważanych atrybutów. Wymaga to w każdym kroku sprawdzania wewnętrznej zgodności ocen (spełnienia warunku addytywności).
- **technika wyszukiwania wartości środkowej:**
  Technika zakłada niezależny pomiar wartości poszczególnych atrybutów alternatyw. Decydent wskazuje środkowy punkt pomiędzy skrajnymi wartościami ustalonego atrybutu alternatywy. Jest to punkt, w którym funkcja wartości alternatywy staje się jednowymiarowa, gdyż pozostałe atrybuty zostały wyzerowane. Po określeniu wszystkich jednowymiarowych funkcji (ich liczba jest równa liczbie atrybutów) decydent wskazuje wagi poszczególnych wymiarów. Ostateczna wartość parametrów zostaje ustalona po porównaniu alternatywy opisanej jednoargumentową funkcją rozważanego wymiaru z alternatywą scharakteryzowaną wagą tego atrybutu.
- **techniki bezpośrednie:**
  Do oszacowania funkcji wartości alternatywy wykorzystuje się subiektywne oceny decydenta. Odmiany techniki:
  - bezpośrednia ocena na skalach *(ang. direct rating scale)* - decydent po zdefiniowaniu najgorszej (jej wartość = 0) i najlepszej (wartość = 100) alternatywy dla określonego atrybutu porządkuje (w skali 0 - 100) pozostałe alternatywy ze względu na ten atrybut;
  - bezpośrednie ocenianie stosunków między różnymi wielkościami - decydent określa, ile razy bardziej ceni alternatywę W_i wobec W_j dla określonego atrybutu.

Ze względu na swą prostotę wynikającą z braku sformalizowania, techniki bezpośrednie mogą prowadzić do zestawienia alternatyw nie spełniających założeń addytywnego modelu.

3.2 Model probabilistyczny

Procedura ustalania jednoargumentowej funkcji użyteczności w sytuacji probabilistycznej zakłada sekwencyjne dochodzenie do poszukiwanej postaci.

Po ustaleniu najbardziej (x^*) i najmniej (x^o) atrakcyjnych konsekwencji z punktu widzenia decydenta i przypisaniu im użyteczności: $u(x^*) = 1$ i $u(x^o) = 0$ określa się użyteczność konsekwencji pośredniej znajdując punkt indyferencji decydenta między otrzymaniem z pewnością tej kon-

[4] wynika to z formuły (3)

sekwencji x a loterią [x^*, p, x^o], w której otrzymuje x^* z prawdopodobieństwem p i x^o z prawdopodobieństwem 1 - p. Krzywa użyteczności powstaje po dopasowaniu krzywej do wyznaczonych w powyższy sposób punktów.

Aby wskazać decydentowi działanie o najwyższej dla niego (subiektywnie) oczekiwanej użyteczności niezbędne jest określenie subiektywnej interpretacji, czyli stopnia przekonania osoby o zaistnieniu danego zdarzenia. Służą temu dwie poniższe metody:

- **metoda najmniejszej ceny sprzedaży zakładu**
  Decydent określa, za jaką najniższą cenę sprzedałby prawo do rozegrania loterii, przy czym decydentowi jest wiadome, że ewentualny nabywca ma ustaloną (choć niejawną) ceną, za jaką skłonny jest kupić prawo do rozegrania loterii. Formuła obliczania ceny sprzedaży loterii wynika bezpośrednio z formuły obliczania wartości oczekiwanej działania obliczanej dla dwóch wypłat i ich prawdopodobieństw dopełniających się do jedności.
  Metoda należy do pośrednich, charakteryzujących się wykorzystaniem werbalnej deklaracji decydenta.
- **metoda równoważnej urny**
  W celu określenia subiektywnego prawdopodobieństwa hipotezy poszukuje się równoważnej loterii o tych samych konsekwencjach i znanym prawdopodobieństwie otrzymania. Szukane prawdopodobieństwo otrzymujemy z porównania obu loterii.

Szacowanie wielowymiarowej funkcji użyteczności rozpoczyna wskazanie najbardziej $\{u(x_1^*, x_2^*, ..., x_n^*) = 1\}$ i najmniej $\{u(x_1^o, x_2^o, ..., x_n^o) = 0\}$ atrakcyjnych konsekwencji oraz określenie wag poszczególnych wymiarów. Następnie znajduje się punkty indyferencji między zdarzeniem o najlepszej wartości dla rozpatrywanego wymiaru (przy najgorszych pozostałych) a zdarzeniem dającym z przyjętym subiektywnie prawdopodobieństwem najlepsze wartości wszystkich wymiarów. Wyznaczone w ten sposób punkty tworzą krzywą użyteczności.

4. Koncepcja oprogramowania wspomagającego

Rozważania na temat systemów wspomagania decyzji mają prawie tak długą historię jak prace nad zastosowaniami informatyki w zarządzaniu. Nadal jednak zarówno w polsko- jak i obcojęzycznej literaturze można przeczytać o systemach wyszukiwania danych, które nazywane są systemami wspomagania. Jak powszechnie wiadomo, genezą współczesnych systemów wspomagania decyzji są prace z zakresu badań operacyjnych, skąd pochodzą sformalizowane modele matematyczne opisujące związki między zmiennymi. Modele te nie uwzględniają jednak złożoności problemu wyrażonej jakością i liczbą czynników wpływających na ostateczną postać decyzji.

W niniejszym rozdziale przedstawiono wstępne rezultaty prac mających na celu opracowanie koncepcji otwartego komputerowego systemu w technologii bazy wiedzy, wspomagającego proces decyzyjny. Rozdział ten prezentuje także metodologię tworzenia aplikacji oraz zawiera określenie warunków, które powinny być spełnione, aby możliwym było zastosowanie wspomagania.

Pod pojęciem **wspomagania decyzji** rozumie się działalność tego, który na podstawie jasno wyrażonych, lecz nie koniecznie w pełni sformalizowanych modeli, pomaga znaleźć elementy odpowiedzi na pytania, jakie stawia decydent w procesie decyzyjnym, elementy wyjaśniające decyzję i zwykle zalecające lub uprzywilejowujące pewne zachowanie w celu zwiększenia spójności między ewolucją procesu z jednej strony, a celami i systemem wartości decydenta z drugiej strony [Roy'90].

Analiza przebiegu procesów decyzyjnych, w tym:
- sposobu budowania modelu sytuacji,
- doboru atrybutów,
- kolejności ich uwzględniania w analizie decyzyjnej,

pozwala stwierdzić, iż proces podejmowania decyzji jest indywidualnym, niepowtarzalnym działaniem (procesem twórczym) jednostki. Proces taki trudno jest (bądź nie jest to możliwe) ująć jednoznacznie w formuły i algorytmy statystyczne. Dlatego też zadanie informatyki jako narzędzia wspomagającego decydenta powinno polegać na dostarczeniu narzędzi, które pozwolą użytkownikowi interakcyjnie definiować modele sytuacji decyzyjnych i porównywać wyrażone w różnych jednostkach wartości atrybutów tychże modeli.

4.1 Założenia

Przeznaczeniem projektowanego systemu jest wspomaganie decyzji gospodarczych. Oznacza to, iż analogicznie do działań decydenta, wykorzystującego opracowania danych opisujących przebieg procesów i stany obiektów organizacji, projektowany system zbudowany zostanie z myślą o tej organizacji, która będzie mogła udostępnić dane w sposób bezpośredni. Inaczej mówiąc, warunkiem koniecznym zaimplementowania tej klasy systemu jest baza danych zawierająca wszystkie dane niezbędne w procesie decyzyjnym. Oznacza to, że już na etapie projektowania struktur bazy danych uwzględnione zostaną wymagania podsystemu decyzyjnego. Obecnie można już w Polsce spotkać organizacje, dla których możliwym byłoby zaimplementowanie projektowanego systemu. Przejdźmy więc do omówienia szczegółowych założeń.

Celem prac jest zbudowanie systemu wspomagającego te decyzje, których model opisujący sytuację (strukturę) jest wielowymiarowy. Oznacza to, że liczba czynników (parametrów) sytuacji decyzyjnej jest znacznie większa od możliwej do jednoczesnego ujęcia i równoczesnego porównania w umyśle człowieka. Wspomaganie obejmować będzie wszystkie etapy wyszczególnione w analizie decyzyjnej tzn. proces formułowania problemu (definiowanie struktury problemu) i kryteriów wyboru oraz sugerowanie najkorzystniejszego wariantu w warunkach określonych przez użytkownika i zgodnie z zawartością bazy wiedzy. Zarówno model sytuacji decyzyjnej, jak i ranga poszczególnych zmiennych decyzyjnych (wyrażonych w różnych, nie zawsze porównywalnych jednostkach), zakres ich zmienności i najkorzystniejsze wartości, a także zawartość bazy wiedzy będą w dużym stopniu zależne od użytkownika i rozwiązywanych przez niego problemów. Z punktu widzenia użytkownika oznacza to, że system w analizie decyzyjnej uwzględniać będzie subiektywne gusty, wartościowania i przekonania. Każdy bowiem decydent ma idealny punkt swoich upodobań, od którego odstępstwa, w jednym lub w drugim kierunku, są odczuwane jako mniej atrakcyjne niż sam punkt idealny, a równocześnie zarówno punkt idealny, jak i wszystkie oceniane atrybuty nie są stałe i podlegają fluktuacjom [6].

Ponadto przewiduje się, że system powinien:
- uwzględniać naturalny sposób opisu sytuacji, tj. wykorzystywać nieprecyzyjne, niejednoznaczne wartości atrybutów sytuacji decyzyjnej,
- obejmować proces formułowania problemu, kształtowania ocen poszczególnych atrybutów sytuacji decyzyjnej, ich integrowania w alternatywy i porównywania globalnych ocen wygenerowanych alternatyw [4],
- uwzględniać problemy złożone, w których decydent nie potrafi całościowo porównać alternatyw,
- uwzględniając specyfikę ekstremalnych wartości atrybutów, wspomagać decydenta w dokonywaniu świadomych przeliczeń i porównań w miejsce emocjonalnych wyborów,

- mieć zastosowanie w sytuacjach, gdy cele możliwych wariantów pozostają w konflikcie (konieczność kompromisu).

4.2 Przewidywany przebieg współpracy z systemem

Inspirację do opracowania koncepcji systemu stanowiły powyższe rozważania oraz zbudowany na ich podstawie model wyobrażeniowy. Za najdogodniejszą formę prezentacji przemyśleń uznano zamieszczenie zakładanego scenariusza analizy z charakterystyką poszczególnych etapów.

A. Skonfigurowanie systemu użytkownikowi

opis:

odbiorcą systemu jest konkretny użytkownik, który zanim rozpocznie współpracę merytoryczną z systemem musi przygotować słownik do interpretowania wykorzystywanych przez niego wartości nieprecyzyjnych (takich jak: *ważny, mały, kilkanaście, około* itp.), proces ten polega na określeniu subiektywnego stopnia przynależności obiektu do analizowanej wartości nieprecyzyjnej,

zaangażowanie użytkownika:

określenie funkcji odpowiadających wartościom nieostrym po to, aby możliwe było wykonywanie na nich operacji porównywania i/lub arytmetycznych,

rola wspomagania:

utworzenie, przechowywanie i aktualizowanie zbioru pojęć nieprecyzyjnych wykorzystywanych przez użytkownika wraz z przypisanymi im funkcjami przynależności[5].

B. Sprecyzowanie celu analizy decyzyjnej

opis:

celem analizy może być wybór ze skończonej listy alternatyw lub wygenerowanie najkorzystniejszego wariantu. Zarówno w jednym jak i w drugim przypadku decydent musi określić funkcję celu, która stanowić będzie kryterium oceny i selekcji wariantów,

zaangażowanie użytkownika:

sformułowanie kryterium w kategoriach operacyjnych, zarejestrowanie tekstu kryterium, wskazanie rodzaju problemu (skończona czy otwarta lista wariantów),

rola wspomagania:

zarejestrowanie wprowadzonego przez użytkownika kryterium, udostępnianie na żądanie opisów sytuacji decyzyjnych zawartych w bazie wiedzy; przewiduje się wykorzystać m.in. następujące techniki doradcze: sieci repertuarowe i hierarchizację struktury celu.

C. Zestawienie listy zmiennych decyzyjnych

opis:

etap ten polega na tworzeniu subiektywnej reprezentacji zadania; ma na celu zbudowanie modelu opisującego rozpatrywaną sytuację (określoną w punkcie A), zmienne decyzyjne (atrybuty sytuacji) mogą kształtować sytuacje z różną siłą, wyrażoną w subiektywnej ocenie użytkownika,

zaangażowanie użytkownika:

podaje kolejno nazwy zmiennych, które jego zdaniem kształtują sytuację, własne oceny siły oddziaływania poszczególnych atrybutów na sytuację oraz podaje jednostki lub skale, w których prezentowane i analizowane będą wartości atrybutów,

rola wspomagania:

- podczas wprowadzania - obsługa dialogu; do wspomagania procesu budowania struktury alternatyw zastosowana zostanie technika zwana drzewem decyzyjnym,
- po zakończeniu wprowadzania - analiza listy atrybutów użytkownika w kontekście modeli zgromadzonych w bazie i ewentualnie sugero-

[5] *funkcja przynależności* - pojęcie z zakresu teorii zbiorów rozmytych, określa stopień (siłę) przynależności elementu do zbioru rozmytego.

wanie zmodyfikowania listy, wyświetlenie użytkownikowi jego ocen oraz ocen ekspertów (i wniosków z poprzednich analiz) zawartych w bazie wiedzy w celu ewentualnej modyfikacji,

D. Generowanie alternatywnych rozwiązań:

opis:

etap szczególnie ważny w przypadku problemu, dla którego istnieją więcej niż dwie możliwości działania i możliwości te nie są znane a'priori; użytkownik każdorazowo musi rozstrzygnąć, czy warto ponosić dodatkowe koszty w celu rozszerzenia zbioru alternatywnych kierunków działania,

zaangażowanie użytkownika:

zarejestrowane zostają tylko te alternatywy, które warto poddać dokładniejszej ocenie w celu wyboru jednej z nich,

rola wspomagania:

wiedza o twórczych procesach poznawczych jest nadal dość uboga (jak wynika ze znanych rozważań teoretycznych); możliwą do zastosowania techniką wspomagającą jest udostępnienie zgromadzonych w bazie danych systemu struktur przeanalizowanych dotąd problemów po to, aby stymulować wytwarzanie alternatyw.

E. Określenie wartości granicznych poszczególnych zmiennych decyzyjnych

opis:

użytkownik określi w odniesieniu do każdej zmiennej nieprzekraczalną i/lub dopuszczalną wartość, jaką może ona przyjąć, wartości te nie muszą być podane w sposób jawny, mogą to być wartości funkcji operujących na danych zgromadzonych z bazie danych obiektu (np.: najmniejsze zatrudnienie w ostatnich trzech miesiącach),

zaangażowanie użytkownika:

redagowanie danych,

rola wspomagania:

nadzorowanie użytkownika podczas wprowadzania, bieżące konfrontowanie z danymi zawartymi w bazie wiedzy.

F. Nadanie wag poszczególnym zmiennym decyzyjnym

opis:

ma na celu uporządkowanie atrybutów w kolejności wynikającej z preferencji użytkownika,

zaangażowanie użytkownika:

decydent określa siłę, z jaką, jego zdaniem kolejne zmienne kształtują sytuację decyzyjną,

rola wspomagania:

bieżąca analiza i porównywanie z danymi zawartymi w bazie wiedzy odzwierciedlającymi poprzednie doświadczenia, z ewentualnymi ocenami ekspertów czy z wartościami wyznaczonymi ściśle.

G. Sprawdzanie założeń wybranego modelu

opis:

niezależność atrybutów modelu sprawdzana będzie w drodze dialogu z użytkownikiem, który odpowiadając na pytania zestawi listę niezależnych atrybutów; automatyczne wykonanie testów pozwoli zweryfikować inne warunki zgodnie z typem modelu, niespełnianie założeń powoduje konieczność zredefiniowania przyjętego celu (powrót do punktu B) i/lub zmodyfikowania listy zmiennych decyzyjnych (punkt C) i/lub zmodyfikowania albo wygenerowania nowych wariantów działania (punkt D) i/lub zmodyfikowania wag nadanych zmiennym decyzyjnym (punkt F),

zaangażowanie użytkownika:

procedura dialogowego weryfikowania założeń jest nie tyle trudną, co czasochłonną, wymaga od użytkownika ustalenia w pierwszym kroku hierarchii wariantów, wynikającej ze zmiany wartości rozpatrywanego atrybutu (od wartości najgorszej do najlepszej), przy

stałych wartościach pozostałych atrybutów, odpowiedzenia na pytanie: *czy zmiana wartości któregoś z pozostałych atrybutów wpłynie na kolejność wariantów wyznaczoną jak wyżej - jeśli nie to analizowany atrybut jest preferencyjnie niezależny od pozostałych*; procedura ta powtarzana jest dla wszystkich atrybutów.

H. Uporządkowanie i selekcja alternatyw

opis:

automatyczne uporządkowanie zgodnie z preferencjami użytkownika, następnie odrzucenie tych alternatyw, które są nierealne, nie zmierzają do wyznaczonego celu, nie uwzględniają ograniczeń lub w inny sposób nie odpowiadają oczekiwaniom,

rola wspomagania:

umożliwienie przeglądania wyników ostatecznych a także pośrednich w celu odtworzenia toku, w wyniku którego otrzymano te rezultaty.

Powyższa procedura (punkty B - H) powtarzana będzie do wyłonienia wariantu spełniającego założenia i ograniczenia, kilku równoważnych wariantów lub wykazania braku rozwiązań.

Warto na koniec zauważyć, że wyłoniony wariant działania zostanie zasugerowany z mocą wynikającą z subiektywizmu opisu wariantów; tak więc, jest prawdopodobne, że ocena szans i ryzyka dokonana przez inną osobę może doprowadzić do innych rezultatów.

5. Zakończenie

Projektowane wspomaganie analizy decyzyjnej powinno poprzez zmniejszenie pracochłonności procesu generowania i dokonywania oceny wariantów umożliwić decydentowi dokonanie samooceny. Zredukowanie niejasności i niepewności związanej z celem czyli strukturalizacją problemu, pozwoli na zbudowanie zhierarchizowanej struktury celu zwiększającej zdolność do przetwarzania informacji co pomoże w jaśniejszym ustalaniu oczekiwań. Różne konsekwencje decyzji oraz preferencje do tych konsekwencji zostaną zestawione w sposób systematyczny. Powinno to wpłynąć na stopień precyzji oceny alternatyw dla poszczególnych atrybutów a tym samym na uświadomienie sobie struktury własnych preferencji.

6. Bibliografia

[1] Ansoff H. J.: *Implanting Strategic Management*, Prentice - Hall International, London 1984.

[2] Ansoff H. J.: *Zarządzanie strategiczne*, PWE, Warszawa 1985.

[3] Fishburn P. G.: *Utility theory for decision making*, Wiley, New York 1970.

[4] Katzan H. Jr.: *Management support systems. A pragmatic approach*, Van Nostrand Reinhold Company, New York 1984.

[5] Kozielecki J.: *Psychologiczna teoria decyzji*, PWN, Warszawa 1977.

[6] Lindsay P. H., Norman D. A.: *Procesy przetwarzania informacji u człowieka, wprowadzenie do psychologii*, PWN, Warszawa 1984,

[7] Tyszka T.: *Analiza decyzyjna i psychologia decyzji*, PWN, Warszawa 1986.

[8] Szaniawski K.: *Kryteria podejmowania decyzji*, w: Kozielecki J. (red.): *Problemy psychologii matematycznej*, PWN, Warszawa 1971.

[9] Wołoszyn J.: *Grafy rozmyte i możliwości ich wykorzystania w ekonomii*, Zeszyty Naukowe, Akademia Ekonomiczna w Krakowie, 1990.

Wojciech M. Jaworski
Computer Sc. Dept., Concordia University, Montreal

Andrzej Zaliwski, Marian Kuraś
Katedra Informatyki Akademii Ekonomicznej w Krakowie

INFOMAPY - inna metda gromadzenia wiedzy

Nie ma nic bardziej praktycznego
niż dobra teoria.
(Pan Edek powołujący się na W.I.L.
a ostatnio na A. Einsteina)

1. Wprowadzenie

Głównym problemem na jaki napotyka każdy, kto zajmuje się projektowaniem lub konserwacją wielkich systemów oprogramowania, jest ich złożoność. Systemy oprogramowania znacznie różnią się od innych systemów. Projektant musi operować na wielu różnych poziomach abstrakcji oraz zajmować się wieloma szczegółami. System oprogramowania - o czym należy pamiętać - przyjmuje złożoność swoich aplikacji. Oprócz tego sam proces wytwarzania oprogramowania jest szczególnie złożony. Typowymi produktami towarzyszącymi procesowi tworzenia oprogramowania są raporty, podręczniki, diagramy, teksty źródłowe itp. Do sporządzenia każdego z tych dokumentów są używane inne techniki notacyjne. Nierzadko do utworzenia pojedynczego dokumentu używa się kilku metod. W takiej sytuacji występuje silna zależność i pokrywanie się informacji w poszczególnych dokumentach, będących produktami kolejnych faz procesu rozwoju oprogramowania. Jest to nie do uniknięcia, zwłaszcza, że wspomniane dokumenty są rezultatem stosowania różnych podejść, metod, technik reprezentacji wizualnej, konwencji opisu, języków, narzędzi wspomagania itp. W związku z tym dokumenty związane z procesem rozwoju oprogramowania są fragmentarycznymi opisami i częściowo wzajemnie nakładającymi się na siebie obrazami systemu oprogramowania. Do dokumentowania SI stosowane są różne techniki diagramowania w celu prezentacji różnych aspektów funkcjonowania systemów (np. DFD - Data Flow Diagrams, ERD - Entity-Relationship Diagrams, STD - State-Transition Diagrams, Structure Charts, Jackson Data Structure Diagrams, Nassi-Shneiderman Diagrams). Każdy z diagramów może opisywać ten sam system i prezentuje jego komponenty. Żaden z nich nie reprezentuje całego systemu - daje tylko pewien specyficzny obraz, przekrój systemu. W konsekwencji modyfikacja jakiegoś komponentu musi spowodować odpowiednie zmiany w innych reprezentacjach. Takie pochodne zmiany nie zawsze są od razu jasne i czytelne. Utrzymanie spójności takiej dokumentacji wymaga znaczących nakładów pracy.

Tworzenie i rozwój oprogramowania możemy potraktować jako proces pozyskiwania wiedzy. Na określonym etapie tworzymy bazę wiedzy zawierającą informacje o tworzonym systemie. Rozwój następuje w wyniku pozyskiwania dalszych informacji i ich włączania do bazy wiedzy lub jako efekt przetwarzania informacji

zgromadzonej wcześniej. Przechowywana wiedza jest zróżnicowana w zależności od fazy projektu. Występuje na różnych poziomach szczegółowości począwszy od założeń programu a kończąc na kodzie. Rzadko zdarza się by powiązania między tymi poziomami były formalne lub kompletne. Przed techniką rozwoju oprogramowania stawia się następujące wymagania:

- formalnej notacji,
- technologii, która wspomaga i jest wspomagana przez notację,
- metod reprezentacji obiektów i wiedzy w obrębie danej dziedziny,
- reguł odwzorowania modeli pomiędzy różnymi konwencjami notacji.

Celem niniejszej publikacji jest przedstawienie techniki notacyjnej spełniającej powyższe wymagania, zatem mogącej stanowić podstawę rozwoju oprogramowania. Technika ta nosi nazwę: **'infoMapy'**. Jej możliwości pozwalają na opisywanie systemu oprogramowania na wielu poziomach abstrakcji oraz na uniknięcie trudności związanych z translacją i konwersją informacji projektowych, gdyż ta sama konwencja notacji może być użyta we wszystkich fazach procesu produkcji oprogramowania. Ponadto, notacja **infoMap** umożliwia modelowanie aplikacji i ich systemu oprogramowania na wszystkich poziomach rozwoju, począwszy od jego założeń i specyfikacji a kończąc na kodzie źródłowym programu.

2. infoMAPy jako technika notacji

W celu rozwinięcia formalizmu, który może być użyty dla szerokiego zakresu aplikacji na różnych poziomach abstrakcji, potrzebujemy podstawowych struktur i technik do manipulowania nimi. Notacyjnie technologia infoMap jest oparta na podstawowych pojęciach matematyki: zbiorach i relacjach. W sensie matematycznym zbiór jest kolekcją obiektów spełniających pewne warunki lub mających tę samą charakterystykę. Standardowymi strukturami matematyki dyskretnej są zbiory, funkcje, relacje i grafy. Relacje są szeroko używane w teorii baz danych, podczas gdy grafy są używane dla tworzenia narzędzi do rozwoju oprogramowania [15].

Relacja jest zdefiniowana jako odwzorowanie produktu kartezjańskiego pewnej liczby zbiorów na zbiór {0, 1}. Innymi słowy, relacja między zbiorami istnieje lub nie. Taka definicja relacji jest niewłaściwa dla wielu zastosowań deskryptywnych. W rzeczywistości bowiem relacja może także przenosić znaczenie. Przykładowo relacja "matka", matki do jej dziecka nie tylko istnieje ale także zawiera rodzaj zależności. Definicja relacji powinna więc zostać uogólniona jako odwzorowanie produktu kartezjańskiego pewnej liczby zbiorów na zbiór symboli, gdzie każdy symbol definiuje jednoznacznie rodzaj relacji. InfoMapa jest reprezentacją tak uogólnionej relacji.

Notacja infoMap [6]-[12] łączy wizualną prostotę grafu z gęstością informacyjną tablicy. Z dziedziny baz danych została zapożyczona koncepcja słownika danych. Reprezentacja wizualna została zapożyczona z dziedziny tablic decyzyjnych. Powstaje w ten sposób nowa, macierzopodobna reprezentacja o dużej pojemności informacyjnej.

Przeanalizujmy prosty przykład. InfoMapa z rysunku 1 przedstawia plan wykładu na temat infoMap. InfoMapa ta zawiera następujące informacje dotyczące:

- zastosowania przekroju,
- struktury wykładu,
- rysunków.

W jMapie wydzielone są dwie zasadnicze części:

- obszar definicji zbiorów
- obszar definicji relacji.

Kolumny 15 i 16 odpowiadają obszarowi definicji zbiorów, kolumny od 1 do 13 obszarowi definicji relacji.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1																	
2		A	A	A	A	A	A	A	A	A	A	A	A	A		{ZASTOSOWANIE PRZEKROJU}	
3		v														Instytucja	
4			v													Wykładowca	
5				v	v	v										Struktura wykładu	
6							v	v	v	v	v	v	v			Użycie rysunków	
7														v		Liczebność	
8		M	M	H	H	H	o	o	o	o	o	o	o	16		{CZĘŚCI WYKŁADU}	
9		v														**Cracow School of Economics**	
10		v														**Canadian -Polish Management Centre**	
11			v													Wojciech M. Jaworski, Concordia University, Montreal	
12				h			o									***infoMaps: a System for Knowledge Transfer Automation***	
13				1	h											Wprowadzenie	
14					1			o								Efektywność transferu wiedzy	
15					2				o							Dokladność transferu wiedzy	
16				2		h										Dlaczego "Automatyzacja transferu wiedzy"?	
17						1										Człowiek-, Komputer-, Wiedza "wykonawcza"	
18						2										Proces pozyskiwania wiedzy i kontrola jakości	
19						3										Zastosowanie transferu wiedzy	
20						4										Użycie wiedzy - Operator, Komputer	
21				3						o						Funkcjonalność "infoFarm"	
22				4							o					Produkcja oprogramowania - bardzo prosty przykład	
23				5								o				Składniki bazy wiedzy "infoFarm"	
24				6									o			Odzyskiwanie wiedzy	
25							M	M	M	M	M	M	M	8		{RYSUNKI}	
26							v									Model wykładu1	
27								v								Fig. A. Tryby transferu wiedzy	
28									v							Fig. B. "Captive Mind (wise joke)"	
29										v						Fig. 1. infoMap - przykład oceny studentów	
30										v						Fig. 2 .. 7.4 - "infoFarm"	
31											v					Program przeszukiwania binarnego: infoMapy 1 .. 12	
32												v				Przestrzeń Systemu: infoMapy 13 .. 17	
33													v			Przykład: MIS : infoMapa 18	
34																	
35		**Rys. 1. InfoMapa zawierająca plan wykładu n/t infoMAP**															
36																	

Nazwy zbiorów i ich elementy są zdefiniowane w obszarze definicji zbiorów. Relacje między zbiorami oraz relacje między elementami zbiorów są zdefiniowane w obszarze definicji relacji.

W obszarze definicji zbiorów, nazwy zbiorów są zestawione w tej samej kolumnie, ujęte w nawiasy klamrowe i napisane dużymi literami. Zdefiniowane w przykładzie z rysunku 1 nazwy zbiorów to: {ZASTOSOWANIE PRZEKROJU}, {CZĘŚCI WYKŁADU} oraz {RYSUNKI}. Elementy wchodzące w skład tych zbiorów są wymienione odpowiednio poniżej nazw tych zbiorów. Każda kolumna jMapy reprezentuje relację. Relacje są niepowtarzalne, z czego wynika, że nie powinno być dwóch identycznych kolumn. Dzięki temu unikamy redundancji danych. Każda komórka w obszarze definicji relacji odpowiadająca wierszowi elementu zbioru uczestniczy w danej relacji.

Uczestniczenie zbioru lub jego elementu w danej relacji jest oznaczane przez umieszczenie odpowiedniego elementu notacji (symbolu specjalnego) na skrzyżowaniu kolumny relacji z wierszem, w którym jest nazwa zbioru (lub nazwa elementu zbioru). Pusta komórka w jakimś miejscu kolumny relacji oznacza, że zbiór

lub jego element w odpowiadającym wierszu nie uczestniczy w tej relacji. Symbole specjalne są podzielone na dwie grupy *'Rola zbiorów'* oraz *'Rola elementów zbioru'*. Stosowane oznaczenia oraz zależności pomiędzy tymi grupami (infoSYNTAX) zostały zamieszczone na rysunku 10. Należy w tym miejscu zaznaczyć, że składnia infoMap jest otwarta. Użytkownik ma całkowitą swobodę w definiowaniu lub redefiniowaniu przyjętych oznaczeń, znaczenia obiektów infoMapy a także dopuszczalnych operacji na tych obiektach.

W jMapie istnieją dwa różne poziomy relacji. Pierwszy poziom specyfikuje relacje między zbiorami. Drugi poziom, niższy, relacje między elementami zbiorów. Pierwszy ze wspomnianych poziomów nosi nazwę **infoSCHEMy** *(infoFRAME)*. InfoSCHEMA jest modelową strukturą dla reprezentowania określonej dziedziny wiedzy. InfoSCHEMA stanowi ramową strukturę, która jest stopniowo wypełniana i uszczegóławiana aż do otrzymania infoMapy. Stąd infoMapy mogą być traktowane jako implementacja infoSCHEMy. Generalnie rzecz biorąc z jednej infoSCHEMy można otrzymać wiele infoMap. Podobnie jak w przypadku programów: na podstawie jednej specyfikacji można zakodować wiele programów. Przejście od infoSCHEMy do infoMapy wymaga udziału człowieka, podczas gdy operacja odwrotna może być przeprowadzana automatycznie.

Na rysunku 2 jest przedstawiona infoSCHEMA dla przykładu z poprzedniego rysunku. W przypadku interpretowania relacji w infoMapie należy najpierw analizować relacje między zbiorami (czyli relacje przedstawione na infoSCHEMie) a dopiero potem relacje między elementami zbiorów. W naszym przykładzie relacje zawartą w kolumnie 7 możemy interpretować w sposób następujący: *'Jeden (One) element zbioru {CZĘŚCI WYKŁADU} jest powiązany z wieloma (Many) elementami zbioru* {RYSUNKI}. Oznacza to, że między tymi zbiorami istnieje relacja typu 1:M. Ponadto czytamy z rysunku, że relacja w kolumnie 7 jest fragmentem relacji zagregowanej (Aggregation) wskazującej na autora tej kolumny/relacji wymienionego w zbiorze {AUTHOR}. Stopniowo, przez uzupełnianie infoSCHEMY o relacje między elementami zbiorów otrzymujemy infoMapę taką jak na rysunku 1.

Ważnym elementem każdej infoMapy jest specjalny zbiór o nazwie {VIEW} lub {VIEW's PURPOSE} (W wypadku rysunku 1 i 2 jest to {ZASTOSOWANIE PRZEKROJU}). Każda relacja, stanowiąca przecież określony przekrój dziedziny, jest opisana przez komentarze zawarte w tym zbiorze. Przykładowo na rysunku 1, w wierszu *'Struktura wykładu'* znajdują się trzy litery 'v' w kolumnach 4,5,6 oznacza to, że te kolumny opisują strukturę wykładu. Oprócz tego istnieje specjalny przekrój oznaczony jako *'cardinality'* (liczebność). Związana z nim kolumna zawiera informacje o liczebności zbiorów. Informacje te są przydatne przy weryfikacji mapy.

	1	2	3	4	7	14	15	16
1								
2		A	A	A	A	A		{ZASTOSOWANIE PRZEKROJU}
8		M	M	H	O	16		{CZĘŚCI WYKŁADU}
25					M	8		{RYSUNKI}
34								
35		**Rys. 2. InfoSCHEMA dla rysunku 1.**						
36								

Kolejny przykład - infoMapa przedstawiona na rysunku 3 zawiera zestawienie ocen dotyczących sytuacji w Polsce w 1992 roku, uzyskanych od różnych osób. Czytelnikom proponujemy samodzielne odczytanie informacji zawartych w tej infoMapie.

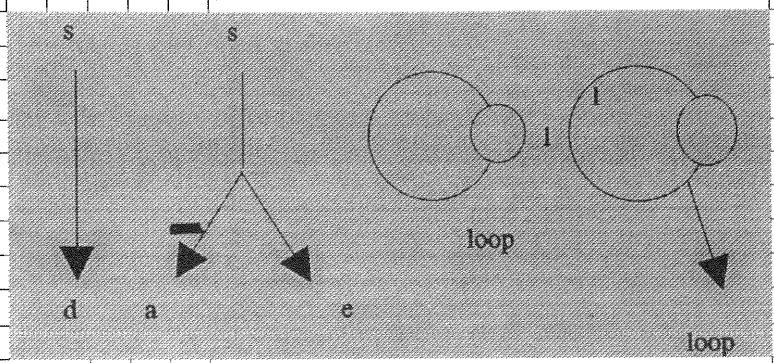
Jak już mieliśmy możliwość zauważyć, notacja infoMap jest notacją bardzo pojepojemną i elastyczną. W celu pokazania walorów tej techniki notacyjnej oraz przekonania, iż jest rzeczywiście notacją uniwersalną, mogącą opisywać każdy aspekt SI, w dalszej części niniejszej publikacji prezentujemy kilka przykładów. Większość z nich dotyczy samego programowania a jeden dziedziny od programowania bardzo odległej.

	1	2	3	4	5	6	7	8	9
1									
2									
3		A	A	A	A	A	A	{PUNKT WIDZENIA}	
4		v	v		v				Polska widziana oczyma Polakow z Zachodu
5				v					Polska widziana oczyma Polakow w Polsce
6		v	v	v	v	v	v		Skala ocen; 1=wyjatkowo dobrze, 2=zupelnie dobrze, 3=troche dobrze, 4=neutralna, 5=troche zle, 6=zupelnie zle , 7=wyjatkowo zle
8		O	O	O	O	O	O	{OPINIODAWCA}	
9		o							A. Targowski (na Zachodzie od 1980 r.)
10			o						J. Nowak Jezioranski (na Zachodzie od II Wojny Swiatowej)
11				o					S. Lem (w kraju)
12					o				W. Jaworski (na Zachodzie od 1968 r.)
13						o			Marian Kuras (roczne stypendium w Kanadzie 1991-92)
14							o		Andrzej Zaliwski (w kraju)
15		V	V	V	V	V	V	{DZIEDZINA}	
16		6	3	6	6	5	5		Sytuacja polityczna
17		7	2	7	7	7	7		Strategia polityczna
18		6	2	6	6	4	5		Sytuacja gospodarcza
19		7	?	7	7	6	6		Strategia gospodarcza
20		?	1	?	5	4	4		Przywodcy/Dzialacze demokratyczni do 1989 r
21		6	5	6	6	6	5		Obecna kadra polityczna/politycy
22		6	?	5	6	6	7		Kultura polityczna
23		6	?	7	7	6	6		Wiedza polskiego spoleczenstwa o strategii gospodarczej
24		?	?	?	6	7	6		Wiedza polskiego spoleczenstwa o pozycji Polski w swiecie
25		?	?	?	6	3	4		Prognoza powaznych inwestycji/pozyczek zagranicznych
26									
27									
28		A	A	A	A	A	A	{ZRODLO}	
29		v	v	v	v	v	v		K.M. Ford and all. ,"An Approach to Knowledge Acquisition Based on the Structure of Personal Construct Systems", IEEE Trans. on Knowledge and Data Eng., March 1991 pp 78-88
30		v	v	v	v	v	v		Z. Pawlak, "Rough Sets: Theoretical Aspects of Reasoning about Data", De Kuyper, Amsterdam, 1992
31		v							A. Targowski, "Rewolucja zjada wlasne dzieci", Studium Newsletter, January 1992
32			v						J. Nowak Jezioranski, "W odpowiedzi Andrzejowi Targowskiemu", Studium Newsletter, February/March 1992
33				v					S. Lem, "Swiat i Polska", Kultura, Paryz, Nr 5/536, 1992
34									
35									
104		**Rys. 3. Ocena sytuacji w Polsce w 1992 r. przedstawiona w postaci infoMAPY.**							
105					*Copyright © W.M. Jaworski 1992*				

3. Zastosowania w programowaniu

W celu ilustracji możliwości opisu przy zastosowaniu konwencji infoMap wybraliśmy sposób reprezentacji najbardziej elementarnych struktur sterujących i przykładów programów.

