

- 10) opracowanie elementów molekularnych,
- 11) opracowanie bionicznych urządzeń łączności człowieka z komputerem,
- 12) protezy elektroniczne.

III. W latach dziewięćdziesiątych:

- 13) urządzenia domowe służące do automatycznego odtwarzania czasopism na odległość (metoda *faksymile*),
- 14) budowa automatycznych kierowców,
- 15) produkcja komputerów dla celów domowych,
- 16) komputery zdolne do samodzielnego rozwiązywania testów na inteligencję.

Z przeprowadzonej analizy wynika, że w rozwoju techniki komputerów jest jeszcze wiele do zrobienia, jednakże punkt ciężkości rozwoju przenosi się do sfery zastosowań.

5.3. Rozwój oprogramowania

Słowo oprogramowanie (*software*) ma kilka znaczeń. W najszerszym sensie zawiera wszystko, co nie jest sprzętem, a więc jest „miękkie”, jak: programy, języki, podręczniki, dokumentacja, a nawet sam proces pisania programów. W znaczeniu zawężonym odnosi się do instrukcji, które tkwią w pamięci komputera w czasie kiedy są egzekwowane. Często w najbardziej zawężonym sensie dotyczy tylko programów „maszynowych” dostarczanych wraz z komputerem, bez pakietów zastosowaniowych.

Przez oprogramowanie będziemy dalej rozumieć podprogram, program uogólniony w ten sposób, że funkcja informatyczna może być stosowana przez większą liczbę użytkowników. W ten sposób nadajemy oprogramowaniu aspekt wymiany między użytkownikami. Aspekt, który nadaje oprogramowaniu znaczenie społeczne.

W punkcie 4.4.2. podzieliliśmy oprogramowanie na następujące podsystemy: system operacyjny, translatory języków programowania, programy usługowe, programy użytkowe. Na rysunku 5.7. podano schemat oprogramowania, oparty na podanej koncepcji podziału. Każdy z wymienionych podsystemów ulega dalszemu podziałowi (proponujemy w tym zakresie por. w pkt. 4.4.2.). Równie pożyteczny jest podział oprogramowania na (por. rysunek 4.7.):

- 1) oprogramowanie niezależne od zastosowań:
 - a) oprogramowanie maszynowe (tzw. *hard software* lub oprogramowanie systemowe — według terminologii IBM); zalicza się do niego:

system operacyjny, programy usługowe i translatory języków programowania,

b) kierowanie zbiorami,

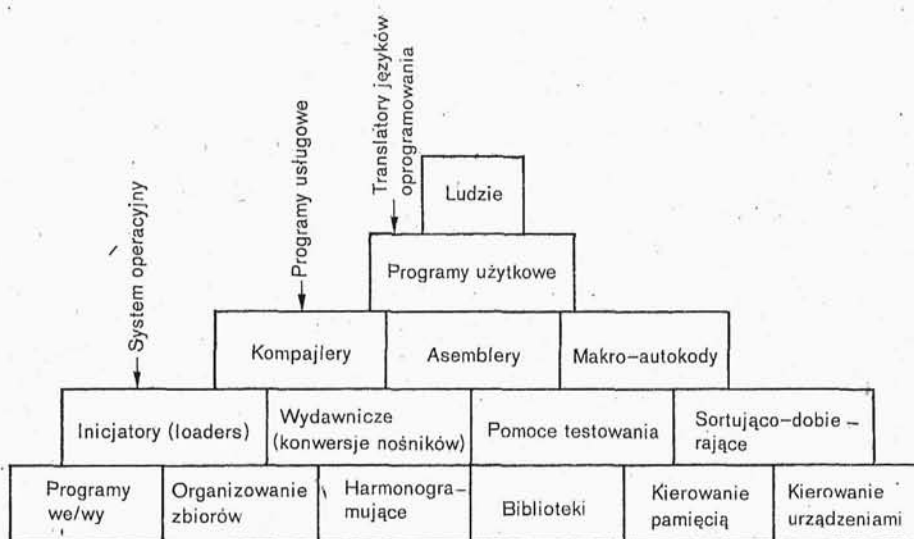
2) oprogramowanie zastosowaniowe:

a) systemów informatycznych, które dotyczą podsystemów (np. według zasobów lub procesów) lub całych systemów według obiektów (pakiety programów systemu zbiorczego, np. IMS),

b) według metod, np. PERT, SIMPLEX, GPSS itp.

Rysunek 5.7.

Schemat ideowy podziału oprogramowania według kryterium celu podsystemu



Na rysunku 5.8. podano schemat podziału oprogramowania oparte-
go na tej koncepcji.

W rozdziale tym zajmiemy się m.in. analizą czynników rozwojowych
w zakresie oprogramowania.

W rozwoju oprogramowania szczególną rolę odegrały kobiety.
W 1842 r. Lady Lowelace (Augusta Ada Byron, 1815—1853) ułożyła język
programowania dla maszyny analitycznej Ch. Babbage'a. W ten sposób
można córkę Byrona uznać za pierwszego w świecie programistę.

Matką współczesnego oprogramowania jest niewątpliwie inna kobie-
ta — G. Hopper²⁴, która rozpoczęła zawód programisty przy maszynie
MARK I, podczas II wojny światowej. Jest autorką pierwszego języka pro-
gramowania zadań gospodarczych — FLOW-MATIC dla UNIVAC I
i UNIVAC II oraz pierwszego sprzedawanego programu usługowego — sor-

²⁴ Przed przejściem na emeryturę otrzymała stopień admirała.

**Rysunek 5.8.**

Schemat podziału oprogramowania według kryterium stosunku do zastosowań (w polach zakreskowanych podano oprogramowanie zastosowaniowe)

owania. Język FLOW-MATIC stał się prekursorem dla późniejszego języka COBOL. Był pierwszym językiem, który został oparty na syntaksie języka angielskiego.

W dalszych rozważaniach zwrócimy szczególną uwagę na rozwój: języków programowania i oprogramowania gospodarki zbiorami²⁵.

5.3.1.

Języki programowania

W latach 1952—1956 (pierwszej generacji komputerów) krystalizował się pogląd na zakres i ograniczenia języków programowania. Pojawiały się koncepcje: pseudokodów, automatycznego kodowania²⁶, programowania, kompajlera, interpretera. Z dzisiejszego punktu widzenia, pseudokod oznaczał kod maszyny zapisany za pomocą symboli; automatyczne kodowanie oznaczało programowanie w języku symbolicznym, a automatyczne programowanie oznaczało coś więcej niż automatyczne kodowanie. Z języków,

²⁵ Zagadnienie pakietów programów zastosowaniowych zostało omówione w pkt. 4.4.2. Analiza w tym punkcie została przeprowadzona pod kątem okoliczności decydujących o przesłankach strategii rozwojowej wybranych elementów oprogramowania. Czytelnika szukającego koncepcji klasyfikacyjnych w zakresie języków oprogramowania odsyła się do licznych na ten temat pozycji literatury.

²⁶ Polski SAKO po latach wziął stąd nazwę: System Automatycznego Kodowania.

które powstały i przetrwały do dziś są: FORTRAN²⁷ (do obliczeń numerycznych) i APT (do sterowania numerycznego).

Lata 1958—1959 (początki II generacji komputerów) można uznać za najważniejsze w dotychczasowej historii rozwoju języków oprogramowania. W okresie tym miały miejsce następujące wydarzenia:

1. Opracowano i ogłoszono raport IAL (*International Algebraic Language*), który znany jest także jako ALGOL 58.

2. Opracowano trzy języki oparte na języku ALGOL 58, tj. NELIAC, MAD i CLIP (dał początek językowi JOVIAL), użytkowane do 1971 r., potem stosowane głównie w zastosowaniach militarnych.

3. J. Backus przedstawił w 1959 r. podczas spotkania UNESCO w Paryżu metody formalizacji języków na przykładzie języka ALGOL.

4. Utworzenie grupy CODASYL w maju 1959 r., później przemianowanej na grupę COBOL, która opracowała do końca roku specyfikację języka COBOL, ogłoszoną w 1960 r.

5. Opracowano języki: IPL-V, AIMACO, FACT (Honeywell) i COMMIT (opis ukazał się w 1957 r.). Rozpoczęto prace nad językiem JOVIAL (1959), DYANA (1958), DYNAMO (1959) AED (1959) i innymi.

W okresie lat 1960—1970 (wejście III generacji komputerów) prace nad językami programowania weszły w okres dojrzałości. Walka o stosowanie języków problemowych została wygrana. Kodowanie w językach symbolicznych było wyjątkiem. Chociaż wielką popularność zyskał „półmaszynowy” assembler PL/360, dzięki popularności maszyn IBM/360. Najpopularniejszymi językami były: FORTRAN, ALGOL, COBOL, PL/I, z tym, że dwa ostatnie i FORTRAN były najbardziej wykorzystywane w USA. Został sformułowany ALGOL 68 i opracowany w 1970 r. Upowszechnienie zdalnego liczenia konwersacyjnego należy przypisać językom JOSS i BASIC oraz APL/360 (K. Iversona). Do manipulacji formułami opracowano języki FORMAC i Formula ALGOL, z których pierwszy był częściej stosowany. Język SNOBOL został opracowany do przetwarzania ciągów („stringów”) i dobierania wzorów. Prawdopodobnie najważniejszymi wydarzeniami w tym okresie było opracowanie normy dla języków FORTRAN, COBOL²⁸ i zapoczątkowanie prac nad normą PL/I. Historyczny rozwój języków programowania został podany na rysunku 5.9²⁹ (na końcu książki).

Listę ważniejszych języków podano w tablicy 5.5. (od 1960 r. i od 1967 r.) i w tablicy 5.6. (od 1971 r.). Tylko dziesięć języków powtarza się na 3 listach, są to:

²⁷ Opracowany przez J. Backusa i zespół z IBM nakładem 25 osobolat.

²⁸ Przed spotkaniem w Paryżu w 1960 r. zginął w wypadku jeden z współtwórców tego języka W. Turański.

²⁹ Rysunek pochodzi z pracy J. Sammet, *Programming Languages: History and Future*, „Communication of ACM” 1972, vol. 14, nr 7, July.

Tablica 5.5.

Języki od 1960 r.

ABC
 ACT
 AIMACO
 ALGO
 ALGOL^a
 ALTAC
 ALTRAN
 API
 APS
 APS III
 APT^a
 APX III
 ARGUS
 BACAIC
 BALGOL
 BIOR
 CAGE
 CL-I
 CLIP
 COBOL
 COLASL
 COMIT
 COMMERCIAL
 TRANSLATOR
 FACT
 FLEXIMATIC
 FLIP
 FLOWMATIC
 FORAST
 FORTRAN
 FORTRAN II
 FORTRAN III
 FORTRANSIT
 GAT
 GOOFUS
 GP
 IPL
 IPL V^a
 IT
 IVY
 LISP^a
 MAD^a
 MADCAP^a
 MAGIC
 MATHMATIC
 MCP
 MISHAP

MYSTIC
 NELLAC^a
 NUIT
 9 PAC
 PACT I
 PRINT
 SALE
 SAP
 SHADOW III
 SLANG
 SMAC
 SOAP
 SOS
 STRAP I
 STRAP II
 SURGE
 TAC
 THETEADED LISTS
 TRIE
 UNCOL
 UNICODE
 USE
 VIPP
 XI
 X2
 XTRAN

Języki od 1967 r.

A-2 and A-3
 ADAM
 AED
 AESOP
 AIMACO
 ALGOL^a
 ALGY
 ALTRAN
 AMBIT
 AMTRAN
 Animated Movie
 APL
 APIX 360
 APT^a
 BACAIC
 BASEBALL
 BASIC
 BUGSYS

C-10
 CLIP
 CLP
 COBOL^a
 COGENT
 COGO
 COLASL
 COLINGO
 COMIT^a
 COMMERCIAL
 TRANSLATOR
 COMPUTER
 COMPILER
 COMPUTER
 DESIGN
 CORAL
 CORC
 CPS
 CULLER-FRIED
 DAS
 DATA-TEXT
 DEACON
 DIALOG
 DIAMAG
 DIMATE
 DOCUS
 DSL/90
 DYANA
 DYNAMO
 DYSAC
 ENGLISH
 Extended ALGOL
 FACT
 FLAP
 FLOW-MATIC
 FORMAC
 Formula ALGOL
 FORTRAN^a
 FORTRANSIT
 FSL
 GAT
 GECOM
 GPL
 GPSS
 GRAF
 Graphic
 ICES

c.d. tabl. 5.5.

Języki od 1967 r.

IDS	OMNITAB
Information	OPS
Algebra	PAT
IPL-V ^a	PENCIL
IT	PL/I
JOSS	PRINT
JOVIAL	PROPOSAL WRITING
KLERER-MAY	PROTOSYNTHEX
L6	473L QUERY
LANING and ZIERLER	QUIKTRAN
LDT	SFD-ALGOL
LINCOLN RECKONER	SHORT CODE
LISP 1.5 ^a	SIMSCRIPT
LISP 2	SIMULA
LOLITA	SIMUL. DIG. SYST.
LOTIS	SNOBOL
MAD ^a	SOL
MADCAP ^a	SPEEDCODING
MAGIC PAPER	SPRINT
MAP	STRESS
MATHLAB	STROBES
MATH-MATIC	SYMBOLIC MATH. LAB.
MATRIX COMPILER	TMG
META 5	TRAC
MILITRAN	TRANDIR
MIRFAC	TREET
NELIAC ^a	UCNOL
OCAL	UNICODE

^aDotyczy „dziesiątki” języków pojawiających się na wszystkich listach. (dotyczy tabl. 5.5. i tabl. 5.6)

Źródło: języki od 1960 r. *Tower of Babel*, „Cover of Communications of the ACM” 1961, vol. 4, No 1, January; języki od 1967 r. J. E. Sammet, *End Paper in Programming Languages: History and Fundamentals*, Prentice Hall, Inc. 1969.

a) języki do obliczeń numerycznych — FORTRAN, ALGOL, MAD, MADCAP, NELLIAC,

b) języki do przetwarzania danych gospodarczych — COBOL,

c) języki do przetwarzania list i ciągów (stringów) — IPL, LISP, COMIT,

d) języki specjalizowane — APT.

Najwcześniejsze języki programowania powstały dla obliczeń numerycznych. Najbardziej zunifikowany i sformalizowany zapis matematyczny ułatwiał zadanie. W 1952 r. w owym czasie lider w budowie komputerów — firma UNIVAC opracowała język SHORT CODE. Ideę języka podał jeszcze w 1949 r. J. Mauchly, który W. Schmitt opracował początkowo dla komputera BINIAC. W rok później w 1953 r. firma IBM opracowała

Tablica 5.6.

Języki w 1971 r.

ACTIVE LANGUAGE I	CULP	L ⁶
AED	CYPHER TEXT	LEAF
AESP	DARE	LEAP
AIDS	DATA BASIC	LINCOLN RECKONER
ALADIN	DATA STRUCTURES	LISP 1,5 ^a
ALGOL 60	LANGUAGE	LISP A
ALGOL 68	DATA-TEXT	LOGIC DESIGN
ALTRAN	DCDL	LANGUAGE
AMBIT	DGL	LOGO
Animator	DIALOG	LPL
APAREL ^f	DIMATE	LRLTRAN
APDL	DML	LSYD
APIX 360	DSL	MAC-360
APT ^a	DYNAMO II	MACSYMA
ARIEL	ECAP II	MAD ^a
ATLAS	ELP	MADCAP ^a
ATOLL	EOL-3	MARSYAS
B-LINE	ETC	MATHLAB 68
BALM	EULER	McG 360
BASIC	Extended ALGOL	MENTOR
BCPL	FLAP	META 5
BLISS	FOIL	MOBSSL-UAF
BUGSYS	FORMAC	NAPSS
CAMAL	FORMAL	NELIAC ^a
CATO	FORTRAN ^a	NPPL
CCL	FSL	NUCLEOL
CESSL	GAN	OLDAS
CHAMP	GASP	OMNITAB II
COBOL ^a	GEA	OSCAR
COGENT	GEDANKEN	PAL
COGO	GENERAL PURPOSE	PDEL
COIF	GRAPHIC LANGUAGE	PIRL
COMIT II ^a	GPSS	PL/I
COMPUTER ANIMATION	GRAF	PL/I-FORMAC
LANGUAGE	GRAIL	PLACE
COMPUTER DESIGN	GRAPHIC LANGUAGE	PLANIT
LANGUAGE	GRIND	PLANNER
COMSL	HINT	PPL
COMTRAN	IAM	PREP
CORAL	ICES	PROTEUS
CSS/II	IDS	REDUCE
COURSEWRITER	IITRAN	REF-ARF
COURSEWRITER III	IMP	REL English
CPS	IPL-V ^a	RTL
CSMP	JOSS	RUSH
CSSL	JOVIAL	SALEM
CTL	Klerer-May	SCRATCHPAD/I

cd. tabl. 5.6.

SCROLL	STIL	TRAC[LANGUAGE
SIMSCRIPT 1.5	STRESS	TRANQUIL
SIMSCRIPT II	STROBES	TRANS
SIMULA	STROUDL	TREET
SLANG	SYMBAL	TROOL
SNAP	TCL	VULCAN
SNOBOLA	TERMAC	WRITEACOURSE
SPEAKEASY	TMG	XPL
SPRINT	TPS	

Źródło: Wykaz z *Roster of Programming Languages*, J.E. Sammet, Computers and Automation, 1971, vol. 20, no 63.

SPEED CODING dla komputera IBM 701. Oba języki były symbolicznymi autokodami. Faktycznie za pierwszą próbę zapisu wyrażeń matematycznych należy uznać pracę Szwajcara H. Rutishausera (Federalny Instytut Technologiczny w Zurychu), który w 1952 r. zaproponował naturalną formę zapisu i tłumaczenia typu kompilacji. W 1953 r. podobną próbę podjęli Laning i Zierler dla WIRLWINDu w MIT, chociaż ich język był interpretowany, a nie kompilowany.

Specjalizacja komputera UNIVAC w przetwarzaniu danych przyhamowała jego wkład do rozwoju języków do obliczeń numerycznych. Autokody A2 i A3 opracowane pod kierunkiem G. Hopper nie zyskały poparcia ze strony polityki handlowej firmy. W tej sytuacji firma IBM znalazła lukę dla numerycznych zastosowań swoich pierwszych rozwiniętych komputerów IBM 700.

W 1954 r. opracowano „IBM Mathematical FORMula TRANslating System FORTRAN” w grupie 13 specjalistów pod kierunkiem J. W. Backusa. Ponieważ koszt programowania był wysoki, zatem rosło zapotrzebowanie dla szybkich i pewnych kompilacji, nawet kosztem rozwiązań optymalnych w programie. Wiele mówiono na temat, że dobry programista może ułożyć lepszy program wynikowy niż FORTRAN. Należało do dobrego zwyczaju pisanie programów w języku maszyny, szczególnie przez najlepszych programistów.

Osiągnięcie firmy UNIVAC w zbudowaniu języka „English-language-like” — pod nazwą FLOWMATIC zmusiło IBM do nadania rozgłosu wokół języka FORTRAN i przystosowanie go w 1962 r. do przetwarzania danych gospodarczych (FORTRAN IV). Nie byłoby w stylu IBM wykorzystanie języka FLOWMATIC (co należy uznać raczej za niepotrzebny izolacjonizm i wątpliwą oryginalność IBM) na swoich maszynach. Natomiast odwrotnie, firma UNIVAC jako pierwsza wykonała w 1961 r. translator języka FORTRAN dla swojego komputera SOLID STATE 80. W 1963 r.

wszyscy najpoważniejsi producenci sprzętu opracowali translatory tego języka dla swoich maszyn (znanych jest 43 implementacji)³⁰.

Język FORTRAN przerwał hermetyczność ośrodków obliczeniowych, zmieniając je z *close shop* na *open shop*. Problemiści jak: inżynierowie, ekonomiści, fizycy, chemicy itp. sami zaczęli układać programy i sami je uruchamiali. FORTRAN był tym dla komputerów, czym automatyczna skrzynia biegów dla samochodów. Mniej zaawansowani użytkownicy zaczęli wykorzystywać z powodzeniem technikę obliczeniową. W 1962 r. grupa użytkowników SHARE rozpoczęła prace nad standaryzacją języka FORTRAN, doprowadzając do opracowania w 1966 r. normy w ramach Stowarzyszenia Amerykańskich Norm (obecnie USASI). Sukces języka FORTRAN wynika z popularności i wymienności między maszynami oraz łatwości programowania, co prowadzi do paradoksu, bowiem każdy użytkownik chciałby tym językiem rozwiązać swój problem, podczas gdy nie zawsze język ten może się do tego nadawać. Stało się ewidentne, że istnieją określone granice dalszego doskonalenia tego samego języka. Pojawia się wówczas konieczność opracowania nowego języka.

W 1957 r. środowisko użytkowników doszło do wniosku, że samo powinno opracować język algorytmiczny, wyrażając inicjatywę firmie IBM. W USA sformułowanie tego wniosku miało miejsce w dniach 9 i 10 maja 1957 r. w Los Angeles na konferencji USE, SHARE i ACM. W Europie na jesieni tego roku ten sam wniosek został sformułowany w Towarzystwie GAMM (*Gesellschaft für Angewandte Mathematik*)³¹. W maju 1958 r. doszło do spotkania w Zurychu Amerykanów (J. W. Backusa, C. Katza i A. J. Perlisa z ACM) i Europejczyków (F. L. Bauera, H. Bottenbrucha, H. Rutishausera i K. Samelсона z GAMM). Ustalono, że nowy język powinien spełniać następujące warunki:

- 1) odpowiadać przyjętym normom zapisu matematycznego,
- 2) łatwo opisywać w publikacjach proces obliczeniowy,
- 3) mechanicznie winien być tłumaczony na program wynikowy.