Właściwości elementarnych struktur sterowania są dyskutowane w wielu publikacjach (np. [2], [1], [3], [16]). Graf przepływu sterowania tych struktur sterujących może być reprezentowany przez cztery elementarne grafy oraz ich złożenie. Te podstawowe konstrukcje są określane jako: *'Sekwencja'*, *'Sekwencja z asercją* lub *wyjątkiem (exception)'*, *'Pętla'*, *'pętla z wyjściem'*. Rysunek 4a przedstawia te cztery elementarne konstrukcje a rysunek 4b sposób ich przedstawienia w postaci infoMap. Nazwy tych czterech podstawowych konstrukcji zostały wymienione jako elementy zbioru {GENERIC GRAPH}. Przecięcie wiersza, w którym jest wymieniona nazwa konstrukcji z kolumną opisującą tę konstrukcję jest oznaczone literą *'o'*. Litera *'s'* została użyta do określenia źródła (*source*) a litera *'d'* jako oznaczenie stanu docelowego (*destination*). Symbol *'l'* oznacza pętlę (*loop*), *'a'* - asercję (*assertion*), zaś *'e'* - wyjątek (*exception*).



branch					branch with fork

(a) Reprezentacja graficzna

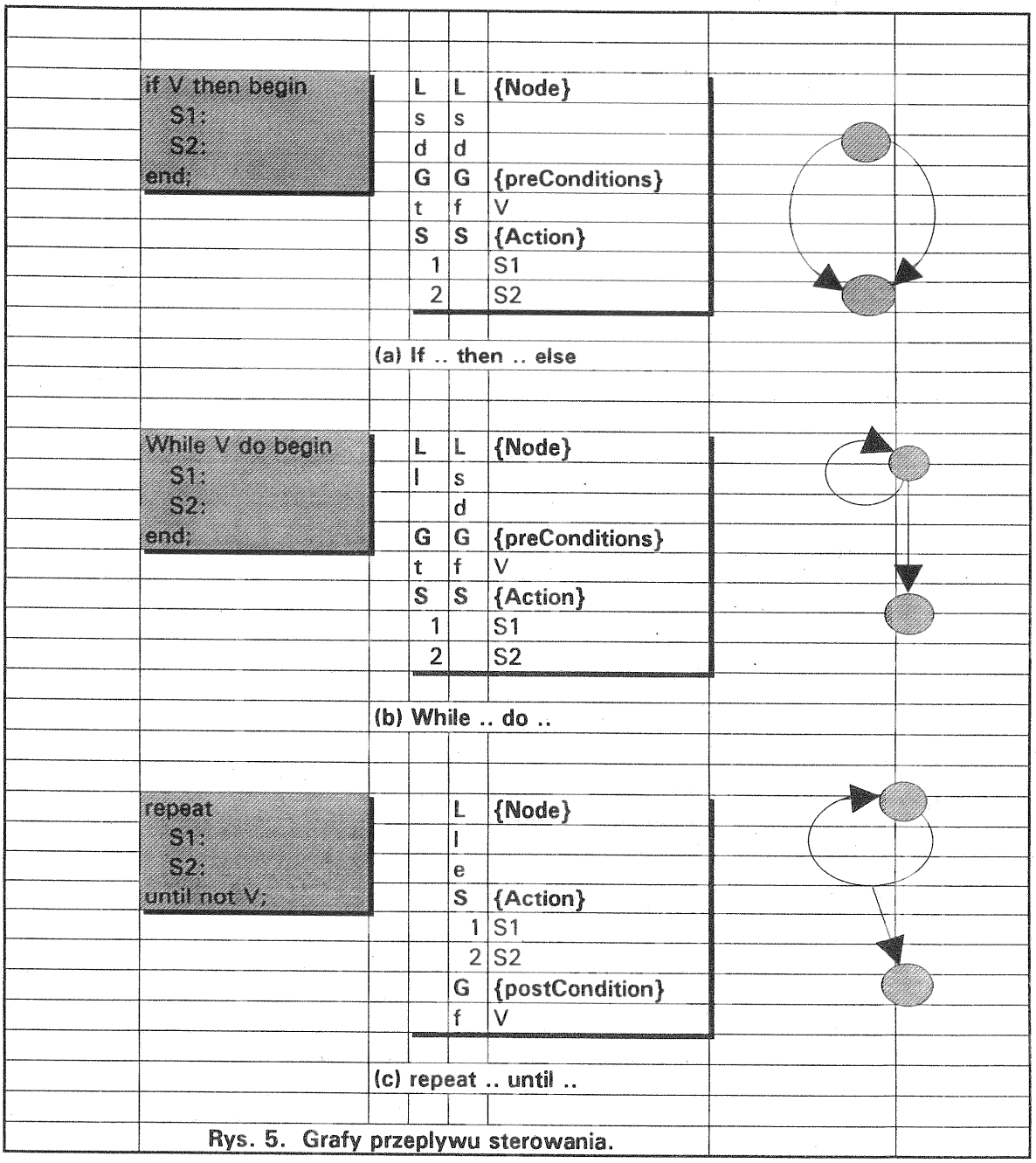
O	O	O	O		{GENERIC GRAPH}
o					branch from s to d
	o				branch from s to either a or e
		o			loop
			o		loop with exit to e
L	L	L	L		{NODE/STATE}
s					start
d					destination
	s				start
	a				asserted destination
	e				exception destination
		l			loop node
			l		loop node
			e		exception destination
A	A	A	A		{AUTHOR}
v	v	v	v		InfoSCHEMA Copyright © 1991 W.M. Jaworski
v	v	v	v		InfoMAP Copyright © 1991 W.M.Jaworski

(b) Reprezentacja w postaci infoMAPY

Rys. 4. Cztery podstawowe grafy przepływu sterowania.

Na rysunku 5 z lewej strony pokazano w postaci kodu strukturalnego zbliżonego do języka Pascal niektóre konstrukcje stosowane w programach. Środkowa część rysunku przedstawia te same konstrukcje w postaci infoMap. Zwróćmy uwagę na zbiór {PRECONDITION} na rysunku 5a. Jeśli wymieniony w nim warunek *'V'* jest spełniony (*true*), zachodzi sytuacja przedstawiona w pierwszej kolumnie. Jest wykonywana sekwencja akcji *S1* i *S2*. Litera *'S'* sygnalizuje operacje sekwencyjne. Cyfry *'1'* i *'2'* pod literą *'S'* określają wówczas kolejność wykonywania operacji. Jeśli warunek *'V'* nie jest spełniony (*false*), to mamy sytuację opisaną w drugiej kolumnie. Tak jak w poprzednim przypadku również następuje przejście z jednego stanu do drugiego ale akcje *S1* i *S2* nie mają miejsca.

Na rysunku 6 pokazano kod strukturalny dla programu przeszukiwania binarnego (a) oraz odpowiadającą mu reprezentację w postaci infoMapy (b). Pozycje *'O'*, *'L'*, *'G'*, *'S'* i *'G'* określają role jakie zostały przydzielone odpowiednim zbiorom tj. {*Transition*}, {*State*}, {*preCondition*}, {*Action*} oraz {*postCondition*}. Znaczenie poszczególnych pozycji jest następujące:



Rys. 5. Grafy przeplywu sterowania.

```
1: found:=false
2: while( first <= last)
    and  not  found do begin
              i := ( first + last ) div 2;
3:            if a[i]<X then first := i+1
4:            else if a[i]>X then last:=i-1
                    else found := true;
    end;
5: if a[i] = X then
            X found at i
    else X not found;
6:
```

(a) kod strukturalny

1	2	3	4	5	6	7	8	9	10	11	12	
O	O	O	O	O	O	O	O	O	9	{Transition}		
L	L	L	L	L	L	L	L	L	6	{State}		
s											1:	
d	s	s	d		d	d					2:	
	d		s	s							3:	
				d	s	s					4:	
							s	s			5:	
							d	d			6:	
G	G	G	G	G	G	G	G	G	4	{preCondition}		
	t	f									first<=last and not found	
			t	f							a[i] < X	
					t	f					a[i] > X	
							t	f			a[i] = X	
S	S	S	S	S	S	S	S	S	7	{Action}		
1											found := false	
	1										i:=(first + last) div 2	
			1								first:=i+1	
					1						last:=i-1	
						1					found := true	
							1				X found at i	
								1			X not found	

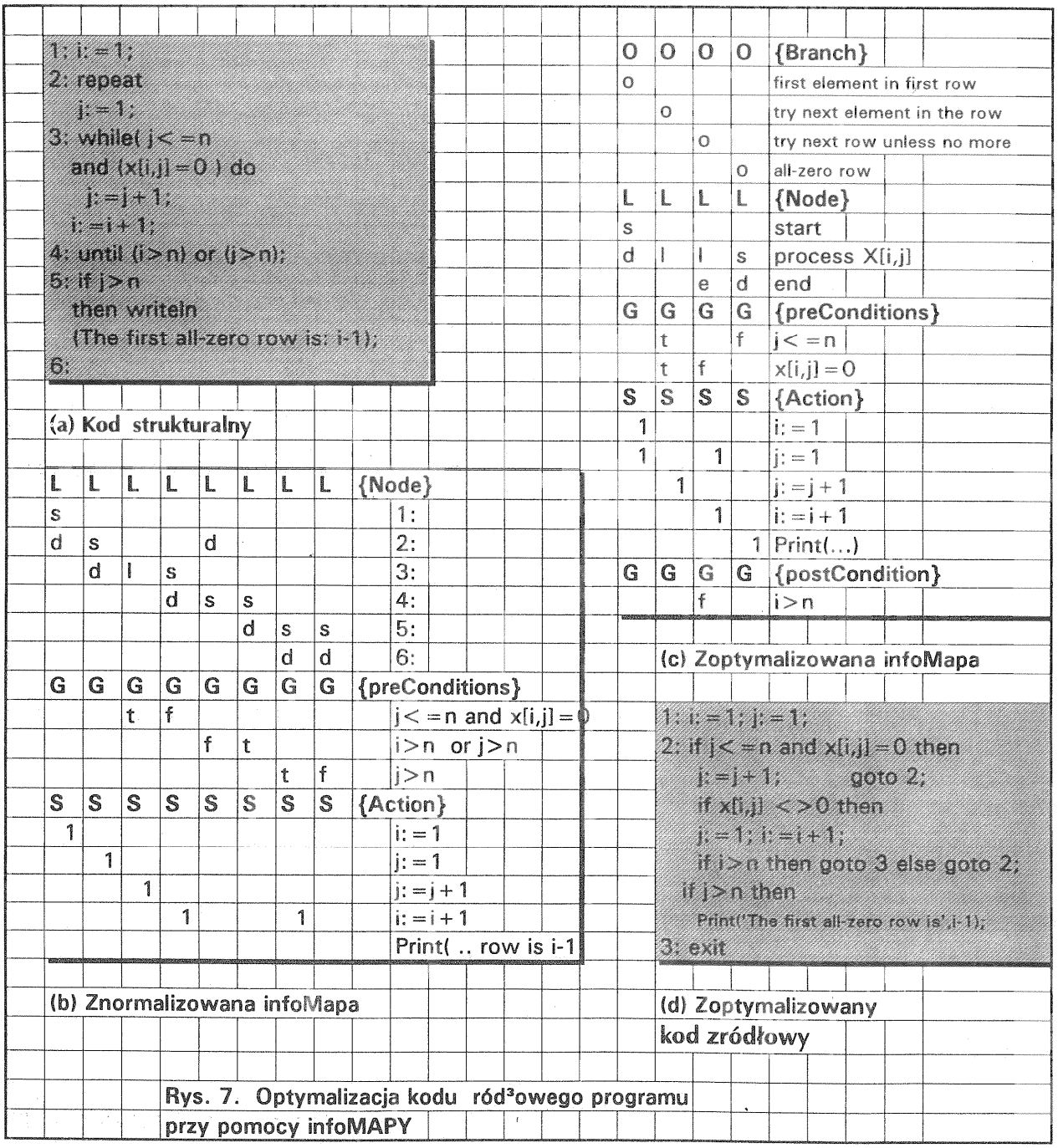
(b) Kod z rysunku (a) zapisany jako infoMAPA

Rys. 6. Algorytm wyszukiwania binarnego.

Każde (*each*) przejście (*transition*) łączy (*Links*) stany (*States*). Każde przejście jest zaimplementowane jako sekwencja (*Sequence*) akcji kontrolowanych (*Guarded*) przez warunki wstępne (*preconditions*) oraz kontrolowana (asserted) przez warunki końcowe (*postconditions*).

Litera *'A'* pozwala użytkownikowi na zaznaczenie faktu, że kilka kolumn jest zgrupowane w relację poziomą. Litera *'S'* oznacza relację pionową i sekwencyjne uporządkowanie elementów zbioru.

Poniżej zamieszczamy przykład oparty na następującym zadaniu, które sformułował Dijkstra w [4]: "Niech X będzie macierzą NxN zawierającą liczby całkowite. Należy napisać program który wyświetla numer



Rys. 7. Optymalizacja kodu ród³owego programu przy pomocy infoMAPY

pierwszego zerowego wiersza macierzy X, o ile taki istnieje."

Rozwiązanie tego problemu pochodzące z [15] i jego translację na infoMapę przedstawia rysunek 7 (a) i (b). Przez przetworzenie wierszy i kolumn infoMapy otrzymano zoptymalizowaną infoMapę (c) oraz zoptymalizowany kod źródłowy (d). Inne rozwiązanie tego samego problemu [13] oraz rozwiązania o nazwach 'Pancode' i 'EPN' zestawiono w postaci infoMapy na rys. 8. Rozwiązanie 'Pancode' reprezentowane jako infoMapa (nie zamieszczona) ma 4 węzły/stany, 6 rozgałęzień/przejść, 3 warunki wstępne i 5 akcji. Oryginalne rozwiązanie 'EPN' ma 5 węzłów/ stanów, 6 rozgałęzień/przejść, 3 warunki wstępne, 5 akcji i 1 warunek końcowy. Rozwiązanie w języku Pascal zaczerpnięte z [15] reprezentowane w postaci infoMapy (rysunek (b) ma 6 węzłów/ stanów, 8 rozgałęzień/przejść, 3 warunki wstępne i 5 akcji. Rozwiązanie zoptymalizowane (c) ma 3 węzły/stany, 4 rozgałęzienia/ przejścia, 2 warunki wstępne, 5 akcji

{Wersja}	A	A	A	A					
Zoptymalizowana	v								
Pancode		v							
EPN-long			v						
EPN-short				v					
{Atrybuty}	V	V	V	V					
branch	2	5	5	5					
branch with fork	0	0	1	0					
loop	1	1	0	1					
loop with exit	1	0	0	0					
state	3	4	5	4					
transition	4	6	6	6					
preCondition	2	3	3	3					
action	5	5	5	5					
postCondition	1	0	1	0					

Rys. 8. Zestawienie różnych rozwiązań zadania

i 1 warunek końcowy.

Rozwiązanie zostało zoptymalizowane - otrzymane przez połączenie stanów, przejść, przez eliminację redundantnych warunków i akcji.

Kompletna infoMapa z rysunku 9 zawiera zestawienie dwóch algorytmów przeszukiwania binarnego. Zostały one oznaczone *'Binary Search Program Version 1'* i *'Binary Search Program Version 2'* jako elementy zbioru {PROCESS}. Kolumny oznaczone literami *'v'* w wierszu *'Binary Search Program Version 1'* opisują ten algorytm. Analogiczna sytuacja zachodzi w przypadku drugiego algorytmu.

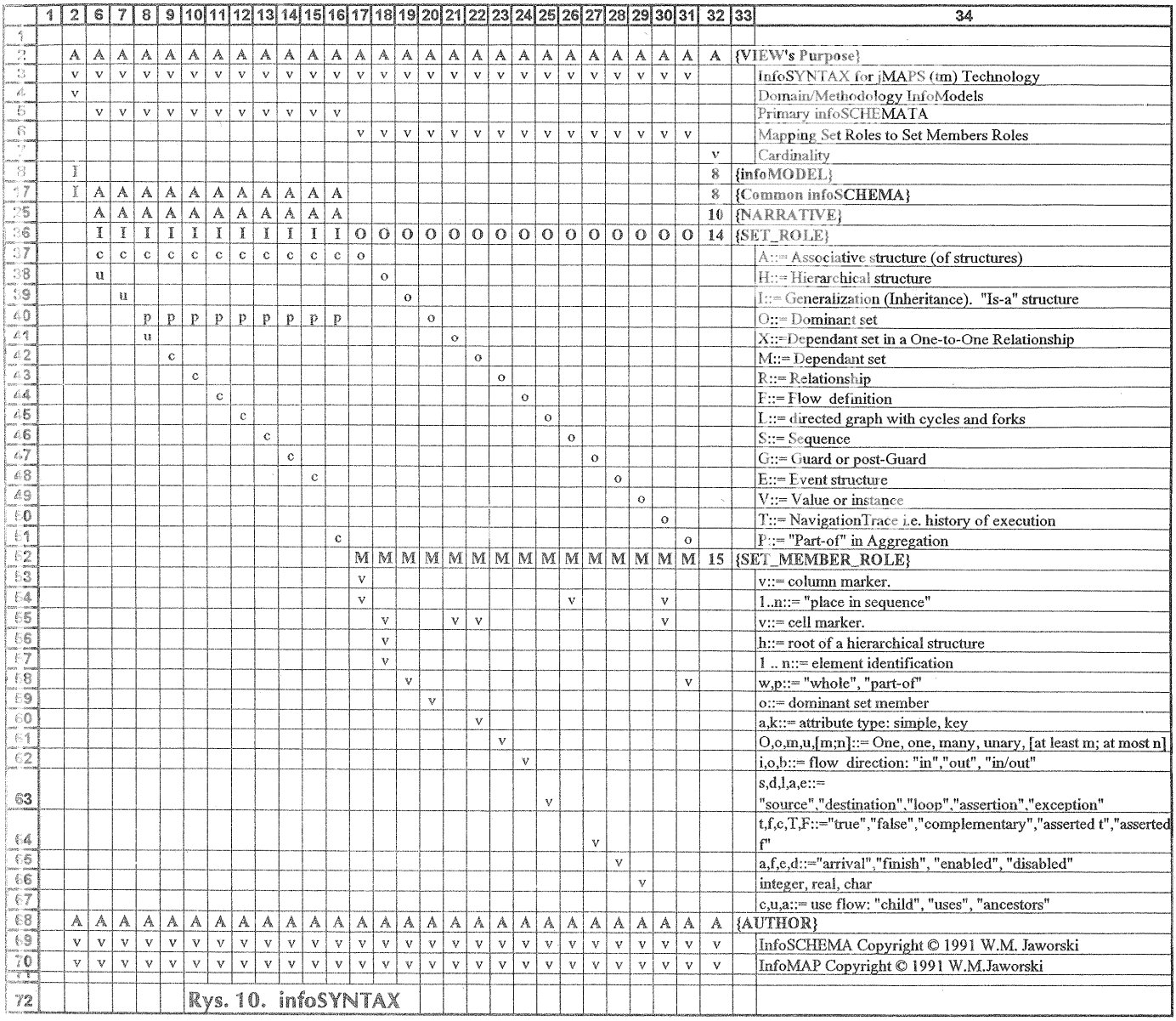
InfoMapę możemy zacząć analizować praktycznie w dowolnym miejscu w zależności od tego co nas interesuje. Zacznijmy może od zbioru {*preCONDITION*}. Zawarty w nim jest warunek *'first<=last'* i jeśli jest on spełniony (tzn. w jego wierszu jest litera *'t'*), analizujemy kolumnę 3, jeśli nie (*'f'*) - kolumnę 4. Zakładamy, że warunek jest spełniony *('t')* i analizujmy kolumnę 3. Idąc w górę kolumny napotykamy litery *'s'* i *'d'*, które mówią, że następuje przejście ze stanu *'1:Start'* do stanu *'2:Process element a[i]'* (Przetwarzanie elementu a[i]). Jeszcze wyżej w zbiorze {*TRANSITION*} badanej kolumnie odpowiada *'try middle'* (spróbuj element środkowy).

Jeśli warunek *'first<=last'* nie jest spełniony *('f')*, mamy sytuację opisaną w kolumnie 4. Następuje przejście od stanu *'1:Start'* do stanu *'3:End'*. Odpowiadająca pozycja dla kolumny 4 w zbiorze {*TRANSITION*} to *'failure'* i w zasadzie nie ma czemu się dziwić. Przeszukiwanie w takiej sytuacji nie ma sensu. Każda kolumna infoMapy przedstawionej na rysunku 9 opisuje inny stan programu. InfoMapa 9 stanowi zestawienie możliwych stanów dwóch porównywanych programów.

Z powyższych przykładów wynika, że infoMapy mogą być wykorzystywane do porównywania oraz optymalizowania algorytmów. Możliwe jest także sprawdzenie czy dany 'nowy' nie jest w gruncie rzeczy tym samym, nieco zmienionym, już istniejącym wcześniej algorytmem oraz, czy rzekomo 'nowa' metoda rzeczywiście wnosi coś nowego do inżynierii oprogramowania.

Jako kolejny przykład opiszemy rozwój analizatora syntaktycznego oryginalnie dyskutowany w [14]. Załóżmy, że na pewnym poziomie tworzenia tego analizatora otrzymaliśmy już graf przepływu

	2	3	4	5	6	7	8	9	10	11	12	13	14
1													
2	A	A	A	A	A	A	A	A	A	A	A		{AUTHOR}
3	v	v	v	v	v	v	v	v	v	v	v		InfoSCHEMA Copyright © W.M. Jaworski
4	v	v	v	v	v	v	v	v	v	v	v		InfoMAP Copyright © WMJ
5	A	A	A	A	A	A	A	A	A	A	A		{VIEW's Purpose}
6	v												Pascal-like Code for Binary Search
7		v	v	v	v	v	v	v	v				Pascal-like Code rewritten as infoMAP (Version 1 and 2)
8										v			infoMap rewritten as Pascal-like Code
9											v		Cardinality
10	S									S	23		{STATEMENT}
11	1												Start: IF (first <= last) THEN i:= (first+last) div 2;
12	2												ELSE X not found; GOTO Exit;
13	3												IF a[i]<X THEN first := i-1; GOTO Start;
14	4												ELSE
15	5												IF a[i]>X THEN last:= i-1 GOTO Start;
16	6												ELSE X found at i GOTO Exit;
17	7												Exit:
18										1			1: IF (first <= last) THEN i:= (first+last)div 2 GOTO 2;
19										2			IF NOT(first <= last) THEN X not found GOTO 3;
20										3			2: IF a[i]<X THEN first := i+1 GOTO 1;
21										4			IF (NOT(a[i]<X)) AND (a[i]>X) THEN last:= i-1 GOTO 1;
22										5			IF (NOT(a[i]<X)) AND NOT(a[i]>X) THEN X found at i GOTO 3;
23										6			3: End;
24		A	A	A	A	A	A	A	A		2		{PROCESS}
25		v	v	v	v	v							Binary Search Program Version 1
26							v	v	v				Binary Search Program Version 2
27		O	O	O	O	O	O	O	O		5		{TRANSITION}
28		o											Try middle
29			o										failure
30				o			o						try upper
31					o			o					try lower
32						o			o				success
33		L	L	L	L	L	L	L	L		6		{STATE}
34		s	s	d	d								1: Start
35		d		s	s	s							2: Process element a[i]
36			d			d							3: End
37							1	1	s				1: Process element a[i]
38							a	a					2: Failure
39									d				3: Success
40		G	G	G	G	G	G	G	G		3		{preCONDITION}
41		t	f										first <= last
42				t	f	f	t	f	f				a[i]<X
43					t	f		t	f				a[i]>X
44		S	S	S	S	S	S	S	S		5		{ACTION}
45				1			1						first := i+1
46					1			1					last:= i-1
47		1					2	2					i:= (first+last) div 2;
48						1			1				X found at i
49			1										X not found
50		G	G	G	G	G	G	G	G		1		{postCONDITION}
51							f	f					first <= last
52		F	F	F	F	F	F	F	F		5		{DATA}
53				i	i	i	i	i	i				a[FIRST..LAST]::= vector
54				i	i	i	i	i	i				X::= searched for existance in vector a
55		i	i	o			b	i					first::= lower limit
56		i	i		o		i	b					last::= upper limit
57		o		i	i	i	b	b	i				i::= middle
58													
59		Rys. 9. Zestawienie dwóch algorytmów przeszukiwania binarnego.											

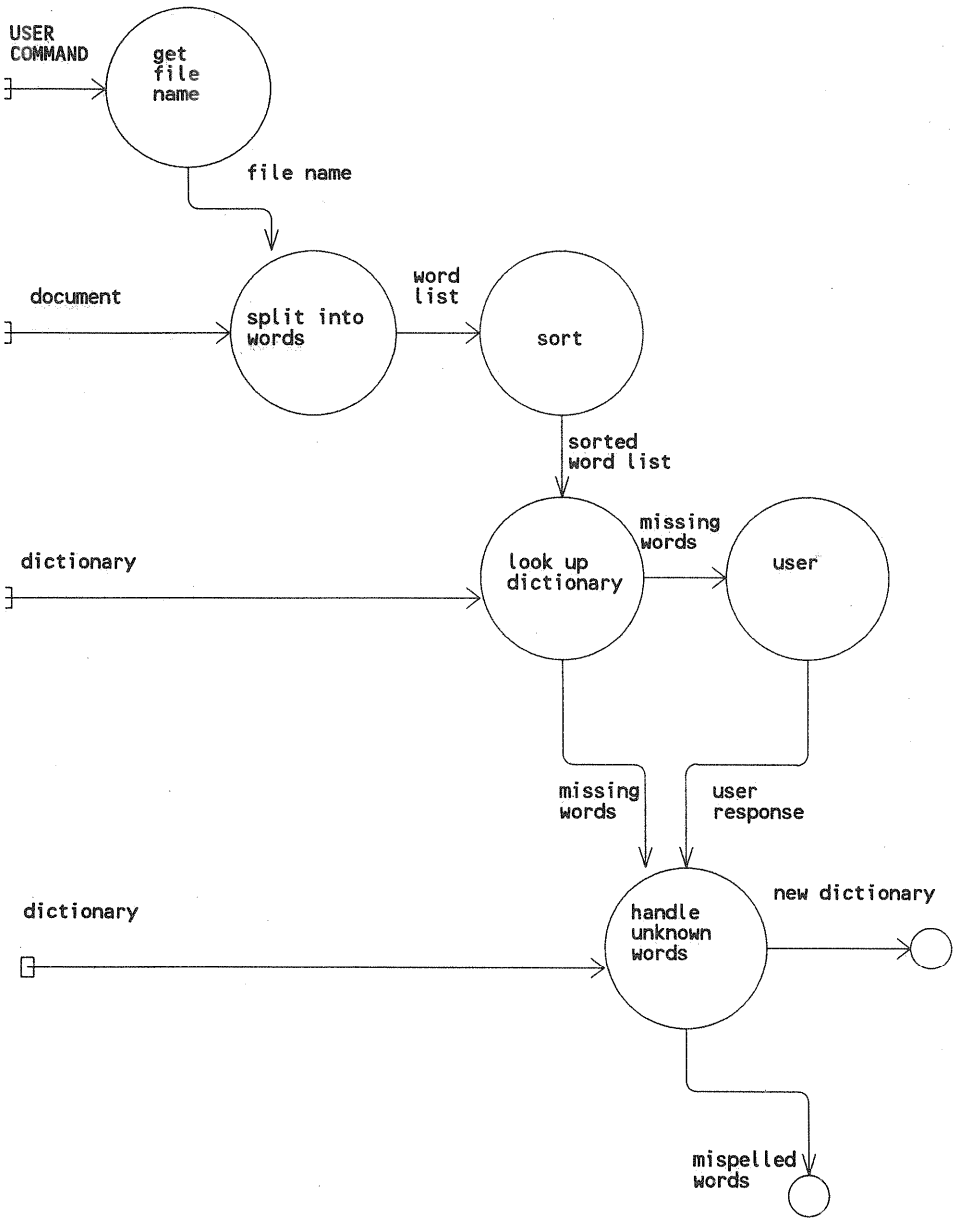


Rys. 10. infoSYNTAX

danych pokazany na rysunku 11 (będący adaptacją rysunku 18 z [18]). Yau daje zbiór relacji między tym grafem przepływu a diagramem struktury z rysunku 13. Graf przepływu i diagram struktury są wyrażone jako infoMapy na rysunkach 12 i 14. Przykładowo schemat rysunku 12(a) mówi nam o składnikach rysunku 12(b): Są to OPERATION i dataOBJECT. Górna część infoMapy z nagłówkiem złożonym z sześciu liter 'O' na rysunku (12b) definiuje sześć operacji odpowiadających sześciu wierzchołkom grafu przepływu danych[1].

Dolna część infoMapy opisuje dwie binarne relacje od operacji do obiektów. Wiersz zawierający *'g'* odpowiada obiektowi wejściowemu (lub inaczej danemu: *given*). Wiersz odpowiadający obiektowi wyjściowemu jest oznaczony symbolem *'r'* (*result*). Schemat infoMapy na rysunku 14 jest bogatszy niż ten z rysunku 12. W części górnej po lewej stronie użyto symbolu *'H'* do oznaczenia formy drzewiastej diagramu struktury, symbole *'g' i 'r'* mają takie samo znaczenie jak zdefiniowane wcześniej. Symbol *'b'* wskazuje, że dany obiekt jest zarówno wprowadzany jak i zwracany przez operację. Przykładowo *'GetWords'* otrzymuje nazwę pliku z *'GetFileName'* i wysyła ją do *'split'*. Użycie kilku różnych symboli w treści

[1] Graf skierowany $G<\{O\},\{D\}>$, może być określony jako zbiór wierzchołków oznaczony literą *'O'* i zbiór krawędzi oznaczony literą *'D'*. Do identyfikacji krawędzi zbioru G, używa się liter *'g','r'* oraz *'b'* (*given, result, both*)

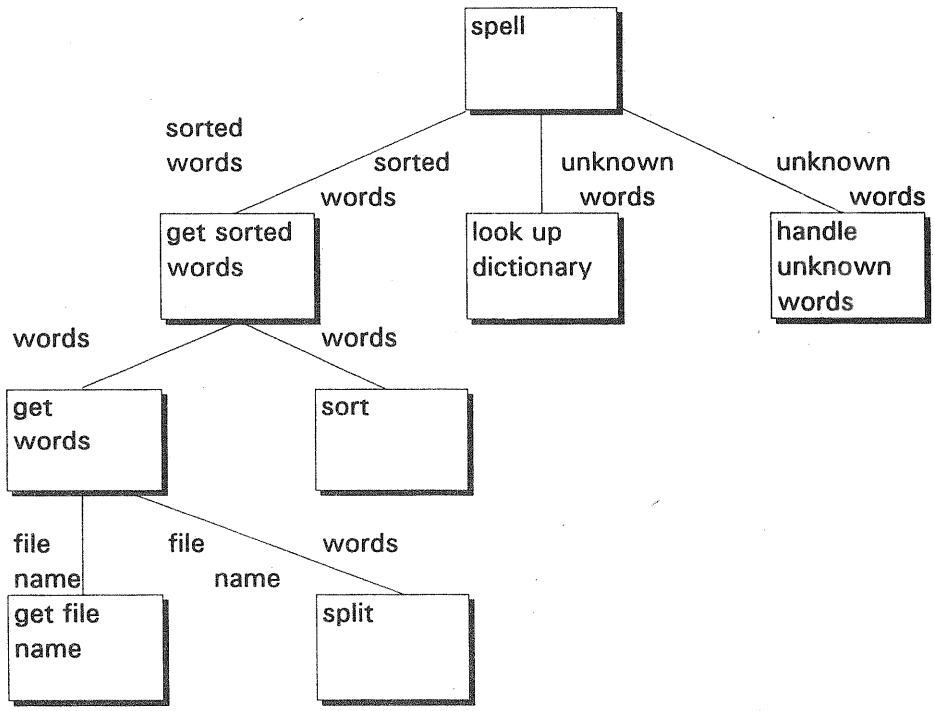


RYS. 11. Graf przepływu danych

jMapy odpowiada użyciu kilku rodzajów krawędzi w grafie.

Niejednoznaczności notacji można uniknąć, jeśli wybierze się symbole z wielkiego alfabetu (małe i duże litery, cyfry, litery greckie itd.). Liczba rodzajów krawędzi jaka może być rozróżniana (jasne, ciemne, przerywane itp.) jest ograniczona. Prezentacja grafów wymaga ponadto właściwego oprogramowania, gdyż trudno jest wyświetlić na ekranie graf mający wiele wierzchołków. Nieetykietowana krawędź grafu zawiera tylko jeden bit informacji; osobie oglądającej mówi tylko czy dwa węzły są w relacji. W celu zinterpretowania grafu trzeba dokładnie wiedzieć, co oznacza obecność krawędzi (kontekst).

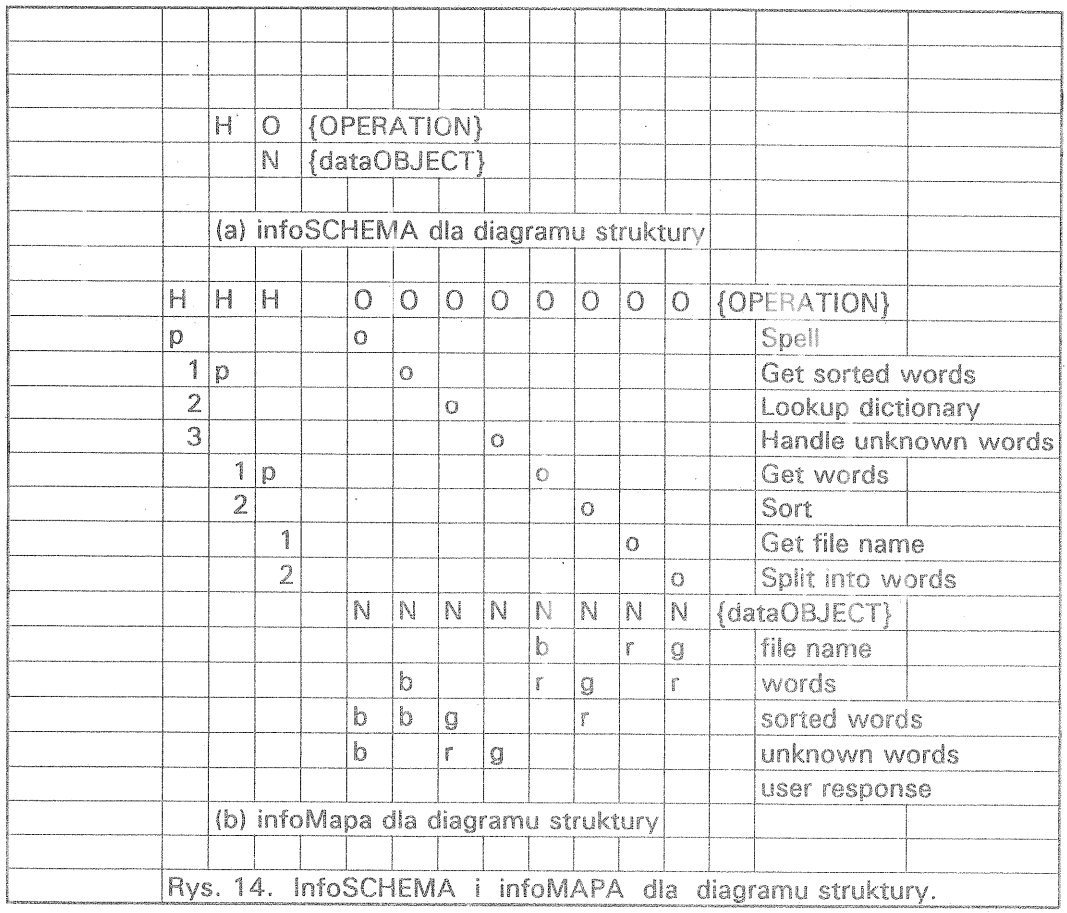
W grafie przepływu danych na rysunku 11 każda krawędź odpowiada przepływowi danych pewnego rodzaju. Przykładowo, krawędź *'sorted word list'* łączy słowa z węzła *'sort'* z węzłem *'lookup dictionary'*. Na diagramie struktury (rys. 13) węzły są połączone, jeśli jest jakikolwiek przepływ danych między nimi.



Rysunek 13. Diagram struktury dla grafu z rysunku 11.

		O	{OPERATION}								
		N	{dataOBJECT}								
	(a) infoSCHEMA dla grafu przepływu danych										
	O	O	O	O	O	O	{OPERATION}				
	o							lookup dictionary			
		o						handle unknown words			
			o					sort			
				o				get file name			
					o			split into words			
						o		user			
	N	N	N	N	N	N	{dataOBJECT}				
	g	g						dictionary			
		r						new dictionary			
					g			document			
				g				user command			
		r						mispelled words			
				r	g			file name			
			g		r			word list			
	g		r					sorted word list			
	r	g				g		missing words			
		g				r		user response			
	(b) infoMapa dla grafu przepływu danych										

Rys. 12. infoSCHEMA i infoMAPA dla diagramu przepływu danych.



Rys. 14. InfoSCHEMA i infoMAPA dla diagramu struktury.

Strzałki wskazują zależność a nie kierunek przepływu. Przykładowo: węzeł *'lookup dictionary'* otrzymuje posortowane słowa a zwraca słowa nieznane.

W reprezenatacji jMapy takie niewidoczne rozróżnienia nie istnieją. Znaczenie każdej pozycji w jMapie jest jasne dzięki zawartości wiersza deskryptorów. W prezentowanej konwencji możemy używać 'krawędzi' różnych rodzajów bez niejednoznaczności.

4. Wnioski

Omówione przykłady pozwoliły - mamy nadzieję - skutecznie zaprezentować użyteczność InfoMap jako narzędzia do analizy konstrukcji stosowanych w językach programowania i algorytmach. Te możliwości odnoszą się także do specyfikacji oraz konstrukcji projektowych. Notacja infoMap jest oparta o relacje, grafy i funkcje. Stanowi doskonałe narzędzie do dokumentowania, zmniejszania redundancji poprzez wykrywanie ukrytych powiązań. Pozwala również na modelowanie różnych obiektów z uwzględnieniem ich aspektów statycznych i dynamicznych. InfoMapy stanowią także poważny krok w kierunku utworzenia 'Przestrzeni Informacyjnej Systemu' [15], wspierającej projektantów systemu we wszystkich fazach rozwojowych tego systemu. Propozycja implementacji tej idei jest przedstawiona w pracy [17] jako 'przestrzeń informacyjna projektu.'