Dokument, który go opisywał nazwano najpierw IAL (*International Algebraic Language*), a potem ALGOL 58. W USA pierwszą implementację tego języka wykonała firma Burroughs dla komputera B220, stąd powstała nazwa BALGOL (B-ALGOL). W Europie pierwszą implementację wykonali Duńczycy z Regnecentralen, na maszynie GIER (stąd maszyna ta cieszyła się dość znaczną popularnością). W 1959 r. podczas paryskiej konferencji pod auspicjami UNESCO, historyczny referat wygłosił J. Bac-

³⁰ W tym okresie w Polsce opracowano język SAKO oparty na rozwiązaniach języka FORTRAN, co odizolowało polskich informatyków od dorobku oprogramowania światowego i było tylko wątpliwą samodzielnością w pracach, nad automatyzacją programowania, kierowanych przez L. Łukaszevicza.

³¹ Czyżby stąd powstała polska nazwa GAM Grupy Aparatów Matematycznych, będącej załącznikiem obecnego Instytutu Maszyn Matematycznych?

kus, w którym przedstawił formalną metodę zapisu syntaktyki języka na przykładzie języka ALGOL, od jego nazwiska nazwaną Backus Normal Form (BNF). Dała ona pomost do lingwistyki matematycznej i gramatyk Chomskyego. Od 1967 r. miesięcznik amerykański ACM Communications przestał publikować opisy programów w języku FORTRAN, zastępując go językiem ALGOL. W 1960 r. powstała wersja ALGOL 60, a w 1968 r. ALGOL 68 (opis van Wijngaardena).

ALGOL został zaprojektowany do obliczeń numerycznych i procedur logicznych (np. sortowania, kompilowania), stąd pokrywał duży zakres potrzeb obliczeniowych. Jest językiem przystosowanym do dydaktyki — uczenia podstaw procesu obliczeniowego. Wadą tego języka jest brak możliwości przetwarzania danych alfanumerycznych, złożonych struktur danych oraz redagowania wejściowo-wyjściowego (ulepszono w języku ALGOL 68). Stąd stał się językiem uniwersyteckim w Europie. W USA wobec wymienionych braków, w sytuacji większego natężenia obliczeń, język FORTRAN nie dał się zastąpić przez ALGOL. Wśród dialektów języka ALGOL można wymienić m.in. MAD, NELIAC i JOVIAL (Jules Schwartz' Own Version of the International Algebraic Language). Język JOVIAL rozszerzył zakres zastosowań języka ALGOL, w kierunku zastosowań uniwersalnych inicjując kierunek rozwojowy, który najlepiej charakteryzuje PL/I.

Rozwój języka ALGOL w USA nastąpił w bardzo pośredniej formie. Ponieważ obliczenia numeryczne zaczęto rozwijać z końcówek, zatem stało się konieczne opracowanie języków raczej konwersacyjnych niż partioowych. Warto dodać, że również próba rozwoju języka FORTRAN w tym kierunku nie powiodła się. Język QUICTRAN (1961) lansowany przez IBM nie przyjął się. Natomiast wielkim językiem konwersacyjnym stał się BASIC (*Beginner's ALL Purpose Symbolic Instruction Code*) opracowany w 1965 r. w Dorthmund College przez J. Kemery i T. Kutza, na maszynie GE 225. Autorzy postawili sobie za cel uproszczenie języka ALGOL w taki sposób, aby BASIC stał się tylko wprowadzeniem do bardziej złożonego programowania. Ponieważ firma GE dysponowała olbrzymim systemem abonenckim MARC (ponad 100 tys. końcówek) nic dziwnego, że przejęty przez nią BASIC zdominował obliczenia konwersacyjne. Odpowiedzią firmy IBM był język APL (*A Programming Language*) opracowany jeszcze w 1962 r. przez K. E. Iversona w Uniwersytecie Harvardzkim. Firma IBM przejęła go i popularyzowała go od 1967 r. na maszynach IBM 360. Język APL powstał jako protest przeciw językowi ALGOL, opracowanego przez „Komisję”. Dzięki jednemu autorowi język zyskał zwartość i wprost „szaloną” prostotę. Podobno kto raz zaczął używać APL, nigdy nie przechodzi na inny język.

W zakresie języków do przetwarzania danych gospodarczych sytu-

acja została szybciej wyjaśniona. Po pierwsze, liczba liczących się języków była niewielka: FLOWMATIC-UNIVAC, COMTRAN (*Commercial Translator*) — IBMu, FACT (*Fully Automatic Compiling Technique*) — Honeywella.

W przeciwieństwie do języka ALGOL, rozwój prac nad językiem COBOL (*Common Oriented Business Language*), został zainicjowany przez producentów sprzętu i Departament Obrony. W 1959 r. powstał CODASYL (*Conference on Data System Languages*), a w jego ramach trzy komisje: Krótkookresowa, Średniookresowa i Długookresowa. Interesujące będzie wspomnieć, że CODASYL nigdy nie był Komitetem w przyjętym tego słowa znaczeniu. Tylko raz się zebrał (maj 1959), by potem działać korespondencyjnie i poprzez Wykonawczą Komisję (*Executive Committee*). Pierwszą specyfikację języka COBOL opracowali w 1959 r.: H. Bromberg i N. Discount (RCA), V. Reeves i J. Sammet (Sylvania) i W. Selden i G. Tierney (IBM)³². Znamienny jest fakt, że po raz pierwszy, a zarazem po raz ostatni przedstawiciele IBM uczestniczyli we wspólnym projekcie z przedstawicielami firm konkurencyjnych. W 1959 r. wielu producentów rozwijało dopiero prace nad językami komercyjnymi, stąd wpływ języka COBOL na dalszy kierunek tych prac miał zasadnicze, pozytywne znaczenie. Jest to przykład podjęcia w porę prac unifikacyjnych. Język FLOWMATIC był główną podstawą koncepcji oparcia języka COBOL na słowach angielskich (np. GROSS PAY or COMPUTE). Z języka COMTRAN przyjęto IF ... THEN oraz zapis danych w postaci PICTURE. Prace nad językiem FACT były prowadzone w tajemnicy, stąd gdy w 1960 r. zostały ogłoszone wyniki, okazało się, że przewyższa on pierwszą specyfikację języka COBOL w zakresie giętkości opisu danych i lepszej listy rozkazowej. W dalszych pracach oba języki wykorzystywały nawzajem swoje zalety. W następnych latach powstały dalsze wersje: COBOL 61, COBOL 65, COBOL 68 i COBOL 70.

W Polsce, jak zwykle, nowości dotarły szybko. Na przełomie 1961/1962 r. L. Łukaszewicz podjął prace w ŻAMie nad polskim językiem COBOL. Była to działalność chybiona. Trend rozwojowy wykazał, że język angielski stał się niejako kodem (Esperanto) języków programowania³³, i że chodzi tu o tworzenie warunków do wspólnego porozumienia, a nie izolowania się od dorobku światowego. Po 10 latach A. Borowiec wykonał implementację języka COBOL na komputer ZAM 41, tj. wtedy, gdy los ZAMów został definitywnie przesądzony. Brak rozeznania w ogólnych

³² Potem zrezygnowano z publikowania nazwisk autorów na korzyść firm uczestniczących w przedsięwzięciu. Pełną listę wszystkich późniejszych współtwórców COBOLu można znaleźć w Dodatku A do Normy COBOLu (USASI).

³³ Polski autokod MOST dla komputera ODRA 1204 został wykonany już w „kodzie” angielskim.

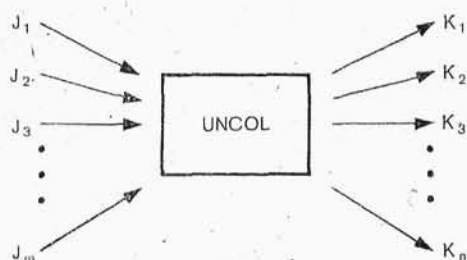
trendach światowych spowodował opóźnienie przygotowania polskich programistów do korzystania z języka COBOL. Wytworzyła się nawet maniera nieposługiwania się językiem COBOL, która spowodowała większą popularność języka PLAN na komputerach ODRA 1300. Skutki tego są wymierne i w czasie i w złotych (oczywiście strat).

Oprócz wymienionych rodzajów języków trzeba wymienić inny, wyodrębniający się kierunek języków do programowania manipulacji symbolami, zwany również przetwarzaniem list. Pierwszy pomysł w tym zakresie podali w 1956 r. A. Newell i H. Simon, kiedy scharakteryzowali potrzeby programowe w zakresie dowodzenia twierdzeń. Koncepcja listowa stała się jedną z podstaw teorii i praktyki języków programowania i była stosowana m.in. do aktualizacji dużych zbiorów informacji, dla których trudno jest przewidzieć z góry wymaganą pojemność pamięci, np. tablic, kartotek, wyrobów i półfabrykatów. Jeżeli owe zbiory są utrzymywane sekwencyjnie, wówczas włączanie i wyłączanie elementów wymaga stałej reorganizacji zbioru. W przetwarzaniu listowym każdy element ma adres (*pointer*) następnego elementu, logicznie z nim powiązanego. Element ten może znajdować się w innym miejscu pamięci, niekoniecznie obok właściwego sąsiada. Do klasy problemów, gdzie wykorzystuje się listowe przetwarzanie, zwykło się zaliczać pisanie translatorów, manipulowanie wyrażeniami algebraicznymi, przetwarzanie obszarów, lingwistykę matematyczną, problemy z zakresu sztucznej inteligencji, czyli przetwarzanie tekstów. Mało natomiast wiadano, że przetwarzanie listowe zrewolucjonizowało technologię przetwarzania danych gospodarczych. Nie bez powodu koncepcja listowa powstała w 1956 r., tj. w okresie, w którym pojawiły się pierwsze pamięci dyskowe RAMAC. Właśnie pamięć dyskowa narzuciła koncepcję przetwarzania listowego, którą przejęła po kilkunastu latach grupa CODASYLu pracująca nad technologią baz danych. W latach 1960–1970, firma IBM wprowadziła system rozwinięć montażowych wyrobów oparty na koncepcji listowej (MASTER FILE i STRUCTURE FILE) początkowo znany pod nazwą BILL of MATERIAL, a potem jako podsystem w PICS (*Production Information and Control System*). Wśród języków do tego celu zbudowanych trzeba wymienić IPL, LIPS, COMIT i SNOBOL. Szczególnie innowacyjną rolę spełnił COMIT, który wprowadził efektywne środki wyszukiwania szczególnych wzorów ciągów.