InfoMapy pozwalają na bardzo łatwe przetwarzanie zawartych w nich informacji np. przez użycie typowych arkuszy kalkulacyjnych oraz narzędzi rozszerzających te arkusze [5].
Narzędzia te pozwalają na generację kodu programu na podstawie specyfikacji programu w postaci infoMapy. I żeby zakończyć prezentację możliwości infoMap możemy powiedzieć, iż stanowią one narzędzie pozyskiwania i gromadzenia

wiedzy o możliwościach nie w pełni jeszcze odkrytych. Jeszcze raz namawiamy Czytelnika na zmierzenie się z rysunkiem 3, na którym jest przedstawiona w sposób ustrukturyzowany wiedza z dziedziny bardzo odległej od przykładów ilustrujących niniejszą prezentację.

Bibliografia

[1] Aho A. V., Hopcroft J. E., Ullman J. D.: *Projektowanie i analiza algorytmów komputerowych*, PWN, Warszawa 1983.

[2] Amstel J. J., Poirters J. A. A. M.: *The design of Data Structures and Algorithms*, Prentice Hall, Academic Service, 1989.

[3] Berztiss A. T.: *Data Structures. Theory and Practice*, Academic Press. New York, London 1971.

[4] Dijkstra E. W.: *Guarded commands, nondeterminancy and the formal derivation of programs*, CACM 18, August 1975, pp. 453-457.

[5] *InfoFARM: User Manual*, GS General Strategies Inc., 1991.

[6] Jaworski W. M., Ficocelli L., O'Mara K. S.: *The ABL/W4 Methodology for System Modelling*, System Research J., 1987, Vol. 4, No. 1, pp. 23-27.

[7] Jaworski W. M., Virard M.: *Converting a Software Company to a New Technology*, Proceedings of the 1986 Canadian Conference on Industrial Computers, Montreal 1986, pp 12-1..12-7.

[8] Jaworski W. M., MacCuaig I., Marinelli T., Nysztor T.: *Executable Specification for a Large Industrial Process*, Proceedings of the 1986 Canadian Conference on Industrial Computers, Montreal 1986, pp. 60-1..60-5.

[9] Jaworski W. M., Cummings T.: *Program Normalization and Optimization: Using infoMaps as an inspection and program processing tool*, Canadian Conference on Electrical and Computer Engineering, Quebec City, September 1991.

[10] Jaworski W. M.: *System Analysis and Design in a Classroom: infoMAPs Teaching Factory*, Modelling Conference, Pittsburg, Pa., May 2-3, 1990.

[11] Jaworski W. M., Deslauriers B.: *Software Process, Maintenance, Products: Software Development Environment based on infoMAPs*, Control and Electrical Eng. Conference, Ottawa, September 1990.

[12] Jaworski W. M., Grogono P. D.: *infoMAPS: a Pragmatic Environment for Seamless and Nondeterministic Software Development*, Concordia University, Computer Science Dept., January 1991.

[13] Kovats T. A.: *Comments on innovative control constructs in Pancode and EPN*, SIGPLAN Notices 25, 11 (November 1990).

[14] Somerville I.: *Software Engineering*, Addison-Wesley, 1986.

[15] Webster D.: *Mapping the design representation terrain: a survey*, IEEE Computer, (December 1988), pp. 8-23.

[16] Wirth N.: *Algorytmy+Struktury danych=Programy*, WNT, Warszawa 1989.

[17] Zaliwski A., Kuraś M.: *CASAD - Nowe podejście do modernizacji SI*, Materiały Piątej Jubileuszowej Wyższej Międzynarodowej Górskiej Szkoły PTI, Szczyrk 1993.

[18] Yau S., Nicholl R., Tsai J. and Liu S. S.: *An integrated life cycle for software maintenance*, IEEE Trans. Soft. Eng., 1988, 14, 8, 1128-44.

Marian Kuraś, Andrzej Zaliwski
Katedra Informatyki Akademii Ekonomicznej, Kraków

Komunikacyjne aspekty tworzenia systemów informacyjnych[1]

I like talking to the brick wall.
It is the only thing in the world
that never contradicts me.

Oscar Wilde

1. Wprowadzenie

Problemy z komunikacją zaczęły ujawniać się w naszej praktyce odkąd rozpoczęły się zmiany w polityce i gospodarce. Wzorce przeniesione z poprzedniego systemu politycznego i gospodarczego w zupełności nie odpowiadają obecnym potrzebom. Demokratyczny system polityczny jak i liberalny ustrój gospodarczy wymagają wszechstronnego, aktywnego i skutecznego komunikowania się.

Przed ponad 40 laty w 'Harvard Business Review' ukazał się artykuł znanych psychologów C. Rogersa i F. Roethlisbergera [25], w którym autorzy omówili bariery w komunikacji w organizacji. Ta klasyczna pozycja miała swój wpływ - wśród wielu innych publikacji z zakresu psychologii - na zmiany w zarządzaniu zapoczątkowane w latach '60 i trwające do dziś. Od lat większość studentów szkół biznesu uczy się lepiej komunikować wiedząc, iż od tych umiejętności będzie zależeć w znacznym stopniu ich zdolność kierowania. Warto w tym miejscu przypomnieć legendę zarządzania - L. Iaccocc'ę, który - będąc z wykształcenia inżynierem - uważał, że najważniejsze dla menadżera są umiejętności związane ze współpracą z ludźmi, w tym umiejętność komunikowania się [11].

Wielokrotnie w literaturze z zakresu systemów informacyjnych są wykazywane analogie między pracą analityka/projektanta systemów informacyjnych a menadżerem (np.: [3], [12]). Nic zatem dziwnego, że problematyka komunikacji wcześniej była dostrzeżona przez twórców systemów informacyjnych (SI) w naszej niedawnej praktyce gospodarczej. Jakkolwiek postrzegana, nie jest uważana za pierwszorzędną. Niech świadczy o tym przykład systemu POLTAX, którego twórcy popełnili całą serię błędów [15], [18]. Można je z powodzeniem interpretować właśnie jako nieumiejętność, a czasami wręcz brak komunikacji z użytkownikami w toku prac projektowych.

Każdy, kto ma doświadczenia z tworzenia SI, może z łatwością podać liczne przykłady niepowodzeń i błędów popełnionych na skutek niesprawności komunikacji. Mimo to jej znaczenie w tworzeniu SI jest wciąż nie w pełni uświadomione i często lekceważone. Autorzy podejmują próbę przedstawienia w niniejszym referacie przeglądu różnych aspektów komunikacji w tworzeniu SI.

Celem referatu jest prezentacja różnych aspektów komunikowania się w organizacji oraz implikacji w sferze SI - ich tworzenia i funkcjonowania. W kolejnych punktach omówiono zagadnienia komunikacji w or-

[1] Referat jest efektem prac badawczych przeprowadzonych dzięki finansowaniu przez Akademię Ekonomiczną w Krakowie

ganizacji, potrzeby komunikowania się w procesie tworzenia SI, celów wprowadzania techniki informacyjnej (TI), znaczenia metodyki jako narzędzia komunikowania się i potrzeb komputerowego wspomagania komunikacji w toku procesu tworzenia SI.

2. Komunikacja w organizacji

Tematyka komunikacji w organizacji *(communication in organization, business communication, managerial communication)* ma bardzo dużo opracowań w literaturze zachodniej tak w postaci monografii jak i podręczników (np.: [2], [4], [6], [7], [22]). W szczególności komunikacja jest szeroko opracowana w literaturze amerykańskiej.

Problematyka komunikacji pojawia się w kontekście tworzenia SI w trzech aspektach:

- komunikacji w organizacji, w której modernizujemy system informacyjny;
- komunikacji z użytkownikami, dla których tworzony jest SI (dotyczy to wszystkich grup użytkowników);
- komunikacji w zespole projektowym, którego jednym z głównych zadań jest zgromadzenie i wykorzystanie wiedzy o organizacji w celu stworzenia nowego SI.

Wszystkie z wymienionych aspektów okazują się ważne w procesie tworzenia SI, przy czym obserwacja dowodzi, że wszystkie są powszechnie zaniedbywane w praktyce. Najmniej chyba uświadomioną i uznawaną kwestią są procesy komunikacyjne w organizacji. Im też będzie poświęcony ten punkt referatu. Podstawom komunikacji między użytkownikami i między członkami zespołu projektowego w zakresie komunikacji interpersonalnej była poświęcona praca [16]. W dalszym ciągu niniejszego referatu omówione są zagadnienia związane z komunikacją rozumianą jako droga akwizycji wiedzy o systemie informacyjnym.

Komunikację można określić poprzez dwa zasadnicze rodzaje działań:

- **nadawanie** i
- **odbiór** wiadomości lub obrazu.

Ponadto wyróżniamy w procesie komunikacji takie składniki jak: *treść, otoczenie, sposób kodowania, medium, zakłócenia* itd., przy czym możemy je uznać za pochodne dwóch podstawowych. Komunikacja zachodzi w każdych warunkach i nie można się nie komunikować. Nawet wtedy, kiedy ktoś unika kontaktu, przekazuje wiadomości poprzez swoje zachowania. Tym większe znaczenie ma świadome i celowe komunikowanie się.

Przedmiotem naszych zainteresowań jest komunikacja w organizacji definiowana jako [6]:

> *Ewoluujący proces kulturowy dzielenia informacji i tworzenia powiązań w środowisku stworzonym w celu zarządzania zachowaniami zorientowanymi na osiąganie celów.*

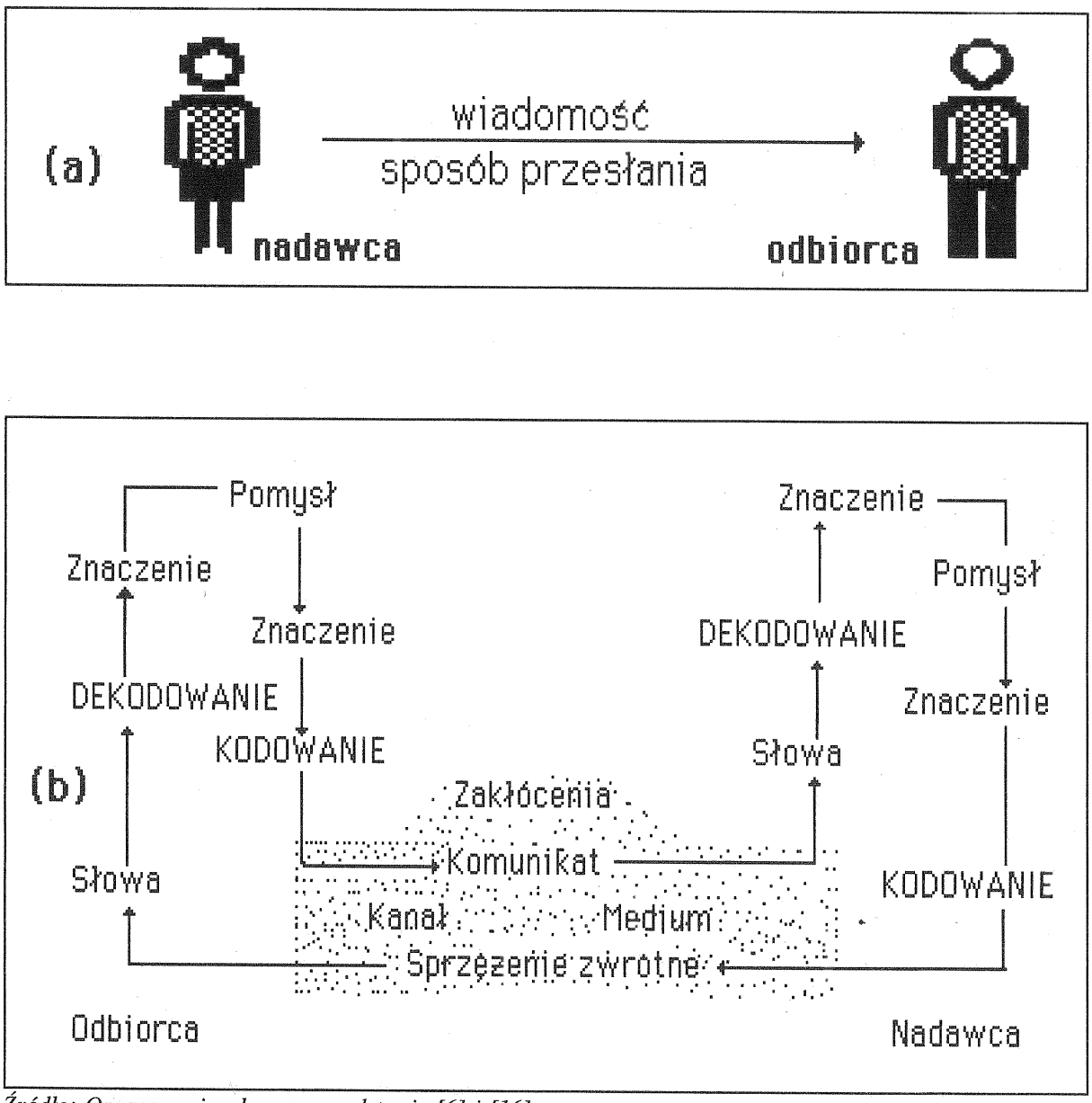
Wyróżnia się pięć podstawowych modeli (rysunek 1) komunikowania się [6]:

a. jednokierunkowy;
b. interakcyjny;
c. relacji między dwiema osobami;
d. kontekstowy;
e. strategiczny (transakcji między organizacją a otoczeniem).

Wymienione modele są przydatne do diagnozowaniu procesów komunikowania. Do naszych celów najbardziej przydatnymi wydają się modele: interakcyjny (rys. 1b) i kontekstowy (rys. 1c), najlepiej wyjaśniający istotę komunikacji w organizacji.

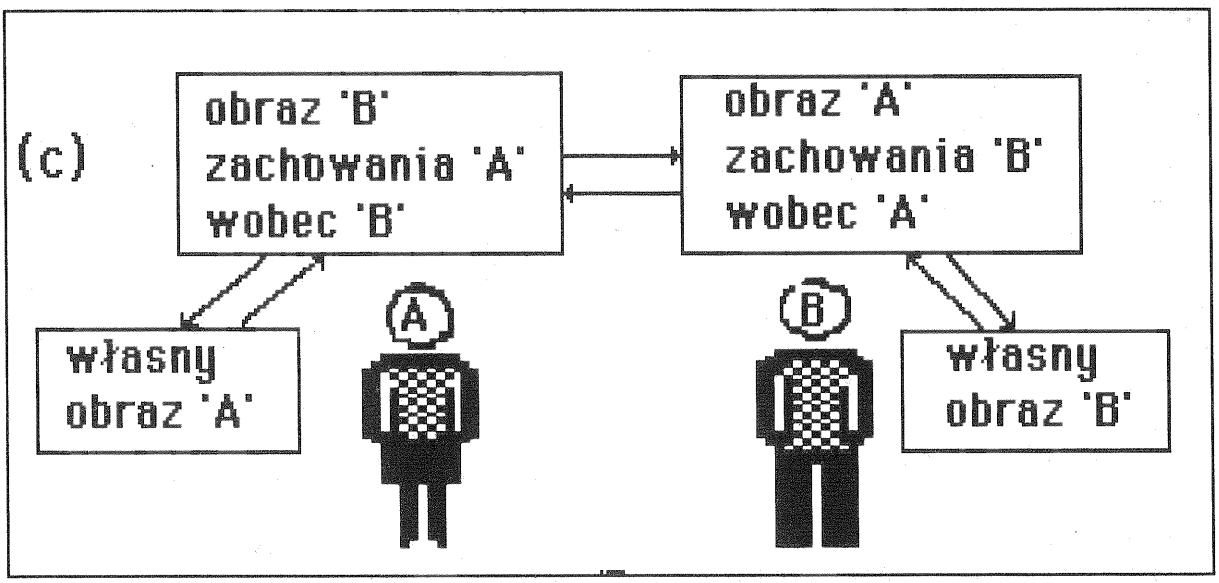
Do analizy procesów komunikacji w organizacji przyjmuje się różne perspektywy, z których cztery mają podstawowe znaczenie i mogą być użyteczne w tworzeniu SI oraz ich ocenie [6], [7]:

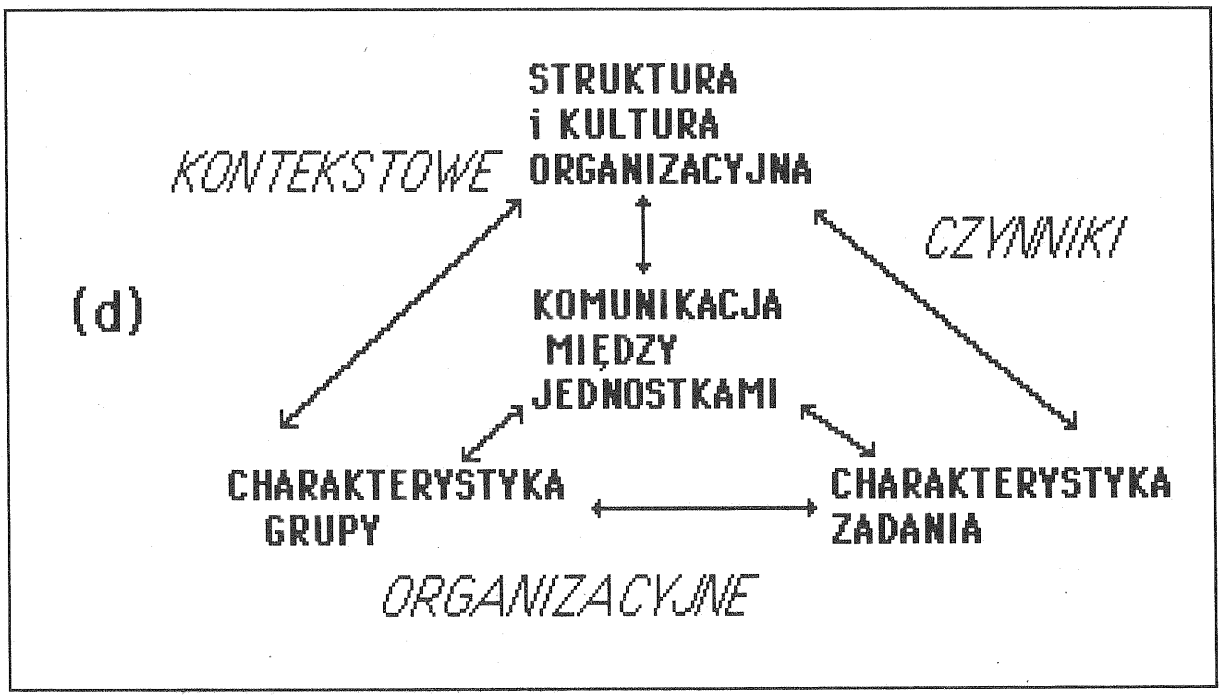
- **mechanistyczna** koncentrująca się na procesie przekazu wiadomości poprzez kanał (związana z modelem jednokierunkowym);
- **psychologiczna** skupiająca się na odbiorcy wiadomości (odpowiada jej model kontekstowy);
- **interpretacyjno-symboliczna** zorientowana na symbole, metafory itp. wytworzone przez kulturę organizacyjną (model kontekstowy);
- **interakcyjno-systemowa** koncentrująca się na sekwencjach wymiany wiadomości (odpowiadają jej dwa modele: kontekstowy i strategiczny).

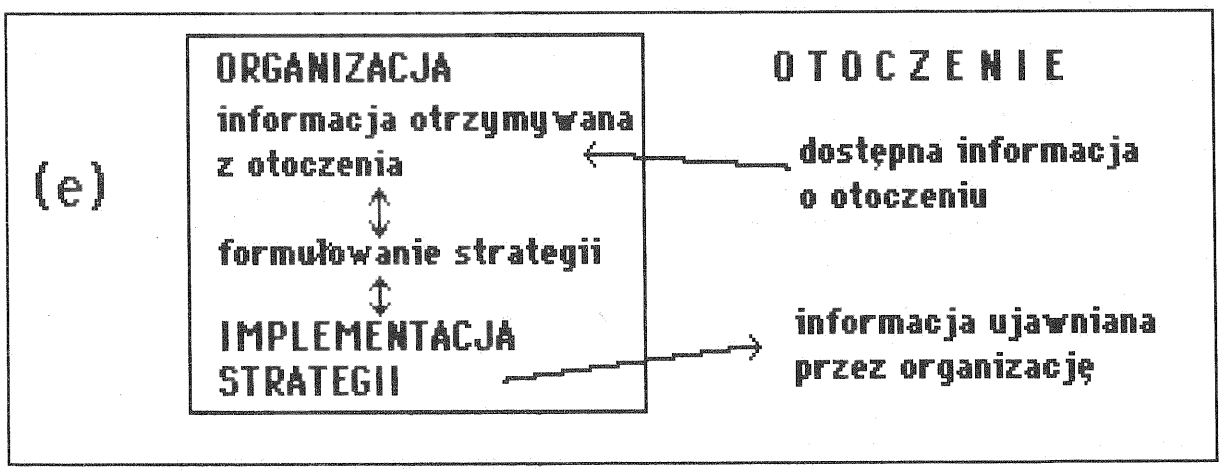


Źródło: Opracowanie własne na podstawie [6] i [16]

RYS. 1. Modele komunikacji w organizacji







RYS. 1. Modele komunikacji w organizacji (c.d.).

Komunikacja organizacyjna natrafia na przeszkody. Ich uświadomienie jest nieodzowne również do prowadzenia analizy SI jak i do organizowania pracy zespołu. W literaturze wymienia się liczne bariery[2] powodujące niesprawności komunikacji, takie jak (por.: [4], [6], [24]:

- selektywna percepcja;
- ocenianie ludzi i faktów;
- z góry przyjmowane założenia, uprzedzenia i przesądy;
- głęboka specjalizacja, nastawienie zadaniowe;
- odmienność języka;
- błędny odbiór i 'twórcze' interpretacje;
- brak sprzężenia zwrotnego, zaniechanie konsultacji;
- czynniki kulturowe (kultura i subkultury organizacyjne);
- czynniki przestrzenne;
- czynniki organizacyjne, takie jak: status, władza itp.;
- ograniczenia: czasowe, finansowe itp.

Działanie wymienionych barier objawia się w postaci:

- zakłóceń w pionowym i poziomym przepływie wiadomości;
- przeciążenia lub niedociążenia komunikacyjnego (informacyjnego);
- utraty informacji;
- zakłóceń w przebiegu procesów decyzyjnych.

Przedstawione bariery komunikacji w organizacji są raczej powszechnie znane, jednak bardzo rzadko w naszej praktyce są poddawane analizie. Analiza skupia się na danych, ich przechowywaniu, przetwarzaniu i przesyłaniu. Zakres analiz wynika z dotychczasowych praktyk, rozwiązań organizacyjnych oraz wyobrażeń twórców systemu. Brak jest przewidywania, planowania przyszłych potrzeb informacyjnych, gdyż nie ma wizji zmian organizacyjnych. Jeżeli nawet taka wizja istnieje, nie jest ujawniana przez kierownictwo - nie jest przekazywana informatykom tworzącym nowe rozwiązania SI. W ten sposób powstają zastosowania zaspokajające co najwyżej dzisiejsze potrzeby.

3. Cele wprowadzania techniki informacyjnej a komunikacja

Usprawnienie komunikacji jest bardzo często stawiane jako cel wprowadzania techniki informacyjnej / techniki komputerowej. We wstępnej dokumentacji systemu informacyjnego jako cel możemy znaleźć takie sformułowania jak *poprawa, usprawnienie komunikacji, informacji.* Takie cele wydają się wskazywać na zrozumienie potrzeby skutecznego i sprawnego komunikowania się. Praktyka dowodzi jednak powierzchowności tej obserwacji.

Przede wszystkim w rozpoznaniu i analizie nie poszukuje się informacji o tym, jak powinny być kształtowane więzi informacyjne, ale jak wyglądają. Nie analizuje się celów, potrzeb i warunków wykorzystania informacji a określa się, gdzie i przez kogo są wysyłane oraz otrzymywane. Nie określa się potrzeb informacyjnych organizacji w zmieniającym się otoczeniu, potrzeb decydentów odpowiedzialnych za podejmowanie zamierzeń mających zapewnić przetrwanie organizacji, a 'doskonali się istniejący system informacyjny' wywodzący się z systemu scentralizowanego. Innymi słowy, oczekuje się, by rozpoznanie i analiza doprowadziły do stwierdzenia stanu faktycznego a następnie jego modyfikacji polegającej na uzupełnieniu nowej techniki.

Również oczekiwany zakres wprowadzania nowej techniki w większym stopniu ma odpowiadać potrzebom gromadzenia, przechowywania, wyszukiwania, przetwarzania i przesyłania danych niż komunikacji.

Komunikacja nie jest postrzegana przez kierownictwo jako element funkcjonowania przedsiębiorstwa. W coraz większym sto-

[2] P. V. Lewis [20] podaje listę 130 przeszkód w komunikacji organizacyjnej opracowaną na podstawie 26 prac poświęconych tej tematyce.

pniu są doceniane przez kierowników i specjalistów potrzeby informacyjne organizacji. Skutkiem tego jest inwestowanie w techniczne wyposażenie sfery informacyjnej, jednak nie jest to jednoznaczne z uznawaniem komunikacji za element systemu sterowania kluczowymi procesami zachodzącymi w organizacji. Informacja nie jest tymczasem jednoznaczna z komunikacją [26]. Nadmiar korespondencji i jej lekceważenie, uważanie narad i spotkań za stratę czasu i związane z tym ich unikanie są typowymi przykładami nieprzywiązywania wagi do procesów komunikacji w organizacjach. Ta cecha kultury organizacyjnej ma swoje implikacje wtedy, gdy przystępuje się do modernizacji, doskonalenia SI.

W warunkach tak scharakteryzowanej kultury organizacyjnej określa się cele nie uwzględniające potrzeb komunikowania się, np. ułatwiania komunikacji nieformalnej tam, gdzie jej wzorce sprzyjają sprawności i skuteczności działania organizacji. Znaczenie tego jest trudne do przecenienia, czego dowodzą opisane przez Th. Petera i R. H. Watermana [23] doświadczenia wielkich kompanii (m.in. IBM, Hewlett-Packard, 3M, City Bank). Z analizy tych przypadków autorzy wyciągnęli następujące podstawowe wnioski dotyczące komunikacji:

- systemy komunikacji w organizacjach są w zasadzie nieformalne;
- intensywność komunikacji okazuje się w dobrze funkcjonujących organizacjach bardzo duża;
- komunikacja potrzebuje wspomagania technicznego;
- wyposażenie techniczne, jeśli jest umiejętnie zastosowane, może wymuszać intensyfikację i poprawę komunikacji;
- intensywna, nieformalna komunikacja działa jako wyjątkowo twardy i skuteczny system sterowania sprzyjający innowacyjności.

Komunikacja ma szczególne znaczenie w organizacjach, w których zachodzi transformacja - zmiana organizacyjna, zmiana kulturowa [8]. Podstawową drogą do przygotowania zmiany jest przekonywanie i motywowanie uczestników całego procesu. Formalne środki pozostające do dyspozycji leadera zmiany są niewystarczające i zwykle nieskuteczne.

Modernizację systemu informacyjnego związaną z wprowadzaniem TI musimy uznawać za zmianę organizacyjną i uwzględniać w jej przygotowaniu zasady oraz stosowne metody [9]. Jest to w szczególności widoczne w warunkach zmian na skalę, z jaką mamy do czynienia w naszym kraju. Konieczność przebudowy systemu zarządzania organizacjami gospodarczymi i administracją, restrukturyzacja, nieodzowne zmiany organizacyjne nie mogą odbywać się bez uwzględnienia potrzeby i możliwości wykorzystania na szeroką skalę TI. Wprowadzanie drogiej TI niekoniecznie musi być przy tym widziane jako jednorazowe zamierzenie, w które już teraz musimy inwestować. Proces zmiany powinien i musi być zaplanowany na okres co najmniej kilkuletni.

Zamierzenia informatyczne nie mogą zatem być izolowane i nie mogą przebiegać w oderwaniu od badań organizacji. W szczególności zaś nie można rozważać zmian organizacyjnych bez uwzględniania różnych aspektów procesów informacyjnych, gdyż celem i istotą organizacji są właśnie procesy informacyjne, procesy komunikowania się.

Reasumując rozważania na temat celów stawianych przed zamierzeniami informatycznymi należy stwierdzić, że obecna praktyka nie pozwala na ich racjonalne przeprowadzanie. Kierownictwo nie jest przygotowane do zmian i nie potrafi stawiać celów, nie są przygotowani użytkownicy, którzy posługują się nieodpowiednimi wzorcami komunikowania się. Nie w pełni są do takich wyzwań gotowi informatycy, którzy odżegnują się od problemów organizacji i preferują podejście bardziej zorientowane na technologię, która sprawia mniej kłopotów niż użytkownicy.

4. Potrzeby komunikowania się w procesie tworzenia SI

Jakość aplikacji, jakie powstaną w rezultacie prac projektowych i implementacyjnych, jest w pełni zależna od zakresu i jakości wiedzy o organizacji - potrzebach i ograniczeniach - jaką uzyskają informatycy od użytkowników. Celem wstępnej fazy prac, identyfikacji potrzeb użytkowników i ich analizy, jest zgromadzenie wszystkich niezbędnych informacji o funkcjonowaniu organizacji. Dopiero na ich podstawie można określić cele i zakres komputeryzacji, strukturę funkcjonalną oraz technologię i organizację SI.

4.1. Uwagi o praktyce komunikowania się w procesie tworzenia SI

Autorzy zbierają na drodze badań ankietowych (prowadzonych wśród informatyków tworzących i implementujących zastosowania) dane o ocenie komunikacji w procesie tworzenia SI. W tablicy 1 przedstawione są wyniki uzyskane w pierwszej fazie prac wśród ponad 30 informatyków z firm (głównie prywatnych) z Krakowa.

Przedstawiane wyniki potwierdzają tezę o docenianiu komunikacji i jednocześnie jej ograniczonym postrzeganiu przez wielu praktyków. Grupa, w której przeprowadzono ankietę jest mało liczna, ale autorzy oczekują zbliżonych wyników w dalszych badaniach. Generalnie respondenci zgadzają się z twierdzeniem o istotnym znaczeniu komunikacji w procesie tworzenia SI. Mniej jednoznacznie wypowiadają się o użytkownikach i współpracy z nimi, przy czym zastanawia duży odsetek respondentów nie udzielających odpowiedzi na pytania dotyczące współpracy z użytkownikami. Chęć biernej współpracy przy tworzeniu SI potwierdza ok. połowa ankiet, ale grupa ta jest wyraźnie niższa w przypadku odpowiedzi na pytanie o propozycje rozwiązań ze strony użytkownika. Rośnie udział ankiet zawierających odpowiedzi twierdzące na pytanie o wpływ zakłóceń w komunikacji powodowanych przez użytkowników. Na to pytanie blisko jedna czwarta ankietowanych odpowiedziała, że nie ma to znaczenia.

Analitykom przypisuje odpowiedzialność za sprawność komunikacji dwie trzecie respondentów (aż ponad czwarta część uważa, że nie analitycy są za to odpowiedzialni). Ocena własnych umiejętności komunikowania się jest bardzo wysoka (84% ocen dobrych i bardzo dobrych). Umiejętności te zostały przyswojone głównie dzięki praktyce i kolegom. Większość respondentów jest zdania, że umiejętności komunikowania można nabyć. Zdecydowana większość jest zdania, że praktyczne umiejętności komunikowania się należy uzyskać w szkole i na kursach oraz w toku praktyki (ta ostatnia opcja zyskała najwięcej zwolenników). Jednocześnie blisko 1/3 respondentów jest zdania, że nie można opanować sztuki komunikowania się bez wrodzonych zdolności, a aż 92% respondentów zrozumiało znaczenie komunikacji w pracy zawodowej (46% w następstwie popełnienia błędów).

Doskonali swoje umiejętności komunikowania się 90% respondentów; 58% korzysta z dostępnej literatury, 38% uzupełnia kwalifikacje na kursach i studiach podyplomowych a 81% korzysta z rad i uczy się od innych.

Około 1/3 respondentów przygotowuje się do wywiadów z użytkownikami, rozmów telefonicznych, negocjacji z użytkownikami i sesji przeglądowych. Jednocześnie blisko trzecia część nie przygotowuje się lub uważa to za nieistotne. Tylko 61% uważa, że metodyka tworzenia SI ma wpływ na sprawność komunikacji, podczas gdy odpowiednio 84% i 85% znaczenie takie przypisuje wyposażeniu w środki techniczne i programowe narzędzia wspomagania tworzenia SI. Aż 88% uczestniczących w ankiecie stwierdziło, że w zespole nie ma bibliotekarza projektu.

Tablica. 1

Opinie o komunikacji wśród analityków i programistów (%%)

O	O	O	O	O	{OCENA}
o					Zdecydowanie tak / Bardzo dobrze
	o				Raczej tak / Dobrze
		o			Obojętne / Średnio
			o		Raczej nie / Raczej słabo
				o	Zdecydowanie nie / Słabo
					{PYTANIE}
77	19	4	0	0	Komunikacja - determinuje sukces projektu
85	12	0	4	0	Komunikacja w początkowych fazach ma szczególne znaczenie
77	19	0	0	0	Jakość kom. w procesie tworzenia SI ma znaczenie dla jakości
4	46	15	27	0	Użytkownicy chętni do współpracy
4	54	15	19	0	Użytkownicy przekazują niezbędne informacje
8	35	15	23	12	Użytkownicy chętnie i kompetentnie oceniają propozycje
0	12	8	50	23	Użytkownicy sugerują rozwiązania
19	35	23	15	0	Zakłócenia w komunikacji powodują użytkownicy
27	38	8	19	8	Odpowiedzialność za sprawność kom. spoczywa głównie na anal.
0	12	0	15	73	W zespole jest bibliotekarz dokumentujący projekt
0	42	35	19	4	Komunikacja z użytkownikami na codzień
12	46	31	0	0	Komunikacja wewnątrz zespołu projektowego
19	65	12	0	0	Swoje umiejętności porozumiewania uważam za:
12	15	19	23	31	Praktyczne umiejętności kom. uzyskałem w szkole
8	27	8	23	27	Praktyczne umiejętności kom. uzyskałem na kursach
65	35	0	0	0	Praktyczne umiejętności kom. uzyskałem w toku praktyki
15	38	15	27	4	Praktyczne umiejętności kom. uzyskałem ucząc się od kolegów
4	15	12	19	42	Praktyczne umiejętności kom. przyszły same (są wrodzone)
46	27	8	19	0	Prakt. umiejętności kom. powinno się uzyskać w szkole
23	50	12	8	4	Prakt. umiejętności komunikacji powinno się uzyskać na kursach
65	27	0	4	0	Prakt. umiejętności kom. powinno się uzyskać w toku praktyki
35	35	8	19	4	Prakt. umiejętności kom. powinno się uzyskać od kolegów
8	23	4	23	42	Prakt. umiej. kom. nie można opanować bez wrodzonych zdoln.
15	31	8	23	23	Znaczenie kom. w procesie proj. uświadomiłem sobie ucząc się
12	50	8	23	8	O znaczeniu komunikacji dowiaduje się z literatury krajowej
12	23	8	19	35	O znaczeniu komunikacji dowiaduje się z literatury zachodniej
50	42	0	0	0	Znaczenie kom. w proj. zrozumiałem w trakcie pracy zawodowej
19	27	15	15	12	Znaczenie kom. w proj. zrozumiałem w następstw. popełn. błędów
4	31	12	27	15	Znacz. kom. zrozumiałem zauważając odmienność postępow. kol.
4	23	19	27	15	Znacz. kom. zrozumiałem w wyniku zwrócenia uwagi przez kier.
35	65	0	0	0	Doskonale swoje umiejętności komunikowania się
27	31	23	12	8	Korzystam z dostępnej literatury
15	23	0	31	27	Uzupełniam kwalifikacje na kursach, studiach podyplomowych
27	54	4	12	4	Ucze się od innych, korzystam z rad
31	50	4	0	4	Pracuje nad sobą, eksperymentuje
27	42	8	8	15	Przygotowuję wcześniej wywiady z użytkownikami
23	42	8	8	15	Przygotowuje wcześniej scenariusz rozmów telefonicznych
27	46	15	4	8	Przygotowuje wcześniej scenariusz sesji przeglądowych
35	42	12	4	8	Przygotowuje wcześniej scenariusz rozmów z użytkownikami
38	23	19	8	8	Wpływ na sprawność kom. w zesp. - stosow. metodyka tworz. SI
58	27	12	0	0	Wpływ na sprawność kom. w zesp. - organizacja zespołu
19	35	15	12	4	Wpływ na sprawność kom. w zesp. - tworzenie zesp.zadaniowych
46	23	4	4	4	Wpływ na sprawność kom. w zesp. - aktywne uczestnictwo użytkow.
46	38	8	4	0	Wpływ na sprawność kom. w zesp. - wyposaż. w środki techn.
58	27	4	4	4	Wpływ na sprawność kom. w zesp. - dostępność narzędzi program.
19	19	15	31	15	Działania mające usprawnić kom. - korzysta się z typowych druków
35	27	4	15	8	Działania mające usprawnić kom. - zespoły zadaniowe
19	23	19	12	15	Działania mające usprawnić kom. - aktywne uczestnictwo użytkown.
12	23	12	38	15	Działania mające usprawnić kom. w pełni zdefiniowana metodyka
23	31	4	31	12	Działania mające usprawnić kom. - standard dokumentacji
19	31	8	35	8	Działania mające usprawnić kom. - wprowadzono standardy danych
4	23	4	42	12	Działania mające usprawnić kom. - oprogr. wspomagające proj.

Źródło: Badania i opracowanie własne

Możliwości usprawniania komunikacji dopatrują się respondenci głównie w tworzeniu zespołów zadaniowych (62%), standaryzacji dokumentacji (54%) i danych (50%). Aktywne uczestnictwo użytkowników (42%), metodyka (35%) i oprogramowanie wspomagające (27%) są uznawane za mniej istotne działania. Zastanawiający jest duży odsetek (8-15%) ankietowanych, którzy nie mieli opinii na temat możliwości usprawnienia komunikacji.

Przedstawione ogólne oceny praktyki komunikowania się w procesie tworzenia SI pozwalają na twierdzenie, że istniejące wzorce zachowań wystarczają do obecnych zastosowań. Zarazem można twierdzić, że nie zagwarantują sukcesu w zamierzeniach bardziej złożonych w warunkach zwiększonych potrzeb organizacji, a w szczególności kierownictwa. Można zarazem uznać, że liczne pozytywne opinie o znaczeniu komunikacji wraz ze świadomością potrzeb dotyczących organizacji i wyposażenia wspomagającego komunikację, mogą wskazywać na zdolność analityków do przystosowania się do tych nowych potrzeb.