Specjalizacja języków oprogramowania nasunęła kuszący problem stworzenia uniwersalnych, ogólnych języków programowania „wszystkiego”. Dwa języki reprezentują ten trend: JOVIAL (wojsko) i PL/I (IBM), (można tu wymienić także Formuła ALGOL).

Ograniczenia języka FORTRAN w programowaniu operacji na znakach (i inne) spowodowały wspólne działania grupy SHARE i IBM w kie-

runku zbudowania nowego języka. Chodziło o wykorzystanie najlepszych cech języków FORTRAN, COBOL i JOVIAL. W efekcie powstał w 1964 r. PL/I (dla serii 360), gdzie I oznacza „one”, czyli do wszystkiego³⁴. Język PL/I mógłby wyprzeć wymienione języki, gdyby nie okazał się zbyt skomplikowany i wymagający długiego okresu nauki. Wprawdzie zawiera lepsze możliwości programowania operacji na zbiorach komunikacyjnych, ale nie równoważy to niezbyt przychylnej opinii użytkowników.



Rysunek 5.10.

Schemat roli języka UNCOL, który redukuje liczbę translatorów z $K \times J$ na $K + J$ (gdzie K oznacza liczbę różnych komputerów, a J liczbę różnych języków programowania)

W sferze rozważań teoretycznych pozostaje koncepcja języka UNCOL (UNiversal Computer Oriented Language). Można wyobrazić sobie K -komputerów i J -języków, które trzeba wyposażyć w $K \times J$ translatorów. Język UNCOL nie byłby językiem programowania, a pośrednikiem między językami programowania a komputerami (por. rys. 5.10.). Wówczas trzeba byłoby opracować nie $K \times J$ translatorów, a $K + J$ „połowicznych translatorów”. Z chwilą pojawienia się nowego języka byłoby konieczne opracowanie translatora tego języka tylko na UNCOL.

Natomiast każda maszyna miałaby już gotowy translator z języka UNCOL na swój język maszyny. I odwrotnie, każdy nowo zbudowany komputer musiałby mieć swój translator na UNCOL, przez co akceptowałby dotychczas funkcjonujące języki, posiadające „połowiczne translatory” na UNCOL. Koncepcja UNCOL, chociaż pociągająca swoją prostotą, nie przyjęła się. Mocni producenci nie są w niej zainteresowani, bowiem zdolność do produkowania translatorów jest częścią kampanii reklamowych i wchodzi w sferę konkurencji. Wysuwa się także argument typu, że w ten sposób hamuje się postęp techniczny. Budowę i translatora, i sprzętu dostosowuje się do rozwiązań oprogramowaniowych i odwrotnie, podczas gdy UNCOL wprowadza zbyt daleko posuniętą stabilizację.

³⁴ Nazwa NPL (New Programming Language) została odrzucona jako zbieżna z brytyjskim National Laboratory of Physics, także MPPL (Multi-Purpose Programming Language) nie została przyjęta.

*

*

*

Na tle przeanalizowanego rozwoju języków programowania, w szczególności z punktu widzenia spraw motywacyjnych można sformułować zarys modelu rozwoju języków programowania.

Warunki niezbędne do upowszechnienia języka można usystematyzować następująco: -

1. Wymagana jest rzeczywiście nowa koncepcja języka lub wyłania się nowa dziedzina zastosowań wymagająca własnego języka (dotyczy języków FORTRAN, APT, FLOWMATIC; IPL, GPSS, COGO).

2. W wyniku doświadczenia w posługiwaniu się określonym językiem staje się konieczna jego modyfikacja lub stworzenie nowego języka bardziej skutecznego (np. COBOL zastąpił FLOWMATIC a SNOBOL wymienił COMIT).

3. Możliwości paru języków są łączone w celu utworzenia jednego języka (por. z PL/I) lub staje się ewidentne, że zaprojektowanie nowego języka jest łatwiejsze od poprawiania dotychczasowego.

4. Znajduje się osoba lub grupa zainteresowana w zbudowaniu nowego języka i często nie zdaje sobie sprawy, że istnieją pewne języki mające zbliżone cechy. Na przykład podjęte przez L. Łukaszewicza prace nad SAKO (wobec języków FORTRAN i ALGOL) izolowały nas od ruchu światowego zarówno zawodowo, jak i naukowo. Mała firma duńska, która opracowała ALGOL dla komputerów GIER zyskała natomiast duży sukces handlowy.

5. Występują osobiste animozje w stosunku do stosowanego języka, powodujące nacisk na zbudowanie nowego języka. Na przykład można tu wymienić stosunek K. Iversona (od APL) do języka ALGOL.

Niektóre języki zyskują popularność i entuzjastyczne użytkowanie ze względu na ich zapładniający wpływ, jak np. ALGOL i APL podczas gdy COBOL i PL/I pomimo szerokiego stosowania nie mają już podobnego przyjęcia.

W okresie lat 1952—1977 powstało ponad 3 tysiące języków, z tego 70% stanowią języki problemowo-kompilowane, a tylko 14 języków zawiera nowe koncepcje teoretyczne i użytkowe. Do nich zaliczają się: APT (regulacja obrabiarek), IPL-V, COMIT, LISP (przetwarzanie list), FORTRAN, ALGOL 60, FORMAC (obliczenia), FLOWMATIC, COBOL (przetwarzanie danych), JOSS, APL (konwersacyjne), GPSS (symulacyjny), JOVIAL i PL/I (ogólne).

Na dalszy rozwój języków programowania wywierają wpływ koncepcje języków: PL/I, ALGOL 68, APL. Chociaż najwięcej pracy włożono w języki FORTRAN i COBOL, można zaryzykować twierdzenie, że AL-

GOL 68 spełnił podobną inspirującą rolę jak ALGOL 60 i z równym średnim powodzeniem w użytkowaniu.

Specjalizacja czy uniwersalizm to główny temat rozważań odnośnie dalszego rozwoju języków. Wydaje się, że różne style porozumiewania się ludzi, w tym z maszynami, przemawiają za różnymi podejściami do tej sprawy. Chyba jeden uniwersalny język tego nie rozwiąże. „Bezkomputerowe” porozumiewanie się między członkami tego samego środowiska (np. lekarzy, inżynierów, matematyków, muzyków, wojskowych) wskazuje na dużą rolę żargonu. Z tego względu specjalizowanie języków programowania ma również sens i uwarunkowania. Natomiast do rozwiązania pozostaje sprawa łatwości programowania. W tym względzie chodzi o to, by użytkownik po zdefiniowaniu, **co** ma wykonać na komputerze, mógł łatwo otrzymać do tego zadania opracowany *ad hoc* język. Chodzi o to, by zamiast szczegółowego programowania **jak** wykonać, użytkownik podawał, co chce otrzymać. Oczywiście swoje polecenie powinien podawać w swoim języku potocznym. Pewną zapowiedzią tego kierunku jest opracowany w 1975 r. przez IBM, system QUERY BY.EXAMPLE (pytanie na przykładzie). Trzeba również dodać, że w miarę rozwoju zastosowań komputerów, wiele problemów zostanie rozwiązanych (przeliczonych) oraz zaprogramowanych na stałe w formie parametryzowanych pakietów programowych. Zatem na niektórych odcinkach można będzie zaobserwować spadek prac w zakresie indywidualnego programowania. Nie jest wykluczone, że może temu towarzyszyć także spadek potrzeb na coraz to nowe języki programowania.

5.3.1.1.

Gospodarka zbiorami

Przez gospodarkę zbiorami rozumiemy: a) kierowanie zbiorami, b) łączność między danymi (*data communication*), c) zapytywanie o dane i odpowiadanie (*query and reporting*), d) katalogowanie danych (*data dictionary/directory facilities*).

Rozwój systemów gospodarowania zbiorami ma długą historię. Rozpatrzmy go w zakresie systemów kierowania zbiorami od najprostszych do najbardziej złożonych. Za kryterium rozróżniające rodzaj systemu przyjmijmy wzajemne zorganizowanie słów (dana elementarna, pole), np. numer pracownika, nazwisko, numer miejsca pracy. Logiczny zbiór tworzy zapis (rekord, segment, grupę). Kilka różnych zapisów mających związek logiczny tworzy pozycję (item), np. pozycja magazynowa, która zawiera zapisy o pojedynczym asortymencie materiałowym. Kilka pozycji mających związek logiczny tworzy z kolei kartotekę (*file*)³⁵.

³⁵ Por. A. Targowski, *Organizacja procesu przetwarzania danych*, wyd. cyt.

Wyróżniamy następujące systemy kierowania zbiorami:

- systemy kartotek prostych (*file system*),
- systemy kartotek odwróconych (*inverted file system*),
- systemy kierowania danymi (*data management*),
- systemy kierowania bazami danych (*data base management*),
- systemy danych rozdzielonych (*distributed system*).

System kartotekowy przeznaczony jest do gospodarowania zbiorami jednostki lub podsystemu zastosowaniowego. Organizacja pozycji jest sekwencyjna (na taśmach) lub indeksowo-sekwencyjna (na dyskach), (por. rys. 5.11 a.). Oprogramowanie zapewnia modyfikowanie i aktualizowanie zbiorów oraz zapytywanie o dane. Przykładem takiego systemu jest INFORMATICS MARK IV, który ma około 1000 użytkowników w świecie.

Potrzeba szybkiego przeszukiwania zbiorów doprowadziła do powstania systemu kartotek odwróconych (por. rys. 5.11 b.). W tym systemie każde słowo zapisu oznaczone jest łącznikiem (*link, pointer*), który oznacza adres, gdzie znajduje się wartość danej. Trzeba podkreślić, że przekazywanie łączników danych zorganizowane jest w ten sposób, że tego samego rodzaju łączniki są zgrupowane obok siebie. Wówczas przeszukiwanie odbywa się bardzo szybko³⁶. Natomiast wadą jest kłopotliwa aktualizacja. Ogólne wymagania dotyczące przechowywania danych gospodarczych polegają na potrzebie przechowywania całego logicznego zapisu w formie zwartej. Kierując się tym warunkiem wprowadzono systemy częściowo odwróconych kartotek lub systemy drugorzędnych indeksów (*secondary indexes*). Stosuje się w nich dotychczasowy układ zwartych logicznych zapisów oraz indeksy łączników dla najczęściej wyszukiwanych danych elementarnych (por. system SIEMENSA — PRISMA zastosowany w polskim ogólnokrajowym systemie MAGISTER). W miarę oceny częstotliwości wyszukiwania określonych danych łączniki można zmieniać. W systemie drugorzędnych indeksów³⁷, każda dana ma swój łącznik, podający adres wartości (zwany indeksem pierwszorzędnym — *primary index*). W indeksie drugorzędnym zawarte są tylko te łączniki, których dane podlegają wyszukiwaniu. Wówczas procedura przeszukiwania polega na kolejnym przeglądaniu indeksów: drugorzędnego i pierwszorzędnego. W nowoczesnych systemach korzystających z dysków zanika funkcja indeksu pierwszorzędnego. Z indeksu drugorzędnego od razu uzyskuje się adres do danej, umieszczonej w dostępnym miejscu pamięci (*hash-organization*), w ramach wyrywkowego dostępu. Warto dodać, że w tym systemie występują pierwsze oznaki eliminowania redundancji danych. Da-

³⁶ Schemat graficzny tej organizacji zbiorów, por. tamże, s. 84; M. Szaniawska, *Przetwarzanie zbiorów danych. Elementy metody i techniki przetwarzania*, Warszawa 1976.

³⁷ Drugorzędność indeksu oznacza również, że indeksowanie dotyczy danych w głębszych polach zapisu, które nie są głównymi kluczami wyszukiwania.

na powtarzającą się występuje tylko raz, dzięki zorganizowaniu zbioru zrelatywizowanych danych (konceptcja wykorzystana później przez grupę CODASYL) (por. rys. 5.11 c.).

Rysunek 5.11.

Proste i odwrócone formy organizacji zbiorów

a) Kartoteka prosta

D #	Nazwisko	Dostawa	Miasto
D 1	Nowak	30	Kraków
D 2	Kowalski	10	Warszawa
D 3	Pawlak	20	Poznań
D 4	Zieliński	10	Poznań
D 5	Kowalczyk	20	Warszawa

b) Kartoteka odwrócona—drugorzędne indeksowanie według miast

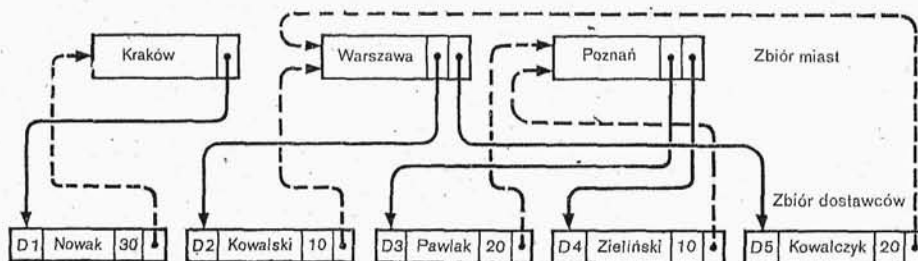
Zbiór miast

Miasto	Dostawca łącznik
Kraków	—
Warszawa	—
Poznań	—

Zbiór dostawców

D #	Nazwisko	Dostawa
D 1	Nowak	30
D 2	Kowalski	10
D 3	Pawlak	20
D 4	Zieliński	10
D 5	Kowalczyk	20

c) kartoteka odwrócona (—) z danymi zrelatywizowanymi (---)



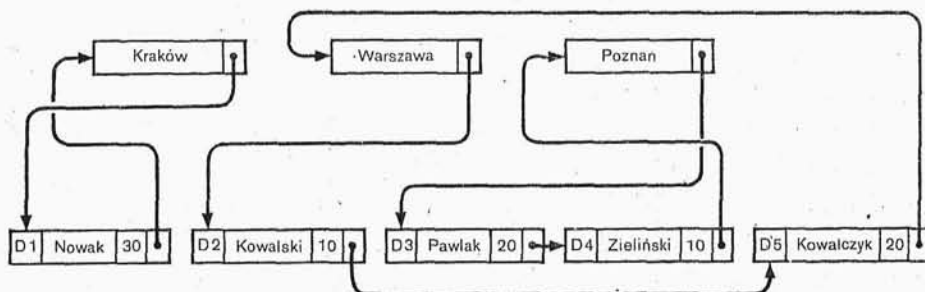
Wadą indeksowania drugorzędnego jest to, że w kartotece odwróconej każdy indeks musi zawierać wszystkie łączniki do indeksu pierwszorzędного. Ponieważ indeks pierwszorzędny jest zmienny, zatem liczba łączników jest też zmienna i wówczas program koordynacyjny musi przewidywać zmienną długość zapisów zawierających łączniki w kartotece odwróconej. Wymienioną wadę można usunąć dzięki wprowadzeniu koncepcji listowej. W naszym przykładzie każde miasto ma łącznik do pierwszego dostawcy z tego miasta, a następnie, na końcu zapisu danego dostawcy, jest łącznik do drugiego dostawcy, który pochodzi z tego samego miasta. Wówczas dla każdego miasta mamy łańcuch (*chain*) wszystkich dostawców z te-

go miasta (por. rys. 5.12 a.). Wadą listownego przetwarzania jest to, że by dostać się do n -tego dostawcy z tego samego miasta, trzeba przeszukać listę od 1, 2, ..., do $(n-1)$ dostawców.

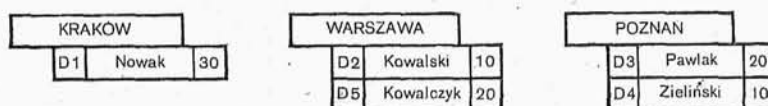
Rysunek 5.12.

Łańcuchowe i hierarchiczne formy organizowania zbiorów

a. Organizacja zbioru łańcuchowa



b. Organizacja zbioru hierarchiczna



Inną organizację reprezentują zbiory hierarchiczne (por. rys. 5.12 b.), w których logicznym zapisie są uwzględnione najczęstsze drogi wyszukiwania (np. poczynając od miast). Potem wszystkie dane są zgrupowane obok siebie, przez co spełniony jest wspomniany postulat, aby w gospodarczym przetwarzaniu dane logiczne powiązane były przechowywane obok siebie.

W systemach kartotek odwróconych występują trudności przy wyszukiwaniu danych, które znajdują się na różnych szczeblach hierarchii i w różnych odgałęzieniach sieciowej struktury danych. Występuje wówczas potrzeba tworzenia zapisów relacyjnych (*relationship record*) między poziomami i rozgałęzieniami. W nowszych systemach oprogramowania rozwiązano to dzięki specjalnej tablicy (w SYSTEM 2000) lub przez łączniki między kartotekami (*coupling* lub *linking indexes*) w ADABAŚ, aż do 80 łączników. Rozwiązanie jest definiowane dopiero po założeniu bazy danych.

Pewnym, najpopularniejszym systemem kartoteki odwróconej był TDMS (*Time-Shared Data Management System*) firmy oprogramowania SDC — *System Development Corporation*. Wśród innych można wymienić IFAM firmy CCA (*Computer Corporation of America*). Pakiet umożliwia wyszukiwanie: sekwencyjne, indeksowo-sekwencyjne, przez indeks

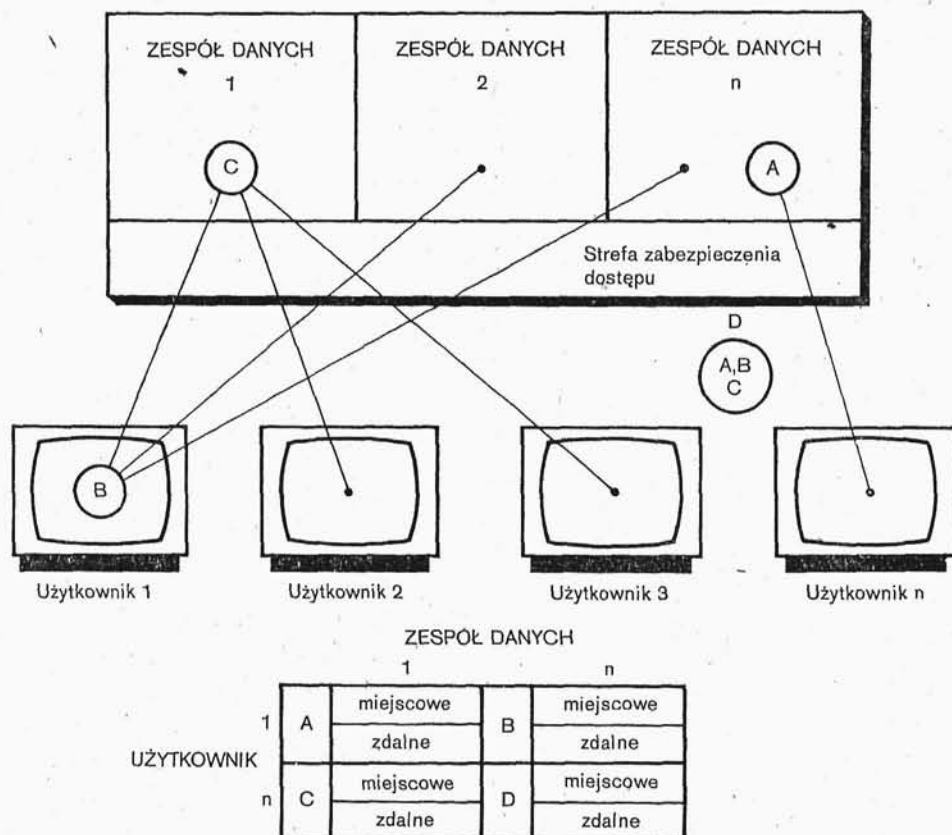
drugorzędny; każdy element może być włączony do tego indeksu. Zbiór może mieć zapisy zmiennej długości oraz zapisy tekstowe. W 1973 r. wymieniona firma wypuściła pakiet METABASE, który umożliwia przechowywanie w zapisie nazwy i wartości danej elementarnej (seks = kobieta) oraz nie rezerwuje pamięci dla nie występujących w zapisie danych. Aż do 4095 danych w zapisie może być indeksowanych. Zapytania mogą być formułowane w językach COBOL, FORTRAN, BAL i PL/I.

Systemy kierowania danymi (SKD) polegają na organizowaniu zbiorów wielokrotnych według określonych zastosowań w warunkach zdalnego ich aktualizowania i zapytywania w ramach określonego sprzętu informatycznego, bez możliwości uniknięcia powtarzalności (redundancji) danych w zbiorach.

Na rysunku 5.13. pokazano różne sposoby dostępu do zbiorów: lokalne i zdalne. Jeden użytkownik może mieć dostęp do wszystkich zespołów danych lub wszyscy użytkownicy mogą mieć dostęp do jednego zespołu

Rysunek 5.13.