4.2. Metodyka jako narzędzie komunikacji

W omówieniu praktyki komunikacji w poprzednim punkcie metodyka tworzenia SI nie zyskała w oczach uczestników ankiety uznania jako narzędzie komunikacji. Można przyjąć, że jest to skutek odmienności warunków, w jakich tworzone są u nas zastosowania. Te warunki to przede wszystkim: otoczenie organizacyjne z wzorcami sformalizowanej komunikacji i subkulturą organizacyjną, ograniczone, nieuświadomione bądź nie artykułowane potrzeby informacyjne, poziom technologiczny i cywilizacyjny itp. W takich warunkach nie ma (**nie było**) zapotrzebowania na zastosowania wspomagające zmianę organizacyjną, permanentne zmiany dostosowujące organizację do zmieniającego się ciągle otoczenia.

Przyjmijmy, że pod pojęciem metodyki rozumiemy *uporządkowany i zorganizowany sposób postępowania w badaniu potrzeb użytkownika, ich analizie, specyfikacji SI, jego implementacji oraz utrzymaniu i modyfikacji wraz z zestawem odpowiednich metod, technik i narzędzi oraz zasadami dokumentowania*. Tak rozumiana metodyka ma zapewnić skuteczną i sprawną współpracę twórców SI (analityków i programistów) oraz użytkowników. Ma wspomagać transfer wiedzy o organizacji od użytkowników oraz wiedzy o nowej technologii od informatyków do organizacji. Ma służyć sprawnemu przebiegowi procesu tworzenia SI, który z natury rzeczy jest procesem komunikacji.

Jednym z głównych celów metodyki, jaka powinna być wykorzystywana do modernizacji SI w naszych organizacjach, powinno być usprawnianie komunikacji we wszystkich jej aspektach dotyczących procesu tworzenia SI. Zostały one wymienione w punkcie 1 ale przypomnijmy je. Są to aspekty:

- komunikacji w organizacji, w której modernizujemy system informacyjny;
- komunikacji z użytkownikami, dla których tworzony jest SI (dotyczy to wszystkich grup użytkowników),
- komunikacji w zespole projektowym, którego jednym z głównych zadań jest zgromadzenie i wykorzystanie wiedzy o organizacji w celu stworzenia nowego SI.

Zachodzi potrzeba, iżby metodyka wspomagała wszystkie z wymienionych aspektów. Z punktu widzenia analizy organizacji najistotniejsze znaczenie ma udostępnienie narzędzi rozpoznawania potrzeb w zakresie komunikacji wewnątrz organizacji. Obecnie wykorzystywane metodyki nie oferują wyspecjalizowanych technik i narzędzi do analizy komunikacji nieformalnej. Uwzględnienie transferu informacji np. w bezpośredniej rozmowie, przez telefon, w toku narad czy negocjacji zależy zatem od możliwości analityka: jego wiedzy, doświadczenia zawodowego, zaakce-

ptowanego zakresu analizy (analiza organizacji) a także narzędzi wspomagania. Bez tych ostatnich praktycznie można uwzględnić jedynie wybrane i to nieliczne niesformalizowane przekazy informacyjne jako uzupełnienie analizy przepływów danych.

Nie można w pełni badać procesów komunikacji bez wykorzystania specyficznych podejść i metod. Metody takie - obecnie będące przedmiotem badań i eksperymentów - korzystają z dorobku lingwistyki i dziedzin behawioralnych [13]. Należy usytuować je w sferze badania organizacji, kultury organizacyjnej. Podejście takie jak np. w pracy [13] polega na poszukiwaniu charakterystycznych dla organizacji metafor. Są one przypisywane organizacji przez pracowników i już na wstępie rozpoznania umożliwiają określenie sytuacji w organizacji, stylu zarządzania, stopnia formalizacji procedur organizacyjnych oraz innych cech charakteryzujących kulturę organizacyjną. W ten sposób analityk zyskuje cenne wskazówki na temat organizacji i sposobu postępowania w dalszym ciągu prac.

Znajomość podstaw komunikacji organizacyjnej oraz umiejętności komunikacji interpersonalnej są nieodzowne do nawiązywania i utrzymywania kontaktów z użytkownikami. Konieczne jest tworzenie warunków do transferu wiedzy - w szczególności od użytkownika, który przyjmuje często postawy obronne w obawie przed utratą pozycji, władzy a nawet zatrudnienia. Stopniowe przełamywanie tych oporów i nieuporządkowane uzyskiwanie informacji jest naturalną koleją rzeczy nawet w przypadku, gdy nie istnieje konflikt. Porządkowanie i strukturyzowanie zgromadzonej wiedzy oraz jej gromadzenie w celu wykorzystania jest zadaniem, z którym nie zawsze radzą sobie zespoły projektowe.

Wynikają z tego trudności komunikacyjne w zespole projektowym. Złożoność obiektu i procesu projektowania, liczba uczestników oraz odmienność języków, w jakich jest opisywana wiedza o systemie, a także wymagania dwukierunkowego transferu informacji między użytkownikami i analitykami - wszystko to powoduje, że zachodzi konieczność:

- rejestrowania wszelkich danych opisujących badany obiekt;
- prezentacji danych w postaci umożliwiającej zrozumienie w celu korygowania, aktualizacji i oceny przez wszystkich zainteresowanych;
- strukturyzowania i standaryzowania tych opisów;
- zapewnienia ich kompletności i spójności;
- umożliwienia wszystkim uczestnikom prac dostępu do zgromadzonych danych;
- zapewnienia aktualności i ciągłości w miarę postępu prac.

Stąd reguły rejestrowania i prezentacji zebranych danych oraz dokumentowania prac są ważnym składnikiem metodyki. W miarę rozwoju zastosowań i krystalizowania się metodyki powinno się ustalać standardy zastępujące sprawdzone reguły.

Doświadczenie i obserwacje dowodzą, że utrzymanie reżymu dokumentowania prac jest trudne nawet przy niewielkich projektach. Przy dużych projektach występuje tak znaczny wzrost ilości gromadzonych informacji i ich złożoności, że praktycznie nie jest możliwe stosowanie się do wymienionych powyżej wymagań. Usprawnienie komunikacji poprzez komputerowe wspomaganie prac projektowych staje się koniecznością.

4.3. Potrzeby i ograniczenia wspomagania komunikacji

Zakres prac analitycznych i projektowych ulega ciągłemu rozszerzaniu. Jeszcze przed kilkoma laty analiza SI w naszej praktyce gospodarczej była często uznawana za nie w pełni uzasadnioną. Wysoki stopień formalizacji organizacji czynił rozpoznanie prostym, a potrzeba transferu i gromadze-

nia dotyczyła ograniczonego zakresu 'twardych' danych opisujących jedynie system przetwarzania.

Obecnie można oczekiwać licznych zmian w praktyce tworzenia SI. Będą one dotyczyły zarówno głębokości zmian (*zarządzanie, organizacja i jej SI*), zakresu analizy (*analiza organizacji, analiza zróżnicowanych potrzeb użytkowników*), stosowanych podejść, metod, technik i narzędzi (*'miękkie' podejście systemowe, 'twarde' metody systemowe, alternatywne formalne techniki opisu i analizy*), dostępnej technologii, uczestnictwa i wymagań użytkowników, organizacji informatyki w przedsiębiorstwie i odpowiedzialności za efekty jej wprowadzania (zastosowania 'u użytkownika'), zaawansowania potrzebnych zastosowań (*systemy nowych generacji, biurowe systemy informacyjne, łączność*).

Zakres i głębokość tych zmian będzie powodować ewolucję praktyki projektowania. Będzie nam wówczas łatwiej zrozumieć przyczyny, dla których komunikacja jest tak eksponowana w zachodnich podręcznikach analizy SI (np.: [5], [12]).

Dotychczasowe doświadczenia z metodyką ISAD [19] oraz próby jej wspomagania komputerowego [17], [19] pozwoliły na stwierdzenie konieczności komputerowego wspomagania prac analitycznych i projektowych. Metody wykorzystywane do przeprowadzenia analizy potrzeb użytkowników oraz specyfikacji SI przy kompleksowym podejściu, jakiego coraz częściej oczekują użytkownicy, wymagają gromadzenia i wykorzystania wielkich ilości danych. Ich strukturyzacja i systematyzacja może być przeprowadzona jedynie przy wspomaganiu. Również ich gromadzenie (w postaci diagramów przepływu danych, DPD) w zasadzie wymaga komputerowego wspomagania.

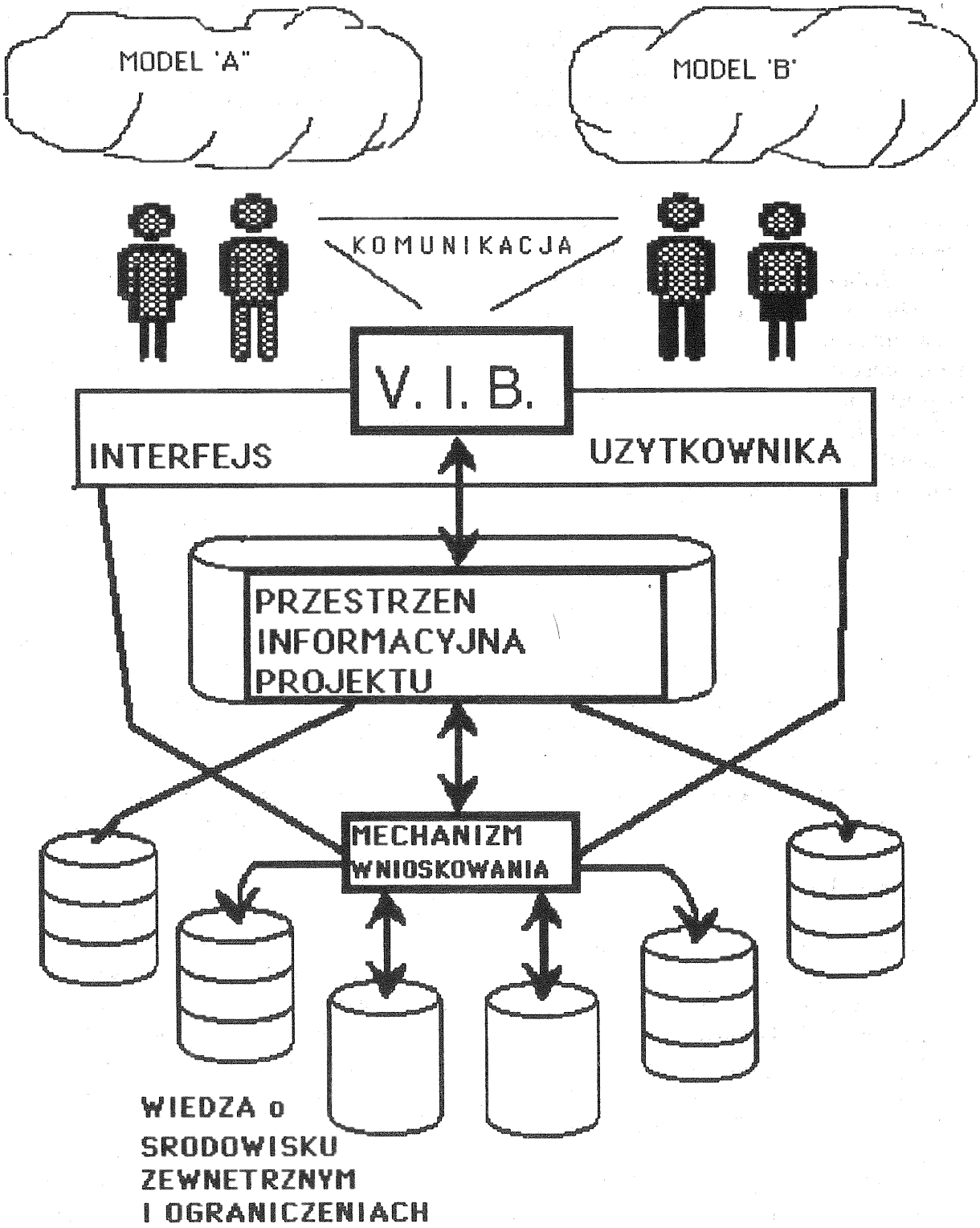
Uczestnictwo użytkowników w fazie analizy potrzeb pozostaje obecnie poza jakąkolwiek dyskusją. Po krótkim szkoleniu użytkownicy akceptują konwencję opisu organizacji w postaci DPD. Częstotliwość zmian wprowadzanych do diagramów czyni bardzo pracochłonnym opracowywanie zmian i możliwość ich dokonywania przy wykorzystaniu edytora diagramów czyni tę pracę znośną. Analiza spójności dokonywana automatycznie pozwala na podniesienie jakości i skrócenie pracy. Następny moduł - baza danych projektowych - pozwoli na dalsze usprawnienie transferu zbieranych informacji i poprawę ich jakości [28].

W wyniku zastosowania tych podstawowych modułów całego pakietu można oczekiwać znacznej poprawy komunikacji między analitykami a użytkownikami. Będzie temu towarzyszyć poprawa jakości gromadzonych danych, które będą na bieżąco rejestrowane podczas wspólnych sesji użytkowników i analityków (rysunek 2).

Metodyka zakłada uwzględnianie w razie potrzeby 'miękkich' wstępnych faz modernizacji SI. Również w tych fazach powinno być dostępne wspomaganie komputerowe ze względu na zakres gromadzonych i analizowanych danych o organizacji oraz zakres komunikacji.
Tworzenie narzędzia wspomagania 'miękkiej' metodyki jest zadaniem o wiele trudniejszym ze względu na ograniczenia formalizacji, rozmytość, bogactwo spojrzeń na problemy, niepewność, ograniczenia organizacyjne jak i wymagania technologiczne (patrz: [1], [14]). Przewidywane potrzeby stosowania tego podejścia czynią jednak wielce celowymi wszelkie próby komputerowego wspomagania.

5. Wnioski

Przedstawione uwagi dotyczące trzech aspektów komunikacji w tworzeniu SI są sygnalizacją problemów, jakie pojawiają się w warunkach szybkiego wprowadzania TI przy jednoczesnych głębokich zmianach systemowych. Czytelnikowi pewne stwierdzenia mogą wydać się przesadzonymi, gdyż - jak można trafnie zauważyć - ewo-



Rys. 2. VIB-CASAD - pakiet komputerowego wspomagania analizy SI.

lucja zastosowań i metod ich tworzenia trwała całe ćwierćwiecze. Uwaga taka jest tyleż trafna, co myląca. Konieczność głębszej niż w krajach rozwiniętych zmiany - zmiany cywilizacyjnej z wprowadzaniem TI - będzie w dużym stopniu uzależniona od możliwości analizy i projektowania nowych rozwiązań organizacyjnych wspomaganych przez TI.

Sytuacja będzie zmuszała firmy do dostosowywania organizacji do wymagań otoczenia. Jednym z nich jest już dostrzegana potrzeba zmian wzorców komunikowania się w organizacji. Komputerowe wspomaganie procesów informacyjnych w organizacji jest pierwszym krokiem w jej usprawnianiu. Kolejny krok to wspomaganie pracownika w komunikowaniu się. Służą temu współczesne interfejsy użytkownika [21].

Tworząc i adaptując dziś narzędzia komputerowego wspomagania prac na stanowisku, wspomagania decyzji, 'automatyzacji' biura, czy wspomagania analizy i projektowania SI, trzeba pamiętać o tym, że są one (były?) rozwiązaniami o ograniczonych możliwościach. Prawdziwe wyzwania wiążą się ze wspomaganiem pracy twórczej, która opiera się na komunikacji.

Bibliografia:

[1] Avison D. E., Golder P. A., Shah H. U.: *A Tool Kit for Sift Systems Methodology*, w: Kendall K. E., Lyytinnen K., DeGross J. I, (eds.): *The Impact of Computer Supported Technologies on Information Systems Development*, IFIP Transactions, A-8, IFIP - North Holland, Amsterdam - London - New York - Tokyo 1992.

[2] Baskin O. W., Aronoff C. E.: *Interpersonal Communication in Organizations*, Glenview Ill., Scott, Foresman & Company 1980.

[3] Bocchino W. A.: *Systemy informacyjne zarządzania*, WNT, Warszawa 1975.

[4] Bovée C. L., Thill J. V.: *Business Communication Today*, Random House Inc., New York 1986.

[5] Flaatten P. O., McCubbrey J. D., O'Riordan P. D., Burgess K.: *Foundations of Business Systems*, Ill. The Dryden Press, Chicago 1989.

[6] Fisher D.: *Communication in Organizations*, 2-nd Ed., West Publishing Company, Minneapolis/St.Paul 1993.

[7] Hanna M. S., Wilson G. L.: *Communicating in Business and Professional Settings*, 2-nd Ed., Random House Inc., New York 1988.

[8] Hayder G., Kuraś M.: *Change in Transitional Countries.: Main Obstacles and Method*, Journal of East-West Business, 1992 (to appear).

[9] Hayder G., Kuraś M.: *Modernizacja systemu informacyjnego jako zmiana organizacyjna*, Zeszyty Naukowe Akademii Ekonomicznej, Kraków 1993 (w druku).

[10] Huseman R. C., Lahiff J. M., Hatfield J. D.: *Business Communication. Strategies and Skills*, The Dryden Press, Chicago, Ill. 1981.

[11] Iaccocca L.: *Autobiografia*, Warszawa 1990.

[12] Kendall K. E., Kendall J. E.: *Systems Analysis and Design*, 2-nd Ed., Englewood Cliffs, Prentice-Hall Inc., 1992.

[13] Kendall K. E., Kendall J. E.: *Existence of Predominant Organizational Metaphores and Their Implications for Information Systems Development*, Rector's Lectures, No. 3, Akademia Ekonomiczna, Kraków 1992.

[14] Kerola P., Oinas-Kukkonen H.: *Hypertext System as an Intermediary Agent in CASE Environment*, w: Kendall K. E., Lyytinnen K., DeGross J. I. (eds.): *The Impact of Computer Supported Technologies on Information Systems Development*, IFIP Transactions, A-8, IFIP - North Holland, Amsterdam - London - New York - Tokyo 1992.

[15] Kotarski J.: *Zamki na lodzie*, Wywiad z drem A. Jankowskim, ComputerWorld, Nr 10 (1993/03/01).

[16] Kuraś M.: *Problemy komunikacji i wewnątrz zespołu*. Informatyka, Vol. 26, Nr 10 i 11, 1991.

[17] Kuraś M.: *Koncepcja komputerowego wspomagania analizy i projektowania systemów informacyjnych*, Zeszyty Naukowe AE, Nr 286, Kraków 1992.

[18] Kuraś M.: *Uczmy się budować zamki*, ComputerWorld, Nr 14 (1993/04/05).

[19] Kuraś M., Sarga D., Zaliwski A.: *Metoda planowania i tworzenia systemów informacyjnych zarządzania*, Raport z pracy badawczej 29/ISEI/ 7/91/, Katedra Informatyki, Akademia Ekonomiczna, Kraków 1992.

[20] Lewis P. V., *Organizational Communication*, Columbus, Ohio, Grid, 1975.

[21] Marcus A.: *Human Communications Issues in Advanced Uls*, Communications of the ACM, Vol. 36, No. 4, (April 1993).

[22] Murphy H. A., Hildebrandt H. W., O'Neil S. A.: *Effective Business Communications*, 2-nd Canadian Ed., McGraw-Hill Ryerson Ltd., Toronto 1990.

[23] Peters Th. J., Waterman R. H., Jr.: *In Search of Excellence. Lessons from America's Best-run Companies*, Warner Books, New York 1984.

[24] Robbins S. P.: *Organizational Behavior: Concepts, Controversies, and Applications*, 5-th Ed., Englewood Cliffs, Prentice-Hall Inc., N. J. 1991.

[25] Rogers C., Roethlisberger F. J.: *Bariers and Gateways to Communication*, Harvard Business Review, Vol. 69, No 6, (November-December, 1991). (Originaly appeared in HBR, July-August 1951.)

[26] Sanderlin R.: *Information is not Communication*, Business Horizons, (March-April 1982).

[27] Wilkinson C. W., Wilkinson D. C., Vik G. N.: *Communicating Through Writing and Speaking in Business*, 9-th Ed., Irwin, Homewood Ill. 1986.

[28] Zaliwski A., Kuraś M.: *CASAD - nowe podejście do modernizacji SI*, Piąta Jubileuszowa Wyższa Międzynarodowa Górska Szkoła PTI, Szczyrk 1993.

Andrzej Zaliwski, Marian Kuraś
Katedra Informatyki Akademii Ekonomicznej w Krakowie

CASAD - nowe podejście do modernizacji SI

Nie można zrozumieć procesu przez zatrzymanie go.
Zrozumienie musi podążać z biegiem procesu,
musi przyłączyć się i płynąć razem z nim.

Frank Herbert: "Diuna"

1. Wprowadzenie

Podstawowym celem referatu[1] jest prezentacja prototypu oprogramowania narzędziowego przeznaczonego do analizy systemów informacyjnych.

Istnieje szereg istotnych barier tworzenia i rozwoju SI. Ważniejsze z nich to trudności techniczne wynikające ze złożoności systemu oraz trudności techniczne, ludzkie i organizacyjne obejmujące m.in. komunikację z użytkownikiem w zespole projektowym, niezrozumienie potrzeb informacyjnych organizacji oraz jej celów strategicznych. Jest to o tyle trudniejsze, że często wśród samych użytkowników nie ma zgodności co do celów i wymagań wobec modernizowanego systemu informacyjnego.

Efektem prac jest pełna koncepcja i moduły główne środowiska projektowego typu ES-GDSS o roboczej nazwie VIB-CASAD. Niniejsza publikacja prezentuje podstawowe założenia tego programu. VIB-CASAD w znacznym stopniu powinien umożliwić pokonanie trudności i zakłóceń w przebiegu prac projektowych, na jakie natknęli się autorzy korzystając z metodyki ISAD [25].

2. Proces projektowania systemów informacyjnych

Projektowanie systemów informacyjnych[2] (SI) jest wciąż uznawane przez licznych technologów za czynność zbędną, bądź ograniczającą się do opisu istniejącego SI. Dzieje się tak wszędzie, gdzie stosuje się technikę informacyjną (TI). Różna jest jednak skala zjawiska, różne są także warunki, w jakich wprowadza się technikę informacyjną, przygotowanie tak użytkowników jak i twórców SI, ich doświadczenia zawodowe oraz efekty. Odmienne są przede wszystkim potrzeby użytkowników i one właśnie decydują o zaawansowaniu metod tworzenia jak i samych SI.

W kolejnych punktach rozważamy istotę procesu projektowania oraz doświadczenia tworzenia SI na podstawie literatury jak również własnej praktyki. Mają one wprowadzić Czytelnika do omówienia propozycji metodycznych oraz wspomagania procesu tworzenia SI.

2.1. Projektowanie systemów informacyjnych

Projektowanie SI jest w swej zasadniczej fazie - analizie - transferem wiedzy między

[1] Niniejszy referat jest wynikiem prac wykonanych w ramach tematów badawczych 13/KI/5/92/S i 14/KI/5/92/S sfinansowanych przez AE w Krakowie.

[2] Autorzy nie rozróżniają projektowania systemów informacyjnych i informatycznych. To drugie dotyczy jedynie technologii, której nie można separować od całego otoczenia organizacyjnego.

uczestnikami procesu: użytkownikami i analitykami. Ma to na celu opanowanie złożoności systemu będącego przedmiotem prac poprzez analizę logiczną i strukturyzację (patrz: [48], [49]). Analiza ma prowadzić do porządkowania postrzeganej rzeczywistości, jej uproszczenia poprzez zreformułowanie wstępnie postawionego problemu [32]. Temu służą nowe podejścia do analizy systemów - jak w latach '70 i '80 podejście strukturalne a obecnie obiektowe [8].

Modelowanie jako metoda i technika akwizycji oraz transferu wiedzy o SI pojawiło się z końcem lat '70. Od tego czasu powstało wiele metod, które pozwalają na prezentację wszelkich aspektów, jakie należy uwzględnić w toku analizy SI. Można przy tym łatwo zauważyć, że znaczny nacisk kładzie się na aspekt organizacyjny i społeczny. W pierwszym przypadku wymusza to konieczność podporządkowania zastosowań potrzebom organizacji, która wprowadza i rozwija zastosowania nowoczesnej techniki TI po to, by uzyskać wymierne korzyści i by zdobywać przewagę konkurencyjną[3]. Uwzględnienie aspektów społecznych jest koniecznością wynikającą z podporządkowania zastosowań potrzebom użytkowników oraz z potrzeby ich uczestniczenia w procesie tworzenia SI.

Rozwój zastosowań TI w krajach rozwiniętych prowadzi do głębokich zmian organizacyjnych. Wpierw są to przesunięcia zastosowań do użytkowników[4] a następnie ich wprowadzanie w celu dokonywania zmian organizacyjnych dostosowujących firmę do wymagań otoczenia (patrz: [38], [42], [45]). Zmiany zachodzące w organizacjach stają się możliwe dzięki zastosowaniom TI i zarazem wymuszają dalsze zmiany. Jedną z istotnych zmian w funkcjonowaniu organizacji jest traktowanie informacji jako kluczowych jej zasobów. To stwarza warunki i zarazem potrzebę rozwoju metod, technik i narzędzi tworzenia SI.

Wśród podejść do projektowania zyskują na znaczeniu podejścia *miękkie*[5] zapewniające uwzględnienie czynników, które nie poddają się skutecznej kwantyfikacji i formalizacji *(twarde podejście systemowe)*. Przydatność miękkiego podejścia systemowego jest powszechnie uznawana, przy czym w projektowaniu SI jest stosowane w kombinacji z podejściem twardym nieodzownym do stworzenia rozwiązania technicznego. Wstępna faza jednak wymaga odkrycia licznych faktów organizacyjnych, które przy podejściu *twardym* nie mogą być skutecznie zidentyfikowane i opisane.

2.2. Doświadczenia projektowania systemów informacyjnych

Pojęcie projektowania pojawiło się w praktyce zastosowań informatyki jeszcze w latach '70. Jednocześnie w tejże praktyce obserwowaliśmy, że stosowane podejścia nie odpowiadały współczesnemu rozumieniu projektowania. Można byłoby określić to *projektowaniem przez rysowanie* (patrz: [11], [20]). Ograniczoność potrzeb

[3] Podręczniki systemów informacyjnych oraz ich projektowania zawierają zwykle rozdział poświęcony znaczeniu SI w osiąganiu przewagi konkurencyjnej (np.: [21], [9]). Takich korzyści oczekuje się od dzisiejszych zastosowań w organizacjach, których celem jest przetrwanie. Jednocześnie literatura z zakresu zarządzania strategicznego akcentuje znaczenie informacji i SI (np.: [5], [34]).

[4] Przetwarzanie u użytkownika (*EUC - end user computing*) to od połowy lat '80 coraz częstsza forma tworzenia i wykorzystywania aplikacji. Oznacza ona - co należy podkreślić w celu uniknięcia nieporozumień - zaawansowanie użytkowników samodzielnie korzystających z techniki komputerowej, uczestniczących w tworzeniu aplikacji i w końcu tworzących samodzielnie własne aplikacje na stanowisku.

[5] *'Miękkie podejście systemowe'* zakłada uwzględnianie sposobu widzenia świata *(Weltanschaung)*, czyli wartości, przekonań, opinii i postaw uczestników zamierzenia w toku analizy (patrz: [6], [46]).

informacyjnych jednostek gospodarczych a także niski stopień zaawansowania technologicznego powodowały, że zakres i struktura SI były z góry określone. Zadaniem projektanta *systemu informatycznego* było w takiej sytuacji 'narysowanie' systemu i opracowanie dokumentacji dla programisty. Na przełomie lat '70 i '80 w praktyce projektanci zanikali.

Rozwój techniki mikrokomputerowej i jej dominacja w naszym kraju spowodowała, że w dużych ośrodkach jest niewielu projektantów o ile w ogóle utrzymali się. Jednocześnie obserwuje się od wielu lat fascynację technologią, od której użytkownicy oczekują cudu, zaś informatycy traktują ją jako wartość samą w sobie. Narzędzia informatyki są tak traktowane w środowisku, w którym są ciągłą nowością, w którym są postrzegane w pierwszym rzędzie jako nowinka, atrybut nowoczesności a nie narzędzie. Są one domeną profesjonalistów a nie użytkowników, co jest charakterystyczne dla wczesnych faz wzrostu zastosowań [33].

W ciągu trzech poprzednich dziesięcioleci byliśmy skazani na korzystania z wzorców przenoszonych z zachodu. Nauczyliśmy się przenosić wszystko dość bezkrytycznie. Trzeba przy tym zauważyć, że z nowości w tworzeniu SI w naszej praktyce przyjmują się te elementy, które mają bezpośrednie powiązanie z technologią. Są one zazwyczaj wykorzystywane przez entuzjastów w sposób wręcz hobbystyczny, czasami wbrew opiniom kierowników czy właścicieli firm.

Jednocześnie powszechną praktyką jest unikanie systematycznego podejścia do projektowania obejmującego kolejno: studium wykonalności, analizę potrzeb użytkowników oraz dokumentowanie tych faz prac. Dokumentacja najczęściej zawiera specyfikację sprzętu (typową w naszych warunkach) a ostatnio coraz częściej oprogramowania podstawowego, narzędziowego i aplikacji. Struktura cyklu życia systemu jest zupełnie niepodobna do tego, co obserwuje się w krajach rozwiniętych. Czas rozpoznania i analizy jest u nas wyjątkowo krótki, podczas gdy wiele czasu poświęca się na oprogramowanie i testowanie a następnie na wdrożenie. Długi czas oprogramowania i próbnej eksploatacji jest bezpośrednim skutkiem powierzchownej analizy.

Coraz częściej w ostatnich latach mamy dostęp do oprogramowania narzędziowego, wspomagającego tworzenie SI. Kosztowne oprogramowanie renomowanych firm zachodnich - podobnie jak wiele typów oprogramowania - staje się gadgetem, z którego może korzystać informatyk ale tworzone przy jego zastosowaniu aplikacje odpowiadają ograniczonym potrzebom użytkowników.

2.3. Potrzeby zmian w praktyce projektowania

Zmiany w gospodarce powodują przyspieszone (wciąż jednak wolne) uczenie się przez organizacje zasad zarządzania oraz korzystania z techniki informacyjnej. Zainteresowanie użytkowników techniką informacyjną oraz uczenie się jej wykorzystania zwiastuje przyspieszenie rozwoju zastosowań. Będą to - jak należy przypuszczać - inne zastosowania niż obecne, gdyż elementarne potrzeby zostaną zaspokojone. Będzie się wzmagał popyt na aplikacje wspomagające zarządzających przy zwiększaniu wymagań wobec dotychczasowych. Należy sądzić, że będzie się to odbywało w mniej komfortowych warunkach, przy ostrzejszych wymaganiach, szczególnie finansowych.

Oznacza to, że pojawi się zapotrzebowanie na metody projektowania zapewniające systematyczne podejście do **projektowania systemów informacyjnych organizacji**. Zastąpi to dotychczas dominujące podejście do komputeryzacji firmy, komórek organizacyjnych czy stanowisk, polegającego na doraźnym doborze lub tworzeniu oprogramowania.

Pojawi się zapotrzebowanie na wiedzę[6] a przede wszystkim na praktyczne umiejętności rozpoznawania potrzeb użytkowników i analizę tych potrzeb, prowadzącą do określenia wymagań wobec systemu informacyjnego. Rozwiązania techniczne będą w coraz większym stopniu podporządkowane potrzebom funkcjonowania organizacji i zarządzania. Zastosowania będą stopniowo oceniane na podstawie kryteriów ustalonych przez użytkowników.

2.4. Wnioski z doświadczeń i obserwacji trendów rozwojowych

W przewidywaniu zmiany podejścia do tworzenia SI autorzy od kilku lat pracują nad stworzeniem metodyki i komputerowego jej wspomagania. Przyjęto założenie, że metodyka powinna:

- obejmować całość prac wykonywanych podczas modernizacji SI;
- być raczej przewodnikiem po metodach, technikach i narzędziach, jakie są dostępne w literaturze, niż nową metodyką zalecającą jedyny sposób postępowania i jedynie dostępne techniki i narzędzia;
- oferować konwencję graficznego opisu struktur SI wymuszającą strukturyzację problemów;
- wymuszać standardowe rozwiązania w ramach dowolności typowej dla miękkiego podejścia systemowego;
- oferować komputerowe wspomagania najbardziej pracochłonnych czynności prac analitycznych;
- być sprawdzona przed przystąpieniem do tworzenia wspomagania komputerowego;
- zawierać dostępną dla użytkownika wiedzę o metodach, technikach i narzędziach (podręcznik, kontekstową 'ściągę' oraz diagnostykę błędów);
- pozostać rozwiązaniem otwartym - w szczególności w zakresie wspomagania;
- przekształcać się w system oparty na wiedzy, wspomagający współpracę użytkowników i informatyków.

Wcześniejsze próby stosowania oryginalnych metodyk zachodnich (ISAC, Gane + Sarson) zakończyły się niepowodzeniem. Jego przyczyny były dwojakiego rodzaju. Pierwszą był negatywny syndrom NIH *(Not Invented Here)* powodujący, że użytkownicy nie ufają metodzie stworzonej i wykorzystywanej w innych warunkach. Druga grupa przyczyn niepowodzeń to niedostępność narzędzi CASE (lata 80). Stało się to przyczyną podjęcia przez nas prac nad własną metodyką a następnie nad jej wspomaganiem komputerowym.

3. Nasza koncepcja wspomagania projektowania

W celu omówienia naszej koncepcji wspomagania projektowania należy wyjść od określenia właściwej **strategii rozwoju oprogramowania**.
Każda organizacja nie działa w próżni, lecz w zmieniającym się dynamicznie środowisku. Warunki działania organizacji są wypadkową czynników zewnętrznych (gra rynkowa, polityka rządu, działania organizacji gospodarczych) jak i wewnętrznych (komplikacja procesu wytwarzania, złożoność struktury wewnętrznej, fluktuacja personelu, kwalifikacje personelu - to tylko niektóre czynniki). W przypadku, gdy tworzymy dla takiej organizacji system informacyjny (oznaczmy go literą A), ***w praktyce będziemy mieli do czynienia z wielowymiarową przestrzenią informacyjną opisującą organizację i jej środowisko.***

Tworzony jest projekt systemu informacyjnego A, który stanowi wielowymiarowy podzbiór przestrzeni informacyjnej. Jest to "migawka" stanu

[6] W Polsce nie można obecnie kupić **podręcznika projektowania SI**. Inne opracowania z tej dziedziny są również nieliczne w przeciwieństwie do niezliczonych opracowań dotyczących sprzętu i oprogramowania. Niech to świadczy o stanie systemów informacyjnych organizacji oraz naszych możliwości ich tworzenia w warunkach generalnej zmiany zarządzania.

organizacji w chwili przeprowadzania analizy. Pomijając błędy jakie mogą zostać popełnione podczas analizy, obraz uzyskany w wyniku analizy - choćby z powodu opóźnień przy opracowywaniu raportu końcowego - może być już w pewnym stopniu nieaktualny. Implementacja systemu A, która także zabiera nieco czasu jest dokonywana na podstawie obrazu uzyskanego w wyniku analizy dokonanej w przeszłości (niedalekiej ale przeszłości).

Specyfikacja istniejącego SI stanowi pewien model rzeczywistości istniejącej w chwili tworzenia systemu. Po pewnym czasie różnica między tym modelem a zmieniającą się rzeczywistością jest tak duża, że konieczna staje się modernizacja SI. W wyniku modernizacji powstaje nowy system B będący pewnym modelem nowej sytuacji w jakiej znalazła się organizacja po pewnym czasie t.

Problem polega na przejściu od stanu aktualnego do pewnego żądanego stanu. Przy czym stan żądany niekoniecznie musi być dobrze zdefiniowany. Zarówno analityk jak i użytkownik wytwarzają w swoim umyśle model będący ich wyobrażeniem stanu żądanego. Model analityka jest oparty o jego:

- wiedzę o możliwościach sprzętu i oprogramowania,
- znajomość metodyki, narzędzi i doświadczenie zawodowe.

Użytkownik natomiast ma:

- wyobrażenie jak system powinien funkcjonować (zestaw wymagań i założeń dotyczących funkcjonowania systemu),
- zestaw "pobożnych życzeń" dotyczących dodatkowych funkcji systemu,
- wiedzę o środowisku wewnętrznym i funkcjonowaniu organizacji.

W wyniku procesu projektowania następuje konfrontacja obu tych modeli. W wyniku wzajemnej interakcji uczestników procesu powinno dojść do maksymalnego nałożenia się tych modeli na siebie. Członkowie zespołu zanim osiągną swój cel muszą pokonać wiele przeszkód. Wymieńmy niektóre z nich:

- Niezrozumienie problemu, u podstaw którego tkwią trudności komunikacyjne między ludźmi różnych zawodów.
  Jest to także powiązane z komunikacją gubienie informacji (niezapisywanie koncepcji pośrednich, idei, pomysłów), wynikające z trudności związanych z prawidłowym dokumentowaniem przebiegu procesu projektowego. Niezrozumienie potrzeb informacyjnych użytkowników [28].

- Redundancja i niekompatybilność form zapisu dokumentacji związanej z różnymi etapami projektowania i rozwoju systemu [14]-[19].
  Powstało wiele technik dla dokumentowania i formalizacji poszczególnych faz rozwoju systemu oprogramowania. Zawierają one różne rodzaje dokumentacji w języku naturalnym, diagramy struktury, diagramy przepływu danych i sterowania, słowniki danych itp. W miarę jak system oprogramowania staje się coraz bliższy realizacji, rośnie liczba wspierających dokumentów. Pomiędzy wszystkimi wyżej wymienionymi dokumentami istnieje silna zależność prowadząca do znacznego pokrywania się informacji. I to zarówno wewnątrz poszczególnych dokumentów jak też i między nimi. Notacja lub język użyty do wytworzenia każdego z tych dokumentów często różni jeden dokument od drugiego (niekompatybilność zapisu informacji). Ponieważ formalna translacja między różnymi reprezentacjami oprogramowania najczęściej nie istnieje, zapewnienie by te wszystkie dokumenty były spójne jest trudne.