Różne sposoby dostępu do zbiorów danych zorganizowanych w zespoły



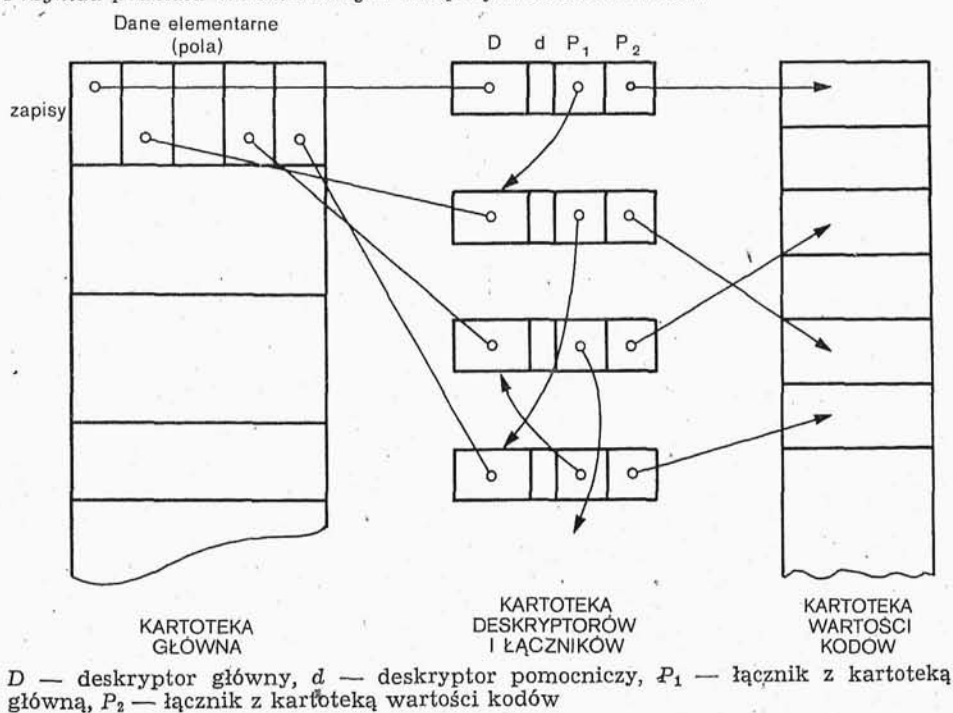
danych. Przez zespół danych rozumie się specjalizowane kartoteki. Na przykład w SKD — INFORMER 1500³⁸ występują trzy kartoteki: główna — przechowująca sekwencyjnie dane zasadnicze (por. rys. 5.14.), deskryptorów i łączników, według których wyszukiuje się i organizuje dostęp do kartotek w ramach zespołu danych, oraz wartości kodów, w której podane są tablice z tekstami symboli, np. stanowisk roboczych, komórek, zawodów (ponieważ w kartotece głównej podane są tylko kody).

W celu zorganizowania wyszukiwania i liczenia danych powstała koncepcja Systemu Kierowania Danymi, którą ilustruje rysunek 5.15. Jak widać z rysunku, SKD (DMS) korzysta ze standardowego oprogramowania w zakresie teleprzetwarzania, organizatora dostępu (np. do zbiorów na dyskach) i programów usługowych. W SKD podstawowym ogniwem jest Język Danych (DML — *Data Manipulation Language*), w którym użytkownik porozumiewa się podczas korzystania z SKD.

Zaletą SKD jest łatwość tworzenia zbiorów, ich aktualizowanie i przeszukiwanie w miarę rozwoju całego systemu informatycznego. Natomiast wadą jest powtarzalność danych w podzbiorach oraz niemożliwość przetwarzania danych zawartych w różnych podzbiorach.

Rysunek 5.14.

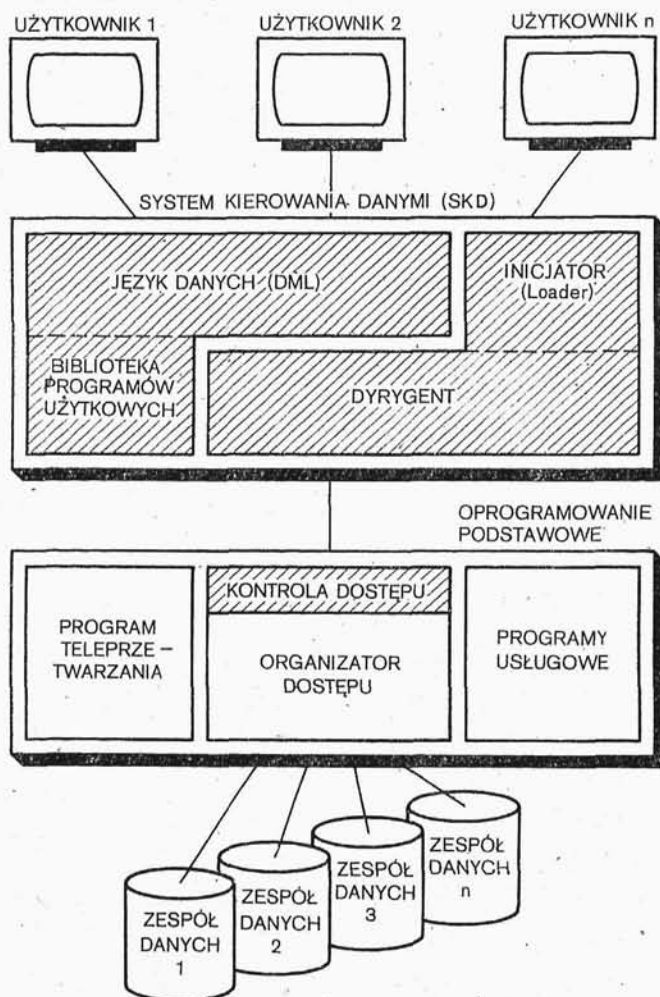
Przykład podziału zestawu danych na specjalizowane kartoteki



³⁸ Współprojektowany przez autora w USA w COGAR Corporation.

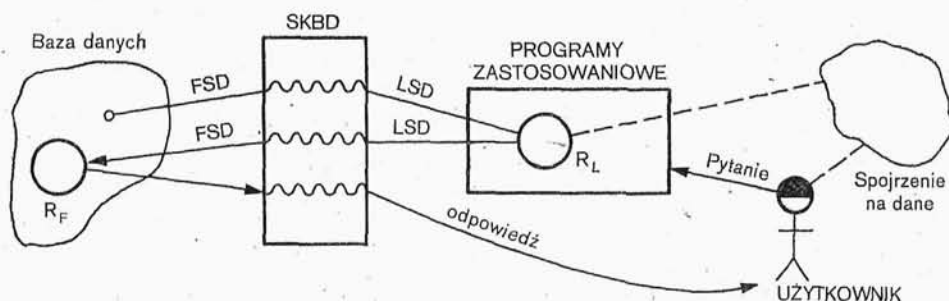
Rysunek 5.15.

Schemat miejsca Systemu Kierowania Danymi w konfiguracji sprzętu i oprogramowania zestawu komputerowego

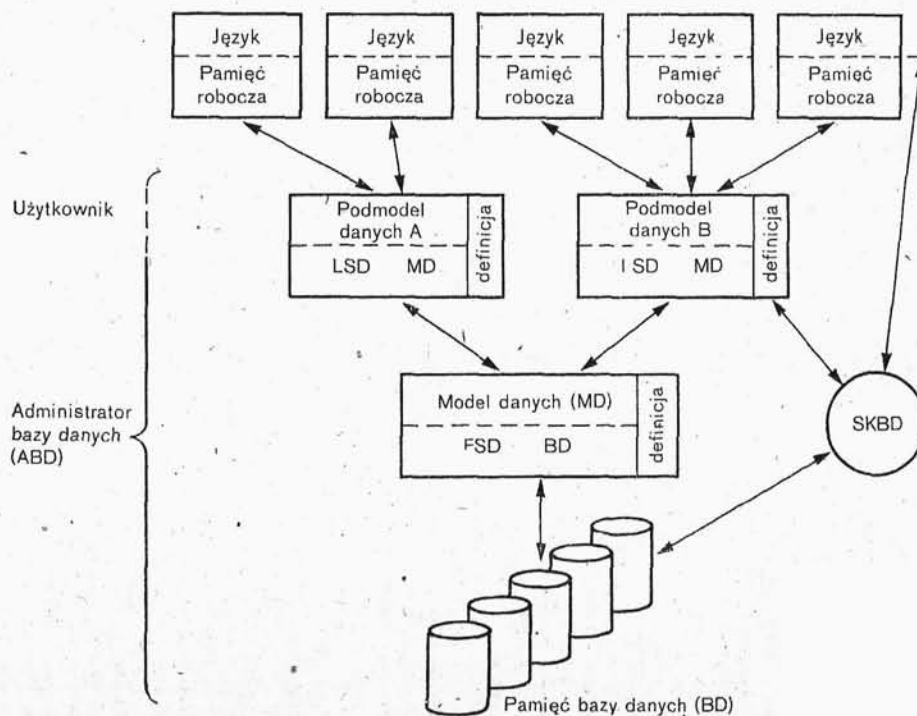


Systemy Kierowania Bazami Danych (SKBD) są rozwiązaniem SKD w zakresie utworzenia z różnych zespołów danych jednego zbioru-bazy danych, w którym dane nie powtarzają się, a wszelkie powiązania między danymi wewnątrz zbioru są możliwe.

Zadaniem SKBD jest pośredniczenie między programem zastosowaniowym a zestawem komputerowym w celu określenia zależności między reprezentacją danych — logiczną (R_L) a fizyczną (R_F). Zatem SKBD musi wybrać odpowiadającą temu fizyczną ścieżkę dostępu (FSD), aby umieścić na niej dane z logicznej ścieżki dostępu (LSD) doprowadzając do postaci

Rysunek 5.16.*Schemat funkcjonowania Systemu Kierowania Bazą Danych*

R_F . Przez FSD rozumie się operacje WE/WY, randomizację lub indeksowanie dostępu itp. Funkcje SKBD ilustruje rysunek 5.16. Posługując się uogólnionym schematem systemu bazy danych (por. rys. 5.17.) przeanalizujemy jego podstawowe składniki na przykładzie dwóch klasycznych rozwiązań: DBTG, CODASYL, IBM IMS.