- Olbrzymia złożoność systemów opro-

gramowania.

System oprogramowania dziedziczy złożoność swoich aplikacji. (Złożoność jest nieodłączną cechą oprogramowania). System rozwoju oprogramowania także sam w sobie jest złożony.

- Brak dobrej metodyki dopasowanej do każdej sytuacji.

  Jak to wynika z doświadczenia analityków nie ma metodologii, która byłaby właściwa do każdej sytuacji [2]. Zastosowanie, czy inaczej odpowiedniość danej metodologii zależy od kontekstu, to jest samej organizacji, użytkowników oraz analityków którzy rozwijają SI. Ponadto istnieją setki opublikowanych często bardzo podobnych metodologii, często niektóre organizacje używają swoich własnych metodologii wymyślonych i narzuconych w danej organizacji [26].

Czynione są próby zmniejszenia kosztów modernizacji lub raczej tworzenia nowych SI przez opracowywanie coraz to nowszych wyszukanych metodologii. Jedną z metod jest tworzenie specyfikacji SI w taki sposób by ich maksymalnie duże fragmenty mogły być wykorzystane w wielu podobnych SI [3]. Przykładem tworzenia tych tzw. 'reusable specification' jest [29].

W literaturze istnieje szereg prób rozwiązania każdego z wymienionych uprzednio problemów. I tak na przykład teorie rozwoju systemu opisują proces uzyskiwania wiedzy na temat problemu, oraz dzięki sformułowaniu odpowiedniego modelu, stanowią przewodnik do konstrukcji rozwiązania. Takim przewodnikiem do konstrukcji rozwiązania są np. podejścia typu *multiview* [1], [2]. Polega to na stworzeniu ogólnej ramy, która pomaga dokonać selekcji metodyki jaka powinna być zastosowana dla każdego z etapów projektu. Znalezienie doskonałej metodologii jest iluzoryczne dlatego też powinno się tworzyć metody projektowania i rozwoju SI pozwalające na dobranie różnych metodologii składowych używanych na poszczególnych etapach rozwoju i tworzenia systemu informacyjnego.

Jak wspomnieliśmy wcześniej organizacja i jej środowisko są opisane przez wielowymiarową przestrzeń informacyjną. Istnieje olbrzymia różnorodność metodyk ale każda z nich opisuje tylko jeden lub rzadko kilka wybranych wymiarów. Przez analogię do geometrii analitycznej: możemy zbadać długość i szerokość prostokąta ale nic nie możemy powiedzieć na temat jego pola. Możemy zbadać każdy z wymiarów sześcianu oddzielnie, nie mając pojęcia o wyglądzie całości. Sytuacja jeszcze bardziej komplikuje się w przypadku większej liczby wymiarów. Tak więc podejście *multiview* również nie jest właściwym rozwiązaniem, gdyż jako efekt takiego działania otrzymujemy kilka 'równoległych' opisów tego samego zjawiska, z których każdy opisuje jego inny aspekt. Tego typu podejście nie zmienia nic!

W dalszym ciągu mamy szereg różnych dokumentów opisujących rozwijany system (tyle, że z przewodnikiem w postaci *framework*), dokumentów w których informacja jest reprezentowana różnymi zwykle niekompatybilnymi formami notacyjnymi. Oczywiście wielość tych ujęć być może pozwoli analitykowi wyrobić sobie we własnym umyśle lepszy model intuicyjny sytuacji z którą ma do czynienia. Jednakże to podejście nie rozwiązuje problemów z niekompatybilnością technik notacyjnych.

Wracając do systemów A i B, idealną sytuacją byłaby taka w której model A ewoluuje dynamicznie tak, że stale stanowi mniej lub bardziej dokładne odwzorowanie sytuacji w jakiej działa organizacja. Dobrym sposobem rozwiązania tego problemu wydaje się być tworzenie nowej specyfikacji określającej zmodernizowany system w sposób możliwie najtańszy. (By tak było musimy w nowej specyfikacji B zawrzeć te elementy starej specyfikacji A,

które nadal są aktualne.) Model A odzwierciedla pewien 'stan wiedzy' o funkcjonowaniu organizacji. Dynamiczne przejście od modelu A do B może następować poprzez uaktualnianie tej wiedzy. (Zdobywana wiedza w miarę ewolucji nawarstwia się na A, tak, że w końcu uaktualnione A staje się podzbiorem B). Pozostaje jeszcze kwestia opracowania narzędzia które mogłoby realizować wspomnianą powyżej transformację. Ponadto narzędzie to powinno zapewnić notację nadającą się do dokumentowania przebiegu procesu projektowego na każdym jego etapie.

Jako punkt wyjścia dla rozwiązania powyżej sformułowanego problemu potraktowaliśmy rozwój oprogramowania jako procesu pozyskiwania wiedzy [14]-[19]. Na odpowiednim poziomie byłaby to baza wiedzy zawierająca aktualnie znane informacje. Postęp następuje poprzez zdobywanie dalszych informacji i włączanie ich w strukturę bazy wiedzy, lub przez przetwarzanie informacji już posiadanej. Baza wiedzy o projekcie programistycznym jest nieformalna, składa się głównie z podręczników, diagramów, plików danych a wiedza istnieje na różnych poziomach, począwszy od założeń programu a kończąc na kodzie programu. Rzadko się zdarza by powiązania między tymi poziomami były formalne lub kompletne.

Strategia rozwoju oprogramowania [14]-[19] musi dostarczać metod dla modelowania systemu oprogramowania na wszystkich poziomach rozwoju. Na każdym poziomie model musi być precyzyjny, jednoznaczny i czytelny.

Nowa technika rozwoju oprogramowania wymaga kilku podstawowych cech:
- formalnej notacji,
- technologii, która wspiera i jest wspierana przez notację,
- metod reprezentacji dla obiektów i wiedzy wewnątrz/w obrębie danej dziedziny,
- reguł rozwoju procedur dla odwzorowania modeli pomiędzy reprezentacjami.

Zarówno notacja jak i reguły przejścia dla odwzorowania modeli pomiędzy reprezentacjami oraz metody reprezentacji obiektów w obrębie danej dziedziny są zapewnione przez infoMapy. Od strony technologicznej rozwój oprogramowania jest wspierany przez narzędzie typu CASE o roboczej nazwie VIB-CASAD.
Jeśli rozwój oprogramowania ma zostać potraktowany jako proces pozyskiwania wiedzy to zadaniem narzędzia wspierającego powinno być pozyskiwanie, gromadzenie oraz udostępnianie wiedzy. Dokładniej mówiąc są to analogiczne funkcje w stosunku do tych jakie spełnia typowy system zarządzania bazą danych (DBMS) ale w odniesieniu do 'wiedzy' a nie 'danych'. Tak dochodzimy do sformułowania zasad funkcjonowania narzędzia wspierającego konstrukcję/modernizację SI na każdym etapie.

4. CASAD - Środowisko projektowe typu ES-GDSS

Konstrukcja oprogramowania CASE może być traktowana jako przedsięwzięcie z pogranicza trzech dziedzin: zarządzania, technologii oraz otoczenia socjalno-społecznego. Ludzie działający w każdej z tych trzech dziedzin używają różnych charakterystycznych dla swojego zawodu języków i zakresów skojarzeniowych. Trudność porozumienia się różnych grup zawodowych między sobą bywa nie tylko dobrą okazją do dowcipów rysunkowych, ale również przyczyną niepowodzeń wielu kosztownych przedsięwzięć programistycznych.

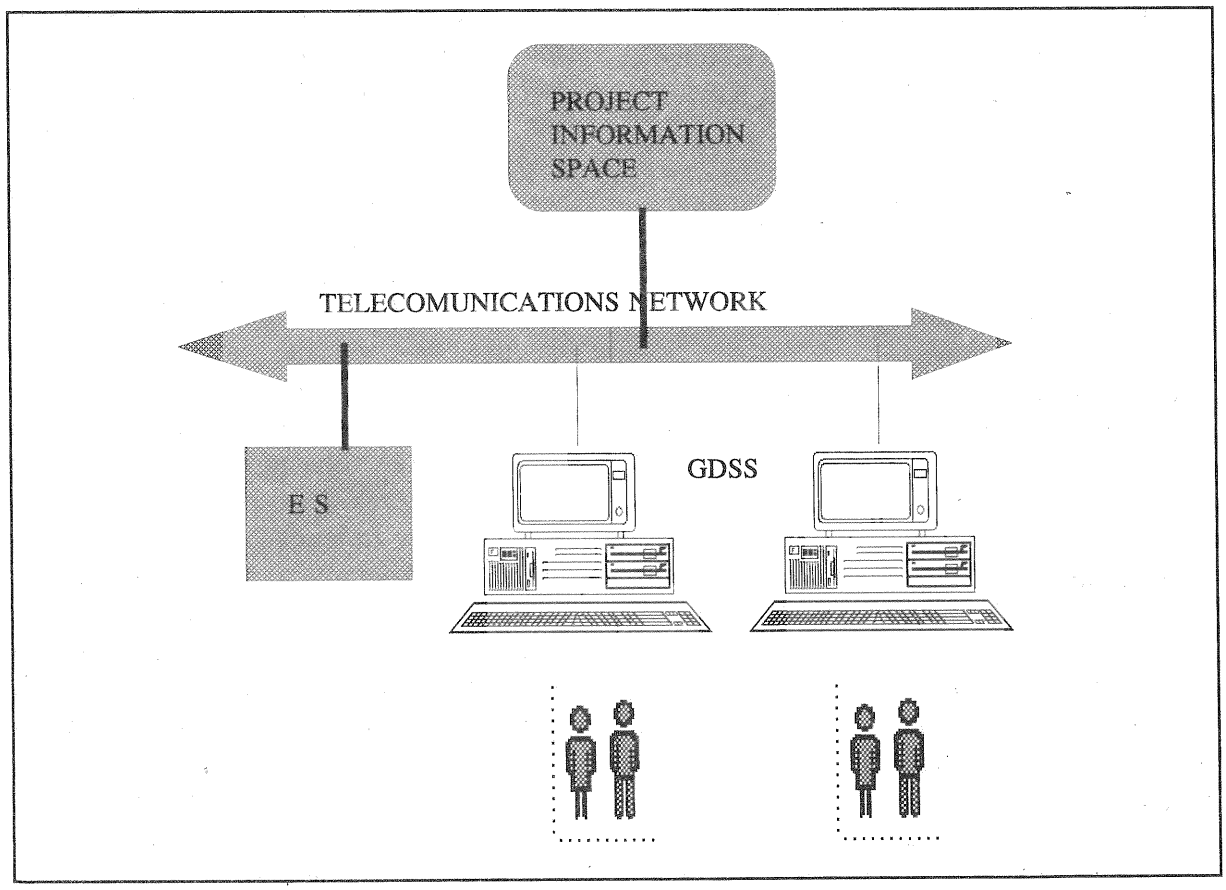
Zarówno użytkownik jak i projektant tworzą we własnej świadomości model będący wyobrażeniem tworzonego/modernizowanego systemu. By uniknąć błędów wynikających ze wzajemnego niezrozumienia konieczne jest nałożenie się tych modeli na siebie.

W tym celu konieczny jest szereg przedsięwzięć umożliwiających nałożenie się na siebie modeli użytkownika i analityka/projektanta. Temu celowi właśnie służą różnego rodzaju metodologie. Buduje się także ciąg kolejno udoskonalanych prototypów i konsultuje się je z użytkownikiem. Jednakże droga do momentu w którym projektant usłyszy od użytkownika upragnione: "Tak, to jest właśnie to o co mi chodziło.", jest bardzo długa. Tak więc większość problemów zawiązanych z projektowaniem można sprowadzić do trudności we wzajemnej komunikacji.

Wyobraźmy sobie system ze sztuczną inteligencją, który tworzy model systemu na podstawie wprowadzonych do niego informacji. Możliwe byłoby wtedy wprowadzanie do niego informacji w różnych postaciach, właściwych dla danego etapu tworzenia i rozwoju systemu. Informacje te byłyby konwertowane wewnątrz programu na jednolitą postać i gromadzone w pojemniku, który nazwiemy **przestrzenią informacyjną projektu** (PIP). Zakładamy, że przestrzeń informacyjna projektu istnieje przez cały czas tworzenia projektu. Jej stan odzwierciedla bieżący stan projektu. Przestrzeń informacyjna projektu jest wielopoziomowym i wielowymiarowym modelem badanego wycinka rzeczywistości. Z przestrzenią informacyjną współpracuje cały szereg pomocniczych baz danych zawierających dodatkowe informacje.

Model systemu w Przestrzeni Informacyjnej może być potraktowany jako pewnego rodzaju zbiór hipotez i będzie stanowić, niezależnie od innych możliwych działań, przedmiot badań prowadzonych przez system doradczy (ES), w którego bazach wiedzy powinny znaleźć się:

- Model środowiska zewnętrznego tj.
  - informacje o uwarunkowaniach zewnętrznych w jakich działa organizacja, które mają(/mogą mieć) wpływ na jej działanie.



RYS. 4.1. Środowisko projektowe typu ES-GDSS.

- Model środowiska wewnętrznego systemu informacyjnego tj.
  - Fakty o analizowanej organizacji,
  - Specyfikacja potrzeb informacyjnych,
  - Wymagania użytkownika/klienta.

Ponadto powinny one zawierać:
- ograniczenia i warunki brzegowe projektu,
- informacje o podobnych projektach,
- podręcznik(i) metodyki.

Wprowadzając do rozważanego systemu użytkownika (podejście partycypacyjne) i analityka dochodzimy do sformułowania koncepcji środowiska projektowego typu ES-GDSS (rys. 4.1) [36].

Zadaniem takiego środowiska będzie wspomaganie prac projektowych związanych z konstrukcją i modernizacją SI na każdym etapie jego tworzenia.

5. Architektura systemu CASAD

Przeanalizujmy architekturę systemu VIB-CASAD w powiązaniu z realizowanymi funkcjami. Schemat funkcjonalny środowiska projektowego typu ES_GDSS jest przedstawiony na rysunku 5.1.1.

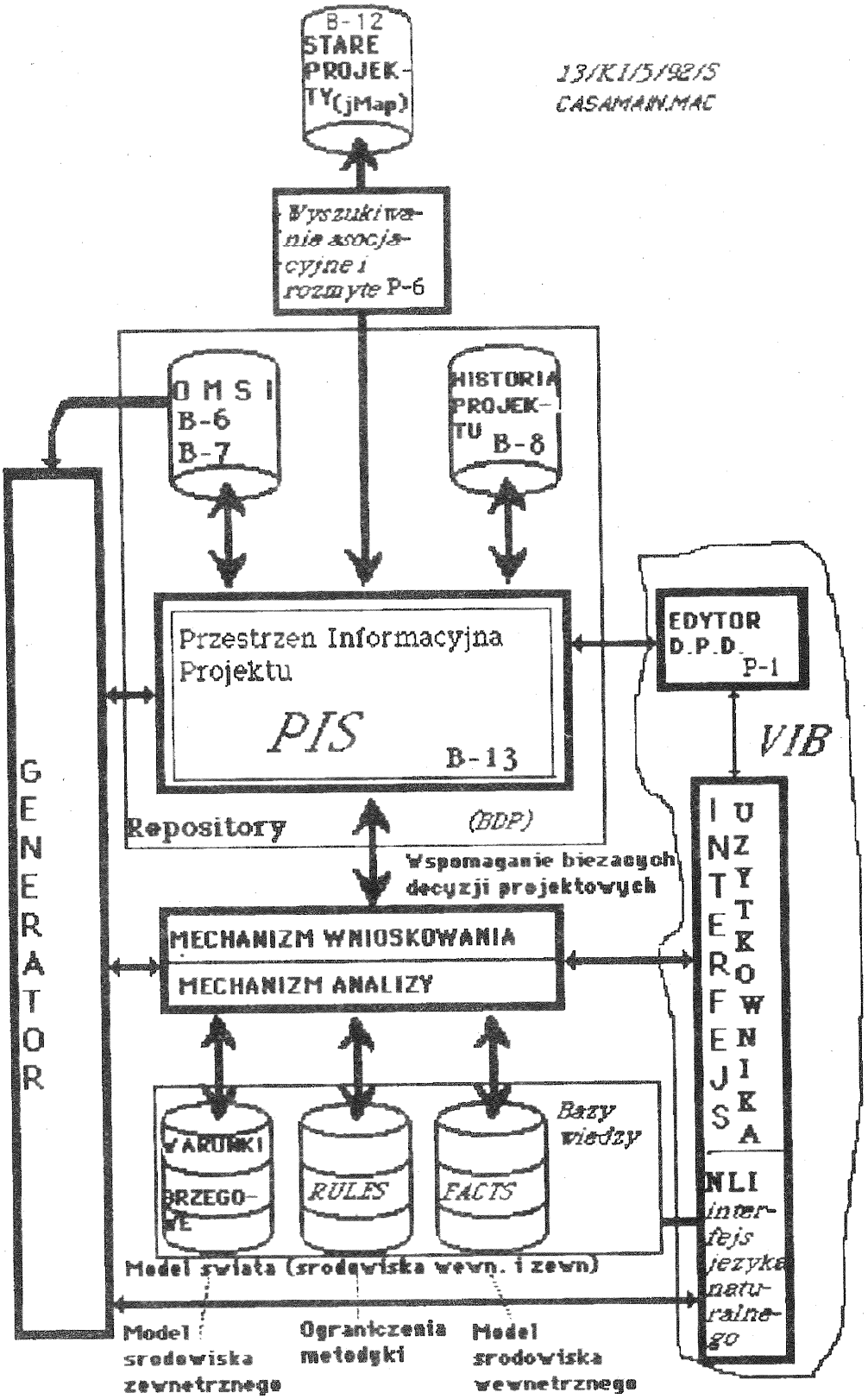
By system mógł spełnić zakładane cele niezbędne jest:

- **pozyskiwanie wiedzy** od członków zespołu projektowego,
- **gromadzenie wiedzy**: gromadzenie faktów uzyskanych w trakcie sesji, w szczególności od użytkowników, kontrola poprawności i redundancji tych faktów.
- **wyszukiwanie i udostępnianie wiedzy** o przypadkach podobnych do analizowanego. Doradztwo. Sugerowanie jakie dane mogą być jeszcze potrzebne.
- przechowywanie i udostępnianie 'wiedzy podręcznikowej'
- **kontrola merytoryczna i składniowa,** w celu wykrywania sprzeczności w nowo projektowanym systemie (np. przez użycie InfoMapy).
- **tworzenie specyfikacji SI** na podstawie danych zebranych w bazach podczas wielu sesji (zakładana jest pewna ingerencja człowieka).
- generacja aplikacji, co nastąpi w dalszych etapach prac.

Na rysunku 5.1.1 są uwidocznione podstawowe moduły wchodzące w skład VIB-CASAD. Pozyskiwanie wiedzy jest realizowane przez VIB, gromadzenie wiedzy przez Przestrzeń Informacyjną Projektu i Bazy wiedzy. Wyszukiwanie i udostępnianie wiedzy odbywa się również za pośrednictwem VIB oraz zespołu specjalizowanych interfejsów. Elementy kontroli merytorycznej i składniowej występują w każdym z modułów.

5.1. Pozyskiwanie wiedzy. VIB - "Virtual Intelligent Board"

Głównym mechanizmem służącym pozyskiwaniu wiedzy w trakcie sesji zespołu projektowego jest VIB - Virtual Intelligent Board. Istniejące programy tego typu (np. 'MathCAD'), pozwalają na dokonywanie przeliczeń wg podanych wzorów, przy czym każda zmiana danych wejściowych lub treści wzoru powoduje automatyczne przeliczenie wszystkich pozostałych wzorów tak, że końcowy wynik jest stale aktualny. Istnieje możliwość tworzenia wykresów, również automatycznie aktualizowanych. Program MathCAD jest więc czymś w rodzaju specjalizowanego z punktu widzenia matematyków arkusza kalkulacyjnego z tym, że poszczególne formuły są ze sobą powiązane nie przez adresy komórek lecz przez użycie tych samych nazw zmiennych. W wielu filmach S-F ekran komputera ma postać biurka, gdzie powierzchnia ekranu stanowi jednocześnie blat biurka i umożliwia pisanie po niej. Łącząc ze sobą oba rozwiązania otrzymujemy coś w rodzaju komputera dla naukowca. Komputer na ekranie którego tak jak na tablicy można wypisywać różnego rodzaju wzory (np. nowej teorii matematycznej) a komputer będzie oceniał lub doradzał inne rozwiąza-



Rys. 5.1.1. Architektura systemu VIB-CASAD.

nia na podstawie wiedzy zgromadzonej w jego bazach. Tak skonstruowany komputer możemy umieścić na ścianie zamiast tablicy i nasz *decision room* udostępnić użytkownikowi i projektantowi, którzy wykorzystają to wyposażenie do modelowania SI. VIB od strony logicznej jest koncepcją pracy zespołu projektowego opartą na wykorzystaniu inteligentnej tablicy zrealizowanej np. przy pomocy komputera z panelem.

To co proponujemy w ogólnym zarysie jest szczególnym rodzajem GDSS z tą różnicą, że mamy jedną tablicę - ekran kinowy - zamiast sieci komputerowej. Sugerujemy również (przynajmniej na razie) zastosowanie tylko jednego stanowiska do wspomagania współpracy użytkownika i analityka.

System będzie mógł przechwycić jak najwięcej informacji wymienianych w trakcie pracy zespołu projektowego, jeśli dyskusja między członkami zespołu projektowego odbywać się będzie w oparciu diagramy oraz częściowo sformalizowane opisy w języku naturalnym prezentowane na VIB. Uczestnicy dyskusji w jej trakcie modyfikują w sposób interakcyjny zawartość baz wiedzy współpracujących z VIB.

Środowisko projektowe oparte o VIB ma wykrywać błędy logiczne i merytoryczne. Z założenia ma być 'cichym wspólnikiem' zespołu projektowego. VIB stanowi jedną z głównych części środowiska projektowego ES-GDSS. Głównym zadaniem VIB jest przechwytywanie i zbieranie informacji wymienianych między członkami zespołu projektowego a także stanowienie inteligentnego interfejsu między zespołem projektowym a innymi programami wchodzącymi w skład zintegrowanego środowiska projektowego typu ES-GDSS [36]. Tak jak to określił Martin w [30], jest to rodzaj batyskafu dopuszczającego tylko potrzebne informacje.

Z fizycznego punktu widzenia VIB jest zespołem specjalizowanych interfejsów przeznaczonych do przechwytywania informacji wymienianych w trakcie pracy zespołu projektowego. Warunkiem spełniania przez VIB założonych funkcji - od strony realizacyjnej - jest przechwytywanie przez VIB informacji przesyłanej różnymi kanałami komunikacyjnymi między uczestnikami procesu projektowego. Dlatego też VIB musi z założenia stanowić główne medium komunikacji między członkami zespołu projektowego. W tym celu VIB jest wyposażony w następujące elementy składowe:

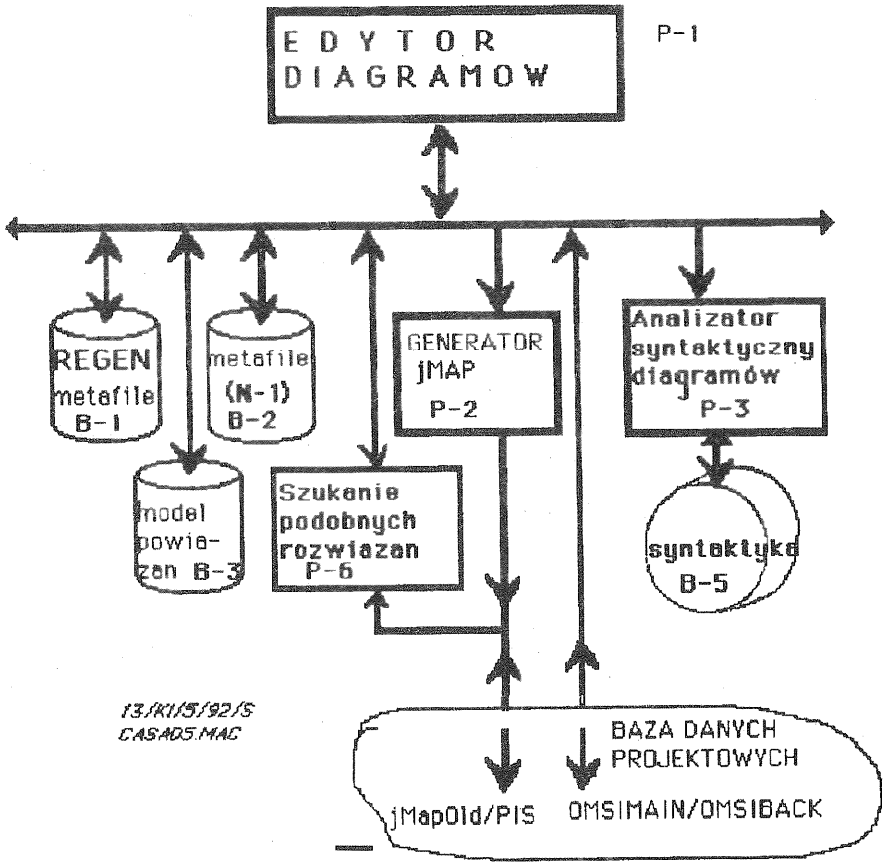
- edytor diagramów, omawiany w następnym podrozdziale oraz
- interfejs języka naturalnego, (którego wstępne założenia są opracowywane i zakłada się jego implementację w następnym etapie),
- interfejsy pomocnicze,
- podsystem informacyjny HELP,
- podręcznik użytkownika.

Te podstawowe moduły obsługujące komunikację i wykorzystanie zasobów będą stopniowo rozszerzane w miarę jak do systemu będą wprowadzane kolejne funkcje oraz metody/techniki/narzędzia modelowania i analizy.

Edytor "DPD"

Edytor diagramów (DPD) jest wyspecjalizowanym edytorem graficznym, którego głównym zadaniem jest wspomaganie pierwszego etapu analizy tj. gromadzenie danych o rozpatrywanym SI (rys. 5.1.2). Edytor DPD działa w sposób interakcyjny i komunikuje się bezpośrednio z członkami zespołu projektowego. Komunikacja odbywa się poprzez użycie (i wspólne tworzenie) diagramów opartych o specjalnie do tego celu opracowaną metodykę.

Zaproponowana metodyka i notacja jest na tyle prosta, że pozwala na czynny udział użytkownika w projektowaniu SI. Doświadczenia wykazują, że użytkownicy przyswajają ją sobie bez kłopotu [23]-[28]. Dodatkowo użytkownicy przez swój czyn-



Rys. 5.1.2. Schemat funkcjonalny edytora diagramow DPD.

ny udział w zespole projektowym czują się potrzebni i mniej zagrożeni (jeszcze raz syndrom NIH), dzięki temu, że uczestniczą w tworzeniu modelu SI, mają wpływ na przyjmowane rozwiązania i że 'panują nad sytuacją'. Jest to ważny czynnik psychologiczny, mogący mieć wpływ na końcowy rezultat całego przedsięwzięcia. Nawet abstrahując od oprogramowania, metodyka i jej schematy rysowane na papierze już same w sobie stanowią dogodne narzędzie komunikacji w zespole projektowym [27], [37].

Schemat funkcjonalny edytora DPD jest przedstawiony na rysunku 5.1.2. Edytor DPD ma obsługiwać tworzenie diagramów przepływu danych opartych o metodykę obiektową a także wykonywać wszystkie operacje związane z przechowywaniem i wyszukiwaniem diagramów, z rejestracją wszystkich wykonywanych operacji graficznych oraz umożliwiać wycofywanie się do poprzednich etapów rysowania diagramu.

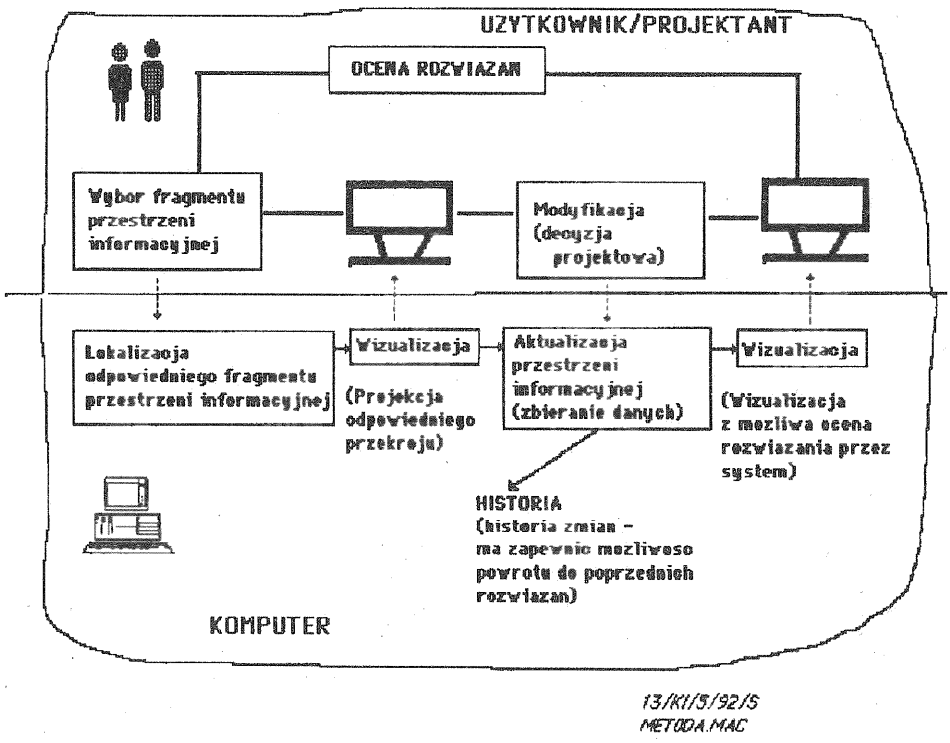
Edytor diagramów jako specjalizowany program typu CAD, ma umożliwiać dokonywanie takich samych operacji na diagramach jakie są przewidziane w programach typu CAD, a więc przesuwanie, przenoszenie fragmentów rysunku, wymazywanie itp. Ponadto musi spełniać szereg wymagań, których nie oczekuje się od typowego systemu typu CAD. Każda operacja dokonywana na diagramie, np. dorysowanie lub usunięcie obiektu jest sprzężona z odpowiednią operacją na bazie danych projektowych (a w szczególności z plikami OMSIMAIN i OMSIBACK omawianymi przy opisie bazy danych projektowych). W szczególności wszystkie operacje przeprowadzane na polu roboczym edytora diagramów natychmiast znajdują swoje odzwierciedlenie w Przestrzeni Informacyjnej Projektu (PiP). Edytor DPD jest

również przewidziany do działania jako jednostka samodzielna.

Reprezentacja całego diagramu jest przechowywana na dwa sposoby. Jeden raz jako zapis powiązań między obiektami diagramu reprezentowany przez struktury właściwe dla PIP/OMSI, drugi raz jako *metafile* programu Windows, tzn. jako plik rejestrujący wszystkie wydane komendy graficzne. Plik *metafile* będzie miał swoją kopię umożliwiającą odtworzenie historii poprawek. Oprócz tego istnieje baza diagramów oznaczona jako HISTORIA, której zadaniem jest przechowywanie przez pewien czas diagramów skasowanych oraz przechowywanie danych o przebiegu wprowadzanych zmian. Wszystkie te rozwiązania mają na celu zapewnienie możliwości cofnięcia się w trakcie procesu analizy do poprzednich rozwiązań, o ile zajdzie taka potrzeba.

Każdy diagram będzie przechowywany zasadniczo w postaci infoMapy. W przypadku załadowania do edytora diagramu w postaci infoMapy z dysku, zostanie przez diagram uruchomiona procedura optymalizacji rozmieszczenia. W efekcie diagram będzie wyświetlony w postaci zbliżonej do optymalnej ze względu np. na rozmieszczenie połączeń między obiektami diagramu. Możliwe jest także przechowywanie diagramu w postaci opisu graficznego w standardzie *metafile*. Dane przechowywane w postaci opisów graficznych będą wykorzystywane na etapie modyfikowania i do prezentacji diagramów. Diagram na ekranie w zależności od życzeń członków zespołu projektowego może być wyświetlany zarówno w formie infoMapy jak i formie graficznej.

Zagadnienie zaprojektowania edytora diagramów można - zdaniem autorów - sprowadzić do konstrukcji bazy danych. Prześledźmy jak może wyglądać cykl współpracy użytkownika lub analityka z interakcyjnym edytorem diagramów (rys. 5.1.3). Ten sam schemat może być zastosowany również w odniesieniu do wielu graficznych systemów interakcyjnych.



Rys. 5.1.3. Sposób współpracy z edytorem DPD.

Główną operacją przy pracy z systemami tego typu jest oglądanie przez analityka/użytkownika wybranego fragmentu diagramu a następnie (ewentualne) dokonywanie pewnych jego modyfikacji. Główną częścią cyklu przedstawionego na rysunku 5.1.3, przypadającą na komputer, jest pobranie i wizualizacja określonego fragmentu PiP zarówno w postaci infoMapy jak i diagramu. Ewentualna modyfikacja i ocena rozwiązań należy do członków zespołu projektowego. Natomiast zadaniem system CASAD jest jak najszybsze udostępnienie potrzebnego fragmentu struktury PIP.

Ze względu na to, że praca analityka ma charakter twórczy, wymagana jest interakcyjna współpraca zespołu z VIB. Efektywna wymiana informacji między zespołem projektowym a systemem poprzez moduł edytora graficznego wymaga krótkich czasów odpowiedzi. Rezultaty wprowadzanych poprawek powinny być natychmiast widoczne. Ze względu na psychologiczne aspekty procesu projektowego wymagany jest krótki czas reakcji systemu (tj. odstęp czasu między poleceniem wydanym przez człowieka a reakcją systemu na to polecenie). Martin [30] wykazał, że czas reakcji systemu dla 90% jego odpowiedzi nie powinien przekraczać 3 sekund a w pozostałych przypadkach 15 sekund. W dialogach graficznych celowe jest uzyskanie czasu reakcji poniżej jednej sekundy. Przekroczenie podanych czasów spowoduje gwałtowny spadek wydajności analityka/użytkownika. Warunkiem spełnienia powyższych wymagań jest wykorzystanie do implementacji systemu sprzętu komputerowego charakteryzującego się dużą mocą obliczeniową i umożliwiającego szybki dostęp do danych. Ponieważ jednak ze względu na szerokie grono potencjalnych użytkowników pakietu nie zakładano użycia sprzętu specjalnego typu np. stacji roboczych, konieczne będzie zastosowanie rozwiązań programowych przyspieszających pracę pakietu bazy danych projektowych [50].

Interfejs języka naturalnego

Jest przeznaczony docelowo do wprowadzania informacji typu opisowego, zwłaszcza takich, które nie dadzą się wprowadzić przy pomocy diagramu. Jest oparty o specjalnie zaprojektowane sieci semantyczne, przy czym słownictwo jest ściśle określone i zredukowane do wybranych podzbiorów tematycznych.

Inne interfejsy użytkownika

Oprócz wymienionych powyżej interfejsów wchodzących bezpośrednio w skład VIB, istnieje szereg szereg wąsko specjalizowanych edytorów stanowiących interfejsy pomocnicze. W niniejszym opracowaniu omawiamy je ogólnie ze względu na ograniczoną objętość pracy. Jednym z nich jest *edytor bazy danych projektowych*. W swojej podstawowej postaci edytor bazy danych projektowych jest narzędziem do uruchamiania systemu. Jest przeznaczony wyłącznie do przeglądania rekordów bazy danych projektowych, w szczególności bazy OMSI. Możliwość dokonywania edycji jest dopuszczalna tylko w sytuacjach awaryjnych. Zakłada się, że moduł jest dostępny tylko dla analityka.

Kolejny edytor to *edytor reguł projektowych*. Umożliwia wprowadzanie do *bazy reguł projektowych*: reguł i ograniczeń projektowych. Większość z wprowadzanych reguł ma za zadanie parametryzować algorytm(y) analizy konfigurując je w miarę potrzeb. Niektóre wprowadzone reguły mogą dotyczyć warunków brzegowych. O ich przekroczeniu użytkownik/analityk będzie informowany na bieżąco. W związku z istnieniem innych źródeł zasilania regułami baz wiedzy, edytor reguł projektowych stanowi tylko narzędzie uruchomieniowe.

Podsystem informacyjny - HELP

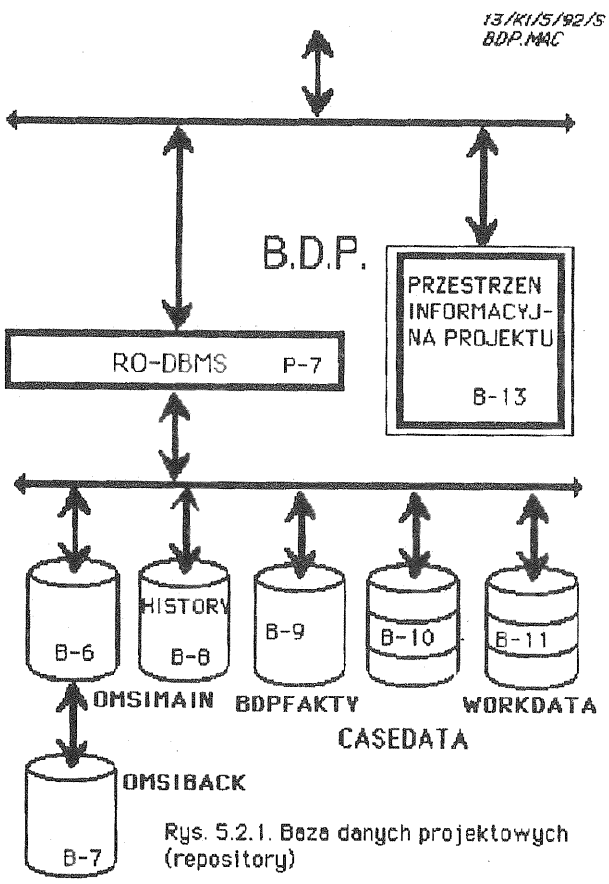
Podsystem ten jest przewidziany zarówno do uzyskiwania informacji na temat działania pakietu jak też do wprowadzania doda-

tkowych danych do plików HELP. Wyświetlana informacja ma charakter kontekstowy. Bazy podsystemu informacyjnego obejmują również podręcznik użytkownika systemu oraz podręczne notatki dokonywane przez analityka/użytkownika w trakcie procesu analizy.

5.2. Baza danych projektowych (repository)

Baza Danych Projektowych BDP przechowuje wszystkie informacje dotyczące projektu na każdym etapie jego realizacji. Pojęciowo BDP odpowiada roli "repository" w innych systemach komercyjnych tego typu. Głównym elementem stanowiącym jądro bazy danych projektowych jest tzw. Przestrzeń Informacyjna Projektu.

Rysunek 5.2.1. przedstawia wewnętrzną strukturę bazy danych projektowych. BDP jest zespołem powiązanych ze sobą baz danych o różnej strukturze sterowanych wspólnie przez system zarządzania BDP (rys. 5.2.1). Główna baza danych w tej strukturze nosi nazwę OMSIMAIN i zawiera dane bezpośrednio stowarzyszone z diagramami. Baza OMSIBACK zawiera dane o skasowanych zasobach, przechowywane w celu odtworzenia poprzednich wersji diagramów. Baza o nazwie SLOWNIK jest słownikiem danych zawierającym opisy słowne a BDPFAKTY - opisy ilościowe modernizowanego SI. Bazy CASEDATA i WORKDATA są bazami roboczymi. Są zespołami zmiennej ilości plików o specjalnej budowie (Są to tzw. bazy oparte o relacje binarne w których całkowicie oddzielono powiązania od danych [31]).



Rys. 5.2.1. Baza danych projektowych (repository)

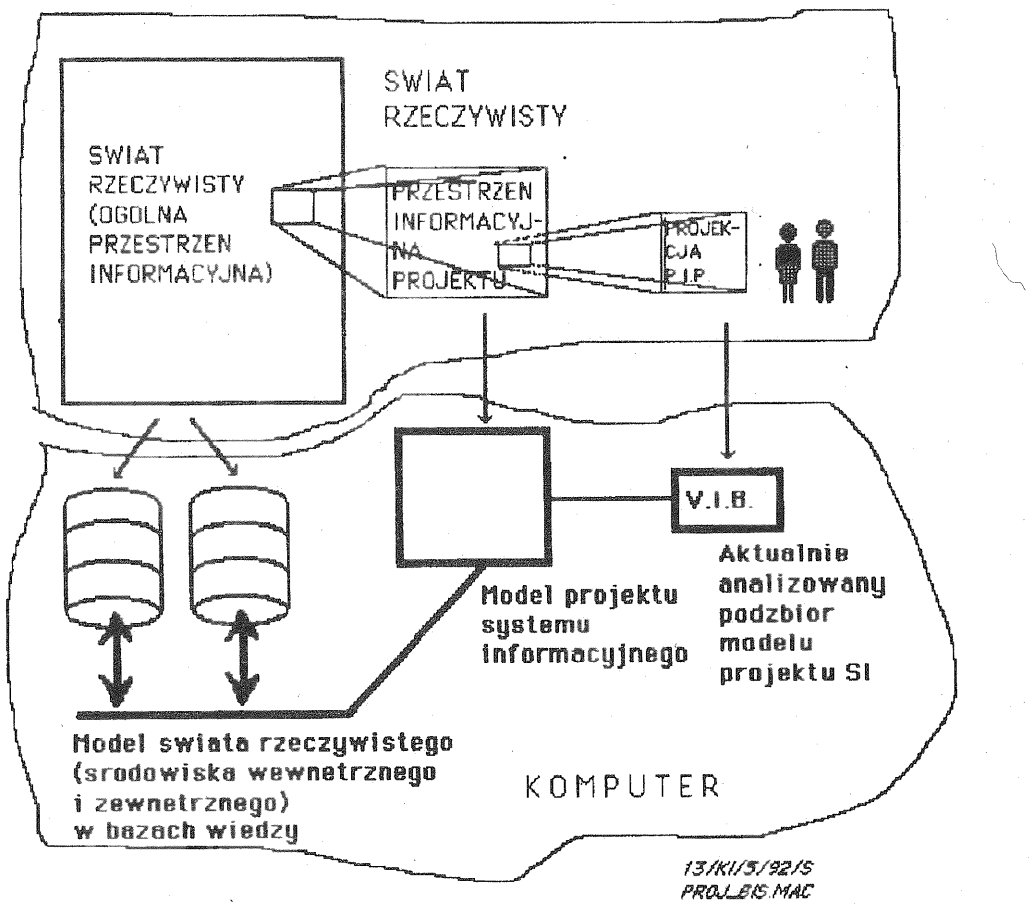
Przestrzeń Informacyjna Projektu

Wielkie wyzwanie w procesie rozwoju oprogramowania jest zawarte w konceptualizacji "*Design Information Space*", która ma dostarczyć mechanizm do

> "zbierania, reprezentacji, powtórnego użycia i prezentacji informacji poprzez cały proces projektowania, dla wspólnego użycia przez ludzi i maszyny w otaczającej socjalnej i edukacyjnej strukturze organizacyjnej" [47].

Przestrzeń informacyjna projektu jest pewnym znanym w danej chwili podzbiorem ogólnej przestrzeni informacyjnej opisującej badaną organizację (Rys. 5.2.2).

Jednym z elementów naszego rozwiązania jest zastosowanie jako techniki notacyjnej tzw. jMAP [14]-[19]. Technika ta została wykorzystana do zaimplementowania tzw. Przestrzeni Informacyjnej Projektu. Przestrzeń Informacyjna Projektu ma z założenia stanowić (wielowymiarowy) model wielowymiarowej przestrzeni informacyjnej w której działa analizowana organizacja. Ze względu na specyficzne walory notacji i techniki jMAP stała się możliwa techniczna realizacja Przestrzeni Informacyjnej Projektu.



Rys. 5.2.2. Przestrzeń Informacyjna Projektu jako model wycinka świata zewnętrznego.

Bazy OMSI

Bazy OMSI przechowują słowne opisy obiektów diagramu (OMSI - obiekt metodyki SI). Z każdym pojedynczym elementem diagramu (tzn. zasobem fizycznym, informacyjnym lub sterującym) związany jest rekord opisujący ten obiekt OMSI. Operacja narysowania na diagramie zasobu powoduje natychmiastową kreację stowarzyszonego z tym zasobem rekordu OMSI.

Mimo iż cały system CASAD jest tworzony w oparciu o język C++ i cała obsługa baz danych opiera się na procedurach napisanych w tym języku, to omawiane bazy danych zostały zrealizowane w standardzie dBase. Do bazy OMSIMAIN mogą być kierowane dwie podstawowe klasy zapytań dotyczących:

- charakterystyk wszystkich lub wskazanych obiektów,
- charakterystyk procesów w poszczególnych klasach,
- prześledzenia drogi od jednego obiektu do drugiego.

Z baz OMSI będzie korzystać analityk w fazie redefiniowania modelu SI wraz z użytkownikiem. Na podstawie danych zawartych w bazach OMSI będzie dokonywana analiza SI oraz jego specyfikacja.

5.3. Zarządzanie projektem

Ten moduł jest typowy dla większości narzędzi typu CASE, i z tego względu pomijamy jego omówienie.

5.4. Mechanizmy kontroli merytorycznej i składniowej

W całym oprogramowaniu środowiska projektowego istnieje szereg mechanizmów kontrolujących poprawność procesu projektowego. Koncentrują się szczególnie w obrębie dwóch modułów:

- edytora diagramów (DPD),
- przestrzeni informacyjnej projektu (PIP).

Mechanizmy w obrębie DFD

Zanim nastąpi zapis diagramu na dysk lub przetwarzanie, na życzenie analityka diagram może zostać zweryfikowany przez analizator syntaktyczny sygnalizujący błędy kardynalne (uniemożliwiające zapis) oraz generujący na życzenie raporty o ewentualnych brakach i niespójnościach. Analizator zawiera również mechanizmy kontroli zapewniające, by nowo wprowadzane dane nie były sprzeczne z regułami projektowymi oraz wcześniej wprowadzonymi danymi. Analizator syntaktyczny ma być wyposażony we własną bazę reguł analizy syntaktycznej. Program wysyła ostrzeżenia o braku kompletnej definicji zasobu lub obiektu, Przewidziano na żądanie operatora, możliwość przetwarzania warunkowego mimo niekompletnych definicji.

Mechanizmy w obrębie PIP

Głównym mechanizmem kontroli badającym PIP jest system doradczy (ES). Jego zadaniem jest badanie czy informacje zawarte w PIP są zgodne z ograniczeniami określającymi warunki wewnętrzne i zewnętrzne oraz warunki brzegowe. Testuje się także zgodność z regułami metodyki. Jest to podstawowe zadanie ES współpracującego bezpośrednio z PIP. Pomocniczym elementem zarówno dla PIP jak i ES jest *blackboard*. ES współpracuje z zestawem baz wiedzy [10]. Przy ich pomocy sprawdza się czy rozwiązanie odpowiada zadanym ograniczeniom.

5.5. Wyszukiwanie analogii w stosunku do innych projektów

Wyszukiwanie analogii i części wspólnych z innymi projektami odbywa się przy pomocy modułu o nazwie HNN ("Fuzzy & Associative Search").

Głównym zadaniem modułu jest wyszukiwanie i udostępnianie rozwiązań podobnych do aktualnie analizowanego. Osiąga się to w ten sposób, że dzięki zastosowaniu notacji jMap, przestrzeń informacyjna problemu może zostać potraktowana jako pewnego rodzaju obraz graficzny stanu systemu. Ponieważ podobne programy będą miały podobne jMapy (Wynika to z właściwości jMap.), możliwe jest użycie typowych algorytmów rozpoznawania obrazu oraz algorytmów wyszukiwania rozmytego. W tym celu konieczne jest przechowywanie starych projektów w postaci jMap. Baza danych zawierająca jMapy dotyczące starych projektów musi być wzbogacona o możliwość wyszukiwania asocjacyjnego [43], [41], [51], [22].

5.6. Dokumentowanie przebiegu procesu projektowego

W trakcie całego procesu projektowego zapewniana jest rejestracja jego przebiegu. Oprócz możliwości powrotu do rozwiązań poprzednich pozwala to także na notowanie bieżąco powstających pomysłów jak również zabezpiecza przed 'gubieniem' informacji w trakcie procesu projektowego. A zwłaszcza przy przejściach między fazami tego procesu. W jaki sposób jest to realizowane? Zarówno VIB jak i PIP współpracują z bazami danych o nazwie HISTORY. Zadaniem tych baz jest odzwierciedlanie przebiegu projektowania SI. W przypadku VIB edytor graficzny pozwala na rejestrowanie każdej przeprowadzonej operacji graficznej. Mechanizm ten jest podwójny. Raz operacje graficzne są odnotowywane w standardzie *WINDOWS metafile*, drugi raz w pliku standardu dBase. Wspomniane mechanizmy dotyczą tylko aktualnie rysowanego

diagramu. Diagramy stare, z już wykonanych projektów są przechowywane na dysku tylko w postaci jMap. W przypadku odtwarzania takiego diagramu uruchamia się mechanizm optymalizacji rozmieszczenia elementów diagramu.
Z każdym elementem diagramu jest powiązany tzw. rekord OMSI. By zapewnić możliwość powrotu do rozwiązań poprzednich, w wypadku kasowania obiektu diagramu na ekranie, odpowiadający mu rekord OMSI nie jest kasowany a tylko przesuwany do rezerwowej bazy OMSIBACK.

5.7. Sterowanie doborem metod/technik projektowania

Autorzy uznają za celowe stosowanie różnych podejść i metodyk odpowiednich do etapu i warunków projektu. Współczesne techniki tworzenia oprogramowania wymagają od analityków produkowania niezliczonych dokumentów opisujących wyniki kolejnych etapów projektu, poczynając od studium wykonalności a na kodzie programu kończąc. Powiązania między tymi dokumentami rzadko są poprawne formalnie i kompletne. Ponadto występuje szkodliwa redundancja. Wiąże się to z dodatkowymi nakładami pracy koniecznymi do porządkowania dokumentacji, jej utrzymania i udostępniania. Istnieje też ryzyko nieuwzględnienia wszystkich powiązań występujących w projekcie, a także - jak to już było wspominane wcześniej - możliwość gubienia pomysłów przy przechodzeniu z jednego etapu do drugiego. W tej chwili widzimy tylko jedno rozwiązanie: wykorzystanie techniki *multiview*, z tym, że produkty każdej metodyki stosowanej na odpowiednim etapie projektu będą transformowane i reprezentowane w BDP w jednolitej postaci, którą zapewniają jMapy.

Reasumując, zaprojektowane środowisko ES-GDSS ma zapewnić wspomaganie doboru właściwej metodyki. Po odpowiednim dostrojeniu ta funkcja ma być realizowana przez ES, który w tym celu ma analizować m.in. zawartość PIP.

Rezultatem przeprowadzonej przez ES analizy ma być sugestia doboru metodyki/techniki odpowiedniej dla danego etapu. Po wybraniu właściwej metodyki przez ES zostaną zrealizowane następujące kroki (rys. 5.7):

(1) Załadowanie do edytora DFD nowego zestawu symboli graficznych i nowego bloku reguł syntaktycznych określających zasady tworzenia diagramów oraz reguły ich poprawności.
(2) Załadowanie do PIP nowej infoSCHEMY.
(3) Uzupełnienie KB_RULES o nowe reguły metodyki (tzw. reparatrytryzacja środowiska projektowego).

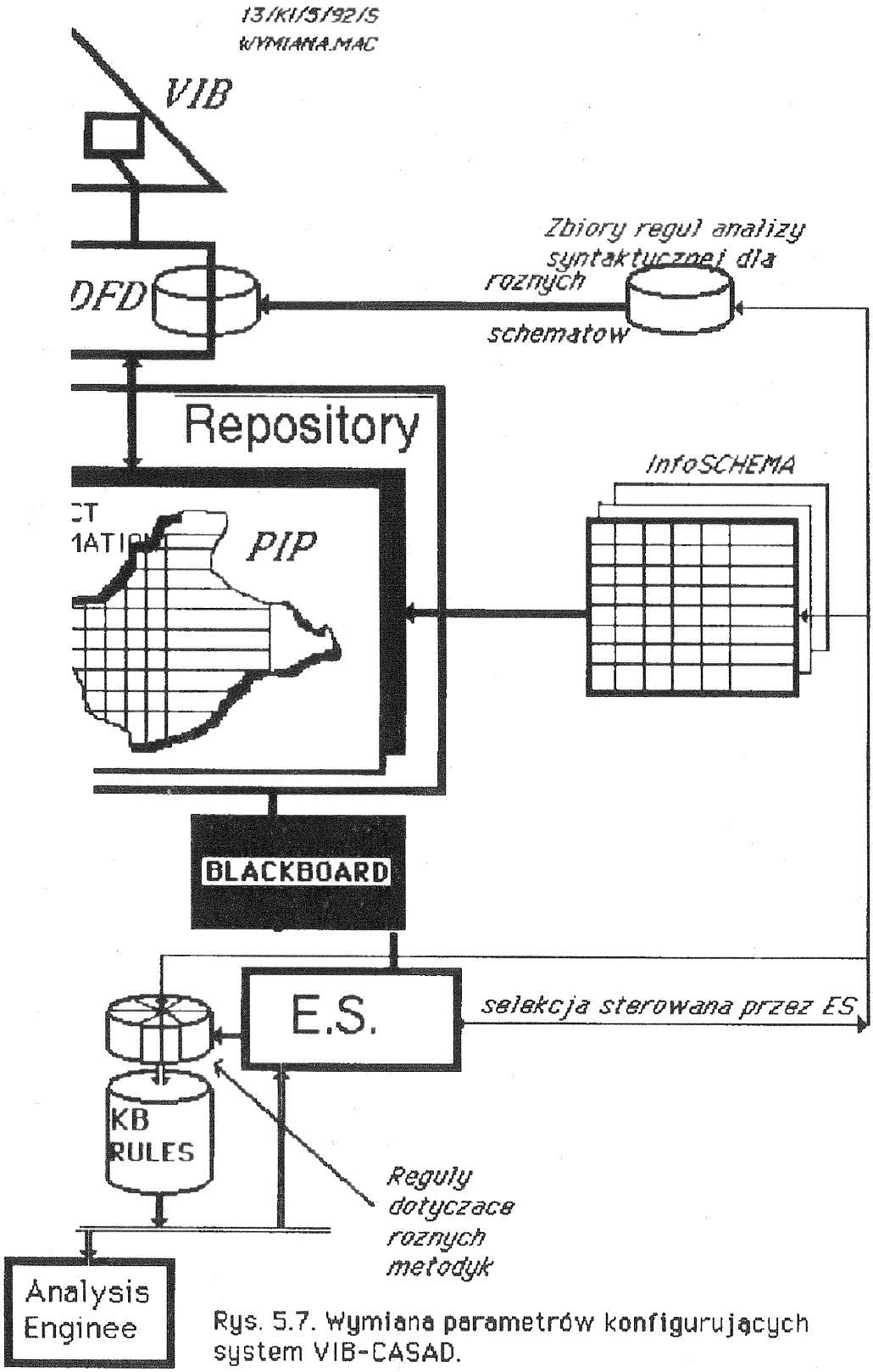
Moduł sterowania doborem metod/technik będzie opracowany i implementowany po uzupełnieniu bazy metod technik oraz mechanizmów konwersji.

6. Wnioski

Arrakis naucza filozofii noża - odrąbujemy to, co niekompletne mówimy: "Teraz jest kompletne, ponieważ tutaj się kończy."

Frank Herbert, cykl "Diuna"

Proponowane rozwiązanie łączy w sobie wiele nowych elementów takich jak na przykład koncepcja VIB oraz implementacja przestrzeni informacyjnej. System VIB-CASAD jest w tej chwili koncepcją najbardziej rozbudowanego środowiska gromadzącego, przechowującego i udostępniającego wiedzę o tworzonym SI. Wykorzystuje notację infoMAP (czy inaczej jMAP) jako nośnik tej wiedzy (infoMAPY są doskonałym narzędziem do pozyskiwania i gromadzenia wiedzy). Zalety notacji infoMAP (m.in. to, że cały proces rozwoju oprogramowania może być dokumentowany w jednolitej postaci) czynią możliwym implementację wielu nowych funkcji. Możliwe jest na przykład 'rozmyto'-asocjacyjne wyszukiwanie rozwiązań podobnych, czy też łatwe przechodzenie z jednej notacji na inną.



Rys. 5.7. Wymiana parametrów konfigurujących system VIB-CASAD.

Sprawdzenie przydatności wyszukiwania 'rozmyto'-asocjacyjnego wymaga jednak wcześniejszego wypełnienia baz wiedzy systemu VIB-CASAD informacjami o wcześniej wykonanych projektach. Zachodzi zatem potrzeba zgromadzenia opisów z kilku projektów wykonywanych z wykorzystaniem tego narzędzia. VIB-CASAD jest obecnie w fazie opracowywania prototypu podstawowych funkcji. Najpoważniejszym zagrożeniem dla wypracowanej koncepcji (częściowo zaimplementowanej) jest brak odpowiednich środków finansowych.

Bibliografia

[1] Avison D. E., Golder P. A., Shah H. U.: *A Tool Kit for Soft Systems Methodology*, w: [13].

[2] Avison D. E., Wood-Harper T.: *Information Systems Development Research: A Pragmatic View*, The Computer Journal 34, Vol. 2.

[3] Baxter I. D.: *Design Maintenance Systems*, CACM, Vol. 35., No. 4. April 1992.

[4] Buckley F. J.: *Some Standards for Software Maintenance*, IEEE Computer, pp.69-70, Nov.1989.

[5] Certo S. C., Peters J. P.: *Strategic Management. Concepts and Applications*, 2-nd Ed. McGraw-Hill Inc., New York 1991.

[6] Checkland P.: *Systems Thinking, Systems Practice*, John Wiley & Sons, Chichester - N. York - Brisbane - Toronto 1986.

[7] Coad P.: *Object-Oriented Patterns*, CACM, Sept. 1992, Vol. 35, No. 9.

[8] Coad P., Yourdon E.: *Object Oriented Analysis*, 2-nd ed., Prentice-Hall Inc., Englewood Cliffs, 1991.

[9] Flaatten P. O., McCubbrey D. J., O'Riordan P. D., Burgess K. *Foundations of Business Systems*, The Dryden Press, Chicago 1989.

[10] Frost R.: *Introduction to Knowledge Base Systems*, McGraw-Hill Publishing Company 1986.

[11] Gasparski W. (ed.): *Projektoznawstwo. Elementy wiedzy o projektowaniu*, WNT, Warszawa 1988.

[12] Heym M., Osterle H.: *A Reference Model for Information Systems Development*, w: [13].

[13] IFIP Transactions A-8, *The Impact of Computer Supported Technologies on Information Systems Development*, Editors: Kendall K. E. et al., Elsevier Science Publishers B.V (North-Holland) (c) 1992 IFIP.

[14] Jaworski W. M., Ficocelli L., O'Mara K. S.: *The ABL/W4 Methodology for System Modelling*, System Research J., Vol.4, No.1, 1987, pp. 23-27.

[15] Jaworski W. M., MacCuaig I., Marinelli T., Nysztor T.: *Executable Specification for a Large Industrial Process*, Proceedings of the 1986 Canadian Conference on Industrial Computers, Montreal, pp. 60-1...60-5.

[16] Jaworski W. M., Cummings T.: *Program Normalization and Optimization: Using infoMaps as an inspection and program processing tool*, Canadian Conference on Electrical and Computer Engineering, Quebec City, September 1991.

[17] Jaworski W. M.: *System Analysis and Design in a Classroom: infoMAPs Teaching Factory*, Modelling Conference, Pittsburg, Pa., May 2-3, 1990.

[18] Jaworski W. M., Deslauriers B.: *Software Process, Maintenance, Products: Software Development Environment based on InfoMAPs*, Control and Electrical Eng. Conference, Ottawa, September 1990.

[19] Jaworski W. M., Grogono P. D.: *infoMAPS: a Pragmatic Environment for Seamless and Nondeterministic Software*

Development, Concordia University, Computer Science Dept., January 1991.

[20] Jones J. C.: *Metody projektowania*, WNT, Warszawa 1977.

[21] Kendall K. E., Kendall J. E.: *Systems Analysis and Design*, 2-nd ed., Prentice-Hall Inc., Englewood Cliffs, New Jersey 1993.

[22] Kosko B.: *Neural Networks and Fuzzy Systems. A Dynamical Systems Approach To Machine Intelligence*, Prentice-Hall Inc. 1992.

[23] Kuraś M.: *Zakres modernizacji systemu informacyjnego*, Zeszyty Naukowe AE Kraków, Nr. 286, 1989.

[24] Kuraś M.: *Koncepcja komputerowego wspomagania analizy i projektowania systemów informacyjnych*, Zeszyty Naukowe AE Kraków, Nr. 366, 1992.

[25] Kuraś M., Sarga D., Zaliwski A.: *Metoda planowania i tworzenia systemów informacyjnych zarządzania*, Raport z pracy badawczej 29/ISEI/7/91/S, Katedra Informatyki, Akademia Ekonomiczna, Kraków 1992.

[26] Kuraś M.: *Wybór strategii rozwoju zastosowań informatyki do obsługi systemu informacyjnego*, Zeszyty Naukowe AE w Krakowie, Nr. 330, 1990.

[27] Kuraś M., Maciołek W., Sarga D., Zaliwski A.: *Komputerowe wspomaganie specyfikacji SI*, Ogólnopolska Konferencja Instytutów, Katedr i Zakładów Informatyki, Rogi k/Łodzi, 9-11 września 1991.

[28] Kuraś M., Sarga D., Węgrzynowski A.: *Specyfikacja potrzeb informacyjnych z udziałem użytkowników*, Zeszyty Naukowe AE w Krakowie, Nr. 330, 1990.

[29] Maiden N. A., Sutcliffe A. G.: *Exploiting Reusable Specifications Through Analogy*, CACM, Vol. 35, No.4, Apr. 1992.

[30] Martin J.: *Dialog człowieka z maszyną cyfrową*, WNT, Warszawa 1976.

[31] Martin J.: *Bazy Danych*, WNT, Warszawa 1980.

[32] Nerson J. M.: *Applying Object-Oriented Analysis and Design*, Communications of the ACM, Vol. 55, No. 9, September 1992.

[33] Nolan R. L.: *Managing the Crisis in Data Processing*, Harvard Business Review, March-April 1979.

[34] Peters T.: *Thriving on Chaos. Handbook for a Management Revolution*, Harper & Row, New York 1988.

[35] Robey D., Markus M. L.: *Rituals In Information System Design*, MIS Quarterly, Vol. 8, No. 1, March 1984.

[36] deSanctis G., Gallupe R. B.: *A Foundation for the Study of Group Decision Support Systems*, Management Science, Vol. 33, No. 5, May 1987.

[37] Sarga D.: *Opis funkcji hipotetycznego przedsiębiorstwa*, AE, Kraków 1993.

[38] Scott Morton M. S. (ed.): *The Corporation of the 1990s. Information Technology and Organizational Transformation*, Oxford University Press, New York - Oxford 1991.

[39] Senn J. A.: *Essential Principles of Information Systems Development*, MIS Quarterly, Vol. 2, No. 2, June 1978.

[40] *Software Engineering Standards*, IEEE, 1987.

[41] Soucek B., Soucek M.: *Neural and Massively Parallel Computers. The six generation*, John Wiley & Sons Inc., 1988.

[42] Sprague R. H. Jr., McNurlin B. C.: *Information Systems Management in Practice*, Prentice-Hall Inc., Englewood Cliffs, New Jersey 1986.

[43] Tadeusiewicz R.: *Problemy*

biocybernetyki, Skrypty Uczelniane, Nr. 1242, AGH Kraków 1991.

[44] Tadeusiewicz R., Flasiński M.: *Rozpoznawanie obrazów*, PWN, Warszawa 1991.

[45] Tapscott D., Caston A.: *Paradigm Shift. The New Promise of Information Technology*, McGraw Hill Inc., New York 1993.

[46] Waring A.: *Systems Methods for Managers*, Blackwell Scientific Publications, Oxford - London - Edinburgh - Boston - Melburn 1989.

[47] Webster D. E: *Mapping the Design Representation Terrain*, Technical Report STP-093-87, MCC, Software Technology Program., July 1987.

[48] Yourdon E.: *Managing the Structured Techniques. Strategies for Software Development for 1990's*, 3-rd ed., Yourdon Press, New York - London 1986.

[49] Yourdon E.: *Modern Structured Analysis*, Prentice-Hall Inc., Englewood Cliffs, 1989.

[50] Zaliwski A.: *Zastosowanie nieelementarnych metod grafiki komputerowej do tworzenia interakcyjnych systemów typu CAD dla potrzeb górnictwa*, Rozprawa doktorska, Wydz. Automatyki AGH, Kraków 1991.

[51] Zurada J. M.: *Introduction to Artificial Neural Systems*, West Publishing Company, St. Paul, New York, Los Angeles, San Francisco 1992.

Spis symboliki użytej na rysunkach

Zbiory/zasoby:

B-1 REGEN metafile
B-2 N-1 metafile
B-3 historia zmian
B-4 <nazwa><kod>.<nr> - pliki przechowujące diagramy
B-5 SYNTAX (reguły składni diagramów rysowanych na edytorze DPD)
B-6 OMSIMAIN (tzw. baza OMSI - Obiektów Metodyki SI)
B-7 OMSIBACK (kopia OMSIMAIN)
B-8 Historia Projektu (bazy HISTORIA)
B-9 BDPFAKTY - opisy ilościowe modernizowanego SI
B-10 CASEDATA
B-11 WORKDATA
B-12 jMapOld (InfoMAPY dotyczące zakończonych projektów)
B-13 PIP - Przestrzeń Informacyjna Projektu
B-14 KB_RULES (Baza wiedzy zawierająca reguły)
B-15 KB_FACTS (Reprezentacja faktów w bazy wiedzy)
B-16 Blackboard (Obszar roboczy dla ES)
B-17 reguły metodyk

Procesy

P-1 Edytor diagramów (DPD)
P-2 Generator jMapy
P-3 Analizator syntaktyczny diagramów
P-4 Moduły optymalizacji rozmieszczenia diagramów
P-5 Operacje dyskowe
P-6 Wyszukiwanie rozwiązań podobnych (Fuzzy & Associative Search)
P-7 RO-DBMS
P-8 ES
P-9 Generator aplikacji
P-10 Edytor reguł
P-11 Generacja reguł
P-12 Interfejs języka naturalnego
P-13 Interfejs użytkownika
P-14 Interfejs dla potrzeb analizy i generacji aplikacji

Jan Baranowski
THOMSON POLKOLOR

Wykorzystanie baz danych opartych na Btrievie i NetwareSQL w środowisku WINDOWS.

1. Wstęp.

System informatyczny THOMSON POLKOLOR oparty jest na sieci Novell. Obsługiwana przez pięć serwerów plików *(ang. file server)*, obejmująca ponad sto stacji roboczych rozrzuconych po terenie całego zakładu, sieć ta pretenduje do miana jednej z największych tego typu instalacji w Polsce. Systemy informatyczne działające w niej obejmują w chwili obecnej cały obszar działalności finansowej przedsiębiorstwa oraz gospodarkę materiałową i zaopatrzenie. Dostosowanie do specyficznych warunków THOMSON POLKOLOR (podwójna ewidencja w złotych i we frankach, dostosowanie do norm zachodnich przy jednoczesnej zgodności z polskimi przepisami, konieczność szybkiego raportowania miesięcznych wyników do centrali w Paryżu), pełne zintegrowanie bazy danych, praca w trybie on-line oraz pełna niezawodność i przyjazność dla użytkowników tych systemów spowodowały zaakceptowanie ich przez ekspertów francuskich jako systemów docelowych i zmieniły początkowe plany zastąpienia ich gotowym oprogramowaniem zachodnim.

Oparcie systemów informatycznych dużego przedsiębiorstwa na produktach firmy Novell takich jak Btrieve i NetwareSQL okazało się wyjątkowo udanym rozwiązaniem. Efektywność działania systemów obsługujących wiele plików zawierających po kilkaset tysięcy rekordów jest w pełni zadowalająca, a jednocześnie w pełni zintegrowana, relacyjna baza danych umożliwia, przy użyciu takich narzędzi jak XTRIEVE lub interpreter komend SQL, błyskawiczny dostęp do dowolnych informacji w dowolnym układzie, co jak okazało się w praktyce, jest niezmiernie potrzebne.

Błyskawiczny postęp techniczny oraz konieczność nadążania za nowymi trendami wywołały potrzebę prezentacji raportów systemów w bardziej przyjaznej użytkownikowi postaci, jaką daje środowisko MS WINDOWS. Zachowując w pełni możliwości funkcjonalne istniejących systemów (w niektórych przypadkach, jak wprowadzanie danych czy generacja wielostronicowych wydruków, prostota i efektywność rozwiązań pracujących pod DOS'em daje o wiele lepsze efekty) podjęliśmy prace nad udostępnieniem danych z naszych systemów dla aplikacji pracujących w środowisku Windows. Przedstawieniu niektórych doświadczeń i możliwości z tym związanych poświęcony jest ten referat.

2. Nowe cechy produktów obsługujących bazy danych firmy Novell.

W 1992 roku firma Novell wprowadziła na rynek nową wersję bazy danych NetwareSQL 3.0 opartą na nowej wersji Btrieve'a 6.0. Wersja ta dostosowana jest wyłącznie do współpracy z Netware 3.x oraz Netware 4.x. Dla Netware 2.x oraz instalacji jednostanowiskowych pozostają NetwareSQL 2.11 i XQL 2.11 będące

znacznym udoskonaleniem wersji 2.01 tych produktów (w szczególności usunięto wiele błędów występujących w wersjach 2.01). W skład pakietu NetwareSQL 3.0 wchodzą również nowe wersje XTRIEVE'a 4.11 oraz innych programów pomocniczych.

Nowe elementy zawarte w NetwareSQL 3.0 to:
- możliwość definiowania do 119 segmentów kluczy sortowania
- możliwość zapewnienia relacyjnej spójności bazy danych *(ang. referential integrity)*
- możliwość dostępu do bazy danych przez nazwę, bez podawania fizycznej lokalizacji *(ang. named database)*
- znaczna poprawa efektywności działania.

Dalszy rozwój NetwareSQL, zgodnie z zapowiedziami firmy Novell, pójdzie w kierunku zapewnienia pełnej zgodności z bazą DB2.

3. Możliwości wykorzystania w środowisku WINDOWS.

3.1 Narzędzia wchodzące w skład pakietu NetwareSQL 3.0.

NetwareSQL 3.0 zawiera nową wersję biblioteki WXQLCALL.DLL umożliwiającą współpracę środowiska WINDOWS z Btrieve'em i SQL'em. Biblioteka ta może być również wykorzystywana do współpracy z XQL.

NetwareSQL 3.0 zawiera również nowy interpreter komend języka SQL, o nazwie SQLScope, przeznaczony do pracy w środowisku WINDOWS . Interpreter ten pozwala na składanie (wybór z menu) komend języka SQL, w szczególności "podpowiada" nazwy tablic i kolumn. Możliwości prezentacji wyników są jednak mocno ograniczone.

NetwareSQL 3.0 zawiera również sprzęg programowy *(ang. router)* do bazy danych firmy GUPTA. Możliwości wykorzystania tego sprzęgu omówione zostaną w dalszej części referatu.

Należy zwrócić uwagę że tzw. *runtime'y* umożliwiające pracę jednostanowiskową (BTRIEVE, XQL, XQLO, XQLP, XQLPO) mogą być bezpłatnie rozpowszechniane z aplikacjami wytworzonymi przez firmy posiadające legalne narzędzia do tworzenia takich aplikacji.

3.2 Visual Basic.

Pakiet Visual Basic firmy Microsoft, szczególnie w wersji 2.0 Professional Edition, stanowi bardzo wygodne i efektywne narzędzie do pisania programów działających w środowisku WINDOWS. Wygodne projektowanie ekranów, programowanie obiektowo-zdarzeniowe *(ang. event-driven programming)* oraz możliwość wykorzystywania praktycznie wszystkich API środowiska WINDOWS (niektórych w wyjątkowo wygodnej formie) stawia Visual Basic na równi z takimi narzędziami jak C++. Wydaje się przy tym, że jest to najwygodniejsze i najprostsze narzędzie do tworzenia aplikacji pracujących w środowisku WINDOWS i pozwalające na wykorzystywanie wszystkich możliwości tego środowiska. Visual Basic zalecany jest przez firmę Novell do tworzenia aplikacji sieciowych opartych na Btrievie i NetwareSQL.

Dla umożliwienia dostępu do baz danych opartych na Btrievie, konieczne jest załadowanie odpowiednich programów (Btrieve lub Brequest, Nsreq lub Xql) do pamięci komputera przed uruchomieniem programu WINDOWS. Niezbędna jest również dostępność bibliotek WBTRCALL.DLL dla Btrieve'a oraz WXQLCALL.DLL dla NetwareSQL'a i XQL'a. Najnowsze wersje tych produktów wraz z dokładnymi opisami ich wykorzystywania dostępne są w rozpowszechnianej na dyskach optycznych encyklopedii novellowskiej oraz w sieci CompuServe.

Praktyczne doświadczenia z wykorzystywania pakietu Visual Basic są bardzo pozytywne. Efektywność dostępu do baz danych jest bardzo dobra, a obsługa sprzęgu graficznego w pełni zadowalająca. Pewnym mankamentem jest ubogi, w stosunku do Btrieve'a, zasób typów pól rozpoznawanych przez Basic. Rozwiązaniem może być tutaj wykorzystanie SQLowej dyrektywy CONVERT umożliwiającej konwersje typów pól. Również możliwości generowania wydruków pozostawiają wiele do życzenia. Pomimo to Visual Basic wydaje się być optymalnym rozwiązaniem dla osób pragnących pisać aplikacje działające w środowisku WINDOWS z wykorzystaniem narzędzi trzeciej generacji.

3.3 Microsoft ACCESS.

Szeroko reklamowany nowy pakiet firmy Microsoft o nazwie ACCESS przeznaczony jest do obsługi relacyjnych baz danych i tworzenia aplikacji działających w środowisku WINDOWS. Oprócz możliwości bezpośredniego dostępu do baz danych w formie sterowania z menu *(ang. menu driven)*, umożliwiającego zadawanie zapytań *(ang. query)* oraz generowanie raportów, pakiet udostępnia także możliwość pisania aplikacji w języku zbliżonym do Visual Basic.

MS ACCESS zapewnia bezpośredni *(ang. on-line)* dostęp do wielu popularnych baz danych, w szczególności do Btrieve'a (a raczej do NetwareSQL'a, gdyż wymaga SQL'owych definicji tablic). Opcja ta obarczona jest jednak kilkoma poważnymi błędami. Jeden z nich polega na błędnym prezentowaniu zawartości pól typu numeric (wartości numeryczne zapisane w postaci znakowej). Obejściem tego błędu jest deklarowanie tych pól jako znakowe. Uniemożliwia to jednak wykonywanie na nich operacji numerycznych. Innym poważnym błędem jest niewłaściwe wykorzystywanie możliwości Btrieve'a w zakresie pozycjonowania rekordów. Operacje odczytania rekordu o zadanej wartości klucza lub odczytanie ostatniego, według zadanego klucza, rekordu realizowane są poprzez sekwencyjne czytanie pliku.

Wydaje się że praktyczne wykorzystywanie tego pakietu, przynajmniej w zakresie dostępu do baz opartych na Btrievie, będzie możliwe dopiero po usunięciu przez firmę Microsoft tych rażących niedoróbek.

3.4 GUPTA.

Szeroko popularyzowana przez firmę TECHMEX i prezentowana na poprzedniej Szkole Górskiej PTI baza danych GUPTA wydaje się być idealnym wręcz narzędziem do zarządzania bazami danych i generowania aplikacji działających w środowisku WINDOWS. Pakiety wchodzące w jej skład, takie jak QUEST, SQLWindows i SQLTalk zapewniają pełną gamę możliwości jakie nowoczesna baza danych mieć powinna: dostęp poprzez wybór z menu *(ang. menu driven)*, interpreter komend języka SQL, generator raportów, API i preprocesory do C i COBOL'u oraz zintegrowane środowisko do pisania, a raczej generowania, aplikacji.