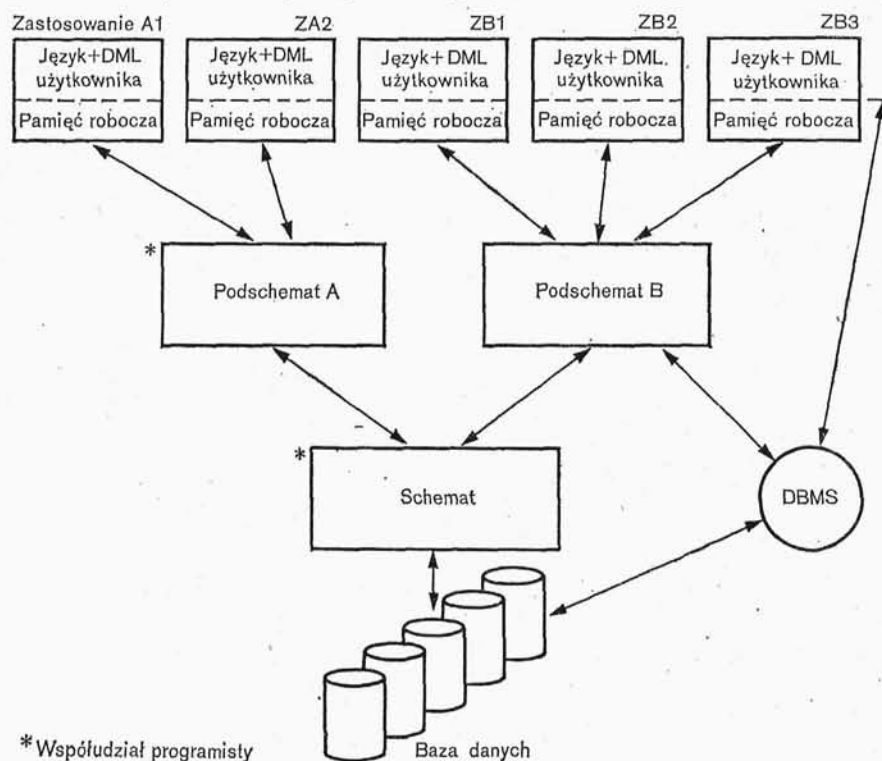
Rysunek 5.17.*Uogólniona architektura systemu bazy danych*

Model danych MD opisuje strukturę elementarnych danych zawartych w bazie danych. W DBTG CODASYL model danych zwany schematem oparty jest na koncepcji łańcuchowej (por. rys. 5.12 a.). W IBM IMS model danych DBD (zwany *Data Base Description*) oparty jest na przeciwstawnej koncepcji hierarchicznej (por. rys. 5.12b.)³⁹. Warto dodać, że struktura hierarchiczna jest szczególnym podzbiorem struktury łańcuchowej.

Każde zastosowanie operuje podmodelem danych (PMD) dostosowanym do jego wymagań. W rozwiązaniu DBTG CODASYL, PMD zwany jest podschematem, który składa się z logicznego podzbioru schematu (por. rys. 5.18.). Ponieważ schemat zawiera opis różnych rodzajów zapisów (rekordów) występujących w MD, dlatego podschemat zawiera wykaz rodzajów stosowanych zapisów, wykaz danych elementarnych schematu (*schema data — item*), które będą występowały w zapisie i jakie mogą być ich powiązania (*schema relations ship-sets*). Z tego wynika, że w podschemata-

Rysunek 5.18.

Architektura systemu bazy danych według DBTG CODASYL



³⁹ Warto zauważyć typową dla IBM chęć zachowania odrębności wobec reszty branży informatycznej.

cie nie może wystąpić żaden rekord i jego powiązania, które nie byłyby uprzednio zdefiniowane w schemacie.

W systemie IBM — IMS model danych przewiduje występowanie wielokrotnych, oddzielnych, logicznych baz danych, które przez programy IMS traktowane są za jedną bazę danych⁴⁰. Każda logiczna baza danych (LBD) definiowana jest w bloku programu łącznościowego (PCB). W PCB zawarta jest definicja LSD (logicznej ścieżki dostępu) do fizycznej bazy danych (PDB), opisanej w modelu danych prac DBD. Innymi słowy, każda LDB ma swój PCB, który z kolei ma swój opis w postaci DBD, kiedy określony program zastosowaniowy, jaki ma korzystać z IMS, wywołuje swój PCB, będący strefą łącznościową między nimi.

Określenie LSD w DBTG CODASYL odbywa się w języku COBOL, a w IBM-IMS realizowane jest w makrorozkazach ASSEMBLERA 360, w ramach bloku specyfikacji programu (PSB).

Każdy użytkownik dla swoich LDB ma swój zbiór PSB.

Definiowanie FSD (fizycznej ścieżki dostępu) w systemie DBTG CODASYL odbywa się w „*device media control language*” (DMCL), języku zbliżonym do OS/360 JOB CONTROL LANGUAGE.

W IBM-IMS definiowanie FSD (fizycznej ścieżki dostępu) odbywa się w ASSEMBLER SYSTEM/360.

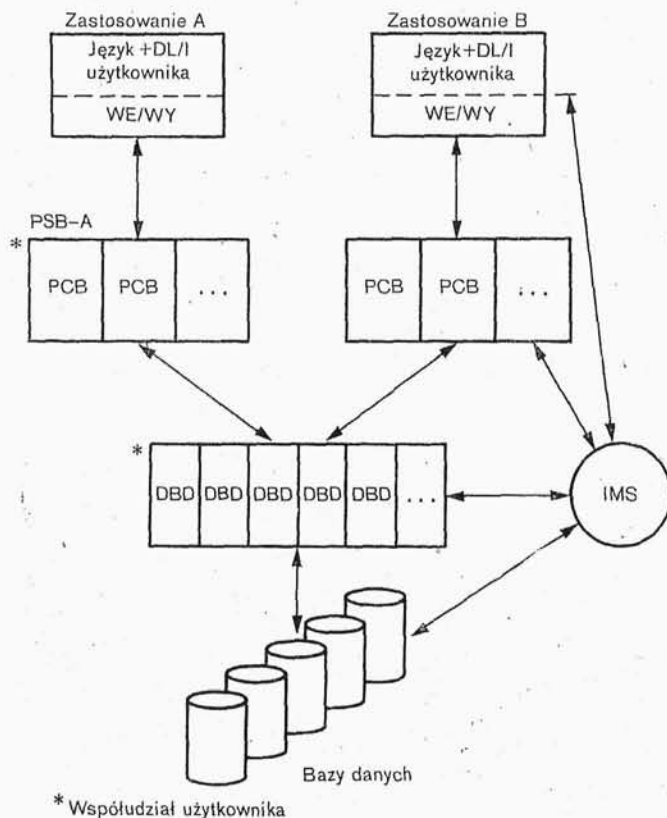
Każde zastosowanie wykorzystujące system bazy danych może władać własnym językiem, np. COBOL, PL-1, ASSEMBLER SYSTEM/360 lub innym. Musi jednak stosować również język danego systemu bazy danych. W DBTG CODASYL język ten nazwano DML (*Data Manipulation Language*), który włączony jest do ANES COBOL, będąc po prostu rozszerzeniem języka COBOL o operacje na zbiorach bazy danych. Do definiowania podszchematu wykorzystuje się podszchematowy DDL (*Data Description Language*), a do określania schematu korzysta się ze schematowego DDL. W systemie IBM-IMS funkcje DML spełnia język DL/I (*Data Language/I*).

Prace nad systemami baz danych rozpoczęły się w połowie lat sześćdziesiątych. W 1965 r. została utworzona Grupa Zadaniowa ds. Baz Danych (*Data Base Task Group*) w Komitecie CODASYLu, przy Komisji Języków Programowania. W 1969 r. została ogłoszona pierwsza COBOLowska wersja działań na bazie danych, obejmująca koncepcje modelu danych (schematu i podszchematu opisywanego przez DDL) oraz 16-czasownikowego języka manipulowania danymi (DML). Po uzyskaniu opinii od zainteresowanych, DBTG opracowała w 1971 r. Raport końcowy. Na podstawie

⁴⁰ Można to rozwiązanie porównać do wielu kartotek, w których wyeliminowano powtarzalność tych samych danych i zabezpieczono łatwe przejście z jednej kartoteki do drugiej. Rysunki 5.17., 5.18. i 5.19. pochodzą z pracy C. J. Dane, *An Introduction to Data Base Systems*, Addison-Wesley, 1975.

Rysunek 5.19.

Architektura systemu bazy danych IBM IMS



tego raportu Zarząd CODASYLu podjął dwa działania. Utworzył Komisję ds. Języka Opisu Danych (DDL — *Data Description Language Committee*), tym samym sankcjonując koncepcję schematu (*schema*) DDL jako typowego wspólnego (*common*) języka do opisu danych, który może być wykorzystywany przez główne języki programowania. Natomiast języki DML i schemat (*subschema*) DDL uznane zostały za rozszerzenie języka COBOL, w celu opracowania specyfikacji tych języków, spójnej ze specyfikacją języka COBOL została utworzona Grupa Robocza ds. Języków Bazy Danych (DBLTG — *Data Base Language Task Group*). W 1973 r. DBLTG przedstawiła swoje opracowanie, w którym m.in. zmieniła w DML i DDL (podschemat) słowa zastrzeżone w COBOLu jak: AREA na REALM, RENAME na ALIAS itp.

Prace Komitetu CODASYL prowadzą do unifikacji rozwiązań języka COBOL i systemu bazy danych. Natomiast Amerykański Krajowy Insty-

tut Norm wykorzystując wymieniony dorobek unifikacyjny prowadzi prace nad normalizacją omawianych rozwiązań, które powinny znaleźć się w normie ANS COBOL.

Warto dodać, że Raport CODASYLu został opracowany na podstawie dorobku firmy GE w zakresie pakietu IDS. Na bazie Raportu CODASYLu zostały opracowane m.in. następujące systemy baz danych:

1) DMS 1100 (Univac) różni się od pierwowzoru brakiem zabezpieczeń na poziomie elementarnych danych (są na poziomie systemu operacyjnego i monitora łącznościowego) oraz brakiem podschematów. Natomiast posiada rozwinięte ułatwienia zapytywania z końcówek.

2) EDMC (Xerox), który w dużym stopniu pokrywa się ze specyfikacją CODASYLu.

3) IDMS (Culliname) opracowany dla maszyn IBM 360/370, SPECTRA 70 i SIEMENS, UNIVAC seria 90 i 70, DEC PDP-11, ICL 1900 i ICL 2900 — odpowiada specyfikacji CODASYLu. Uznany jest za łatwy w użyciu. Jeden z użytkowników stwierdził, że przejście z jego S/I na IBM-IMS wymagałoby 25 osobołat, podczas gdy przejście na IDMS kosztowało około 6 osobołat. Koszt nabycia pakietu, wynosi 37 tys. dolarów (według cen z 1974 r.).

4) DBMS-10 (DEC) dla maszyn PDP-10 odpowiada ściśle specyfikacji CODASYLu.

5) DM6700 (Burroughs), który różni się znacznie w porównaniu z wzorcem.

6) QUERRY UPDATE (CDC), zawiera moduły, które ulepszają przetwarzanie tradycyjnie zorganizowanych kartotek oraz ułatwiają przejście do wspólnej bazy danych.

7) TOTAL (Sincom Systems) jest najpopularniejszym pakietem posiadającym dwukrotnie większą liczbę użytkowników niż IBMowski IMS (na koniec 1977 r. około 1000). Może być wykorzystywany na małym sprzęcie, bowiem niezbędna pamięć jest od 3 K do 30 K bajtów. Opracowany został dla prawie wszystkich popularnych maszyn. Cena od 26 do 34 tys. dolarów.