Jednym z elementów wspomnianego wcześniej pakietu NetwareSQL 3.0 jest sprzęg programowy umożliwiający bezpośredni dostęp do baz danych opartych na NetwareSQL lub XQL z poziomu produktów firmy GUPTA. Dostępna jest również wersja SQLWindows oparta na Btrievie. Oba sprzęgi są jednokierunkowe. Nie istnieje możliwość dostępu do baz danych GUPTA z poziomu Btrieve'a czy NetwareSQL. Oba sprzęgi posiadają również jeden poważny mankament. Nie tłumaczą zapytań odnoszących się do plików systemowych, a więc zapytań o tablice, kolumny, prawa dostępu itp. Ponadto w nazwach plików systemowych GUPTY występuje znak "." co uniemożliwia imitowanie ich na poziomie NetwareSQL. Nie działają więc wszystkie tak przydatne opcje typu dostępu przez wybór z menu, w szczególności QUEST.

W przypadku rozwiązania powyższego problemu (co nie wydaje się zbyt trudne) pakiety firmy GUPTA stałyby się wręcz idealnym narzędziem do implementacji baz danych opartych na Btrievie i NetwareSQL w środowisku WINDOWS.

3.5 OBJECT VISION.

Kolejnym narzędziem przydatnym do umożliwiania dostępu do baz danych z poziomu środowiska WINDOWS jest pakiet OBJECT VISION firmy Borland International Inc. Pakiet umożliwia projektowanie formatek ekranowych w sposób zbliżony jak przy pomocy Visual Basic'a, choć o znacznie ograniczonych możliwościach. Pakiet nie zawiera generatora raportów, chociaż umożliwia drukowanie formatek ekranowych. Istotną cechą pakietu jest możliwość łączenia pól z formatek ekranowych z polami z tablic szeregu baz danych m.in. opartych na Btrievie (właściwie na XQL, gdyż niezbędne są tablice systemowe w tym standardzie). Szereg dalszych możliwości filtrowania danych oraz definiowania dodatkowych funkcji czyni ten produkt dość przydatnym narzędziem dostępu do baz danych, chociaż o znacznie ograniczonych, w stosunku do innych omówionych pakietów, możliwościach.

4. Podsumowanie.

Omówione powyżej pakiety nie wyczerpują oczywiście wszystkich możliwości dostępu do baz danych opartych na Btrievie w środowisku WINDOWS. Wymienić należy chociażby inne języki programowania dostępne w tym środowisku z C++ na czele. Wymieniono jedynie te pakiety, których możliwości zostały praktycznie sprawdzone. Jednak już te przykłady dają wystarczające podstawy do stwierdzenia, że przejście z aplikacji DOS'owskich opartych na Btrievie i NetwareSQL w środowisko WINDOWS nie nastręcza większych trudności. Wszystkie omówione pakiety działać mogą zarówno w sieci Novell jak i na pojedynczym komputerze. W przypadku systemów wielodostępnych możliwy jest równoczesny dostęp do tych samych danych zarówno z poziomu DOS'u jak i ze środowiska WINDOWS. Możliwe są w szczególności systemy mieszane: wprowadzanie danych przy pomocy aplikacji DOS'owskiej i generowanie "ładnych" raportów w postaci graficznej w środowisku WINDOWS. Możliwe jest również stopniowe przechodzenie z DOS'u na WINDOWS'y z zachowanie tych samych baz danych, a co za tym idzie przy niezakłóconej pracy aplikacji. Jest to dodatkowa zaleta stosowania aplikacji opartych na standardowych bazach danych, do których dostęp zapewniają producenci innych pakietów. W dobie coraz bardziej posuniętej standaryzacji, trzymanie się wytyczonych dróg ma coraz większe znaczenie.

W poniższej tabeli zestawiono najistotniejsze cechy omówionych produktów.

	Interpreter komend SQL	Dostęp poprzez wybór z menu	API	Programowanie obiektowe
Netware SQL	+	-	-	-
Visual Basic	-	-	+	+
ACCESS	-	+	+	+
GUPTA	+	- +	+	+
Object vision	-	+	-	-

Jacek Stochlak
ICL POLAND

Systemy przetwarzania transakcyjnego OLTP. Modele i standardy

Wprowadzenie

Środowisko gospodarowania w latach dziewięćdziesiątych charakteryzuje się dynamicznym postępem technologicznym, coraz większą konkurencją, powstawaniem nowych rynków zbytu, globalizacją i decentralizacją procesów gospodarowania. Przedsiębiorstwa obecnie działające są pod nieustanną presją usprawnienia działalności, obniżenia kosztów i świadczenia coraz lepszych usług dla swoich klientów. Wiąże się to z poprawieniem i ułatwieniem dostępu do posiadanych informacji oraz możliwości ich natychmiastowego przetwarzania. Informacje stają się obecnie jednym z najbardziej cennych i istotnych zasobów w procesie gospodarowania. Wiele przedsiębiorstw w celu zapewnienia dostępu do strategicznych usług informatycznych wykorzystuje systemy przetwarzania transakcyjnego OLTP *(On-Line Transaction Processing)*.

Systemy transakcyjne OLTP są szeroko stosowanymi systemami przetwarzania danych, zorientowanymi na obsługę dużej liczby transakcji - elementarnych, niepodzielnych i dających się odtworzyć operacji, które wymagają dostępu, przetwarzania i aktualizacji danych przechowywanych w bazie danych systemu. Systemy OLTP są projektowane w taki sposób aby zapewnić wysoki stopień rzetelności, bezpieczeństwa dostępu do danych i usług systemu. W dzisiejszym bardzo konkurencyjnym środowisku gospodarowania dobrze zaprojektowane i działające systemy OLTP mogą w znaczny sposób przyczynić się do uzyskania przewagi konkurencyjnej oraz oznaczać szansę odniesienia sukcesu. Decentralizacja procesów gospodarowania i konieczność ich ciągłego dopasowywania do zmieniających się warunków otoczenia, uwidacznia się coraz silniejszą tendencją do budowania wspomagających je systemów informatycznych w oparciu o koncepcję systemów otwartych. Tylko to podejście bowiem jest w stanie zapewnić wymaganą elastyczność i "otwartość" środowiska systemowego przy zachowaniu rozsądnych kosztów inwestycyjnych.

Zgodnie z wynikami badań rynku systemów informatycznych, systemy przetwarzania transakcyjnego OLTP są największym i najszybciej rozwijającym się segmentem komercyjnego przetwarzania danych. Jego roczny wzrost wynosi 35% przy 10% wzroście całego rynku systemów informatycznych. Wzrost segmentu systemów OLTP pracujących w środowisku systemów otwartych jest przy tym o 18% większy od segmentu obejmującego pozostałe systemy OLTP (InfoCorp).

Przetwarzanie transakcyjne

Przetwarzanie transakcyjne OLTP to środowisko systemowe, w którym duża liczba użytkowników ma zapewniony natychmiastowy dostęp i możliwość przetwarzania kluczowych informacji, istotnych w obsłudze procesów gospodarowania. Środowisko to charakteryzuje przewidywalny czas reakcji systemu, konieczność współpracy z dużymi bazami

danych oraz bezpieczeństwo i rzetelność informacji.

Obecnie wykorzystywane systemy przetwarzania transakcyjnego OLTP wykorzystują najczęściej duże systemy komputerowe *(mainframes)* jako platformy do implementacji usług transakcyjnych. W ostatnich latach, ze względu na coraz większe uzależnienie procesów gospodarowania od dostępności systemów informatycznych oraz coraz większą liczbę wymaganych usług i użytkowników, zaczęły pojawiać się rozwiązania oparte o wieloprocesorowe systemy o ciągłej *(continous processing)* lub podwyższonej dostępności *(high availability)* systemu. Rozwiązania te cechuje z jednej strony zamknięta architektura systemowa kontrolowana przez dostawcę systemu, z drugiej natomiast, ograniczone możliwości integracji i współpracy z innymi systemami co w dobie globalizacji i decentralizacji procesów gospodarowania jest istotnym czynnikiem ograniczającym rozwój systemów informatycznych. Są one najczęściej systemami scentralizowanymi, wykorzystującymi centralne bazy danych oraz protokoły komunikacyjne i urządzenia zewnętrzne kontrolowane przez jednego producenta.

Środowisko gospodarowania lat dziewięćdziesiątych stawia przed systemami informatycznymi dużo większe wymagania. Nie są ich w stanie spełnić tradycyjne systemy OLTP bazujące na centralnych systemach klasy mainframe i proste terminale znakowe. Podstawowym czynnikiem, który zadecydował o tak dynamicznym rozwoju systemów przetwarzania transakcyjnego OLTP, jest zmiana modelu gospodarowania ostatnich lat. Technologia przetwarzania informacji i systemy OLTP przestały być traktowane jako koszt gospodarowania lecz stały się sercem i układem krwionośnym procesu, który może dać przewagę konkurencyjną i szansę odniesienia sukcesu. Obserwowana w latach obecnych globalizacja i decentralizacja procesów gospodarowania, prowadzi do powstawania autonomicznych jednostek gospodarczych (centrów zysków), które ściśle ze sobą współpracują. Nie bez znaczenia jest również konieczność integracji nowych systemów OLTP z wykorzystywanymi obecnie, które są w dalszym ciągu potrzebne i pochłonęły niekiedy znaczne nakłady inwestycyjne.

Innym czynnikiem, który uwidacznia się w procesie powstawania nowych systemów przetwarzania transakcyjnego OLTP jest zjawisko zwane "downsizing". W dużym skrócie zjawisko to można przedstawić jako migrację usług informatycznych ze środowiska mainframe do środowiska wykorzystującego jako platformy systemowe systemy klasy mikro/mini oraz rozwiązania sieciowe o architekturze klient/serwer. Warto odpowiedzieć sobie na pytanie dlaczego zjawisko to tak znacznie wpływa na tworzone obecnie systemy informatyczne obsługi procesów gospodarowania. "Downsizing" oznacza zmniejszenie kosztów, poprawę czasu reakcji systemu, łatwość dopasowywania do zmieniających się warunków gospodarowania, a dedykacja systemów do świadczenia określonych usług informatycznych przybliża system do użytkownika i czyni go łatwiejszym w obsłudze i zarządzaniu. Tam gdzie zjawisko to wywarło już wpływ na rozwój systemów informatycznych, obserwowane jest rozproszenie baz danych i usług informatycznych na różne klasy systemów, od klasy mainframe przez serwery departamentowe, stacje robocze aż po mikrokomputery PC. Systemy te są najczęściej połączone różnego rodzaju sieciami LAN *(Local Area Network)* oraz WAN *(Wide Area Network)* tworząc globalną sieć informatyczną przedsiębiorstwa. Jednocześnie wzrasta liczba i złożoność usług transakcyjnych potrzebnych do wspomagania procesu gospodarowania.

Tendencje te prowadzą do zupełnie nowego typu środowiska przetwarzania transakcyjnego różnego od obecnie dostępnych systemów OLTP. Środowisko to musi zapewnić bezpieczny i niezawodny

dostęp do informacji i usług informatycznych obsługi procesów gospodarowania. Nie są one już zgromadzone w jednym dużym systemie komputerowym, lecz są rozproszone w obrębie całego przedsiębiorstwa niekiedy na systemach komputerowych pochodzących od różnych dostawców, wykorzystujących różne środowiska operacyjne oraz odmienne bazy danych. Jest to środowisko rozproszonego przetwarzania transakcyjnego, które łączy w ramach jednej "globalnej" transakcji zbiór luźno zintegrowanych systemów informatycznych, obsługujących procesy gospodarowania przedsiębiorstwa.

W latach obecnych gdy procesy gospodarowania stają się coraz bardziej złożone i konieczne jest ciągłe ich dopasowywanie do zmieniającego się środowiska ekonomicznego, nie można zapominać o formie i sposobie udostępniania usług transakcyjnych użytkownikom systemu. Graficzny system dialogowy *(Graphical User Interfejs)* oraz sposób współpracy z systemem zaprojektowany w taki sposób aby był przyjazny użytkownikowi, może w znaczny stopniu przyczynić się do poprawy wydajności pracy, zmniejszenia kosztów szkolenia i wdrażania nowych usług informatycznych.

Przetwarzanie transakcyjne w świecie Systemów Otwartych

W ostatnich latach coraz bardziej uwidacznia się tendencja do tworzenia systemów informatycznych w oparciu o standardy systemów otwartych. Odnosi się to również do systemów przetwarzania transakcyjnego OLTP. Standardy systemów otwartych, zorientowane obiektowo przetwarzanie rozproszone, architektura klient/serwer oraz zjawisko "downsizing'u" przyczyniło się do powstania nowych modeli przetwarzania OLTP dla środowiska systemu operacyjnego UNIX, jednego z filarów świata systemów otwartych. Środowiska operacyjne świata systemów otwartych zapewniają elastyczność, skalowalność i przenaszalność systemów aplikacyjnych oraz możliwości integracyjne w takim stopniu, który nie jest możliwy do osiągnięcia dla systemów wykorzystujących platformy systemowe klasy mainframe.

O coraz szybszym rozwoju systemów przetwarzania transakcyjnego OLTP pracujących pod kontrolą systemu operacyjnego UNIX zadecydowało również jego dojrzewanie jako środowiska komercyjnego przetwarzania danych. System operacyjny UNIX SVR4 opracowywany przez USL *(UNIX System Laboratory)* zgodnie z linią rozwojową nakreślaną przez organizację UI *(UNIX International)*, udostępnił nowe cechy środowiska operacyjnego, które umożliwiają projektowanie i implementację bezpiecznych i bardzo wydajnych systemów przetwarzania transakcyjnego OLTP. Cechy te obejmują między innymi nowe systemy plików VjFS *(Veritas Journaling File System)* i VxFS *(Veritas Commercial File System)*. Systemy te charakteryzują się duża efektywnością obsługi dostępu do danych, zapewniają wysoki stopień bezpieczeństwa i rzetelności danych, szybkie odtworzenia systemu plików po awarii systemu oraz przeprowadzanie pełnego lub inkrementacyjnego składowania systemu plików i innych prac administracyjnych bez konieczności przerywania pracy systemu. Innym nowym elementem systemu UNIX SVR4, mającym bezpośredni wpływ na efektywność i bezpieczeństwo systemów OLTP, jest system zarządzania i obsługi pamięci masowych VxVM *(Veritas Disk Manger)*. Wprowadza on pojęcie dysku logicznego, który może obejmować kilka dysków fizycznych *(spanning)* usuwając ograniczenia na wielkość plików w systemie, może być rozmieszczony na kilku dyskach w postaci tzw. skrawków *(stripping)* pozwalając na zrównoleglenie operacji dostępu do danych i poprawienia wydajności systemu plików. Pozwala również na tworzenie kilku kopii przechowywanych informacji *(mirroring)* czyniąc system mniej wrażliwym na awarie pamięci dyskowych.

W systemach przetwarzania transakcyjnego OLTP jednym z najbardziej istotnych czynników branych pod uwagę przy ich ocenie jest bezpieczeństwo oraz poufność danych i usług systemowych. Zagadnienia te nie należały dotychczas do mocnych stron systemu operacyjnego UNIX, lecz ostatnio zwracana jest na nie coraz większa uwaga. Dostępne są już środowiska operacyjne systemu UNIX, w których spełnione są kryteria C2 i B2 bezpieczeństwa systemu zgodne z zaleceniami TCSEC *(Trusted Computer System Evaluation Criteria)* zwanymi także Pomarańczową Księgą i opracowanymi w centrum komputerowym NSC *(National Computer Security Centre)* w USA. Pomarańczowa Księga TCSEC zaleca klasę bezpieczeństwa C2 do stosowania w prawie wszystkich zadaniach komercyjnych i większości zastosowań systemów informatycznych w administracji państwowej. Klasa B2, która jest klasą bardziej wymagającą, wymagana jest w systemach informatycznych narażonych na dywersyjną penetrację. Innym nie mniej ważnym zagadnieniem z obszaru bezpieczeństwa i poufności systemu jest bezpieczeństwo sieci komputerowych. Zagadnienie to nabiera coraz większego znaczenia ze względu na wykorzystywanie technologii przetwarzania rozproszonego typu klient/serwer przy tworzeniu systemów OLTP. Lecz i te aspekty środowiska operacyjnego są już uwzględniane przez systemy operacyjne UNIX. W najnowszych rozwiązaniach spotykane są elementy bezpiecznej architektury systemowej DESD *(Data Elements and Service Definitions)* opracowanej przez organizację ECMA *(European Computer Manufacturers Association)* oraz elementy architektury przetwarzania rozproszonego DCE *(Distributed Computing Environment)* MIT Kerberos *(Cerber)* odpowiedzialne za poufność pracy w systemach sieciowych. System MIT Kerberos został zaaprobowany przez UNIX International oraz OSF *(Open Software Foundation)* jako zalecany do implementacji bezpiecznych i poufnych systemów przetwarzania rozproszonego i najprawdopodobniej będzie miał znaczny wpływ na dalszy kształt architektury DESD.

Kolejnym bardzo istotnym czynnikiem decydującym o przydatności środowiska operacyjnego do implementacji systemów OLTP jest sposób szeregowania zadań w systemie oraz możliwość powiększania mocy obliczeniowej systemu w miarę rosnących potrzeb. System operacyjny UNIX był typowym systemem z podziałem czasu, pozbawionym możliwości szeregowania zadań zgodnie z zadanymi priorytetami i wywłaszczania procesora. Cech, które są konieczne do zapewnienia krótkiego i przewidywalnego czasu reakcji systemu transakcyjnego. Wszystkie te niedostatki zostały obecnie usunięte, co w połączeniu z coraz szerszą dostępnością symetrycznych systemów wieloprocesorowych pracujących pod kontrolą systemu operacyjnego UNIX sprawiło, że środowisko to stało się idealną platformą systemową do implementacji rozproszonych systemów transakcyjnego przetwarzania danych OLTP.

Systemy OLTP są najczęściej systemami, które muszą być nieprzerwanie dostępne (7 dni w tygodniu - 24 godziny dziennie). Jest całkowicie nie do przyjęcia, aby konieczne było przerwanie pracy systemu w celu przeprowadzenia składowania lub odtwarzanie bazy danych, konserwacji sprzętu lub instalacji nowej wersji oprogramowania. Czynności te muszą być wykonywane bez przerywania pracy systemu transakcyjnego, a gdy konieczne jest wyłączenie jednego z procesorów systemu jego usługi muszą być przeniesione w inne miejsce, bez najmniejszych zakłóceń obsługi użytkowników. Zagadnienia odporności na błędy i awarie znalazły również swe odbicie w najnowszych wersjach systemu UNIX. Zostało opracowanych wiele nowych technologii i rozwiązań zapewniających ciągły dostęp do usług i danych systemów informatycznych. Należą do nich:

- pełne i inkrementacyjne składowanie systemu *(On-line Archiving)*,
- obsługa urządzeń podtrzymywania napięcia UPS *(Uninterruptible Power Systems)*,
- urządzenia zewnętrzne o kilku portach *(Dual-ported peripherals)*,
- dyski zwierciadlane *(Disk mirroring)*,
- systemy dysków macierzowych RAID *(Redundant Array of Inexpensive Disks)*,
- przetwarzanie rozproszone typu klient/serwer *(Client/Server Architecture)*.

Wszystkie wymienione powyżej nowe możliwości systemu operacyjnego UNIX nie obejmują jednak tego co jest niezbędne do implementacji efektywnego, niezawodnego i bezpiecznego systemu OLTP - systemu zarządzania transakcjami *(Transaction Manager)*. Środowisko operacyjne UNIX musi być uzupełnione o dodatkowe warstwy oprogramowania systemowego obejmujące system obsługi bazy danych, system obsługi przetwarzania transakcyjnego i rozproszonego oraz systemy usług sieciowych LAN i WAN. Dopiero tak zbudowane środowisko systemowe może być fundamentem budowania pełnowartościowego systemu przetwarzania transakcyjnego OLTP.

Na zakończenie tego fragmentu pracy warto odpowiedzieć na pytanie dlaczego w latach obecnych uwidacznia się tak silna tendencja tworzenia systemów OLTP w środowisku systemów otwartych, a szczególnie pracujących pod kontrolą systemu operacyjnego UNIX. Do zalet takiego podejścia można zaliczyć:

- przenaszalność i skalowalność aplikacji czyli możliwość pracy na platformach systemowych o wymaganej mocy obliczeniowej i pochodzących od różnych producentów,

- bardzo korzystny stosunek ceny systemu do jego wydajności, szczególnie w porównaniu ze scentralizowanymi rozwiązaniami bazującymi na komputerach klasy mainframe,

- przystosowanie i możliwości pracy w sieciach komunikacyjnych LAN i WAN oraz w środowiskach przetwarzania rozproszonego,

- szeroka rodzina narzędzi tworzenia oprogramowania aplikacyjnego pozwalającego na szybkie projektowanie, implementację i wdrażanie systemów transakcyjnych OLTP wykorzystujących najnowsze graficzne systemy dialogowe (GUI) współpracy z użytkownikiem.

Atrybuty przetwarzania transakcyjnego

Wiele systemów obsługi procesów gospodarowania wymaga wykorzystania zasad przetwarzania transakcyjnego OLTP do implementacji koniecznych usług informatycznych. Różne środowiska systemów informatycznych - informacyjne systemy zarządzania i wspomagania podejmowania decyzji, systemy przetwarzania wsadowego, systemy interakcyjne, systemy uwarunkowane czasowo *(real-time)* - posiadają odmienne charakterystyki. Najbardziej istotną różnicą pomiędzy systemami OLTP i pozostałymi środowiskami przetwarzania danych jest stopień w jakim system zapewnia:

- rzetelność i integralność danych,
- bezpieczeństwo i poufność danych i usług,
- zdolność odtwarzania po awarii,
- wydajność i czas reakcji systemu,
- standaryzację rozwiązań.

Systemy OLTP są projektowane w taki sposób, aby optymalnie adresowały powyższe aspekty środowiska aplikacyjnego. Inne typy systemów są najczęściej systemami uniwersalnymi, które pomimo faktu, że mogą zapewniać usługi transakcyjne użytkownikom systemu informatycznego, nie są w pełni systemami OLTP. Dla przykładu brak automatycznego

odtwarzania stanu systemu po awarii sprzętu lub oprogramowania, wymagający ręcznego odtworzenia transakcji, jest czynnikiem dyskwalifikującym system jako system transakcyjny, który zabezpiecza strategiczne usługi informatyczne przedsiębiorstwa.

Cechami przetwarzania transakcyjnego mającymi największy wpływ na architekturę systemu są:

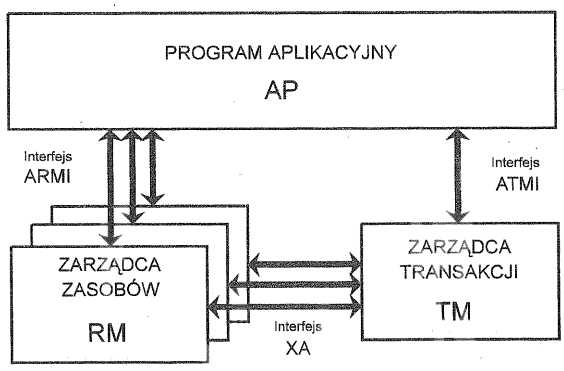
- predefiniowany zakres usług transakcyjnych o zróżnicowanych priorytetach,
- wiele krótkich interakcji i duża liczba aktywnych użytkowników systemu,
- krótki i przewidywalny czas reakcji oraz wysoka wydajność systemu,
- konieczność współpracy z dużymi bazami danych.

Wymagają one aby:

- moduły komunikacyjne i sieciowe systemu OLTP były efektywne i nie wprowadzały dodatkowego obciążenia środowiska operacyjnego *(system overhead)*,
- zasoby systemu były wykorzystywane efektywnie aby zapewnić dużą przepustowość systemu OLTP,
- wydajność systemu wzrastała proporcjonalnie wraz ze wzrostem obciążenia systemu OLTP (większa liczba użytkowników),
- istniała możliwość dynamicznych zmian środowiska systemowego dopasowujących je do aktualnych potrzeb i warunków bez konieczności przerywania pracy systemu,
- system OLTP był systemem pewnym, odpornym na awarie sprzętu i oprogramowania oraz zapewniał pełne zakończenie transakcji, a w przypadku błędu, jej anulowanie i przywrócenie bazy danych do stanu pierwotnego.

Model DTP X/OPEN

W roku 1987, styczniowe wydanie dokumentu X/Open Portability Guide (XPG) zobowiązało grupę X/Open do opracowania standardów przetwarzania transakcyjnego dla komercyjnych programów aplikacyjnych pracujących w środowisku systemu operacyjnego UNIX. W odpowiedzi na to, już w lipcu roku 1987 grupa X/Open opublikowała pierwszą wersję modelu OLTP dla systemów otwartych *(TP Reference Model)*.
Prace nad udoskonaleniem modelu i uzupełnieniem go o elementy przetwarzania rozproszonego oraz architektury klient/serwer, trwały jednak w dalszym ciągu i doprowadziły do opublikowania, w lipcu roku 1989, przejściowego modelu rozproszonego przetwarzania transakcyjnego DTP *(Interim Reference Model for Distributed Transaction Processing)*. Celem tego dokumentu było poinformowanie o kierunkach standaryzacji przetwarzania transakcyjnego w świecie systemów otwartych prowadzonych przez grupę roboczą organizacji X/Open *(Transaction Processing Working Group)*. Obecnie trwają końcowe prace nad ostateczną wersją modelu DTP X/OPEN i standardami OSI-TP (ISO 10026).



RYS. 1. Model DTP X/Open

Model X/Open DTP definiuje następujące elementy systemu przetwarzania transakcyjnego (rys. 1):

- **Program aplikacyjny (AP)** - element środowiska transakcyjnego definiujący

transakcję (sekwencja operacji) oraz wykorzystywane przez nią zasoby systemowe (terminale - współpraca z użytkownikiem, bazy danych - dostęp i aktualizacja danych).

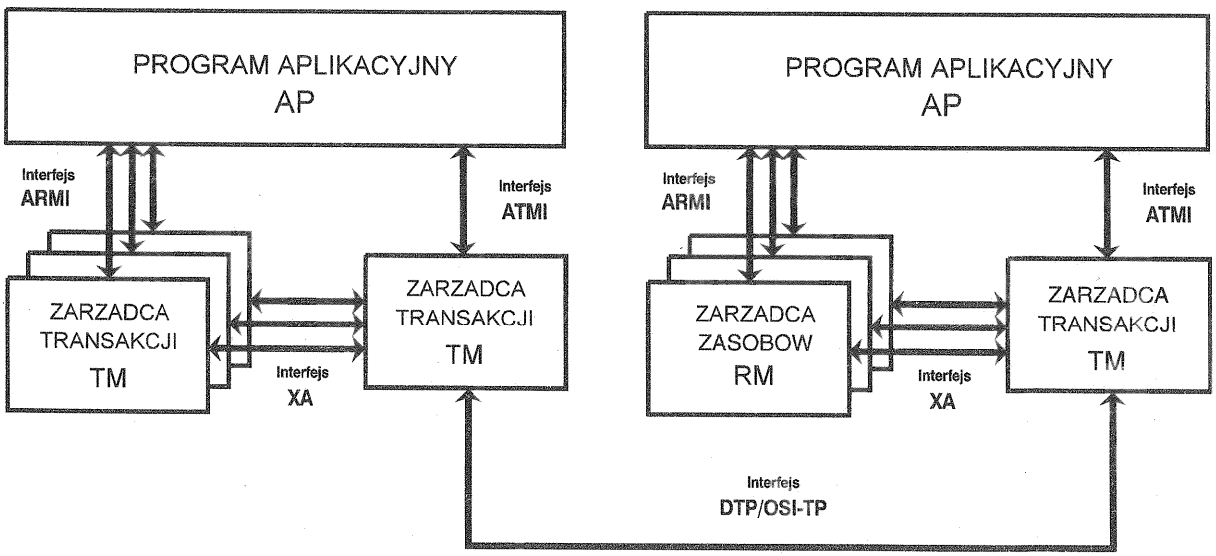
- **Zarządca zasobów (RM)** - element środowiska transakcyjnego odpowiedzialny za współpracę ze współdzielonymi zasobami systemu informatycznego (bazy danych, urządzenia wprowadzania i wyprowadzania danych oraz składowania i odtwarzania danych).

- **Zarządca transakcji (TM)** - element środowiska transakcyjnego odpowiedzialny za zarządzanie transakcją "globalną" tzn. transakcją obejmującą swoim zasięgiem wiele elementów zarządzania zasobami (RM), śledzący jej przebieg, potwierdzający jej poprawne zakończenie i odtwarzający stan środowiska w przypadku awarii.

Elementy te, w połączeniu z innymi elementami środowiska systemowego takimi jak: system operacyjny, system obsługi bazy danych oraz systemy komunikacyjne i sieciowe, tworzą środowisko przetwarzania transakcyjnego OLTP. Tym co odróżnia ten model od modeli systemów OLTP wykorzystujących własne rozwiązania producentów są standardy interfejsów i protokołów pomiędzy jego elementami.

Model DTP X/Open definiuje transakcję jako elementarną, niepodzielną i dającą się anulować jednostkę pracy systemu informatycznego. Systemu, zbudowanego w oparciu o niejednorodną sieć systemów komputerowych. Opracowanie tego modelu miało na celu określenie takich standardów, aby mogły współpracować ze sobą systemy komputerowe pochodzące od różnych producentów, pracujące pod kontrolą odmiennych systemów operacyjnych i wykorzystujących różne systemy zarządzania bazami danych, powiązanych ze sobą sieciami LAN i WAN, tworząc jednorodne środowisko usług transakcyjnych (rys. 2).

Rozproszone przetwarzanie transakcyjne OLTP jest środowiskiem systemowym, które wymaga bardzo precyzyjnego zdefiniowania zasad współpracy i protokołów pomiędzy poszczególnymi elementami systemu. Model DTP X/Open określa obsługę transakcji i sposób dostępu do zasobów w środowisku rozproszonym. Zapewnia również integralność i rzetelność danych oraz pewność i dostępność usług transakcyjnych. Za wszystkie te zagadnienia są odpowiedzialni zarządcy transakcji (TM) poszczególnych systemów OLTP, które komunikują się za pośrednictwem



RYS. 2. Model DTP X/Open w środowisku przetwarzania rozproszonego

protokołu rozproszonego przetwarzania transakcyjnego DTP (OSI-TP).

Istnieją cztery podstawowe kryteria oceny systemów transakcyjnych OLTP. Środowiska systemowe, które nie spełniają tych kryteriów nie mogą być uważane za w pełni transakcyjne. Kryteria te noszą nazwę testu ACID i są nimi:

- Niepodzielność *(Atomicity)* - transakcja jest elementarną i niepodzielną jednostką pracy systemu transakcyjnego, która musi być zakończona całkowicie lub anulowana przywracając stan współdzielonych zasobów do stanu początkowego.

- Wiarygodność *(Consistency)* - transakcja może jedynie przekształcać współdzielone zasoby systemowe ze stanu poprawnego do nowego stanu poprawnego.

- Wzajemna izolacja *(Isolation)* - zmiany stanu współdzielonych zasobów występujące w ramach transakcji mogą być widoczne dla pozostałych transakcji dopiero po jej zakończeniu.

- Nieodwracalność *(Durability)* - z chwilą pełnego zakończenia transakcji wszystkie zmiany stanu współdzielonych zasobów muszą być trwałe i odporne na mogące wystąpić awarie systemu informatycznego.

Interfejsy i protokoły modelu DTP X/Open

Interfejsy i protokoły modelu przetwarzania transakcyjnego DTP X/Open można podzielić na dwie podstawowe kategorie. Pierwsza z nich obejmuje te z nich, które są wykorzystywane przez projektantów i programistów programów transakcyjnych oraz administratorów systemów OLTP. Należą do niej:

- ARMI *(X/Open Application-to-Resource Manager Interface)* - interfejs definiujący zasady i sposób współpracy pomiędzy programami aplikacyjnymi (AP) i zarządcą (RM) np. język SQL dostępu do relacyjnej bazy danych.

- ATMI *(TX X/Open Application-to-Transaction Manager Interface)* - interfejs współpracy programu aplikacyjnego (AP) z zarządcą transakcji (TM) definiujący parametry i zakres transakcji.

Druga kategoria interfejsów i protokołów modelu DTP X/Open określa zasady współpracy pomiędzy poszczególnymi jego elementami i nie jest widoczna dla twórców systemu. Obejmuje ona dwa protokoły:

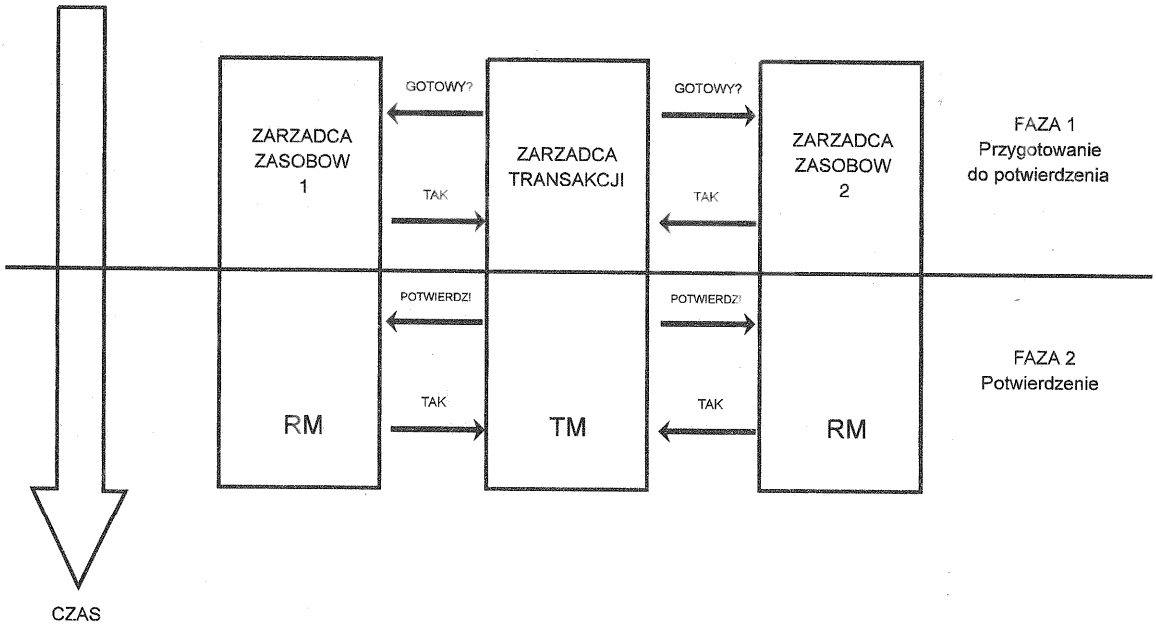
- DTP *(Distributed Transaction Processing Protocol)* - interfejs pomiędzy zarządcami transakcji (TM) uczestniczącymi w obsłudze "globalnej" transakcji rozproszonego systemu OLTP.

- XA *(X/Open Transaction Manager-to-Resource Manager Interface)* - określający zasady współpracy pomiędzy zarządcą transakcji (TM) i zarządcą zasobów (RM). Służy on do przekazywania informacji o samej transakcji (identyfikator, początek i zakończenie transakcji) oraz implementacji dwufazowego protokołu potwierdzeń 2PC *(Two-Phase Commit Protocol)*.

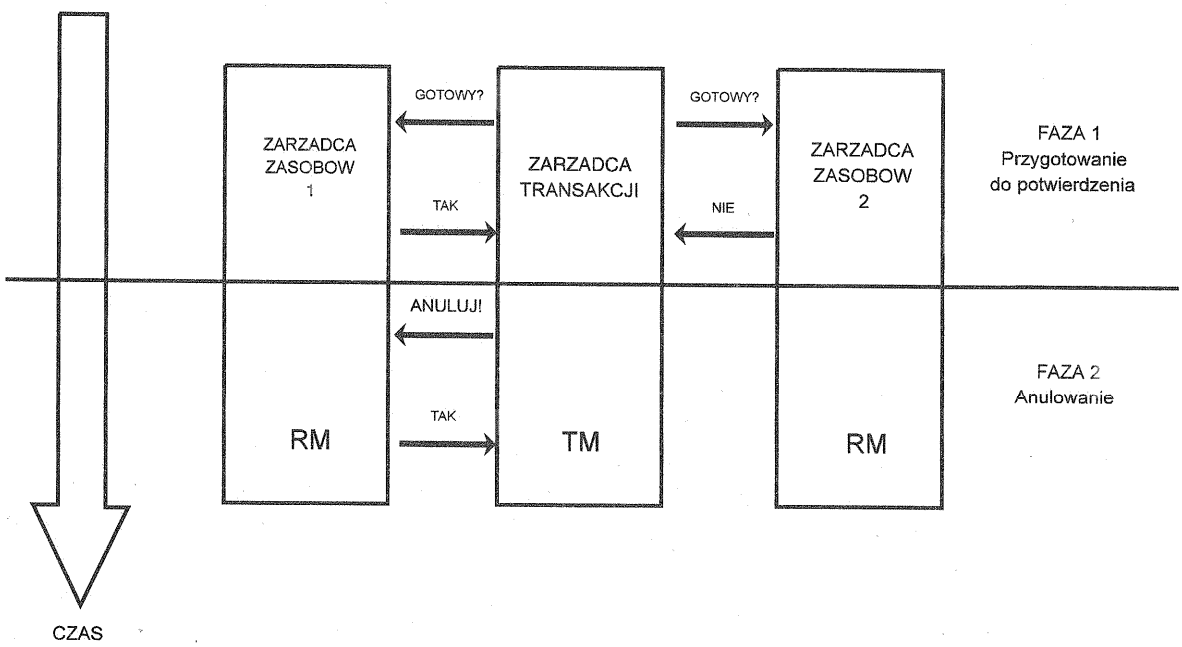
W środowisku rozproszonego przetwarzania transakcyjnego OLTP, tzw. "globalna" usługa transakcyjna może wymagać dostępu do zasobów i usług systemu znajdujących się w różnych miejscach sieci komputerowej. Jest to szczególnie istotne gdy w ramach jednej transakcji istnieje konieczność aktualizacji wielu baz danych, które mogą pochodzić od różnych dostawców i pracować w odmiennych środowiskach operacyjnych. Dla tych transakcji, aby

zapewnić rzetelność i integralność baz danych systemu, zarządca transakcji (TM) kontroluje przebieg transakcji i współpracę z wymaganymi zarządcami zasobów (RM) za pomocą dwufazowego protokołu potwierdzeń (2PC).

Przebieg transakcji z wykorzystaniem dwufazowego protokołu potwierdzeń (2PC) jest następujący. Zarządca transakcji (TM) przekazuje żądania obsługi do wymaganych zarządców zasobów (RM), które pracują dalej niezależnie. Potwierdzenie transakcji odbywa się w dwóch fazach (rys. 3). W pierwszej z nich, TM sprawdza czy każdy z zarządców zasobów (RM) zakończył pracę i czy jest gotowy do potwierdzenia transakcji. Faza druga to właściwe potwierdzenie zakończenia transakcji, lecz następuje ono jedynie wtedy gdy TM otrzyma wszystkie odpowiedzi pozytywne. W sytuacji gdy choć jeden RM nie jest gotowy do potwierdze-



RYS. 3. Dwufazowy protokół potwierdzeń 2PC - potwierdzenie transakcji



RYS. 4. Dwufazowy protokół potwierdzeń 2PC - anulowanie transakcji

nia, zarządca transakcji przerywa transakcję i inicjuje proces odtworzenia stanu zasobów, który istniał przed jej rozpoczęciem (rys. 4). Dwufazowy protokół potwierdzeń (2PC) jest kluczowym elementem zapewniającym rzetelność i integralność danych rozproszonego systemu OLTP.

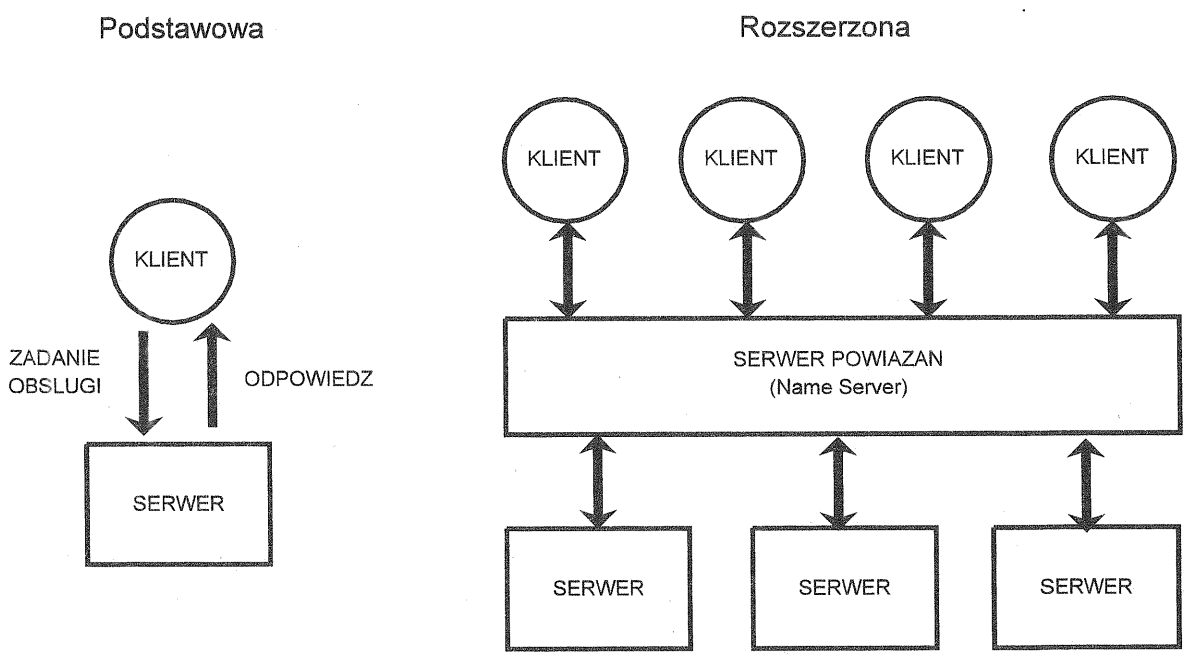
Architektura klient/serwer przetwarzania transakcyjnego

Systemy przetwarzania transakcyjnego OLTP, ze względu na sposób obsługi użytkownika, są przykładem architektury klient/serwer. Klientem jest podzbiór zasobów systemowych potrzebnych do zapoczątkowania transakcji i obsługi współpracy z użytkownikiem. Proces klienta przygotowuje i przekazuje zlecenia obsługi do serwerów systemu. Serwer to podzbiór zasobów systemowych wymaganych do obsługi zleceń transakcyjnych. Serwer może być przygotowany do świadczenia wielu usług transakcyjnych, które są jego procedurami obsługującymi dany typ żądania.

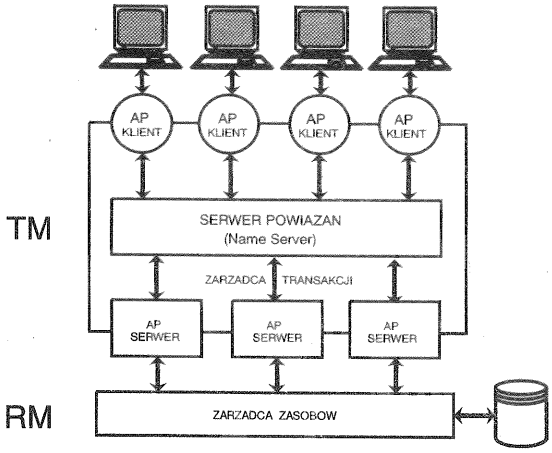
Znane są dwa modele architektury klient/serwer: podstawowy i rozszerzony (rys. 5). W modelu podstawowym proces klienta przekazuje zlecenie obsługi do znanego mu procesu serwera. Serwer przetwarza zlecenie i zwraca rezultaty do procesu klienta. Model ten wymusza relację jeden-do-jednego pomiędzy klientem i serwerem systemu przekreślając możliwość współdzielenia zasobów serwera. Wymaga również, aby klient znał szczegóły serwera a on sam był przygotowany do obsługi wszystkich typów zleceń transakcyjnych.

Z tych też powodów, do implementacji systemów OLTP bardziej przydatny jest rozszerzony model klient/serwer. W modelu tym, został wprowadzony dodatkowy serwer powiązań "Name Server" pomiędzy procesy klientów i serwerów. Uniezależnia on obsługę zlecenia od położenia serwera oraz umożliwia symboliczna adresację usług transakcyjnych. Serwer powiązań jest odpowiedzialny za przyjmowanie zleceń, ich szeregowanie i przekazywanie ich do obsługi w dostępnych serwerach systemu.

Elementy modelu DTP X/Open przetwarzania transakcyjnego współpracują ze sobą zgodnie z architekturą klient/serwer (rys. 6). Klient, program aplikacyjny (AP), obsługuje współpracę użytkownika z syste-



RYS. 5. Architektura Klient/Serwer



RYS. 6. Model DTP X/Open - Architektura Klient/Serwer

mem transakcyjnym *(CUI - Computer User Interface)*, przygotowuje zlecenia i przekazuje je do zarządcy transakcji (TM). Zarządca transakcji (serwer powiązań) przekazuje otrzymane zlecenie do serwera, którym jest inny program aplikacyjny (AP). Serwer przetwarza zlecenie i zwraca odpowiedź za pośrednictwem zarządcy zasobów (TM).

Architektura klient/serwer, ze względu na jej własności i wykorzystywane zasady pracy jest szczególnie przydatna do implementacji systemów OLTP w środowisku przetwarzania rozproszonego. Podstawowymi jej zaletami w stosunku do przetwarzania scentralizowanego są: współbieżność przetwarzania, większa wydajność i niezawodność systemu sieciowego oraz możliwość inkrementacyjnego zwiększania mocy obliczeniowej i zakresu usług systemu.

W systemie rozproszonym współbieżność przetwarzania oznacza równoległą pracę serwerów i klientów systemu w wielu węzłach sieci. Klienci, których zadaniem jest zapoczątkowanie transakcji oraz obsługa współpracy z użytkownikiem, nie muszą znać rozmieszczenia w sieci usług transakcyjnych *(Location Transparency)* i mogą pracować blisko obsługiwanego użytkownika. Serwery natomiast, mogą być umieszczone w najlepszym miejscu sieci komputerowej biorąc pod uwagę najlepsze wykorzystanie zasobów systemu. Wszystkie te czynniki mają olbrzymi wpływ na całkowitą przepustowość i czas reakcji systemu *(Scalability)*. Administrator systemu, w zależności od obciążenia systemu, może w prosty sposób uruchomić kolejne serwery lub przenosić je w inne miejsca sieci dla zapewnienia optymalnego wykorzystania mocy obliczeniowej systemu.

Większa niezawodność systemu przetwarzania rozproszonego jest ponownie wynikiem rozdzieleniem funkcji klienta i serwera, uniezależnieniem systemu od ich wzajemnego położenia oraz możliwości jednoczesnej pracy wielu serwerów. W sytuacji gdy nastąpi awaria jednego z serwerów, jego funkcje mogą być przeniesione i uruchomione na innym serwerze dostępnym w sieci. Zapewnia to natychmiastowy restart istotnych usług transakcyjnych bez zakłócania i przerywania pracy systemu. W systemach scentralizowanych, gdy nastąpi awaria systemu komputerowego, z wyjątkiem systemów klasy "Fault Tolerant" lub Continous Processing", dalsza praca systemu transakcyjnego jest niemożliwa.

Kolejną zaletą przetwarzania rozproszonego opartego na model klient/serwer jest jego elastyczność *(Flexibility)* w zakresie świadczonych usług oraz liczby obsługiwanych użytkowników *(Throughput)*. Wynika to z modularności systemu sieciowego. Klienci i serwery systemu są zamkniętymi elementami systemu OLTP, które współpracują między sobą, a ich liczba może być zwiększana w miarę potrzeb. Dodatkowe serwery to kolejne usługi transakcyjne oraz poprawa czasu reakcji systemu w miarę wzrostu jego obiążenia. Koszty związane z uruchomieniem kolejnego węzła systemu rozproszonego są jednocześnie znacznie mniejsze od nakładów, które są konieczne do rozbudowy centralnego systemu komputerowego przedsiębiorstwa.

System przetwarzania transakcyjnego TUXEDO ETP 4.2

System TUXEDO został zaprojektowany do obsługi systemów rozproszonego przetwarzania transakcyjnego OLTP pracujących w świecie systemów otwartych. Systemów, charakteryzujących się modelami przetwarzania typu klient/serwer i komunikacji równy-z-równym *(peer-to-peer)*, koniecznością aktualizacji wielu baz danych w ramach jednej "globalnej" transakcji oraz równoważeniem obciążenia systemu transakcyjnego w obrębie sieci.

System TUXEDO wkroczył już w drugie dziesięciolecie obecności na rynku jako komercyjny system przetwarzania transakcyjnego OLTP. Został on opracowany w roku 1978 w firmie AT&T Bell Laboratory do obsługi usług telekomunikacyjnych i uzyskał pełną akceptację użytkowników. Od tego czasu, system TUXEDO był uważany za system przetwarzania transakcyjnego pracujący w środowisku systemu UNIX dedykowany dla przemysłu telekomunikacyjnego. Jednocześnie uzyskiwał on coraz większe uznanie w systemach obsługi banków, hoteli, handlu, szpitali i przemysłu. Pierwsze komercyjne wersje systemu TUXEDO pojawiły się w roku 1984 (TUXEDO Release 4 w roku 1990) i chociaż sam system został opracowany w AT&T Bell Laboratory, obecnie jest licencjonowany i rozwijany przez USL *(UNIX System Laboratory)*. USL udostępnia kod źródłowy systemu TUXEDO dostawcom systemów informatycznych, którzy przenoszą go na własne platformy systemowe, uzupełniają o nowe elementy i oferują jako pełny system przetwarzania transakcyjnego dla swoich klientów. Do dnia dzisiejszego, USL udzieliło licencji na system TUXEDO dla około 25 dostawców systemów informatycznych. Lista tych dostawców obejmuje między innymi: AT&T, Bull, Data General, IBM, HP, ICL, Motorola, Olivetti, Pyramid, Sequent, SCO, Sun, Unisys.

W 1991 roku USL *(UNIX System Laboratory)* wprowadził na rynek najnowszą wersję systemu TUXEDO ETP 4.2 *(Enterprise Transaction Processing)* dla środowiska UNIX SVR4. Wersja ta jest pierwszą próbą uwzględnienia w systemie OLTP środowisk operacyjnych innych niż UNIX i objęcia swoim zasięgiem systemów komputerowych począwszy od mainframe, poprzez serwery departamentowe aż po mikrokomputery PC.

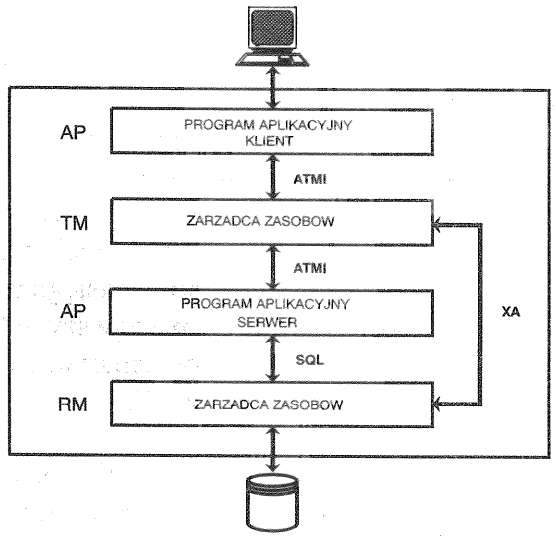
Od chwili pojawienia się pierwszej komercyjnej wersji, system TUXEDO staje się "de facto" standardem systemów OLTP świata systemów otwartych, oferując wiele możliwości niedostępnych w innych "otwartych" produktach OLTP. W swojej pracy wykorzystuje on silne strony środowiska operacyjnego UNIX System V, dla którego został opracowany, uzupełniając je o dodatkowe cechy: wydajność, niezawodność, rzetelność i bezpieczeństwo oraz rozszerza platformy systemowe ich implementacji o systemy PC (MS-DOS, OS/2) i Macintosh (Finder) oraz systemy mainframe (MVS/CICS, VME/TPMS).

System TUXEDO ETP 4.2 obejmuje następujące elementy:

- System/T - zarządca transakcji systemu,
- System/WS - brama stacji roboczych (MS-DOS, OS/2, Finder, UNIX),
- System/HOST - brama środowiska MVS/CICS,
- System/DTS - brama środowiska VME/TPMS,
- System/D - system zarządzania bazą danych.

Kręgosłupem i podstwowym elementem systemu TUXEDO jest zarządca transakcji System/T (rys. 7). Do jego zadań należy:

- obsługa komunikacji pomiędzy klientami i serwerami systemu transakcyjnego,
- adresacja usług transakcyjnych systemu *(name and data dependent routing)*,

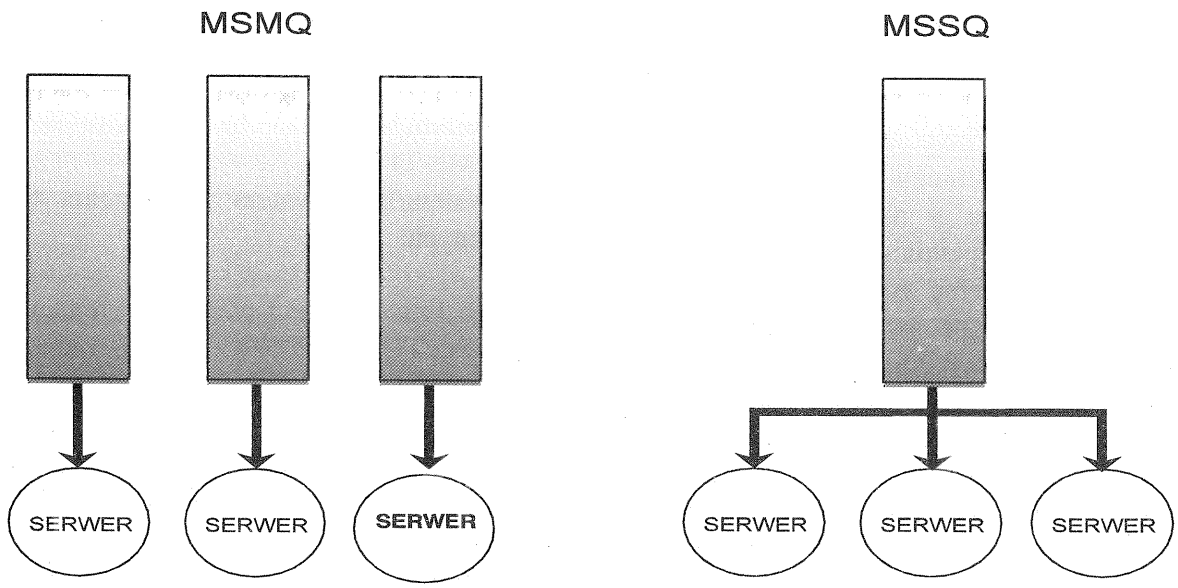


RYS. 7. TUXEDO System/T

- kontrola zasobów systemu informatycznego (procesory, pamięć, dyski, komunikacja),
- zarządzanie transakcjami, obsługa żądań (kolejki), rejestracja transakcji *(logging)*, kontrola współbieżności dostępu i odtwarzanie po awarii systemu,
- koordynacja potwierdzeń i anulowania transakcji (awaria węzła lub sieci komunikacyjnej, globalny zastój "deadlock" systemu),
- zarządzanie systemem transakcyjnym w taki sposób aby zapewnić zrównoważone obciążenie systemu i maksymalną jego przepustowość.

Zapewnienie stabilności i pewności pracy systemu transakcyjnego jest jednym z podstawowych wymagań stawianych systemowi informatycznemu. TUXEDO System/T posiada wiele cech i własności, które w znacznym stopniu poprawiają dostępność systemu transakcyjnego. Obejmuje to wykorzystywanie dwufazowego protokółu potwierdzeń transakcji (2PC), badanie żywotności procesów *(process viability checks)*, badanie maksymalnego czasu reakcji serwera *(time-out checks)*, automatyczny restart i odtwarzanie serwerów oraz definiowane przez użytkownika procedury odtwarzania systemu po awarii. Dodatkowo, dostępność usług transakcyjnych może być zwiększona poprzez uruchomienie wielu serwerów na kilku węzłach pracujących w sieci.

TUXEDO System/T został zaprojektowany zgodnie z rozszerzoną architekturą klient/serwer. Architektura ta umożliwia współbieżną obsługę zleceń transakcyjnych oraz ich szeregowanie zgodnie z priorytetami procesów gospodarowania. TUXEDO System/T dopuszcza dwie metody równoważenia obciążenia systemu i szeregowania zadań (rys. 8). Pierwsza z tych metod zwana MSMQ *(Multiple Servers, Multiple Queues)* wykorzystuje tzw. współczynniki obciążenia *(load factors)*. Są one przypisywane poszczególnym usługom transak-



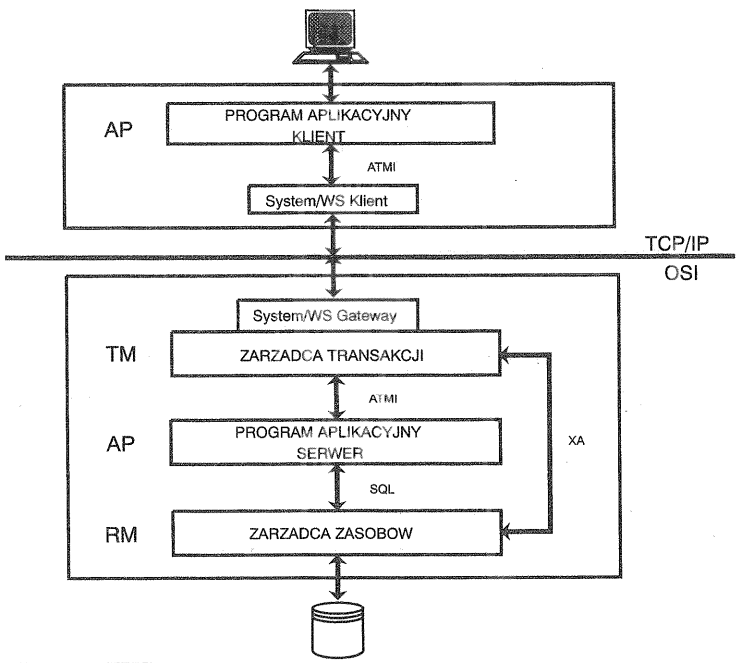
RYS. 8 TUXEDO System/T - Szeregowanie żądań obsługi

cyjnym w ramach serwera. Metoda wykorzystuje algorytm równoważenia obciążenia przy umieszczaniu żądania w kolejce serwera, który zapewnia jego najszybszą obsługę. Druga metoda, MSSQ *(Multiple Servers, Single Queue)*, wykorzystuje jedną kolejkę żądań dla kilku serwerów a kolejne żądania obsługi są przekazywane do pierwszego dostępnego serwera. Usługi mogą mieć jednak przypisane priorytety i poszczególne serwery przeglądają kolejkę oczekujących żądań pobierając do obsługi żądanie o najwyższym priorytecie. System zapewnia równocześnie, że żądania o niskim priorytecie nie będą oczekiwały na obsługę nieskończenie długo na skutek pojawiających się żądań o wyższym priorytecie.

Administracja i zarządzanie systemem rozproszonego przetwarzania transakcyjnego jest kolejnym bardzo istotnym kryterium oceny systemu OLTP. TUXEDO System/T udostępnia bogaty zestaw narzędzi do dynamicznej administracji, konfigurowania i śledzenia wykorzystania zasobów systemu informatycznego. Definicja systemu transakcyjnego, która może być dynamicznie zmieniana, obejmuje: zasoby, systemy, sieci, serwery, usługi, zasady adresacji usług i użytkowników systemu. Możliwa jest również dynamiczna rekonfiguracja systemu transakcyjnego w zależności od jego obciążenia i wymaganych usług. Rekonfiguracja ta obejmuje zmianę liczby i rozmieszczenia serwerów, zmianę zestawu dostępnych usług, współczynników obciążenia i ich priorytetów.

TUXEDO System/T jest dojrzałym i szeroko dostępnym produktem. Pozostałe elementy systemu, System/WS, System/HOST i System/DTS, rozszerzają definicję otwartego systemu OLTP poza środowisko systemu operacyjnego UNIX o środowiska systemowe MS-DOS, OS/2, MVS/CICS i VME/TPMS.

TUXEDO System/WS jest bramą stacji roboczych systemu OLTP pozwalającą na pracę klientów i serwerów systemu na rozłącznych systemach komputerowych (rys. 9). Rozwiązanie takie zmniejsza obciążenie serwerów systemu. Przesuwa bowiem część programów aplikacyjnych odpowiedzialnych za inicjowanie transakcji i obsługę systemu dialogowego (klient) bliżej użytkownika na jego stację roboczą lub PC. TUXEDO System/WS jest obecnie dostępny dla środowiska operacyjnego MS-DOS, OS/2 i UNIX SVR4.

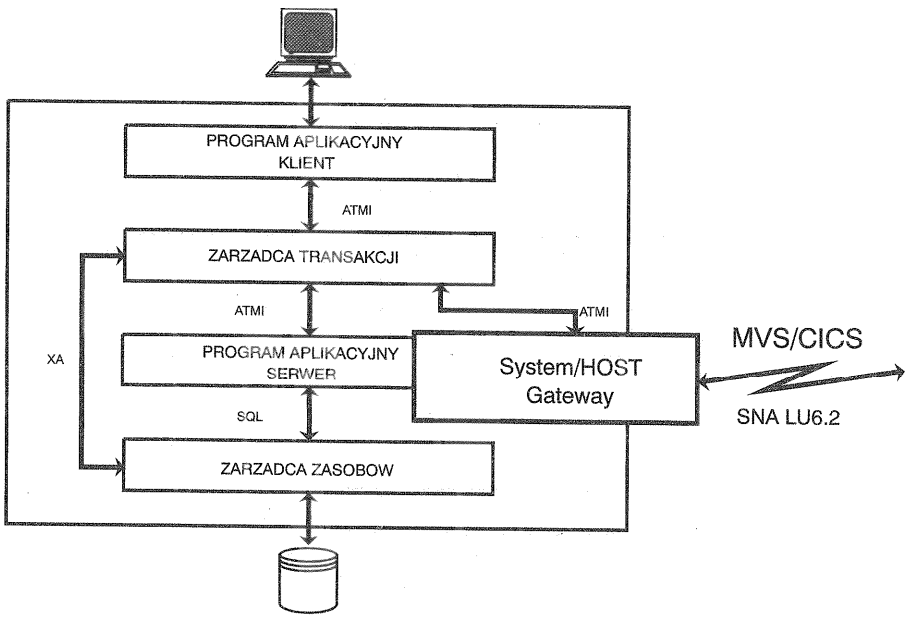


RYS. 9. TUXEDO System/WS

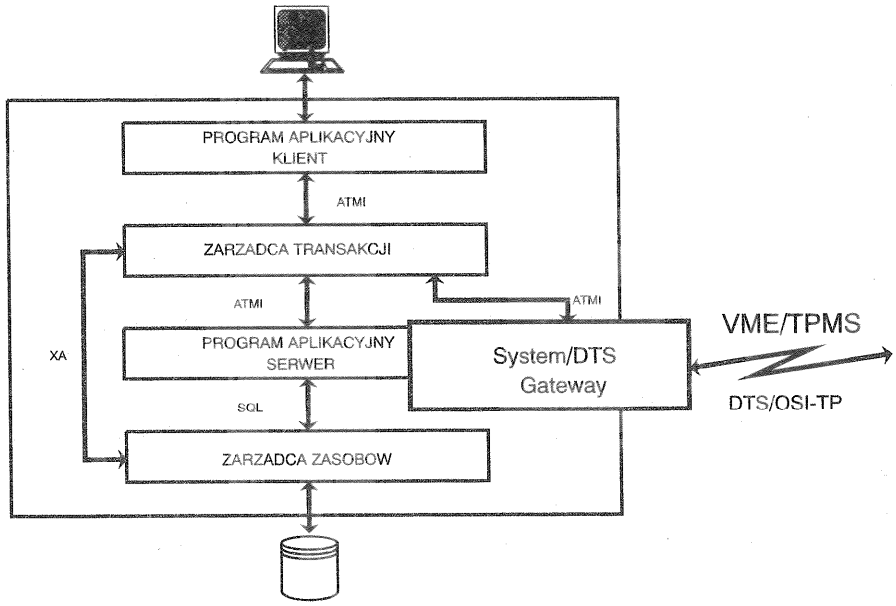
TUXEDO System/HOST i TUXEDO System/DTS są bramami komunikacyjnymi pozwalającymi na integrację systemu otwartego przetwarzania transakcyjnego TUXEDO z systemami OLTP o architekturze zamkniętej opracowanej przez dostawców systemów informatycznych IBM i ICL. TUXEDO System/HOST (rys. 10) umożliwia klientom systemu transakcyjnego żądanie usług transakcyjnych dostępnych w środowisku systemowym MVS/CICS łączem komunikacyjnym SNA LU6.2. Brama System/HOST jest serwerem systemu TUXEDO o jednej usłudze transakcyjnej. Serwer przyjmuje żądanie od klienta TUXEDO i przekazuje go do systemu MVS/CICS. Żądanie to jest następnie skierowywane do obsługi przez odpowiedni program aplikacyjny MVS/CICS (w zależności od jego typu), a odpowiedź jest zwracana do klienta TUXEDO za pośrednictwem bramy System/HOST.

W podobny sposób pracuje brama TUXEDO System/DTS. Umożliwia ona integrację systemu TUXEDO ze środowiskiem systemowym VME/TPMS firmy ICL. TUXEDO System/DTS (rys. 11) wykorzystuje opracowany przez ICL protokół DTS (w przyszłości OSI-TP) łącz komunikacyjnych OSI (OSLAN, X25) do współpracy z systemem przetwarzania transakcyjnego TPMS *(Transaction Processing Management System)*. Jest on serwerem TUXEDO o jednej usłudze transakcyjnej odpowiedzialnej za przekazywanie żądań usług transakcyjnych i odpowiedzi na te żądania z/do systemu TPMS.

USL *(UNIX System Laboratory)* poza powyżej wymienionymi elementami systemu TUXEDO, dla kompletności OLTP dostarcza dwa dodatkowe: TUXEDO System/D i DES. TUXEDO System/D jest systemem zarządzania bazą danych specjalnie zaprojektowaną dla systemów OLTP. System ten bazuje na sieciowym modelu danych lecz uwzględnia relacyjny obraz bazy danych oraz biblioteki dostępu ISAM. Dostępne są: język ANSI SQL (interakcyjny i w postaci bibliotek języka C), DDL *(Data Definition Language)* i DML *(Data Manipulation Language)* dla sieciowych i hierarchicznych modeli bazy danych, generator raportów oraz pełny zestaw narzędzi przeznaczonych do administracji, konserwacji i zarządzania bazą danych w środowisku OLTP. TUXEDO DES natomiast, jest systemem zarządzania formatkami, wspomagającym proces tworzenia aplikacji OLTP przy projektowaniu dialogowego systemu współpracy z użytkownikiem.



RYS. 10. TUXEDO System/HOST



RYS. 11. TUXEDO System/DTS

Zarządcy zasobów

Model rozproszonego przetwarzania rozproszonego DTP X/Open, definiuje zarządcę zasobów (RM) jako element oprogramowania systemowego, który pracuje na zlecenie programów aplikacyjnych oraz obsługuje dwufazowy protokół potwierdzeń (2PC) interfejsu XA. Najczęściej zarządcą zasobów jest system zarządzania bazy danych (DBMS), lecz model DTP dopuszcza aby zarządcą zasobów był dowolny dostawca usług dla programów aplikacyjnych np. serwer obsługujący wysoko wydajną drukarkę laserową.

TUXEDO System/T, zarządca transakcji świata systemów otwartych, jest przygotowany do współpracy ze wszystkimi dostępnymi obecnie systemami obsługi relacyjnych baz danych (RDBMS). Nie wszystkie one jednak są obecnie przygotowane do współpracy z TUXEDO za pośrednictwem protokołu XA. W miarę jak producenci RDBMS będą implementowali protokół X/Open XA w swoich produktach, TUXEDO będzie zdolne do koordynacji "globalnej" transakcji obejmującej wiele niejednorodnych baz danych pracujących w sieci komputerowej systemu informatycznego.

Obecnie dostępne są następujące systemy zarządzania bazami danych obsługującymi protokół X/Open XA:

- INFORMIX OnLine 5.0 (INFORMIX-TP/XA),
- ISAM-XA,
- INGRES 6.5,
- ORACLE 7.0.

Pozostałe nie są przygotowane do obsługi protokółu XA. Ponieważ TUXEDO System/T nie jest w stanie rozpoznać, czy protokół XA jest dostępny przy współpracy z wykorzystywanym zarządcą zasobów (RM), odpowiedzialność za potwierdzanie i anulowanie transakcji, w takiej sytuacji, spoczywa na programie aplikacyjnym. Służą do tego celu zlecenia języka SQL.

Pewne systemy zarządzania relacyjną bazą danych wykorzystują zlecenie SQL BEGIN TRANSACTION do wskazania początku transakcji. Zlecenie to musi poprzedzać pierwsze zlecenie SQL dostępu do bazy danych, a SQL COMMIT potwierdzające transakcję lub SQL ROLLBACK anulujące transakcję, musi być ostatnim zleceniem definiującym zakres transakcji. W języku SQL zgodnym ze standardem ANSI, początek transakcji wyznacza pierwsze zlecenie SQL występujące po ostatnim SQL COMMIT LUB SQL ROLLBACK.

Program aplikacyjny może kontrolować przebieg transakcji zleceniami SQL nawet wtedy, gdy zarządca zasobów obsługuje protokół XA, gdy dyktują to przyczyny wydajnościowe systemu.

Rosnąca liczba systemów zarządzania bazami danych (DBMS) obsługujących protokół XA jest tylko jednym z czynników decydujących o tak szybkim rozwoju systemów transakcyjnych wykorzystujących TUXEDO System/T. Nie mniej ważna jest bogata lista oprogramowania narzędziowego przeznaczonego do tworzenia programów transakcyjnych. Do niedawna lista ta ograniczała się do bibliotek procesorów języków programowania trzeciej generacji C i COBOL. Obecnie dostępne są już języki programowania czwartej generacji (4GL), specjalnie przystosowane do współpracy ze środowiskiem TUXEDO. Należą do nich:

- INFORMIX-TP/ToolKit,
- INGRES WINDOWS/4GL-TP,
- ACCELL/TP,
- ALLY 4GL,
- FOCUS 4GL.

Zakończenie

Przetwarzanie rozproszone OLTP stało się już powszechnie spotykaną rzeczywistością. Wiele przedsiębiorstw wykorzystuje standardy systemów otwartych do budowania infrastruktury informatycznej integrującej systemy komputerowe różnych producentów począwszy od systemów klasy mainframe, poprzez systemy serwery departamentowe aż do mikrokomputerów PC. Problemem z jakim przyjdzie uporać się w najbliższym czasie będzie dalsze niezawodne i bezpieczne rozproszenie danych oraz usług transakcyjnych w taki sposób aby było to niezauważalne dla użytkowników systemu.

Systemy otwarte w połączeniu z modelem DTP X/Open rozproszonego przetwarzania transakcyjnego OLTP, ułatwiają zbudowanie takiego środowiska systemowego. Opracowywanie systemów transakcyjnych zgodnie ze standardami modelu DTP X/Open , pozwala na wykorzystywanie i współdzielenie zasobów systemowych w obrębie całej sieci informatycznej przedsiębiorstwa, obejmującej niejednorodne platformy systemowe, odmienne systemy operacyjne i systemy zarządzania bazami danych.

Model DTP X/Open daje następujące korzyści przy tworzeniu systemów OLTP:

- uniezależnienie od typu i zasięgu sieci komputerowej,
- uniezależnienie od rodzaju wykorzystywanych urządzeń zewnętrznych,
- uniezależnienie od miejsca dostępności usług,
- uniezależnienie od typu bazy danych i środowiska operacyjnego,
- poprawę rzetelności i integralności danych,
- poprawę niezawodności i dostępności systemu,
- poprawę wydajności i czasu reakcji systemu,
- architekturę systemu zgodną z obowiązującymi standardami,
- ułatwienie integracji systemu z innymi środowiskami transakcyjnymi.

Systemy przetwarzania transakcyjnego OLTP były zawsze i są w dalszym ciągu jednym z najważniejszych obszarów technologii przetwarzania informacji dla firmy ICL. Obecnie wyraźnie obserwowane są tendencje do odejścia od scentralizowanych systemów OLTP, zbudowanych na bazie dużych systemów komputerowych klasy "mainframe" o zamkniętej architekturze systemowej, w kierunku systemów rozproszonych klient/serwer, wykorzystujących standardy systemów otwartych. Odpowiedzią firmy ICL na to wyzwanie ostatnich lat, jest produkt o nazwie TP Manager - system przetwarzania transakcyjnego OLTP dla środowiska systemu operacyjnego UNIX SVR4 (DRS/NX). TP Manager bazuje na syste-

mie TUXEDO ETP 4.2 z UNIX System Laboratory i jest zgodny z modelem DTP X/Open oraz standardem OSI-TP (ISO 10026).

Firma ICL, w ramach programu OPEN VME *(Virtual Machine Environment)* wprowadzającego systemy SERIES 39 do świata systemów otwartych, zamierza uzupełnić dostępny w tym środowisku system przetwarzania transakcyjnego TPMS *(Transaction Processing Management System)* o standardy modelu DTP X/Open (OSI-TP). Pozwoli to na implementację rozproszonego przetwarzania transakcyjnego w środowisku systemów operacyjnych UNIX i VME.

SPIS TREŚCI

Program zajęć 3

Założenia, projekt i realizacja sieci transmisji danych KOLPAK
(Wojciech Głazek) 7

Sieci komputerowe w polskim przemyśle lotniczym (Włodzimierz Adamski) 12

Elektroniczna postać dokumentacji konstrukcyjnej i technologicznej
(Włodzimierz Adamski) 21

Praktyczne doświadczenia opracowywania i wdrażania w przedsiębiorstwach
kompleksowych systemów informatycznych (Mariusz Klapper) 31

Szablony klas i listy elastyczne w pakiecie narzędziowym POLY do języka C
(Zbigniew A. Nowacki) 44

Metodologiczne aspekty modelowania procesów gospodarczych
(w ujęciu Dynamiki Systemowej) (Jerzy Skrzypek) 55

SYMULACJA SYSTEMÓW - Symulacja procesów dyskretnych (Marek Miłosz) . . . 73

Obiektowo-zorientowane analiza i projektowanie (Artur Kasprzyk) 86

Co daje obiektowe podejście do analizy i projektowania systemów informatycznych
(Zbigniew Benedykt) 108

Modelowanie obiektów dynamicznych (Stanisław Kędzierski) 115

Tworzenie obiektów poprzez dziedziczenie mono- i polimorficzne
(Marek Wierzbicki) 122

Podstawowe zagadnienia programowania zorientowanego obiektowo -
polimorfizm i dziedziczenie klas w języku C++ (Mirosław Wieczorek) 128

Podejście obiektowe do specyfikacji na przykładzie języka CCITT SDL'92
(Marek Średniawa) 135

Projekt i realizacja obiektowo zorientowanego podsystemu komunikacji z użytkownikiem (Roman Siminski) 156

LBMS Systems Engineer. Zintegrowane środowisko do tworzenia systemów informatycznych (Borys Stokalski, Andrzej Maciej Wierzba) 166

SuperMemo as a new tool increasing the productivity of a programmer. A case study - Programming in Object Windows (Piotr Woźniak) 171

System ekspertowy EKSSTER dla wspomagania projektowania i dokumentowania układów automatyki przemysłowej na bazie sterownika STERISS-2 (Joanna Kurzok-Derda) 178

Komputerowe systemy wspomagania decyzji a modele sytuacji decyzyjnych (Danuta Kajrunajtys) 184

InfoMAPY - inna metoda gromadzenia wiedzy (Wojciech M. Jaworski) 194

Komunikacyjne aspekty tworzenia systemów informacyjnych (Marian Kuraś) 210

CASAD - nowe podejście do modernizacji SI (Andrzej Zaliwski) 224

Wykorzystanie baz danych opartych na Btrievie i NetwareSQL w środowisku WINDOWS (Jan Baranowski) 246

Systemy przetwarzania transakcyjnego OLTP. Modele i standardy (Jacek Stochlak) 250