8) ADABAS (RFN) jest częściowo oparty na łańcuchowym modelu danych z dużym naciskiem położonym na indeksowanie drugorzędne.

Kiedy Komitet CODASYL rozpoczął wspomniane prace, firma IBM, zgodnie ze swoim zwyczajem, nie włączyła się do tych prac, aby wypracować konieczne odmienne rozwiązanie do próbującego się zorganizować środowiska użytkowników. W 1965 r. IBM podjął z użytkownikiem — North American Rockwell Company (NAR) wspólny projekt nad IMS-1, który w 1968 r. został uznany za Program Product. Koncepcja hierarchiczna wprowadzona przez IBM z samej filozofii hierarchiczności jest koncep-

cją restrykcyjną (antydemokratyczną), ograniczającą sprawy rozwojowe, podczas gdy koncepcja łańcuchowa ma charakter otwarty i umożliwia różnokierunkowy rozwój.

A

D #	Nazwisko	Dodatek	Miasto
D1	Nowak	30	Kraków
D2	Kowalski	10	Warszawa
D3	Kowalczyk	20	Warszawa

Rysunek 5.20.

Model danych w zależnościowej bazie danych

B

C #	Nazwa części	Cena	Lokalizacja
C1	Tuleja	500	Warszawa
C2	Wałek	800	Radom
C3	Trzpień	100	Kielce

AB

D #	C #	Wielkość zapasu
D1	C1	100
D1	C2	50
D2	C1	200
D2	C3	60
D3	C2	100

Okazuje się, że najwięcej nadziei pokłada się w rozwoju zależnościowych baz danych, które są o wiele prostsze od dotąd omówionych i bardziej przystosowane do przyzwyczajenia przetwarzaniowych człowieka. Model danych podany został na rysunku 5.20. Charakterystyczny jest dla nich tabelkowy lub kartotekowo-sekwencyjny sposób organizowania zbiorów. Każda tabelka ma jeden typ zapisu i główny klucz. Wartości danych tych samych kolumn (np. numer części w B i AB) tworzą domenę. Dzięki domenie można określać związki między danymi na poziomie ich wartości występujących w domenie⁴¹. Poprzez tworzenie kolejnych tabel otrzymu-

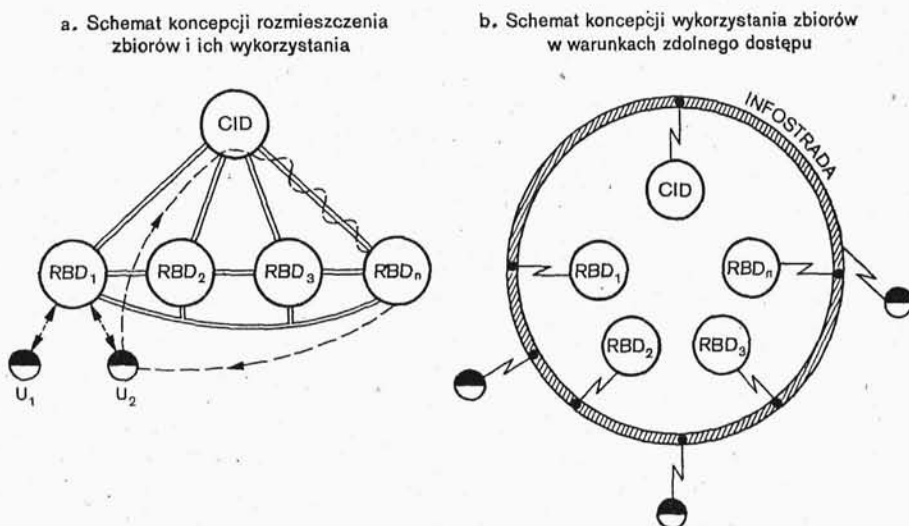
⁴¹ Czytelnik znajdzie opis zależnościowej bazy danych w pracy M. Gniłka. Por. M. Gniłek, *Czy będziemy produkować zależnościowe bazy danych*, „Informatyka” 1977, nr 5.

je się odpowiedź na pytanie. Do ważniejszych systemów zależnościowych baz danych można zaliczyć: LEAP, TRAMP, RELATIONAL DATA FILE, STDS, CAMBRIDGE, Mac AIMS, RDMS i MORIS.

Systemy danych rozdzielonych. Wszystkie dotychczas wymienione systemy kierowania danymi i bazami danych funkcjonują na pojedynczym komputerze, do którego może być zabezpieczony zdalny dostęp. Powoduje to centralizację kierowania i koncentrację danych, jak również zbędną komplikację systemu. Charakterystyczne jest, że żaden pakiet SKBD nie daje się łączyć z innym takim samym na zasadzie *host-end front*. Rozwiązania SKBD wynikają z potrzeby jednokorporacyjnych czy jednoprzedsiebiorstwowych systemów informacyjnych. Gdyby chcieć zorganizować system ogólnokrajowy typu MAGISTER w zdecentralizowanej formie, to żaden z wymienionych pakietów do tego celu by się nie nadawał.

Rysunek 5.21.

Schemat organizacji systemu danych rozdzielonych



W systemie danych rozdzielonych chodzi o to, by zbiory danych (np. w formie baz danych o różnych modelach danych) były przy źródłach danych (chodzi o łatwość i zdecentralizowane aktualizowanie), a dostęp do nich mógł być organizowany z każdego zdecentralizowanego miejsca użytkownika S/I oraz z centralnego punktu. W centralnym punkcie umieszczony jest indeks drugorzędny, który skierowuje zainteresowanego do miejsca, w którym znajduje się odpowiedni zbiór danych, czyli gdzie można uzyskać odpowiedź.

Na rysunku 5.21. pokazano schemat ideowy rozmieszczenia rozdzielonych baz danych (RBD_i), centralnego indeksu drugorzędnego (CID) oraz użytkowników (U_j). Użytkownicy mogą korzystać ze swojego rejonowego RBD_i lub z RBD_{i+n} z innego rejonu. Wówczas o lokalizację RBD_{i+n} zapytują w CID, który go odpowiednio skierowuje. Linie ciągłe ilustrują przepływ danych (aktualizacja, modyfikacja) w systemie RBD, a linie przerywane ilustrują przepływ pytań i odpowiedzi.

W celu realizacji systemu RBD konieczna jest infostrada, która prócz dróg transportu zabezpiecza adresowanie przepływu danych i zapytań — odpowiedzi. W tym celu niezbędne są węzły w infostradzie, zaznaczone ciemnymi kółkami na rysunku 5.21 b. Z rysunku wynika, że infostrada przejmuje funkcje scalania systemu RBD poprzez logikę działania węzłów.

Łączność między danymi (data communication). Pakiety oprogramowania zabezpieczające łączność między zbiorami a użytkownikami zyskują autonomię w stosunku do pakietów oprogramowania gospodarki zbiorami. Do uniwersalnych pakietów łącznościowych można zaliczyć: Intercomm, Environ/1, BEST, Taskmaster, CICS, Hyper-Faster, Multi-Faster.

Na dowód postawionej tezy można wymienić: a) system TOTAL, który wykorzystuje Environ/1 lub Intercomm, b) ADABAS wykorzystujący Intercomm lub CICS oraz także Taskmaster, c) IMS ma własny pakiet, ale również umożliwia wykorzystywanie CICS (IBM).

Katalogowanie danych. Wobec intensywnego rozwoju systemu baz danych rola katalogowania danych zmalała. Słownik danych (*data dictionary*) zawiera źródłowe (*source*) definicje elementarnych danych stosowanych w systemie informatycznym. W spisie danych (*data directory*) umieszcza się wynikowe (*object*) definicje elementarnych danych, po ich przetłumaczeniu z definicji źródłowych. Definicje wynikowe zawierają adresy fizycznej lokalizacji w pamięci.

Katalogowanie danych określane również jako system DD/D przede wszystkim służy do utrzymywania w porządku dokumentacji S/I. System DD/D pomaga w przejściu na system bazy danych.

Można podać przykłady następujących rodzajów definicji:

- 1) definicja danej elementarnej,
- 2) definicja powiązań logicznych danej elementarnej,
- 3) definicja struktury pamięci, w której dana jest przechowywana,
- 4) wykaz synonimów lub dotąd stosowanych definicji,
- 5) definicje metod kontroli danej przed jej wprowadzeniem do S/I,
- 6) definicja prezentacji danej w wydrukach,
- 7) definicja zabezpieczenia dostępu do danej,
- 8) definicja strategii wyszukiwania danej.

Jednym z bardziej popularnych pakietów programowych dla DD/D jest wspomniany w rozdziale 4 pakiet PSL/PSA.

W gospodarce zbiorami istnieje tendencja wysuwania na plan pierwszy funkcji kierowania zbiorami, która przejmuje nadzór nad pozostałymi funkcjami: łączności, zapytywania i katalogowania. Chociaż równocześnie wymienione funkcje drugorzędne często rozwiązywane są w sposób autonomiczny.

5.4. Modele rozwoju teleprzetwarzania danych

Rozwój systemów teleprzetwarzania danych przede wszystkim zależy od wyboru zasady pracy sieci teleprzetwarzaniowej oraz jej struktury.

Przyjmując za kryterium ładunek informacyjny podlegający przesyłaniu można wyodrębnić następujące trzy zasady pracy sieci teleprzetwarzaniowej:

- a) zasada przełączania kanałów (*circuit switching*),
- b) zasada przełączania komunikatów (*message switching*),
- c) zasada przełączania pakietów (*packed switching*).

W zasadzie a) węzeł sieci łączy kanał nadawczy z odbiorczymi i utrzymuje połączenie w czasie trwania transmisji. W wypadku zajętości kanału inicjacja połączenia jest ponawiana, ponieważ węzeł nie ma pamięci. Wymieniona trudność nie występuje w zasadzie b), która polega na tym, że informacja zorganizowana w komunikat (*message*) zostaje przesłana do węzła i sprawdzona pod względem poprawności przesłania, po czym kanał nadawczy rozłącza się z węzłem. Przekazanie komunikatu z węzła do odbiorcy odbywa się w momencie niezajętości odbiorcy. Ten tryb transmisji określany jest również jako *store and forward*.

Zasada c) różni się od b) tym, że porcje informacji przesyłane są według stałej pojemności zbioru, zwanej pakietem. Komunikat do danego odbiorcy może się składać z paru skompletowanych pakietów. Zadaniem węzła jest m.in. uporządkowanie pakietów.

Wybór zasady pracy sieci teleprzetwarzaniowej zależy od przeznaczenia sieci, głównie od częstotliwości przesyłania informacji. Im częstotliwość jest niższa, tym lepiej stosować jest prostszą zasadę.

Wśród struktur sieci teleprzetwarzaniowych wyróżnia się następujące struktury:

- gwiazdzistą,
- gwiazdzisto-niepełną,
- pierścieniową,
- pierścieniowo-gwiazdzistą.